A Categorical and Logical Analysis of the π-calculus
（π 計算の圏論的および論理的分析）

by

Ken Sakayori
酒寄健

A Doctor Thesis
博士論文

Submitted to
the Graduate School of the University of Tokyo
on December 4, 2020
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Information Science and Technology
in Computer Science

Thesis Supervisor: Naoki Kobayashi　小林直樹
Professor of Computer Science

ABSTRACT

A commonly accepted mathematical formalism for describing and proving properties of concurrent systems is the π-*calculus*. The π-calculus provides various operational techniques for reasoning about the behavior of systems, and this is one of the reasons why the π-calculus has been widely used.

Although the operational aspects of π-calculus have been well-studied, the connection between the π-calculus and other mathematical objects is not well-understood. This is in contrast to the case of λ-calculus, which is a formal calculus often used as the basis for functional programming. The correspondence between simply typed λ-calculus, cartesian closed categories and intuitionistic logic is well-known. Such correspondence has not been discovered for the π-calculus, except for session-typed variants of the π-calculus in which only a limited form of concurrency is expressible. The lack of such correspondence has been preventing the transfer of established techniques from logic and category theory to the field of concurrency.

This thesis makes an effort to extend the three-way correspondence between computation, categories and logic to the π-calculus (not limited to the session-typed variants).

First, we develop a correspondence between a categorical structure, which we call *compact closed Freyd category*, and a variant of the π-calculus, which we call the π_F-*calculus*. Both the compact closed Freyd category and the π_F-calculus are novel and are carefully defined so that they correspond to each other. Although compact closed Freyd category is introduced for a particular purpose, i.e. establishing a correspondence with the π-calculus, its definition is fairly standard: compact closed Freyd categories combines two well-known structures, namely, closed Freyd category and compact closed category. The former is a model of higher-order effectful language, and the latter describes connections via channels. To demonstrate the relevance of the categorical model, we reconstruct the classical results on the relation between higher-order languages and the π-calculus by a simple semantic consideration using this model.

The categorical analysis of π_F-calculus reveals a fundamental difficulty in developing a categorical type theory for the π-calculus. We show (modulo some reasonable assumption) that conventional behavioral equivalences for the π-calculus are inherently incompatible with categorical semantics. The root cause of this problem lies in the mismatch between the operational and categorical interpretation of a process called the forwarder. From the operational viewpoint, a forwarder may add an arbitrary delay when forwarding a message, whereas, from the categorical perspective, a forwarder must not add any delay when forwarding a message. As an attempt to overcome this gap, we introduce a novel operational semantics for the π_F-calculus in which forwarders do not cause any delay. We then show that this new operational semantics (i) is compatible with the categorical semantics and (ii) can simulate the standard operational semantics.

As for the relation with logic, we discuss the relation between π_F-calculus and linear logic. The relation between linear logic and the π-calculus has been intensively explored since the early phase of the study of the π-calculus. Among others, Abramsky (1994) and Belling and Scott (1994) showed that linear logic proofs can be interpreted using π-calculus processes. Later Caires and Pfenning (2010) showed that if we limit our focus to session-typed processes, the converse also holds, i.e. session typed processes can be interpreted as linear logic proofs. However, it remained an important open problem whether there exists a proof system that is not only interpretable by π-calculus, but that can also interpret the traditional π-calculus processes. We provide a novel observation to this question by constructing an (inconsistent) proof system that one could reasonably expect to correspond to the π_F-calculus. Our construction makes use of the relation between linear logic and compact closed Freyd category. Since a compact closed Freyd category is a specific instance of a categorical model of linear logic, analyzing the difference between general categorical models for linear logic and compact closed Freyd categories leads us to an extension of linear logic that corresponds to π_F-calculus.

論文要旨

π **計算**は並行システムを記述したり，並行システムの性質を証明する際に広く使われている形式体系である．π 計算が広く使われている理由のひとつに π 計算はシステムの性質を解析する際に使える様々な操作的な手法を提供してくれる点が挙げられる．

π 計算の操作的側面は広く研究されているものの，π 計算と他の数学的対象との関係性はよく理解されているとはいえない．この状況は，関数型言語の論理的基盤である λ 計算をとりまく状況とは異なっている．単純型付 λ 計算，デカルト閉圏，直観主義論理の間の対応は十分理解されている．一方，セッション型付 π 計算という並行性が強く制限された π 計算を除くと，π 計算に対してはそのような対応関係は発見されていない．対応関係がないことは，圏論や論理の手法を並行計算の分野へ流用することの妨げになっていた．

本論文は，計算，圏，論理の対応を（セッション型付でない）π 計算に拡張することを試みる．

初めに，本論文は**コンパクト閉フライド圏**という圏論的構造と π_F **計算**という π 計算の部分体系の対応をあらわにする．コンパクト閉フライド圏も π_F 計算もともに本論文で導入されるものであり，両者は対応関係をもつように注意深く設計されている．コンパクト閉フライド圏は π 計算と対応することを目的として設計されたものであるが，その定義は標準的なものである．コンパクト閉フライド圏はコンパクト閉圏と閉フライド圏という 2 つのよく知られた圏論的構造を組み合わせることによって定義されている．前者は計算効果を含む高階言語のモデルであり，後者はチャネルのネットワーク構造を記述するのに使われる構造である．本モデルの有用性を示すために，π 計算と高階のプログラミング言語の関係性に関した古典的な結果を本モデルを使った意味論的考察から導出できることを示す．

π_F 計算の圏論的分析は，π 計算の圏論的型理論を発展させる上での本質的な難しさを明らかにする．本論文では（適当な仮定のもとで）伝統的な π 計算の振る舞い等価性は圏論的意味論と相容れないことを示す．この問題の根本的な原因はフォワーダーとよばれる特殊なプロセスの操作的・圏論的解釈の相違である．操作的な観点からはフォワーダーはメッセージを転送する際に遅れを発生させることができるのに対し，圏論的な解釈ではフォワーダーが転送時に遅れを発生させることは許されていない．この差異を埋めるための試みとして，本研究では π_F 計算の操作的意味論をフォワーダーが遅れを発生させないように変更する．そして，この新しい操作的意味論が (i) 圏論的意味論と矛盾がないこと (ii) π 計算の標準的な操作的意味論を模倣できることを示す．

論理との関連については，π_F 計算と線形論理の関係性について議論する．線形論理と π 計算の関係は π 計算の研究の黎明期から調べられてきた．中でも，Abramsky(1994) および Bellin と Scott(1994) は線形論理の証明がプロセスを使って解釈できることを発見し，その後 Caires と Pfennning が，対象をセッション型付プロセスに絞れば，プロセスを線形論理の証明として解釈できることを発見した．しかし,「証明をプロセスとして解釈できるだけでなく，伝統的な π 計算のプロセスを証明として解釈できる論理体系が存在するか」は重要な未解決問題である．本論文では，π_F 計算と対応すると期待できる（無矛盾でない）論理体系を構成することで，この問いに対する新たな観察を与える．この論理体系の構成には線形論理とコンパクト閉フライド圏の関係を利用する．コンパクト閉フライド圏は線形論理の圏論的モデルの特殊な場合であるから，一般の線形論理の圏論的モデルとコンパクト閉フライド圏を比較することで π_F 計算と対応する線形論理の拡張を導出する．

Acknowledgements

I would like to express my best gratitude to Naoki Kobayashi and Takeshi Tsukada for their tremendous support during my PhD program. My supervisor Naoki Kobayashi gave many influential comments on my work. His dedication to research and teaching also had a great influence on my attitude to research; I always took him as a role model. Most of the work of this thesis is based on joint work with Takeshi Tsukada. Although he was not my official supervisor, I feel like I am his apprentice. Words cannot describe how much I want to thank him.

I am also grateful to my thesis committee members, Masami Hagiya, Masahito Hasegawa, Akihiko Takano, Yusuke Miyao and Shinya Takamaeda, for their careful reading. In particular, I was fortunately given opportunities to discuss with Masahito Hasegawa of Kyoto University at the very early stage of my research process. From him, I learned a lot about categorical semantics.

I would like to thank the members and ex-members of the Kobayashi Laboratory for their help and companionship that made my last five years enjoyable. Especially, I would like to thank Ryosuke Sato, Kazuyuki Asada and Ryoma Sin'ya for teaching me various topics on theoretical computer science. Special thanks go to Keiichi Watanabe for the daily chat that often relieved my stress, which somehow kept going using Slack after he graduated.

I appreciate my family for their warm encouragement and support not only during the PhD program but ever since I was born. Thanks to my parents, I had an opportunity to spend my childhood in Europe, and this experience definitely helped me interact with researchers from other countries.

The research described in this thesis could not have been accomplished without the financial supports I got during my PhD program. I was supported by JSPS's Research Fellowship for Young Scientists and the IST-RA program of the Graduate School of Information Science and Technology.

Contents

Chapter 1

Introduction

1.1 Background

1.1.1 Reasoning concurrent programs

Writing concurrent program is hard. Programmers have to consider how different threads of programs interact or how they should synchronize. One can easily write a program that ends up in a deadlock or a race condition.

One way of writing concurrent programs is by message passing. In this approach, threads (or processes) interact by explicitly passing data via channels. This approach has benefits over the shared memory approach that are common in practice. In the shared memory approach one has to deal with locks in order to guarantee that a thread has exclusive access. On the other hand, the message passing style abstracts away the notion of locks and this style of programming keeps the code structured. Nowadays, this message passing style is becoming more popular, and there are several languages that support this style of programming. For instance, languages such as Go [2], Rust [26] and Erlang [1] have lightweight threads and message passing between them as builtin language features.

Though message passing is an elegant way to express concurrent computation, reasoning about programs written in message-passing style is still challenging. We still need to consider things such as whether the nondeterministic feature of the program does not affect the expected behavior, whether communications do not fall into deadlocks at runtime, or whether a particular channel will be eventually used. Also asking whether two concurrent programs have the same behavior is far from trivial.

To rigorously reason about concurrent programs, we need a suitable mathematical foundation to work on. Ideally, the mathematical foundation should have a high-level of abstraction but at the same time be able to describe existing systems.

Very broadly speaking, this thesis is an attempt to give a (yet another) mathematical foundation against message passing style communication. The weight is put on the study of the (denotational) semantic and logical aspects of the message passing concurrency.

1.1.2 π-calculus

This thesis is concerned with the π-*calculus*, which is the one of the most studied formalisms for describing (message passing style) concurrent systems. The π-calculus is a kind of a process calculus originally introduced by Milner, Parrow and Walker [94, 95]. The distinguished feature of π-calculus is that π-calculus allows channel names to be communicated along the channels themselves. Actually, in π-calculus, channel passing is the only computational step: every computation is modeled by channel passing.

One of the characteristics of the π-calculus is the conciseness of its syntax. The syntax of (an asynchronous) π-calculus is given by the following grammar:[1]

$$P ::= \mathbf{0} \mid \bar{a}\langle x\rangle \mid a(x).P \mid (P \mid Q) \mid (\boldsymbol{\nu} a)P \mid !P$$

Arguably the most important constructs of the π-calculus are $\bar{a}\langle x\rangle$, $a(x).P$ and $P \mid Q$. These expresses *output action*, *input prefixing* and *parallel composition*, respectively. The remaining constructs $\mathbf{0}$, $(\boldsymbol{\nu} a)P$ and $!P$ are *inaction* (i.e. a process that does nothing), *name restriction* that is used to restrict the scope of a name and *replication* of P that is used to create as many parallel replicas of P as needed. The name restriction $(\boldsymbol{\nu} a)P$ is also called as *name creation* because it can be regarded as an operation that creates a private name.

In the π-calculus, computation proceeds by communications. For instance the following rule is the base case of the reduction relation:

$$a(x).P \mid \bar{a}\langle y\rangle \longrightarrow P\{y/x\} \mid \mathbf{0} \tag{1.1}$$

The above relation expresses that the channel y is transmitted along the channel a. When the receiver $a(x).P$ receives the name y, the continuation $P\{x/y\}$ is executed.

Though the syntax and the reduction relation of π-calculus are very simple, π-calculus is very expressive. It is well-known that π-calculus can encode functional languages such as λ-calculus [90] as well as object oriented models [134].

Because of its simplicity and expressiveness, π-calculus has been (and probably will be) a mathematical model for computation used in various researches.

1.1.3 Curry-Howard-Lambek correspondence

However, what we are interested in is not the syntactical aspects of the π-calculus. This thesis investigates the semantic and logical aspects of the π-calculus. History tells us that insight from logic and category theory are very useful in the study of programming languages. We shall briefly review the impact of logic and category theory on the study of programming languages.

The case of λ-calculus When we turn our attention to the study of (sequential) functional programming languages, the role λ-calculus has played is significant. For example, it has been used for proving various properties of programs and typed lambda calculi play an essential role in the design of type systems for programming languages. In a sense, π-calculus is a calculus that aims to follow the success of the λ-calculus.

One reason why λ-calculus has played a central role in the study of functional programs and programming languages is that λ-calculus has a deep connection with logic. The (simply typed) λ-calculus and intuitionistic propositional logic are the two side of the same coin: types are identified with propositions and programs are identified with proofs. This correspondence, also known as the Curry-Howard correspondence [27, 58], enabled the cross-fertilization of the study on

[1]The calculus presented in this section is different from that of the target calculus used in this thesis. We will use this calculus throughout this *chapter* for presentational purposes; this calculus is simpler than the target calculus of this thesis and is closer to the original π-calculus given by Milner et al. [94, 95], which some readers might be familiar with.

programming languages and logics. The correspondence also demonstrates that the design of the λ-calculus is not arbitrary, but canonical. The correspondence between typed λ-calculi and logical systems are surprisingly robust in the sense that it scales well beyond the identification of simply typed λ-calculus and intuitionistic propositional logic. For example, the tight connection between quantification over propositional variables and polymorphism; classical logic and control operators [100] and modal logic and staged computation [31] has been discovered.

In some cases, there is a third object, namely categorical structure, that fits into this correspondence. In the case of the simply typed λ-calculus, Lambek [76, 77] discovered that simply typed λ-calculus (with pairing) is essentially the same thing as a categorical structure called cartesian closed category. Hence, the correspondence among a programming language, logical system, and categorical structure is called the *Curry-Howard-Lambek correspondence*. From a programming language perspective, this correspondence is important because it says that cartesian closed categories capture the semantics of the simply-type λ-calculus. This allows us to quickly recognize whether a certain mathematical structure is a model of the λ-calculus without going into the details, and serve as a guide when one looks for a specific model of λ-calculus. The connection between category theory and programming language also allows us to employ categorical notions to describe features of programming languages. This is very useful because category theory often captures analogous notions in a unified manner. For example, monads have become standard tools for describing various "computational effects" [97, 98].

Because of these advantages Curry-Howard-Lambek correspondence offer, designing a language in a way that it corresponds to certain logic or categorical structure is a principle that is adopted in the research of programming languages.

The case of π-calculus The connection between π-calculus and logical systems, especially *linear logic*, has been intensively explored since the birth of the π-calculus. Linear logic [43] is a logic that emphasizes the role of formulas as resources. In linear logic, assumptions can be considered as resources that interact and consumed as the proof goes on. Influence of linear logic to π-calculus can already be found in the syntax of the π-calculus. The replication operator $!P$ seems to be inspired by the exponential modality $!A$, which is the unlimited copy of the formula A.

An early work by Abramsky [3], which was then followed by Bellin and Scott [11], revealed that linear logic proofs can be interpreted as π-calculus processes. However, these studies were not considered as a Curry-Howard correspondences, since the opposite direction, i.e. interpreting processes using proofs, was not possible.

The work by Caires and Pfenning [18] was then a turning point, providing a precise correspondence between (dual intuitionistic) linear logic and *session-typed* π-calculus. They identified session-types with linear logic formula, typed processes with proofs and process reduction with cut elimination of a proof. Session types [53, 56] are types that describes the protocol that (two) processes of a concurrent system needs to obey. This precise relationship against linear logical operators and session types was later extended to classical linear logic by Wadler [132] and Caires et al. [19]. These correspondences can be regarded as Curry-Howard-Lambek correspondences, though the connection to category theory was not explicitly given in these work, since the connection between linear logic and category theory has already been well-known [120, 12, 84].

The influential work by Caires and Pfenning [18] and Wadler [132] inspired lots of work on session-typed calculi (not limited to the π-calculus) that are based on logical insights or techniques (e.g. [79, 8, 29, 70]).

1.2 Research Problem and Aim of This Thesis

Though there is no doubt that the correspondence between session-type π-calculus and linear logic is a remarkable result, this correspondence is not completely satisfactory for a Curry-Howard correspondence of the π-calculus.

One limitation of the linear logic based session typed-calculi is the lack of expressiveness. The session-typed calculi [19, 133] corresponding to linear logic have only well-behaved processes because the session type systems guarantee deadlock-freedom and race-freedom of well-typed processes. This strong guarantee is often useful for programmers but can be seen as a significant limitation of expressive power. Wadler—the founder of a linear logic based session typed calculus called CP—also mentions to this point. The following text is taken from his paper [133]:

> a foundation for concurrency based on linear logic will be of limited value if it only models race-free and deadlock-free processes. Are there extensions that support more general forms of concurrency?

Another problem is that there are quite a few type systems of the π-calculus that are not based on session type. Historically, types (aka sorting) for π-calculus was introduced to "specify what kind of data the channel can receive or send" [93] rather than to describe the interactive behavior between processes. This type has been refined and extended in many ways: by distinguishing the input and output capability [106], adding the notion of linearity [69], adding polymorphism [130] and so on. We shall call these types "standard types" following [30]. Since "standard types" and session types are quite different, transferring techniques used in logic to the studies on π-calculus that uses "standard types" via the Curry-Howard correspondence between linear logic and session-typed calculi is not that easy. Moreover, "standard types" for π-calculus may be considered more primitive than session types because session types can be encoded into standard types (with linear channels [69]) [67, 30]. "Standard types" are also the classification that is used in the majority of the practical languages that supports message-passing style programming. Languages such as Go, Erlang and Rust uses the "standard types" for channels, and session types are not supported as primitive language features. Therefore, a logical or categorical foundation for "standard types" is also of demand.

The aim of this thesis is to give a solution to these problems. We introduce a variant of the "standard typed" (**i**/**o**-typed [106] to be specific) π-calculus and a new categorical structure that corresponds to this calculus in the same way that a cartesian closed category is a model of the λ-calculus. The calculus we introduce is expressive enough to model wild concurrent systems or programs that might fall into deadlocks or race conditions.

1.3 Our Approach

In order to establish a Curry-Howard-Lambek correspondence for the π-calculus we first need to exhibit the algebraic structure underlying the π-calculus. Here we briefly discuss how we identify the categorical structure that corresponds to the π-calculus.

The starting point of our work is the observation that π-calculus can be regarded as a *higher-order language* that communicates programs rather than a (first-order) language that communicates names. This observation is not new and has been pointed out since the early days of research on the π-calculus; for instance Sangiorgi [115] revealed that adding higher-order features to π-calculus does not increase the expressive power of the π-calculus.

The key idea is to decompose the input prefixing $a(x).P$ into the name a and an abstraction $(x).P$. Let us write $a \mathbin{@} (x).P$ for $a(x).P$ to emphasize this decomposition. The abstraction $(x).P$ is just an alternative notation for the λ-abstraction of the λ-calculus. Then the reduction relation (1.1) can also be decomposed as follows:

$$\begin{aligned} a \mathbin{@} (x).P \mid \bar{a}\langle x\rangle &\longrightarrow \mathbf{0} \mid ((x).P)\ \langle y\rangle \\ &\longrightarrow \mathbf{0} \mid P\{y/x\} \end{aligned}$$

(The order of the parallel composition is not important: $\mathbf{0} \mid P\{y/x\}$ and $P\{y/x\} \mid \mathbf{0}$ are processes with same behavior) The second step of the above reduction is the β-reduction (in the λ-calculus), i.e. $(\lambda x.P)y \longrightarrow P\{y/x\}$. The first step expresses a communication where the function $(x).P$ is passed from a to $\bar{a}$. (Note that the direction of the communication is the opposite of that of the message passing interpretation of (1.1).) Therefore, in this view, we regard

- an output action $\bar{a}\langle y\rangle$ as a function application
- an input prefixing as an abstraction $(x).P$ (or $\lambda x.P$) located at a location (or a name) a

Now the core operations of the π-calculus are decomposed into the better known constructs of a higher-order (functional) language. Therefore, the π-calculus can be thought of as a functional language with some uncommon operations such as $a \mathbin{@} (P)$ or name creation $(\boldsymbol{\nu} a)P$. It should be emphasized that this language is *effectful*; communications and name creations are usually considered as a computational effect.

This observation leads us to base our categorical model for the π-calculus on a categorical model for effectful functional languages. The categorical models for effectful functional languages are well-studied. The seminal work by Moggi [97] presented the idea that computational λ-calculus can be modeled by (strong) monads. Our categorical model for the π-calculus uses a categorical structure that is equivalent to the monadic model of the computational λ-calculus, which is called the *closed Freyd category* [111].

Now it remains to identify the structure behind the operations $a \mathbin{@} (-)$ and $(\boldsymbol{\nu} a)P$. Our claim is that these operations can be modeled using the *compact closed* structure. *Compact closed categories* [63] are categories that has an associated graphical language of "string diagrams" that is expressive enough to depict various networks. The importance of compact closed categories in the context of concurrency theory has been pointed out by Abramsky et al. [5]. They pointed out that compact closed categories (with some additional structures) can nicely model CCS-like processes interconnected via ports. Since $a \mathbin{@} (x).P$ can be seen as a "connection of a name a and abstraction $(x).P$", compact closed structure can be used to model $a \mathbin{@} (-)$. The name creation $(\boldsymbol{\nu} a)$ can also be modeled by a morphism of a compact closed category. Each compact closed category is equipped with a special morphism (parameterized by objects A) $\eta_A \colon I \to A \otimes A^*$ called the unit. Here the object A^* is called the dual of A. If we consider I

as the unit type and $\otimes$ as a product η_A can be considered as an operation that returns a pair of elements of type A and its dual A^*. This can be considered as an operation that creates the two endpoints of a channel (by considering the input-end as the dual of output-end), which is the operation $(\boldsymbol{\nu} a)$.

Putting the two structures described above together, we introduce a categorical structure called *compact closed Freyd category* as a categorical model of the π-calculus. Despite its simplicity, compact closed Freyd category captures the strong expressive power of the π-calculus. The compact closed structure allows us to construct the network topology in an arbitrary way, in return for the possibility of deadlocks whereas the Freyd structure allows arbitrary duplication of channels, in return for the possibility of race conditions.

1.4 Contributions

The main contribution of this thesis is the introduction of a new variant of the **i**/**o**-typed π-calculus, which we call π_F-calculus. A remarkable feature of the π_F-calculus is that it has a categorical counterpart, called compact closed Freyd category. Using the π_F-calculus and compact closed Freyd category, this thesis investigates and re-examine various aspects of the π-calculus, including its relation to linear logic, relation to higher-order calculi and its operational semantics.

More concretely, contributions of this thesis can be summarized as follows:

Introduction of a variant of the i/o-typed π-calculus with a categorical counterpart As mentioned, this thesis introduces a new variant of the **i**/**o**-typed π-calculus, which we call the π_F-calculus. The syntax and the axiomatic semantics of the π_F-calculus is carefully designed so that it has a corresponding categorical structure. The corresponding categorical structure, the compact closed Freyd structure, is also introduced by this thesis. The correspondence is fairly firm: the categorical semantics is sound and complete, and the term model is the classifying category.

Indication of a fundamental difficulty in developing a categorical type theory for the π-calculus The categorical analysis of this thesis reveals that many conventional behavioral equivalences for the π-calculus are problematic from the viewpoint of categorical type theory. The problem is that they induce only semi-categories, which may not have identities for some objects. This is reminiscent of the β-theory of the λ-calculus, of which the categorical model is given by semi-categorical notions [50]. We formally show that this problem is inevitable modulo some mild assumptions.

Provision of insights on the relationship between the π-calculus and linear logic Though we now know that session-typed π-calculus corresponds to linear logic, it is still an open problem to find a proof system that corresponds to the standard π-calculus. We provide a reasonable candidate by using π_F-calculus and compact closed Freyd categories. We construct an extension of linear logic in which we conjecture that π_F-processes can be interpreted. This logical system is obtained by making use of the fact that compact closed Freyd categories are specific instances of a categorical model of linear logic (linear/non-linear models [12]). By investigating the difference between general categorical models of linear logic and compact closed Freyd category, we discover the additional logical axioms that should be added to linear logic in order to interpret π_F-processes.

Semantic explanation against the relationship between higher-order calculi and the π-calculus We demonstrate the relevance of compact closed Freyd category by giving a semantic reconstruction of Sangiorgi's translation between the higher-order π-calculus and the (first-order) π-calculus. This is done by introducing another calculus, which can be seen as an extension of the higher-order π-calculus, that corresponds to the compact closed Freyd category. Since this calculus and π_F-calculus both corresponds to the same categorical structure, we immediately get a semantic-guided syntactic translation between these calculi. We show that these translations are essentially the same as the ones given by Sangiorgi [118]. The relationship between the π-calculus and other higher-order calculi, such as the λ-calculus, is also re-examined using this coincidence.

Development of an operational semantics that harmonizes with categorical semantics As explained above, asynchronous π-calculus modulo most of the behavioral equivalences do not form categories. We diagnose the nature of this problem and attempt to fill the gap by introducing a new operational semantics that is compatible with the categorical semantics. We introduce a novel reduction semantics to the π_F-calculus and show that π_F-calculus modulo weak barbed congruence, defined on top of the new reduction semantics form a compact closed Freyd category. The new operational semantics is quite complex since it treats a multi-step reduction in the conventional reduction semantics as a one-step reduction. To reason about such a complicated semantics, we develop an intersection type system, or equivalently a system of linear approximations [128, 83], that captures the behavior of a process, which we think would be of independent technical interest.

1.5 Outline of the Thesis

The rest of this thesis is organized as follows. Chapter 2 covers preliminaries for the thesis. Chapter 3 introduces the π_F-calculus, the target language of this thesis. Chapter 4 introduces the notion of compact closed Freyd category, and shows the correspondence between the π_F-calculus and compact closed Freyd category. Chapter 5 compares the π_F-calculus with linear logic, and Chapter 6 re-examines the relationship between higher-order calculi and the π-calculus through the lens of categorical semantics. Chapter 7 introduces a novel operational semantics to the π_F-calculus and shows that it harmonizes with the categorical semantics. Chapter 8 discusses related work and Chapter 9 concludes the thesis.

First appearance Most of the part of Chapter 3, 4 and 6 are based on a previously published paper [114]. Chapter 5 is also an extended version of the discussion given in [114] with some new results. (The paper [114] is an open access paper licensed under cc by 4.0.)

Chapter 2

Preliminaries

This chapter is devoted to preliminaries. Section 2.1 introduces some notations used throughout this thesis. In Section 2.2, we review some categorical notions that are used in the following chapters.

2.1 Notational Conventions

The set of natural numbers (starting from 0) is denoted by **Nat**. We define the set $[n]$ as $\{1, \ldots, n\}$ for a natural number n; if $n = 0$ then $[n]$ is the empty set $\emptyset$.

The notation $\stackrel{\text{def}}{=}$ is used for "equality by definition". We also use $::=$ for the introduction of a syntactic category defined by the Backus-Naur form. We write $\vec{x}$ for a possibly empty sequence $x_1, \ldots, x_n$, provided that every x_i belongs to the same syntactic category, and write $|\vec{x}|$ for the length n of the sequence $\vec{x}$. The capture-avoiding substitution is denoted by $N\{M/x\}$ and we also write $N\{\vec{M}/\vec{x}\}$ for the simultaneous substitution provided that $|\vec{x}| = |\vec{M}|$.

2.2 Category Theory

The aim of this section is to provide some background material on category theory that will be used in this thesis. Definitions that are given in this section include monoidal category, monoidal functors and compact closed category. Readers familiar with these notions may skip this section.

2.2.1 Prerequisites

We assume that the readers are familiar with basic categorical theory. It is sufficient to know the definition of a (plain) category, functors, adjunctions, cartesian closed category and (co)monads. Standard references are [80, 113].

2.2.2 Notations

We use the following notations.

$\mathcal{C}, \mathcal{D}, \dots$	categories
$f; g \colon A \to C$	composition of morphisms $f \colon A \to B$ and $g \colon B \to C$
$\mathbf{Obj}(\mathcal{C})$	collection of objects in $\mathcal{C}$
$\mathcal{C}(A, B)$	homset consisting of morphisms from A to B in $\mathcal{C}$
$\langle f, g\rangle \colon X \to A \times B$	pairing of $f \colon X \to A$ and $g \colon X \to B$
$\pi_i^{A_1 \times A_2} \colon A_1 \otimes A_2 \to A_1$	the i-th projection where $i \in \{1, 2\}$
$\mathbf{d}_A \colon A \to A \times A$	diagonal map of A in a category with finite products
$!_A \colon A \to 1$	the unique morphism to the terminal object

Be warned that we write composition in "diagram order".

2.2.3 Monoidal categories and compact closed categories

Here we review the notion of (symmetric) monoidal category and compact closed category and the structure preserving functors between them. The notion of natural transformations between these structure preserving functors is also reviewed.

Definition 2.1 (Monoidal category [80])**.** A *monoidal category* is a category $\mathcal{C}$ with

- a bifunctor $\otimes \colon \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ called the monoidal product
- an object I of $\mathcal{C}$ called the unit object
- a natural isomorphism $\alpha_{A,B,C} \colon (A \otimes B) \otimes C \cong A \otimes (B \otimes C)$ called the associator
- a natural isomorphism $l_A \colon I \otimes A \cong A$ called the left unitor and a natural isomorphism $r_A \colon A \otimes I \cong A$ called the right unitor

such that the following two diagrams commute:

$$\begin{tikzcd}
((A \otimes B) \otimes C) \otimes D \arrow[r, "\alpha_{A\otimes B,C,D}"] \arrow[d, "\alpha_{A,B,C}\otimes \mathrm{id}_D"'] & (A \otimes B) \otimes (C \otimes D) \arrow[r, "\alpha_{A,B,C\otimes D}"] & A \otimes (B \otimes (C \otimes D)) \\
(A \otimes (B \otimes C)) \otimes D \arrow[rr, "\alpha_{A,B\otimes C,D}"] & & A \otimes ((B \otimes C) \otimes D) \arrow[u, "\mathrm{id}_A\otimes \alpha_{B,C,D}"]
\end{tikzcd}$$

$$\begin{tikzcd}
(A \otimes I) \otimes B \arrow[r, "\alpha_{A,I,B}"] \arrow[dr, "r_A\otimes \mathrm{id}_B"'] & A \otimes (I \otimes B) \arrow[d, "\mathrm{id}_A\otimes l_B"] \\
& A \otimes B
\end{tikzcd}$$

A *strict monoidal category* is a monoidal category of which $\alpha_{A,B,C}$, l_A and r_A are identities, for every $A, B, C \in \mathbf{Obj}(\mathcal{C})$. ■

Definition 2.2 (Symmetric monoidal category [80])**.** A monoidal category $(\mathcal{C}, \otimes, I)$ is a *symmetric monoidal category* if it is equipped with a natural isomorphism

$c_{A,B} \colon A \otimes B \cong B \otimes A$ called the *symmetry* (or braiding) such that

$$\begin{tikzcd}
(A \otimes B) \otimes C \arrow[r, "\alpha_{A,B,C}"] \arrow[d, "c_{A,B} \otimes \mathrm{id}_C"'] & A \otimes (B \otimes C) \arrow[r, "c_{A,B\otimes C}"] & (B \otimes C) \otimes A \arrow[d, "\alpha_{B,C,A}"] \\
(B \otimes A) \otimes C \arrow[r, "\alpha_{B,A,C}"] & B \otimes (A \otimes C) \arrow[r, "\mathrm{id}_B \otimes c_{A,C}"] & B \otimes (C \otimes A)
\end{tikzcd}$$

$$\begin{tikzcd}
A \otimes B \arrow[r, "c_{A,B}"] \arrow[dr, "\mathrm{id}_{A\otimes B}"'] & B \otimes A \arrow[d, "c_{B,A}"] \\
& A \otimes B
\end{tikzcd}$$

commutes. ■

It is known that any monoidal category is equivalent to a strict monoidal category, thanks to coherence theorem [80]. However, it is not always possible to strictify symmetry.

Example 2.1. The category **Rel** is a category whose objects are sets and morphisms between X and Y are relations $R \subseteq X \times Y$. The identity is the diagonal relation $\{(x, x) \mid x \in X\}$ and compositions are defined by the usual relational composition. The category **Rel** is a symmetric monoidal category; the bifunctor $\otimes$ is defined by the cartesian product $\times$ and the identity object I is given as a singleton set $\{*\}$. Morphisms $\alpha_{X,Y,Z}$, l_X, r_X and $c_{X,Y}$ are defined by the canonical ones. It is easy to check that these morphisms satisfy the required diagrams. ■

Example 2.2. Every cartesian category $(\mathcal{C}, \times, 1)$ is a symmetric monoidal category. The associator, left unitor, right unitor and symmetry is given by

$$\alpha_{A,B,C} \stackrel{\text{def}}{=} \langle \pi_1^{(A\times B)\times C}; \pi_1^{A\times B}, \langle \pi_1^{(A\times B)\times C}; \pi_2^{A\times B}, \pi_2^{(A\times B)\times C} \rangle\rangle$$
$$l_A \stackrel{\text{def}}{=} \pi_2^{1\times A} \quad r \stackrel{\text{def}}{=} \pi_1^{A\times 1} \quad c_{A,B} \stackrel{\text{def}}{=} \langle \pi_2^{A\times B}, \pi_1^{A\times B} \rangle.$$

■

We now review the notion of monoidal functor, which is a functor that preserves the monoidal structure.

Definition 2.3 (Monoidal functor [80])**.** A *(lax) monoidal functor* between monoidal categories $(\mathcal{C}, \otimes, I, \alpha^{\otimes}, l^{\otimes}, r^{\otimes})$ and $(\mathcal{D}, \odot, E, \alpha^{\odot}, l^{\odot}, r^{\odot})$ is a functor $F \colon \mathcal{C} \to \mathcal{D}$ equipped with natural transformations

$$m^2_{A,B} \colon F(A) \odot F(B) \to F(A \otimes B) \qquad m^0 \colon E \to F(I)$$

that makes the following three diagrams commute.

$$\begin{tikzcd}
((FA \odot FB) \odot FC) \arrow[r, "m^2_{A,B} \odot \mathrm{id}_{FC}"] \arrow[d, "\alpha^{\odot}"'] & F(A \otimes B) \odot FC \arrow[r, "m^2_{A\otimes B,C}"] & F((A \otimes B) \otimes C) \arrow[d, "F\alpha^{\otimes}"] \\
FA \odot (FB \odot FC) \arrow[r, "\mathrm{id}_{FA} \odot m^2_{B,C}"] & FA \odot F(B \otimes C) \arrow[r, "m^2_{A,B\otimes C}"] & F(A \otimes (B \otimes C))
\end{tikzcd}$$

$$\begin{tikzcd}
FA \odot E \arrow[r, "r^{\odot}"] \arrow[d, "\mathrm{id}_{FA} \odot m^0"'] & FA & E \odot FA \arrow[r, "l^{\odot}"] \arrow[d, "m^0 \odot \mathrm{id}_{FA}"'] & FA \\
FA \odot FI \arrow[r, "m^2_{A,I}"] & F(A \otimes I) \arrow[u, "Fr^{\otimes}"'] & FI \odot FA \arrow[r, "m^2_{I,A}"] & F(I \otimes A) \arrow[u, "Fl^{\otimes}"']
\end{tikzcd}$$

A *strong monoidal functor* is a monoidal functor whose mediating maps, i.e. $m^2_{A,B}$ and m_0, are isomorphisms. A monoidal functor (F, m^2, m^0) is a *strict monoidal functor* if all components of m^2 and m^0 are identities. ■

Definition 2.4 (Symmetric monoidal functor [80])**.** Let (F, m^2, m^0) be a lax monoidal functor between two symmetric monoidal categories $(\mathcal{C}, \otimes, I, \alpha^{\otimes}, l^{\otimes}, r^{\otimes}, c^{\otimes})$ and $(\mathcal{D}, \odot, E, \alpha^{\odot}, l^{\odot}, r^{\odot}, c^{\odot})$. Then F, m^2, m^0 is a *symmetric monoidal functor* if it satisfies the following diagram.

$$\begin{tikzcd}
F(A) \odot F(B) \arrow[r, "c^{\odot}_{FA,FB}"] \arrow[d, "m^2_{A,B}"] & F(B) \odot F(A) \arrow[d, "m^2_{B,A}"] \\
F(A \otimes B) \arrow[r, "F(c^{\otimes}_{A,B})"] & F(B \otimes A)
\end{tikzcd}$$

A strict monoidal functor $F \colon \mathcal{C} \to \mathcal{D}$ is a *strict symmetric monoidal functor* if F is a also symmetric monoidal, i.e. $F(c^{\otimes}) = c^{\odot}$. ■

Example 2.3 (Strict finite product preserving functor)**.** A *strict finite product preserving functor* from a cartesian category $\mathcal{C}$ to a cartesian category $\mathcal{D}$ is a functor $F \colon \mathcal{C} \to \mathcal{D}$ that maps the chosen product cones (resp. the chosen terminal object) in $\mathcal{C}$ to the chosen product cones in $\mathcal{D}$ (resp. chosen terminal object). It is routine to check that F is a strict (symmetric) monoidal functor. ■

Definition 2.5 (Monoidal natural transformation [80])**.** Let (F, m^2, m^0) and (G, n^2, n^0) be monoidal functors from $(\mathcal{C}, \otimes, I)$ to $(\mathcal{D}, \odot, E)$. Then a *monoidal natural transformation* $\tau \colon (F, m^2, m^0) \to (G, n^2, n^0)$ is a natural transformation $\tau \colon F \to G$ such that the following diagrams commute.

$$\begin{tikzcd}
F(A) \odot F(B) \arrow[r, "m^2_{A,B}"] \arrow[d, "\tau_A \odot \tau_B"] & F(A \otimes B) \arrow[d, "\tau_{A \otimes B}"] \\
G(A) \odot G(B) \arrow[r, "n^2_{A,B}"] & G(A \otimes B)
\end{tikzcd}
\qquad
\begin{tikzcd}
 & E \arrow[dl, "m^0"'] \arrow[dr, "n^0"] & \\
FI \arrow[rr, "\tau_I"] & & GI
\end{tikzcd}$$

■

If F and G are strict, then τ is a natural transformation that satisfies $\tau_{A \otimes B} = \tau_A \otimes \tau_B$ and $\tau_I = \mathrm{id}_I$.

We now introduce the notion of compact closed category. To make the definition simple, we will define the notion over strict monoidal categories and in the sequel we may write monoidal category to mean a strict monoidal category.

Definition 2.6 (Compact closed category [63])**.** A symmetric monoidal category $\mathcal{C}, \otimes, I$ is a *compact closed category* if every object $(A \in \mathbf{Obj}(\mathcal{C}))$ has the *dual* A^*. The object A^* is a (left) dual of A if there exist morphisms $\eta_A \colon I \to A \otimes A^*$ and $\varepsilon_A \colon A^* \otimes A \to I$, called *unit* and *counit*, respectively, that satisfies the two triangle identities.

$$\begin{tikzcd}
A \arrow[r, "\eta_A \otimes \mathrm{id}_A"] \arrow[dr, "\mathrm{id}_A"'] & A \otimes A^* \otimes A \arrow[d, "\mathrm{id}_A \otimes \varepsilon_A"] \\
 & A
\end{tikzcd}
\quad
\begin{tikzcd}
A^* \arrow[r, "\mathrm{id}_{A^*} \otimes \eta_A"] \arrow[dr, "\mathrm{id}_{A^*}"'] & A^* \otimes A \otimes A^* \arrow[d, "\varepsilon_A \otimes \mathrm{id}_{A^*}"] \\
 & A^*
\end{tikzcd}$$

The dual of A is unique up to isomorphism. ■

The operator $(-)^*$ induces a functor $(-)^* \colon \mathcal{C}^{\mathrm{op}} \to \mathcal{C}$: the action on morphisms is defined by

$$f^* \overset{\text{def}}{=} B^* \xrightarrow{\mathrm{id}_{B^*} \otimes \eta_A} B^* \otimes A \otimes A^* \xrightarrow{\mathrm{id}_{B^*} f \otimes \mathrm{id}_{A^*}} B^* \otimes B \otimes A^* \xrightarrow{\varepsilon_B \otimes \mathrm{id}_{A^*}} A^*$$

where $f \colon A \to B$. It is easy to see that this functor $(-)^*$ is involutive (up to isomorphism), i.e. $A^{**} \cong A$, and distributes over monoidal products, i.e $(A \otimes B)^* \cong A^* \otimes B^*$.

Example 2.4. One of the simplest example of a compact closed category is **Rel**, which we defined in Example 2.1. The dual of A, unit and counit is given by

$$\begin{aligned} A^* &\overset{\text{def}}{=} A \\ \eta_A \colon I \to A \otimes A^* &\overset{\text{def}}{=} \{(*, (a,a)) \mid a \in A\} \\ \varepsilon_A \colon A^* \otimes A \to I &\overset{\text{def}}{=} \{((a,a), *) \mid a \in A\}. \end{aligned}$$

■

Now we define the notion of *compact closed functor*, which is the "structure preserving functor" between compact closed categories. Probably, the definition of a compact closed functor is a folklore; in fact, the term "compact closed functor" appears in various papers (e.g. [4, 64, 24]). Unfortunately, however, its definition is rarely found. We think that this situation is not desirable because compact closed functors can be defined in varying levels of strictness and strength. Here we explicitly define our version of compact closed functor.

Definition 2.7 (Compact closed functor)**.** Let $(\mathcal{C}, \eta^{\mathcal{C}}, \varepsilon^{\mathcal{C}})$ and $(\mathcal{D}, \eta^{\mathcal{D}}, \varepsilon^{\mathcal{D}})$ be compact closed categories with chosen unit and counit and (F, m^2, m^0) be a strong symmetric monoidal functor from $\mathcal{C}$ to $\mathcal{D}$. Then $F(A^*)$ is a dual of $F(A)$: units η' and counits ε' are given by

$$\eta'_A \overset{\text{def}}{=} m^0; F(\eta^C); (m^2_{A,A^*})^{-1} \quad \text{and} \quad \varepsilon'_A \overset{\text{def}}{=} m^2_{A^*,A}; F(\varepsilon^{\mathcal{C}}); (m^0)^{-1}.$$

Hence, we have a canonical isomorphism $\theta_A \colon F(A)^* \cong F(A^*)$ defined as $(\mathrm{id}_{F(A)^*} \otimes \eta'_A); (\varepsilon^{\mathcal{D}}_A \otimes \mathrm{id}_{F(A^*)})$ for each object $A \in \mathbf{Obj}(\mathcal{C})$. The functor F is a *strong compact closed functor* if θ satisfies the diagram below for all $A \in \mathbf{Obj}(\mathcal{C})$.

$$\begin{tikzcd} F(A^*)^* \arrow[r, "\theta_{A^*}"] \arrow[d, "(\theta_A)^*"] & F(A^{**}) \arrow[d, "\cong"] \\ F(A)^{**} \arrow[r, "\cong"] & F(A) \end{tikzcd}$$

A strong compact closed functor (F, m^2, m^0, θ) is a *strict compact closed functor* if F is a strict symmetric monoidal functor and $\theta_A = \mathrm{id}_F(A)^*$ for all A. ■

Remark 2.1. Our definition of compact closed functor is built upon that of $*$-autonomous functor given by Cockett et al. [25]. Since compact closed categories are special kinds of $*$-autonomous categories, compact closed functors are also defined as particular instances of $*$-autonomous functors.

The following proposition gives us an alternative characterization of strict compact closed functors.

Proposition 2.2. Let $(\mathcal{C}, \eta^{\mathcal{C}}, \varepsilon^{\mathcal{C}})$ and $(\mathcal{D}, \eta^{\mathcal{D}}, \varepsilon^{\mathcal{D}})$ be compact closed categories and $F \colon \mathcal{C} \to \mathcal{D}$ be a strict symmetric monoidal functor. Then F is a strict compact closed functor if and only if $F(\eta^{\mathcal{C}}) = \eta^{\mathcal{D}}$ and $F(\varepsilon^{\mathcal{C}}) = \varepsilon^{\mathcal{D}}$.

Proof. First, we show the only if direction. Since $F(\eta^{\mathcal{C}}) = \eta^{\mathcal{D}}$ and $F(\varepsilon^{\mathcal{C}}) = \varepsilon^{\mathcal{D}}$ and F is a strict symmetric monoidal functor, we have

$$\begin{aligned} \theta_A &= (\mathrm{id}_{F(A)^*} \otimes F(\eta^{\mathcal{C}}_A)); (\varepsilon^{\mathcal{D}}_A \otimes \mathrm{id}_{F(A^*)}) \\ &= (\mathrm{id}_{F(A)^*} \otimes \eta^{\mathcal{D}}_A); (\varepsilon^{\mathcal{D}}_A \otimes \mathrm{id}_{F(A^*)}) \\ &= \mathrm{id}_{F(A)^*}. && \text{(triangle identity)} \end{aligned}$$

We are left to check the coherence condition for θ. As required, the two isomorphisms of the coherence diagram coincides because $F(\eta^{\mathcal{C}}) = \eta^{\mathcal{D}}$, $F(\varepsilon^{\mathcal{C}}) = \varepsilon^{\mathcal{D}}$ and $F(c^{\mathcal{C}}) = c^{\mathcal{D}}$. Here $c^{\mathcal{C}}$ and $c^{\mathcal{D}}$ are the symmetries of $\mathcal{C}$ and $\mathcal{D}$ respectively.

Next, we show the if direction. Since $\theta_A = \mathrm{id}_{F(A)^*}$, we have

$$\begin{aligned}
\eta^{\mathcal{D}}_A &= \eta^{\mathcal{D}}_A; (\mathrm{id}_A \otimes \theta_A) \\
&= \eta^{\mathcal{D}}_A; (\mathrm{id}_A \otimes \mathrm{id}_{F(A)^*} \otimes F(\eta^{\mathcal{C}}_A)); (\mathrm{id}_A \otimes \varepsilon^{\mathcal{D}}_A \otimes \mathrm{id}_{F(A^*)}) \\
&= F(\eta^{\mathcal{C}}_A); (\eta^{\mathcal{D}}_A \otimes \mathrm{id}_A \otimes \mathrm{id}_{F(A)^*}); (\mathrm{id}_A \otimes \varepsilon^{\mathcal{D}}_A \otimes \mathrm{id}_{F(A^*)}) \\
&= F(\eta^{\mathcal{C}}_A); (\mathrm{id}_A \otimes \mathrm{id}_{F(A)^*}) && \text{(triangle identity)} \\
&= F(\eta^{\mathcal{C}}_A).
\end{aligned}$$

We can show $\varepsilon^{\mathcal{D}}_A = F(\varepsilon^{\mathcal{C}}_A)$ in a similar manner. □

This proposition implies that our definition of "strict compact closed functor" is equivalent to (one of the few explicit) definition of "compact closed functor" given by Abransky and Coecke [4], which says that compact closed functors are functors that strictly preserves the monoidal structure, unit and counit.

Remark 2.1. It should be noted that strong symmetric monoidal functors already preserve dualities. If $F\colon \mathcal{C} \to \mathcal{D}$ is a strong symmetric monoidal functor and $\mathcal{C}$ is compact closed, we can easily check that $F(A^*)$ is a dual of $F(A)$ by applying F to the triangle identities. Hence, the notion of strong compact closed functor is redundant unless we specify the chosen duals (or chosen units and counits) of compact closed categories. For this reason we imagine that compact closed functor often refers to strict compact closed functor. ■

To discuss the precise relationship between compact closed categories, we also introduce the notion of isomorphisms between compact closed functors.

Definition 2.8 (Isomorphisms between compact closed functors)**.** Let (F, θ^F) and (G, θ^G) be strong compact closed functors from $\mathcal{C}$ to $\mathcal{D}$. Then a monoidal natural isomorphism $\tau\colon F \cong G$ is an *isomorphism between strong compact closed functors* (or a *compact closed isomorphism* for short) if it satisfies the following diagram.

$$\begin{tikzcd}
G(A)^* \arrow[r, "(\tau_A)^*"] \arrow[d, "\theta^G_A"] & F(A)^* \arrow[d, "\theta^F_A"] \\
G(A^*) \arrow[r, "(\tau_{A^*})^{-1}"] & F(A^*)
\end{tikzcd}$$

■

This definition is essentially the same as that of isomorphisms between $*$-autonomous functors given by Cockett et al. [25].

Chapter 3

π_F-calculus

This chapter defines the target calculus of this thesis, the *π_F-calculus*, which is based on an asynchronous variant of the polyadic π-calculus with **i**/**o**-types [106]. Using the axiomatic semantics and the operational semantics of the π_F-calculus, we also discuss a fundamental difficulty in developing a categorical type theory for the π-calculus in Section 3.4.

3.1 Syntax and Sorts

The set of *sorts*,[1] ranged over by S and T, is given by

$$S, T ::= \mathbf{ch}^o[T_1, \ldots, T_n] \;\mid\; \mathbf{ch}^i[T_1, \ldots, T_n] \qquad (n \geq 0).$$

The sort $\mathbf{ch}^o[T_1, \ldots, T_n]$ is for output channels that are used to send n arguments of sorts $T_1, \ldots, T_n$. The sort $\mathbf{ch}^i[T_1, \ldots, T_n]$ is for input channels. The *dual* $T^\perp$ of sort T is defined by $\mathbf{ch}^o[\vec{T}]^\perp \overset{\mathrm{def}}{=} \mathbf{ch}^i[\vec{T}]$ and $\mathbf{ch}^i[\vec{T}]^\perp \overset{\mathrm{def}}{=} \mathbf{ch}^o[\vec{T}]$. For a sequence $\vec{T} \overset{\mathrm{def}}{=} T_1, \ldots, T_n$ of sorts, we write $\vec{T}^\perp$ for $T_1^\perp, \ldots, T_n^\perp$.

An important difference from the original **i**/**o**-type system [106] is that no name has both input and output capabilities. We will refer this feature of π_F-calculus as **i**/**o**-*separation*.

The set of *processes*, ranged over by P, Q and R, is defined by

$$P, Q, R \;::=\; \mathbf{0} \mid (P \mid Q) \mid (\boldsymbol{\nu}_{\mathbf{ch}^o[\vec{T}]}\, xy)P \mid x\langle y_1, \ldots, y_n\rangle \mid\, !x(y_1, \ldots, y_n).P \quad .$$

Here $x, y, \ldots$ range over a denumerable set of *names.* Each name is either input-only or output-only, because of **i**/**o**-separation. As usual, $\vec{y}$ in $!x(\vec{y}).P$ and x, y in $(\boldsymbol{\nu} xy)$ are called *bound names*; the other names are called *free names.* The set of free names and bound names of P are written as $\mathbf{fn}(P)$ and $\mathbf{bn}(P)$ respectively. The set $\mathbf{n}(P)$ is defined as $\mathbf{fn}(P) \cup \mathbf{bn}(P)$. We allow tacit renaming of bound names, and identify α-equivalent processes. The sort annotation in $(\boldsymbol{\nu}_T\, xy)P$ is often omitted and $(\boldsymbol{\nu} x_1 y_1) \ldots (\boldsymbol{\nu} x_n y_n)P$ is abbreviated to $(\boldsymbol{\nu} \vec{x}\vec{y})P$.

Let us briefly explain the intuitive meanings of the constructs. The process $\mathbf{0}$ is the inaction, which is a process that does nothing; $P \mid Q$ is a parallel composition; $x\langle\vec{y}\rangle$ is an output; and $!x(\vec{x}).P$ is a replicated input. The restriction $(\boldsymbol{\nu}_T\, xy)P$ is slightly unusual and is a bit different from the name restriction used in conventional π-calculi. It hides the names x and y of sort T and $T^\perp$ and, at the same time, establishes a connection between x and y. Communication takes place only over bound names explicitly connected by $\boldsymbol{\nu}$ (cf. Section 3.2.1). This is in contrast to the conventional π-calculus, in which input-output correspondence

[1] The term *sort* instead of *type* is used in order to avoid confusion with intersection types introduced later in this thesis (Chapter 7).

$$\frac{}{\Delta \vdash \mathbf{0} : \diamond}\ (\text{S-Nil}) \qquad \frac{\Delta \vdash P : \diamond \qquad \Delta \vdash Q : \diamond}{\Delta \vdash P \mid Q : \diamond}\ (\text{S-Par})$$

$$\frac{(x{:}\mathbf{ch}^o[\vec{T}]) \in \Delta \quad \vec{y} : \vec{T} \subseteq \Delta}{\Delta \vdash x\langle \vec{y} \rangle : \diamond}\ (\text{S-Out}) \qquad \frac{(x{:}\mathbf{ch}^i[\vec{T}]) \in \Delta \quad \Delta, \vec{y}{:}\vec{T} \vdash P : \diamond}{\Delta \vdash !x(\vec{y}).P : \diamond}\ (\text{S-In})$$

$$\frac{\Delta, x : \mathbf{ch}^o[\vec{T}], y : \mathbf{ch}^i[\vec{T}] \vdash P : \diamond}{\Delta \vdash (\boldsymbol{\nu}_{\mathbf{ch}^o[\vec{T}]}\, xy)P : \diamond}\ (\text{S-Nu}) \qquad \frac{\Delta, x : S, y : T, \Delta' \vdash P : \diamond}{\Delta, y : T, x : S, \Delta' \vdash P : \diamond}\ (\text{S-Ex})$$

Figure 3.1: Sort assignment rules for processes.

is *a priori* (i.e. $\bar{a}$ is the output to a). However, for readability, we will often use the names $\bar{a}, \bar{b}, \dots$ and $\bar{x}, \bar{y}, \dots$ for output names. This is just a convention and it is possible to use names such as $\bar{a}$ for input names; e.g. $\bar{a}(\vec{y}).P$ is a valid process.

This calculus is *polyadic* [93] as it can send or receive multiple names at the same time and it is *asynchronous* [55] because the output action $\bar{a}\langle \vec{x} \rangle$ does not have a continuation. Polyadic and asynchronous variants of π-calculus are widely used and thus being polyadic or asynchronous is not something unique to the π_F-calculus.

An important characteristic of π_F-calculus is the absence of non-replicated input $x(\vec{y}).P$. This allows us to consider input prefixing as λ-abstractions (that is located at a certain address) as we described in the introduction. However, considering a calculus without non-replicated input itself is not our contribution. For example, a typed asynchronous π-calculus without non-replicated inputs has been considered in the work by Honda and Laurent [54] that studies the relationship between polarized proof nets and π-calculus.

Sort assignment rules

A *sort environment* Δ is a finite sequence of bindings of the form $x\colon T$ such that all the names appearing in Δ are pairwise distinct. We write $\vec{x}\colon \vec{T}$ for $x_1\colon T_1, \dots, x_n\colon T_n$ provided that $\vec{x} = x_1, \dots, x_n$ and $\vec{T} = T_1, \dots, T_n$ and also write $(\vec{x}\colon \vec{T}) \subseteq \Delta$ to mean $x_i\colon T_i \in \Delta$ for every i.

A *sort judgement* is of the form $\Delta \vdash P : \diamond$, meaning that P is a well-sorted process under the environment Δ. Here the symbol $\diamond$ is the unique sort for processes. The sort assignment rules are listed in Fig. 3.1. The rules are standard and there is not much to explain. For instance the rule for $x(\vec{y}).P$ says that x should have a sort $\mathbf{ch}^i[\vec{T}]$ for input channels and the body P must be well sorted under the environment Δ extended with $\vec{y} : \vec{T}$. We note that the rule for name restriction stipulates that only names with dual sorts can be restricted.

3.2 Operational and Axiomatic Semantics

This section defines the operational and the axiomatic semantics of the π_F-calculus. The operational semantics is unnecessary to obtain a theory/model correspondence between a calculus and a categorical structure; we only need the axiomatic semantics. We, however, introduce an operational semantics because it is more intuitive than the axiomatic semantics and also helps us understand the axiomatic semantics.

3.2.1 Operational semantics

We now introduce the reduction semantics for π_F-calculus and some behavioral equivalences that can be defined using the reduction relation.

The *one-step reduction relation* on processes, written $\longrightarrow$, is defined by the base rule

$$(\boldsymbol{\nu}\vec{w}\vec{z})(\boldsymbol{\nu}\bar{a}a)(!a(\vec{x}).P \mid \bar{a}\langle\vec{y}\rangle \mid Q) \longrightarrow (\boldsymbol{\nu}\vec{w}\vec{z})(\boldsymbol{\nu}\bar{a}a)(!a(\vec{x}).P \mid P\{\vec{y}/\vec{x}\} \mid Q)$$

and the structural rule which concludes $P \longrightarrow Q$ from $P \equiv P' \longrightarrow Q' \equiv Q$ for some processes P' and Q'. Here the relation $\equiv$ is the structural-congruence defined below. It should be emphasized that, unlike conventional π-calculi, communication only occurs over bound names created by the same ν operator. We write $\longrightarrow^*$ for the reflexive and transitive closure of $\longrightarrow$.

Now we define the structural congruence (in a slightly non-standard way). For a technical reason, we shall first define *structural precongruence* $\Rrightarrow$. A precongruence is like a congruence, but it is just reflexive and transitive, not necessarily symmetric. We define $\Rrightarrow$ as the smallest precongruence relation on processes that satisfies the following rules ($P \Lleftarrow\!\!\Rrightarrow Q$ means $P \Rrightarrow Q$ and $Q \Rrightarrow P$):

$$P \mid \mathbf{0} \Lleftarrow\!\!\Rrightarrow P \qquad P \mid Q \Lleftarrow\!\!\Rrightarrow Q \mid P \qquad (P \mid Q) \mid R \Lleftarrow\!\!\Rrightarrow P \mid (Q \mid R)$$
$$(\boldsymbol{\nu}wx)(\boldsymbol{\nu}yz)P \Lleftarrow\!\!\Rrightarrow (\boldsymbol{\nu}yz)(\boldsymbol{\nu}wx)P \qquad ((\boldsymbol{\nu}xy)P) \mid Q \Rrightarrow (\boldsymbol{\nu}xy)(P \mid Q)$$

where w, x, y, z are distinct in the fourth rule and $x, y \notin \mathbf{fn}(Q)$ in the fifth rule. It is similar to the structural congruence, but the restriction of the scope of $(\boldsymbol{\nu}xy)$ is not allowed. The *structural congruence* $\equiv$ is the symmetric closure of $\Rrightarrow$ and this definition coincides with the standard definition of the structural congruence. The structural precongruence will be used in Chapter 7, in which we define a novel reduction semantics for the π_F-calculus.

Remark 3.1. Although some calculi based on the π-calculus have the rule $(\boldsymbol{\nu}\bar{a}a)P = P$ included in structural congruence, we do not include this rule. This is because the rule $(\boldsymbol{\nu}\bar{a}a)P = P$ is incompatible with the categorical semantics. That is $[\![(\boldsymbol{\nu}\bar{a}a)]\!]P = [\![P]\!]$ does not hold in our categorical semantics defined in Chapter 4. ■

Example 3.1. Let us consider a process $(\boldsymbol{\nu}\bar{b}b)(\boldsymbol{\nu}\bar{a}a)(!a(x).\bar{b}\langle x\rangle \mid \bar{a}\langle x\rangle \mid P)$. Then the following is a valid reduction:

$$(\boldsymbol{\nu}\bar{b}b)(\boldsymbol{\nu}\bar{a}a)(!a(x).\bar{b}\langle x\rangle \mid \bar{a}\langle m\rangle \mid P) \longrightarrow (\boldsymbol{\nu}\bar{b}b)(\boldsymbol{\nu}\bar{a}a)(!a(x).\bar{b}\langle x\rangle \mid \bar{b}\langle m\rangle \mid P)$$

This represents that a message m has been sent via the channel $\bar{a}$ and received by the channel a and now the message is m is ready to be sent by $\bar{b}$. Or one can think that the message m has now been stored in a communication medium $\bar{b}$ and is available to any unguarded subprocess of the form $!b(y).Q$ because $\bar{b}$ is connected with b. Note that the process $!a(x).\bar{b}\langle x\rangle$ remains after the reduction because it is a replicated process. Processes of the form $!a(\vec{x}).\bar{b}\langle\vec{x}\rangle$ are called *forwarders* and will be written as $a \hookrightarrow b$. As the name suggests they forward a message from one place to another. Forwarders will play an important role throughout this thesis because they will be treated as the syntactic counterpart of the identity morphisms of categories. When $x : T$ and $y : T^{\perp}$, we write $x \leftrightharpoons y$ to mean $x \hookrightarrow y$ if $T = \mathbf{ch}^i[\vec{S}]$ and otherwise $y \hookrightarrow x$. ■

We now introduce two behavioral equivalences: testing equivalence and barbed congruence. These equivalences are two of the most studied behavioral equivalences among the various equivalences defined for π-calculi. We will later use these equivalences when we do a “sanity check” on the axiomatic semantics.

In both testing equivalence and barbed congruence, two processes are considered to have the same behavior if "no difference can be observed when the processes are put into arbitrary context". To formalize this idea we formally introduce the notion of observables and contexts. The observables of a process are the names it can use for sending. For each name $\bar{a}$, we write $P{\downarrow_{\bar{a}}}$ if $P \equiv (\boldsymbol{\nu}\vec{x}\vec{y})(\bar{a}\langle\vec{z}\rangle \mid Q)$ and $\bar{a}$ is free, and $P{\Downarrow_{\bar{a}}}$ if $P \longrightarrow^{*} Q{\downarrow_{\bar{a}}}$ for some Q. We do not consider input names as observables because input actions are not observable in an asynchronous setting. This is because output actions in an environment, which is also described as π_F-process, have no continuations. The set of *contexts* ranged over by C, D is defined by the following grammar:

$$C, D ::= [\,] \mid (C \mid P) \mid (P \mid C) \mid\, !a(\vec{x}).C.$$

A (Δ'/Δ)-*context* is a context C such that $\Delta' \vdash C[P] : \diamond$ for every $\Delta \vdash P : \diamond$.

Definition 3.1 (May-testing equivalence [33, 15])**.** Well sorted processes $\Delta \vdash P : \diamond$ and $\Delta \vdash Q : \diamond$ are *may-testing equivalent at* Δ, written $\Delta \vdash P =_{may} Q$, if $C[P]{\Downarrow_{\bar{a}}} \Leftrightarrow C[Q]{\Downarrow_{\bar{a}}}$ for every (Δ'/Δ)-context C and name $\bar{a}$. ■

Definition 3.2 (Barbed bisimulation/Barbed congruence [96])**.** A *strong barbed bisimulation* is a symmetric relation $\mathcal{R}$ on processes such that, whenever $P \mathrel{\mathcal{R}} Q$

1. $P{\downarrow_{\bar{a}}}$ implies $Q{\downarrow_{\bar{a}}}$ and
2. $P \longrightarrow P'$ implies $Q \longrightarrow Q'$ and $P' \mathrel{\mathcal{R}} Q'$ for some Q'.

Similarly, a *weak barbed bisimulation* is a symmetric relation $\mathcal{R}$ on processes such that,

1. $P{\downarrow_{\bar{a}}}$ implies $Q{\Downarrow_{\bar{a}}}$ and
2. $P \longrightarrow P'$ implies $Q \longrightarrow^{*} Q'$ and $P' \mathrel{\mathcal{R}} Q'$ for some Q'.

The *strong barbed bisimilarity* $\overset{\bullet}{\sim}$ and the *weak barbed bisimilarity* $\overset{\bullet}{\approx}$ are the largest strong barbed bisimulation and the largest weak barbed bisimulation, respectively.

Sorted processes $\Delta \vdash P : \diamond$ and $\Delta \vdash Q : \diamond$ are *strong (resp. weak) barbed congruent at* Δ, written $\Delta \vdash P \simeq^{c} Q$ (resp. $\Delta \vdash P \approxeq^{c} Q$), if $C[P] \overset{\bullet}{\sim} C[Q]$ ((resp. $C[P] \overset{\bullet}{\approx} C[Q]$)for every (Δ'/Δ)-context C. ■

3.2.2 Axiomatic semantics

We define a class of equivalence, called the π_F-*theory*, by the rules listed in Figure 3.2. The π_F-theory is the theory that will be shown to correspond to compact closed Freyd category.

Definition 3.3. An equivalence $\mathcal{E}$ is a π_F-*theory* if it is closed under the rules in Figure 3.2. Here $a \hookrightarrow \bar{b}$ represents a forwarder process $!a(\vec{x}).\bar{b}\langle\vec{x}\rangle$ (cf. Example 3.1). Any set Ax of equations-in-context has the minimum theory $Th(Ax)$ that contains Ax. We write $Ax \rhd \Delta \vdash P = Q$ if $(\Delta \vdash P = Q) \in Th(Ax)$. ■

Let us examine each rule in Figure 3.2.

The rule (E-Beta) is reminiscent of the reduction relation. When $C = ([\,] \mid Q)$, (E-Beta) becomes

$$(\boldsymbol{\nu}\bar{a}a)(!a(\vec{x}).P \mid \bar{a}\langle\vec{y}\rangle \mid Q) = (\boldsymbol{\nu}\bar{a}a)(!a(\vec{x}).P \mid P\{\vec{y}/\vec{x}\} \mid Q)$$

$$\frac{a \notin \mathbf{fn}(P, C) \qquad \bar{a} \notin \mathbf{bn}(C)}{\Delta \vdash (\boldsymbol{\nu}\bar{a}a)(!a(\vec{x}).P \mid C[\bar{a}\langle\vec{y}\rangle]) = (\boldsymbol{\nu}\bar{a}a)(!a(\vec{x}).P \mid C[P\{\vec{y}/\vec{x}\}])} \text{ (E-BETA)}$$

$$\frac{a, \bar{a} \notin \mathbf{fn}(P)}{\Delta \vdash (\boldsymbol{\nu}\bar{a}a)!a(\vec{y}).P = \mathbf{0}} \text{ (E-GC)} \qquad \frac{\bar{a}, a \notin \mathbf{fn}(\bar{c}\langle\vec{x}\rangle)}{\Delta \vdash \bar{c}\langle\vec{x}\rangle = (\boldsymbol{\nu}\bar{a}a)(a \hookrightarrow \bar{b} \mid \bar{c}\langle\vec{x}\{\bar{a}/\bar{b}\}\rangle)} \text{ (E-FOUT)}$$

$$\frac{b, \bar{a} \notin \mathbf{fn}(P)}{\Delta \vdash (\boldsymbol{\nu}\bar{a}a)(b \hookrightarrow \bar{a} \mid P) = P\{b/a\}} \text{ (E-ETA)}$$

$$\frac{P \equiv Q}{\Delta \vdash P = Q} \text{ (E-SCONG)} \qquad \frac{\Delta \vdash P = Q \qquad C\colon \Delta'/\Delta\text{-context}}{\Delta' \vdash C[P] = C[Q]} \text{ (E-CTX)}$$

Figure 3.2: Inference rules of equations-in-context. Each rule has implicit assumptions that the both sides of the equation are well-sorted processes.

where $a \notin \mathbf{fn}(P, Q)$, which is indeed a special case of the reduction.

The rule (E-BETA), however, is not a conservative extension of the reduction due to the side condition. The side condition is important because it prevents π_F-theories from collapsing. Without the side condition, every π_F-theory would be forced to contain the symmetric and transitive closure of the reduction relation; thus it would identify $P \mid (\boldsymbol{\nu}\bar{a}a)(!a().P \mid !a().Q)$ with $Q \mid (\boldsymbol{\nu}\bar{a}a)(!a().P \mid !a().Q)$ for every processes P and Q (where $\bar{a}, a$ are fresh), because

$$\begin{array}{rcl}
(\boldsymbol{\nu}\bar{a}a)(\bar{a}\langle\rangle \mid !a().P \mid !a().Q) & \longrightarrow & P \mid (\boldsymbol{\nu}\bar{a}a)(!a().P \mid !a().Q) \\
(\boldsymbol{\nu}\bar{a}a)(\bar{a}\langle\rangle \mid !a().P \mid !a().Q) & \longrightarrow & Q \mid (\boldsymbol{\nu}\bar{a}a)(!a().P \mid !a().Q).
\end{array}$$

Note that the above argument relies on the presence of race conditions.

Another, relatively minor, difference compared to the reduction relation is that application of (E-BETA) is not limited to the contexts of the form $[\,] \mid Q$. We can therefore rewrite a (sub)process guarded by an input prefixing. This kind of extension can be found in, for example, work by Honda and Laurent [54] studying π-calculus from a logical perspective. Nevertheless, this kind of law has been studied since the birth of the **i/o**-typed calculus [106], and thus is not something that was artificially introduced to establish a correspondence between π-calculus and logic.

The rule (E-GC) is a "garbage-collection" law. Because no process in the environment can send a message to the hidden name a, the process $!a(\vec{x}).P$ will never be invoked and thus can be safely discarded. This rule is sound with respect to many behavioral equivalences, including barbed congruence. Rules of this kind often appear in the literature studying logical aspects of concurrent calculi (e.g. [54, 133]).

Another well-known rule is (E-FOUT) that is used to replace an output of free names with that of bound names. This kind of operation has been studied in [14, 88] as a part of translations from the π-calculus to its internal fragments. *Internal π-calculus* [116], is a π-calculus where free outputs are forbidden and only bound outputs, i.e. outputs of newly created names, are allowed.[2]

The rule (E-FOUT) can also be seen as a variant of the η-rule of abstractions, as in the λ-calculus and in the higher-order π-calculus [115]. In the latter, an

[2]Free outputs can be eliminated from π_F-processes by using the rules (E-FOUT) and (E-ETA), i.e. external mobility can be encoded by internal mobility [14, 116]. If the calculus is local [88, 136], then we do not need (E-ETA) to eliminate free outputs.

output name $\bar{b}$ can be identified with an abstraction $(\vec{y}).\bar{b}\langle\vec{y}\rangle$. With this identification (E-FOut) becomes an admissible rule in the presence of the η-rule of abstraction; for example,

$$(\boldsymbol{\nu}\bar{a}a)(a \hookrightarrow \bar{b} \mid \bar{c}\langle\bar{a}\rangle) = (\boldsymbol{\nu}\bar{a}a)(a \hookrightarrow \bar{b} \mid \bar{c}\langle\, (\vec{y}).\bar{a}\langle\vec{y}\rangle\,\rangle) = \bar{c}\langle\, (\vec{y}).\bar{b}\langle\vec{y}\rangle\,\rangle = \bar{c}\langle\bar{b}\rangle$$

where we use (E-Beta) and (E-GC) in the second step.

The rule (E-Eta) requires that forwarders to work as a substitution for input names. To our knowledge, this rule is new and has not been studied elsewhere. This is probably because this law is not valid in most of the behavioral equivalences for π-calculus. For example, let us consider the case where $P \overset{\text{def}}{=} \mathbf{0}$. The process $\mathbf{0}$ and $P' \overset{\text{def}}{=} (\boldsymbol{\nu}\bar{b}b)(!a().\bar{b}\langle\rangle \mid \mathbf{0})$ can be distinguished by a context $C \overset{\text{def}}{=} (\boldsymbol{\nu}\bar{a}a)(\bar{a}\langle\rangle \mid !a().\bar{o}\langle\rangle \mid [\,])$, where $\bar{o}$ is the observable. The process $C[P']$ can reduce as

$$C[P'] \longrightarrow (\boldsymbol{\nu}\bar{a}a)(\boldsymbol{\nu}\bar{b}b)(!a().\bar{b}\langle\rangle \mid \bar{b}\langle\rangle \mid !a().\bar{o}\langle\rangle)$$

and after this reduction we can never observe the output $\bar{o}$. However, there is no reduction from $C[\mathbf{0}]$ that matches this reduction. Though the rule (E-Eta) has a problem from an operational perspective, it is inevitable to include this rule into the logical axioms of π_F-theory in order to achieve a correspondence to a certain categorical structure; we shall explain the details later in this chapter (Section 3.4).

Remaining rules (E-SCong) and (E-Ctx) are easy to understand. The former requires that structurally congruent processes should be identified; the latter says that a π_F-theory is a congruence.

Let us introduce some admissible rules for convenience. The following structural rules are derivable.

$$\frac{\Delta \vdash P = Q \qquad x \notin \mathbf{fn}(\Delta)}{\Delta, x : T \vdash P = Q}\ (\text{E-Wk}) \qquad \frac{\Delta, x : S, y : T, \Delta' \vdash P = Q}{\Delta, y : T, x : S, \Delta' \vdash P = Q}\ (\text{E-Ex})$$

We note that we also have the substitution rule as a derived rule.

Lemma 3.1. Suppose that $\Delta, x : T \vdash P = Q$ and $y \notin \mathbf{fn}(\Delta)$. Then $\Delta, y : T \vdash P\{y/x\} = Q\{y/x\}$. □

Since the proof of this lemma uses the fact that forwarders behaves as substitutions, let us first prove some properties of forwarders.

Lemma 3.2.

1. $(\boldsymbol{\nu}\bar{a}a)(a \hookrightarrow \bar{b} \mid P) = P\{\bar{b}/\bar{a}\}$ if $a \notin \mathbf{fn}(P)$.
2. $(\boldsymbol{\nu}\bar{b}b)(a \hookrightarrow \bar{b} \mid b \hookrightarrow \bar{c}) = a \hookrightarrow \bar{c}$.
3. $(\boldsymbol{\nu}_{\vec{T}}\,\vec{y}\vec{y}')(\vec{x} \leftrightharpoons \vec{y} \mid \vec{y}' \leftrightharpoons \vec{z}) = \vec{x} \leftrightharpoons \vec{z}$. Here we assume that no names are shared between $\vec{x}$, $\vec{y}$, $\vec{y}'$ and $\vec{z}$.

Proof.

1. We first rewrite P into an equivalent process P' such that $\bar{a}$ only appears in subject positions. Let P' be the process obtained by replacing every free output $\bar{c}_i\langle\vec{x}_i\rangle$ that contains $\bar{a}$ in an object position with $(\boldsymbol{\nu}\bar{a}'_i a'_i)(a'_i \hookrightarrow \bar{a} \mid \bar{c}_i\langle\vec{x}_i\{\bar{a}'_i/\bar{a}\}\rangle)$ by using (E-Out). Note that the type system ensures $\bar{c}_i \neq \bar{a}$ for all i.

Now, there exists a multi-hole context C (which may be a context without a hole, i.e. a process) such that $P' = C[\bar{a}\langle\vec{b_1}\rangle]\cdots[\bar{a}\langle\vec{b_n}\rangle]$ and $\bar{a} \notin \mathbf{fn}(C)$. Since $a \hookrightarrow \bar{b} = !a(\vec{x}).\bar{b}\langle\vec{x}\rangle$, we can apply the axiom (E-Beta) n times to show

$$\begin{aligned}(\boldsymbol{\nu}\bar{a}a)(a \hookrightarrow \bar{b} \mid P') &= (\boldsymbol{\nu}\bar{a}a)(a \hookrightarrow \bar{b} \mid C[\bar{b}\langle\vec{b_1}\rangle]\cdots[\bar{b}\langle\vec{b_n}\rangle]) \\ &= (\boldsymbol{\nu}\bar{a}a)(a \hookrightarrow \bar{b} \mid C[\bar{a}\langle\vec{b_1}\rangle\{\bar{b}/\bar{a}\}]\cdots[\bar{a}\langle\vec{b_n}\rangle\{\bar{b}/\bar{a}\}]) \\ &= (\boldsymbol{\nu}\bar{a}a)(a \hookrightarrow \bar{b} \mid P'\{\bar{b}/\bar{a}\}).\end{aligned}$$

Note that each subprocess of the form $(\boldsymbol{\nu}\bar{a}'_i a'_i)(a'_i \hookrightarrow \bar{a} \mid \bar{c}_i\langle\vec{x}_i\{\bar{a}'_i/\bar{a}\}\rangle)$, which is equal to $\bar{c}_i\langle\vec{x}_i\rangle$, has been replaced with $(\boldsymbol{\nu}\bar{a}'_i a'_i)(a'_i \hookrightarrow \bar{b} \mid \bar{c}_i\langle\vec{x}_i\{\bar{a}'_i/\bar{a}\}\rangle)$. Using the rule (E-FOut), each of these process can be replaced by $\bar{c}_i\langle\vec{x}_i\{\bar{b}/\bar{a}\}\rangle$. So, we have $(\boldsymbol{\nu}\bar{a}a)(a \hookrightarrow \bar{b} \mid P'\{\bar{b}/\bar{a}\}) = (\boldsymbol{\nu}\bar{a}a)(a \hookrightarrow \bar{b} \mid P\{\bar{b}/\bar{a}\})$.

Thus, we conclude

$$\begin{aligned}(\boldsymbol{\nu}\bar{a}a)(a \hookrightarrow \bar{b} \mid P) &= (\boldsymbol{\nu}\bar{a}a)(a \hookrightarrow \bar{b} \mid P\{\bar{b}/\bar{a}\}) \\ &= (\boldsymbol{\nu}\bar{a}a)(a \hookrightarrow \bar{b}) \mid P\{\bar{b}/\bar{a}\} && (\bar{a}, a \notin \mathbf{fn}(P\{\bar{b}/\bar{a}\})) \\ &= \mathbf{0} \mid P\{\bar{b}/\bar{a}\} && \text{(E-GC)} \\ &= P\{\bar{b}/\bar{a}\}.\end{aligned}$$

2. This is a direct consequence of (E-Eta), but can also be proved without (E-ETA) by using (1).

3. Follows from (2).

□

Now it is easy to see that the substitution rule is derivable.

Proof of Lemma 3.1. The proof is by a case analysis on the sort T. If T is a sort for input channels, then we have $\Delta, y : T, \bar{x} : T^{\perp}, x : T, \vdash P = Q$ by (E-Wk) and (E-Ex). Since $\Delta, y : T, \bar{x} : T^{\perp}, x : T \vdash y \hookrightarrow \bar{x}$, we have $\Delta, y : T \vdash (\boldsymbol{\nu}\bar{x}x)(y \hookrightarrow \bar{x} \mid P) = (\boldsymbol{\nu}\bar{x}x)(y \hookrightarrow \bar{x} \mid Q)$. Using (E-Eta), we conclude $\Delta, y : T \vdash P\{y/x\} = Q\{y/x\}$.

The proof for the remaining case is similar. The only difference is to use (1) of Lemma 3.2 instead of (E-Eta). □

As briefly explained, all the rules except for (E-Eta) are well-studied and these rules can be justified from the operational viewpoint as well. The following is a well-known result on the **i**/**o**-typed π-calculus (see, e.g. [119, 106]).

Proposition 3.3. Weak barbed congruence is closed under all rules but (E-Eta).

Proof. The proof is essentially the same as in the case of the standard **i**/**o**-typed π-calculus [119]. The **i**/**o**-separation and disallowing non-replicated inputs does not affect the proof. □

In case of may-testing equivalence, which is a rather coarse equivalence, (E-Eta) holds as well. Hence, we have the following result:

Proposition 3.4. May-testing equivalence is a π_F-theory. □

3.3 Expressiveness

The aim of this short section is to clarify what can and cannot be written in the π_F-calculus.

Our calculus can express systems with cyclic structures, which cannot be expressed in session-typed calculi that correspond to linear logic [19, 133]. A typical example is the Milner's cyclic scheduler [89].

Example 3.2. In the π_F-calculus, the Milner's cyclic scheduler can be expressed as follows:

$$\begin{aligned}
\mathrm{Sched} &\overset{\mathrm{def}}{=} (\boldsymbol{\nu}\bar{a}_0 a_0)\dots(\boldsymbol{\nu}\bar{a}_{n-1}a_{n-1})(\boldsymbol{\nu}\bar{b}_0 b_0)\dots(\boldsymbol{\nu}\bar{b}_{n-1}b_{n-1})\\
&\qquad (P_0 \mid \cdots \mid P_{n-1} \mid Q_0 \mid \cdots \mid Q_{n-1} \mid \bar{b}_0\langle\rangle)\\
P_i &\overset{\mathrm{def}}{=} !b_i().(\boldsymbol{\nu}\bar{r}r)(\bar{a}_i\langle\bar{r}\rangle \mid !r().\bar{b}_{i+1\,\mathbf{mod}\,n}\langle\rangle)\\
Q_i &\overset{\mathrm{def}}{=} !a_i(\bar{r}).R_i
\end{aligned}$$

provided that the process R_i sends an acknowledgment using $\bar{r}$ at the end of its computation and that $\bar{r}$ is linearly used. Here the processes P_i are used to express the scheduling policy and each Q_i is the processes that "does the job". The process Q_i is taken care by the process P_i and once the P_i receives the acknowledgement from Q_i it sends a message via $\bar{b}_{i+1\,\mathbf{mod}\,n}$ and then $P_{i+1\,\mathbf{mod}\,n}$ invokes the next job $Q_{i+1\,\mathbf{mod}\,n}$. More complicated scheduling policies can also be expressed by changing the structure of $P_1, \dots, P_{n-1}$. ■

This expressive power comes at the cost of the possibility of deadlock. For example, in π_F-calculus we can write process like $(\boldsymbol{\nu}\bar{a}a)(\boldsymbol{\nu}\bar{b}b)(!a().\bar{b}\langle\rangle \mid !b().\bar{a}\langle\rangle)$.

As we have already seen, the π_F-calculus can model global nondeterminism that arises due to a race condition. For instance, the following process is an example of race condition.

$$(\boldsymbol{\nu}\bar{a}\bar{a})(\bar{a}\langle\rangle \mid !a().P_1 \mid !a().P_2)$$

This kind of process cannot be written in session-typed calculi that correspond to linear logic [19, 133].

Functional programming can also be done using the π_F-calculus. As we shall see in Chapter 6, the π_F-calculus can encode the simply typed λ-calculus.

However, there are things that π_F-process cannot represent due to the absence of non-replicated inputs. It seems impossible to encode imperative features such as reference cells and locks in the π_F-calculus. The following is the typical way to encode a read operation $\mathbf{let}\,x = !\ell\,\mathbf{in}\ P$ of a reference ℓ in the asynchronous π-calculus [51]:

$$\ell(x).(\bar{\ell}\langle x\rangle \mid P).$$

The reference ℓ storing a value v is just an output message $\bar{\ell}\langle v\rangle$ and the read operation is just an input at ℓ.[3] Note that the above process immediately emits a message at $\bar{\ell}$ with the new content of the reference, which is unchanged in the case for read operation. Similarly, a write operation $\ell := v; P$ is encoded as $\ell(\_).(\ell\langle v\rangle \mid P)$, where the underscore represents a name that does not appear in P. Since this encoding relies on the presence of non-replicated input, we cannot use this encoding in the π_F-calculus. On the other hand, non-replicated inputs

[3]Here we are assuming that ℓ and $\bar{\ell}$ are connected by $\boldsymbol{\nu}$ and that there is always exactly one output of the form $\bar{\ell}\langle x\rangle$ that can be accessed.

are usually implemented by using reference cells; a reference cell is used to record whether we have used the input or not. From these observations, we think that the presence of reference cells is equivalent to the presence of the non-replicated inputs. Hence, we conjecture that it is impossible to encode reference cells in the π_F-calculus and the lack of these imperative features is a reason why we succeeded to give a corresponding categorical structure to the π_F-calculus.

3.4 Necessity of the η-rule

When we introduced the axiomatic semantics of the π_F-calculus, we have explained that (E-Eta) is problematic from an operational viewpoint, but is necessarily to have this rule. To make the long story short, (E-Eta) is necessary if π_F-calculus were to correspond to a certain category (not limited to compact closed Freyd category). This section formally proves this argument under some mild assumptions. We think that this gives a certain explanation on why categorical type-theoretic correspondence for π-calculus has not been discovered.

Let us forget about π_F-theory for a moment and reconsider what it means to have a Curry-Howard-Lambek correspondence. To establish a Curry-Howard-Lambek correspondence is to find a nice algebraic or categorical structure of terms. For example, the original Curry-Howard-Lambek correspondence reveals the cartesian closed structure of λ-terms. Such a nice structure would become visible only when appropriate notions of composition and of equivalence could be identified, such as substitution and $\beta\eta$-equivalence for the λ-calculus.

In the context of π-calculus (or other process calculi) "parallel composition + hiding" [52] has been used to compose processes. Let us explain what "parallel composition + hiding" means using well-sorted π_F-processes. Given

$$\vec{x} : \vec{T},\ \vec{y} : \vec{S} \vdash P : \diamond \quad \text{and} \quad \vec{w} : \vec{S}^{\perp},\ \vec{u} : \vec{U} \vdash Q : \diamond,$$

their composite via $(\vec{y}, \vec{w})$is defined as

$$\vec{x} : \vec{T},\ \vec{u} : \vec{U} \vdash (\boldsymbol{\nu}_{\vec{S}}\, \vec{y}\vec{w})(P \mid Q) : \diamond.$$

This kind of composition appears quite often in logical studies of π-calculi [3, 11, 54]. It also plays a central role in *interaction category paradigm* proposed by Abramsky, Gay and Nagarajan [5].

So it remains to determine an equivalence on π_F-processes, appropriate for our purpose. This section examines behavioral equivalences proposed and studied in the literature. Among the various behavioral equivalence, we start by considering weak barbed congruence because it is one of the most widely used equivalences.

Let us consider a category-like structure $\mathcal{C}$ in which an object is a sort and a morphism is an equivalence class of π_F-processes modulo weak barbed congruence $\approxeq^c$. More precisely, a morphism from T to S is a process $x\colon T,\ y\colon S^{\perp} \vdash P\colon \diamond$ modulo weak barbed congruence (and renaming of free names x and y). Then the composition (i.e. "parallel composition + hiding") is well-defined on equivalence classes, because barbed congruence is a congruence. This is a fairly natural setting.

We have a strikingly negative result.

Theorem 3.5. $\mathcal{C}$ is not a category.

Proof. In every category, if $f : A \longrightarrow A$ is a left-identity on A (i.e. $f \circ g = g$ for every $g : A \longrightarrow A$), then f is the identity on A. The process $a : \mathbf{ch}^o[], \bar{b} :$

$\mathbf{ch}^i[] \vdash {!a()}.\bar{b}\langle\rangle : \diamond$ seen as a morphism $(\mathbf{ch}^o[]) \longrightarrow (\mathbf{ch}^o[])$ is a left-identity but not the identity. The former means that $c\colon \mathbf{ch}^o[], \bar{b}\colon \mathbf{ch}^i[] \vdash \Big((\boldsymbol{\nu}\bar{a}a)({!a()}.\bar{b}\langle\rangle \mid P)\Big) \approxeq^c P\{\bar{b}/\bar{a}\}$ for every $c\colon \mathbf{ch}^o[], \bar{a}\colon \mathbf{ch}^i[] \vdash P\colon \diamond$. This holds for weak barbed congruence. Being a right-identity means that $(\boldsymbol{\nu}\bar{b}b)(P \mid a \hookrightarrow \bar{b}) \approxeq^c P\{a/b\}$, which is exactly the same as requiring the (E-Eta) to hold. However, as we explained, weak barbed congruence does not satisfy the rule (E-Eta). $\square$

There are few points worth noting. First, it should be emphasized that this argument also applies to many other equivalences such as *must-testing equivalence* [32]. Second, race condition seems essential for the proof, specifically, for the part proving that the process ${!a()}.\bar{b}\langle\rangle$ is not the identity. Recall that when we demonstrated that ${!a()}.\bar{b}\langle\rangle$ does not satisfy (E-Eta) we used the process

$$(\boldsymbol{\nu}\bar{a}a)(\bar{a}\langle\rangle \mid {!a()}.\bar{o}\langle\rangle \mid (\boldsymbol{\nu}\bar{b}b)({!a()}.\bar{b}\langle\rangle \mid \mathbf{0}))$$

where there are two receivers that can receive a message send via $\bar{a}$. Session-typed calculi in Caires, Pfenning and Toninho [18, 19], which correspond to linear logic, do not seem to suffer from this problem. In our understanding, this is due to the race-freedom of their calculi.

Remark 3.6. The argument in the proof of Theorem 3.5 is widely applicable to **i**/**o**-typed calculi, not specific to the π_F-calculus. In particular, **i**/**o**-separation (i.e. absence of $\mathbf{ch}^{i/o}[\vec{T}]$) has no effect.

The above theorem says that many conventional behavioral equivalences for the π-calculus induce only *semicategories*, which may not have identities for some objects. This situation is reminiscent of the β-theory of the λ-calculus, of which categorical model is given by semi-categorical notions [50].[4] Adding the rule (E-Eta) resolves this problem and this is the reason why we named the rule "η-rule".

[4]To be more precise, the situation of π_F-calculus and β-theory of the λ-calculus is slightly different. The β-theory of the λ-calculus is modeled using semi-functors and semi-adjunctions, but these notions are built upon (ordinary) categories not semicategories.

Chapter 4

Categorical Semantics

Categorical semantics attempts to model programming languages by mapping a (typed) program $\Gamma \vdash M : \tau$ to a morphism $[\![M]\!] \colon [\![\Gamma]\!] \to [\![\tau]\!]$ in some category, where the types are interpreted as objects in that category. In a categorical type theoretic approach, typically the interpretation is not given in a specific category but in categories belonging to a certain class of categories $\mathscr{C}$. There are properties that we would like to expect against the semantics. The following two properties are of particular importance:

Soundness $M \simeq N$ implies $[\![M]\!]_{\mathcal{C}} = [\![N]\!]_{\mathcal{C}}$ for every $\mathcal{C} \in \mathscr{C}$

Completeness $[\![M]\!]_{\mathcal{C}} = [\![N]\!]_{\mathcal{C}}$ for every $\mathcal{C} \in \mathscr{C}$ implies $M \simeq N$

Here we are assuming that a notion of program equivalence $\simeq$ is already given. Soundness ensures that we can prove equivalence of pairs of programs by showing their equality in the model. The completeness, on the other hand, ensures that we can use the semantics for proving that a pair of programs are inequivalent.

The purpose of this chapter is to give, for π_F-calculus, a precise analog of this standard categorical type theory. We introduce the class of *compact closed Freyd categories* and show that the interpretation of π_F-processes in compact closed Freyd categories are sound and complete with respect to π_F-theories. A process in context $\Delta \vdash P : \diamond$ will be interpreted as a morphism $[\![P]\!] \colon [\![\Delta]\!] \to I$, where I is the unit object of a monoidal category. The completeness will be shown by constructing a term model, a compact closed Freyd category that is syntactically built from the π_F-calculus; this is a categorification of the Lindenbaum-Tarski construction. We also show that every model of π_F-calculus is equivalent to a functor with the term model as domain, preserving the compact closed Freyd structure. The correspondence between models and structure preserving functors is the standard criteria required for a correspondence between a programming language (or a type theory) and a categorical structure.

This chapter is organized as follows. Section 4.1 defines *compact closed Freyd categories* as well as the interpretation of π_F-processes. The soundness of the interpretation is also proved in this section. In the beginning of Section 4.1, we also give an informal explanation on the interpretation so the readers may read this part even if they are not familiar with category theory. In Section 4.2, we first define the term model and prove the completeness using the term model. Then we end the chapter by proving the correspondence between models and structure preserving functors.

4.1 Compact Closed Freyd Categories and Interpretation of π_F-processes

This section defines the interpretation of π_F-processes in compact closed Freyd categories and proves the soundness of the interpretation. We first give an informal overview of the interpretation in Section 4.1.1. Section 4.1.2 formally defines compact closed Freyd categories and the interpretation for π_F-processes is given in Section 4.1.3. The proof of the soundness is treated as an individual section (Section 4.1.4).

4.1.1 Overview

Before getting into the technical details of the interpretation, let us explain the ideas behind the interpretation without heavily using categorical notions. We employ the idea that categories can be regarded as a deductive system [75, 77] and give a rather "syntactical" explanation to the interpretation. We informally present compact closed Freyd category as a deductive system and discuss how to interpret π_F-processes by using this deductive system.

As repeatedly mentioned, compact closed Freyd category is a categorical structure that has both closed Freyd and compact closed structures. Therefore, to explain what compact closed Freyd category is, we first need to explain closed Freyd and compact closed categories.

A *closed Freyd category* is a model of higher-order programs with side effects. It has, among others, the structures to interpret the function type $A \Rightarrow B$ and its constructor and destructor, namely, abstraction $\lambda x.t$ and application $t\,u$. Under some simplification, closed Freyd category, when regarded as a deductive system, can be seen as a typed λ-calculus with pairings. As the usual λ-calculus, compact closed Freyd category can duplicate variables; in terms of logic, contraction is admissible.

A *compact closed category* can be seen as multiplicative linear logic [43] (MLL for short) with the left rule:[1]

$$\frac{\Delta, A, A^* \vdash I}{\Delta \vdash I} \qquad \left[\frac{\Delta \vdash A^* \qquad \Delta \vdash A}{\Delta, \Delta \vdash I}\right]. \tag{4.1}$$

(The right rule is the companion, which itself is derivable in MLL.)

Since we define a *compact closed Freyd category* as the combination of the two structures mentioned above, compact closed Freyd category inherits the constructors from the two structures. It has the structures corresponding to the following type constructors:

$$\text{(closed Freyd)} \quad I,\ A \otimes B,\ A \Rightarrow B \qquad \text{(compact closed)} \quad I,\ A \otimes B,\ A^*.$$

Note that the pair type $A \otimes B$ (as well as the unit I) coming from the closed Freyd structure is identified with that from the compact closed structure. Inference rules for a compact closed Freyd category are those for functional languages and the above rules of the compact closed structure.

Roughly speaking, interpreting π_F-calculus in a compact closed Freyd category is to interpret it by using these constructs. As mentioned in Section 1.3, following the observation made by Sangiorgi [115], we regard

[1] From the categorical viewpoint, the left rule corresponds to precomposing the unit morphism $\eta\colon A \otimes A^* \to I$ of a compact closed category (Definition 2.6). The right rule corresponds to postcomposing the counit morphism $\varepsilon\colon A^* \otimes A \to I$.

- an output $\bar{a}\langle\vec{x}\rangle$ as an application of a function $\bar{a}$ to a tuple $\langle\vec{x}\rangle$, and
- an input $!a(\vec{x}).P$ as an abstraction $(\vec{x}).P$ (or $\lambda\vec{x}.P$) located at a.

We interpret the output action by using the function application. Hence the type $\mathbf{ch}^o[T]$ is regarded as a function type $T \Rightarrow I$ (where the unit type I is the type for processes i.e. $\diamond$); then the typing rule for output actions becomes

$$\frac{\Delta, \bar{a}\colon (T \Rightarrow I), x\colon T \vdash \bar{a} : T \Rightarrow I \qquad \Delta, \bar{a}\colon (T \Rightarrow I), x\colon T \vdash x : T}{\Delta, \bar{a}\colon (T \Rightarrow I), x\colon T \vdash \bar{a}\langle x\rangle : I}$$

The type $\mathbf{ch}^i[T]$ is understood as $(T \Rightarrow I)^*$; the input-prefixing rule becomes

$$\frac{\dfrac{}{\Delta, a\colon (T \Rightarrow I)^* \vdash a : (T \Rightarrow I)^*} \qquad \dfrac{\Delta, a\colon (T \Rightarrow I)^*, x\colon T \vdash P : I}{\Delta, a\colon (T \Rightarrow I)^* \vdash (x).P : T \Rightarrow I}}{\Delta, a\colon (T \Rightarrow I)^* \vdash\, !a(x).P : I}$$

This derivation directly expresses the intuition that an input-prefixing is abstraction followed by allocation; here allocation is interpreted by using the compact closed structure, i.e. connection of ports.

The remaining constructs $P \mid Q$ and $(\boldsymbol{\nu}\bar{a}a)P$ are also easy to interpret. We regard parallel composition as pairing. Because we have the compact closed structure, we can use the (right) introduction rule of $\otimes$ in MLL,

$$\frac{\Delta \vdash P : I \quad \Delta \vdash Q : I}{\Delta, \Delta' \vdash \langle P, Q\rangle : I \otimes I,}$$

as a rule for pairing. Since (1) $I \otimes I$ is isomorphic to I and (2) contractions, which comes from the closed Freyd structure, are allowed, the following rule is derivable from the above rule.

$$\frac{\Delta \vdash P : I \quad \Delta \vdash Q : I}{\Delta \vdash P \mid Q : I}$$

Since this rule corresponds to the typing rule of parallel execution, this means that we can interpret parallel composition. The name restriction also has a natural derivation:

$$\frac{\Delta, a\colon (T \Rightarrow I)^*, \bar{a}\colon (T \Rightarrow I) \vdash P : I}{\Delta \vdash (\boldsymbol{\nu}\bar{a}a)P : I}$$

that corresponds to the left rule in (4.1).

4.1.2 Compact closed Freyd category

We now give the formal definition of compact closed Freyd categories. As briefly explained, compact closed Freyd category is defined as a combination of compact closed category and closed Freyd category. We start by defining the missing ingredient, namely the (closed) Freyd category.

Definition 4.1 (Freyd category [111]). A *Freyd category* is given by

- a category with (chosen) finite products $(\mathcal{C}, \otimes, I)$,
- a monoidal category $(\mathcal{K}, \otimes, I, l, r, \alpha)$ and

- an identity-on-object strict symmetric monoidal functor $J\colon \mathcal{C} \to \mathcal{K}$.

■

The categories $\mathcal{C}$ and $\mathcal{K}$ are called the "category of values" and "category of computations" respectively and we will use these term too. The reason they are named as such comes from the fact that Freyd categories were introduced in the study of computational effects [111] and there values and computations were interpreted as morphisms in $\mathcal{C}$ and $\mathcal{K}$ respectively. We note that $\mathbf{Obj}(\mathcal{C}) = \mathbf{Obj}(\mathcal{K})$ because J is an identity-on-object functor.

For simplicity, in what follows, *we only consider Freyd categories whose monoidal and cartesian structures are strict.* Since every Freyd category is equivalent to a strict version of Freyd category the restriction to strictness is not very severe.

Remark 4.1. The above definition is a special case of the original definition of Freyd category [111]. The original definition uses the notion of premonoidal categories [110]: the original definition can be obtained by relaxing the requirement of $\mathcal{K}$ being a monoidal category to premonoidal category in the above definition.

We restrict the definition to monoidal categories to capture concurrency. A premonoidal category is "a monoidal category whose monoidal product is not necessarily a bifunctor" i.e. $(\mathrm{id} \otimes f); (g \otimes \mathrm{id})$ may differ from $(g \otimes \mathrm{id}); (\mathrm{id} \otimes f)$. The difference between these two morphisms models a computation that is sensitive to evaluation order such as computational effects. If we use the syntax of the computational λ-calculus [97], the difference between the two morphisms can be expressed by the following inequality:

$$\mathbf{let}\, x = M\, \mathbf{in}\, \mathbf{let}\, y = N\, \mathbf{in}\, L \qquad \neq \qquad \mathbf{let}\, y = N\, \mathbf{in}\, \mathbf{let}\, x = M\, \mathbf{in}\, L.$$

On the other hand, the bifunctoriality of monoidal products allows us to equate the above two expressions. Hence, we may consider that M and N are concurrently executed because the evaluation order does not matter. ■

Definition 4.2 (Compact closed Freyd category)**.** A Freyd category $(\mathcal{C}, \mathcal{K}, J)$ is a *compact closed Freyd category* if

- $\mathcal{K}$ is a compact closed category with the unit $\eta_A\colon I \to A \otimes A^*$ and the counit $\varepsilon_A\colon A^* \otimes A \to I$ and
- the functor $J\colon \mathcal{C} \to \mathcal{K}$ has a (chosen) right adjoint $I \Rightarrow (-)\colon \mathcal{K} \to \mathcal{C}$.

We write $J\colon \mathcal{C}\underset{\perp}{\rightleftarrows}\mathcal{K}\colon I \Rightarrow (-)$, $J\colon \mathcal{C}\underset{\perp}{\rightleftarrows}\mathcal{K}$ or even J to represent a compact closed Freyd category; the last notation is used when we do not have to deal with the actual data such as morphisms in $\mathcal{C}$ and $\mathcal{K}$. ■

The readers might be confused because we defined compact closed Freyd category without defining *closed* Freyd category. This is not a mistake.

A closed Freyd category is a Freyd category with an additional data, but in the case of compact closed Freyd category, this data can be defined using the other data compact closed Freyd category has. Specifically, a *closed Freyd category* [111] is a Freyd category together with a right adjoint for the functor $J(-) \otimes A$, for each object A. The right adjoint is often denoted by $A \Rightarrow (-)$ and $A \Rightarrow B$ is called the *Kleisli exponential.*

Proposition 4.1. Every compact closed Freyd category $J\colon \mathcal{C}\underset{\perp}{\rightleftarrows}\mathcal{K}\colon I \Rightarrow (-)$ is a closed Freyd category.

Proof. The following natural isomorphisms defines the Kleisli exponential:

$$\begin{aligned}\mathcal{K}(J(A) \otimes B, C) &\cong \mathcal{K}(J(A), B^* \otimes C)\\ &\cong \mathcal{C}(A, I \Rightarrow (B^* \otimes C)).\end{aligned}$$

□

This means that the "higher-order structure" of compact closed Freyd category is derivable; this fact will play an important role in Chapter 6 where we compare higher-order calculi with the π-calculus. We write $\Lambda_{A,B,C}$ for the natural isomorphism from $\mathcal{K}(J(A)\otimes B, C)$ to $\mathcal{C}(A, B \Rightarrow C)$ and $\mathbf{eval}_{A,B}\colon J(A \Rightarrow B)\otimes A \to B$ for the counit of the adjunction. These morphisms will be used to interpret abstractions (which is used to model input prefixing) and applications (which is used to model output actions).

Let us now give some examples of compact closed Freyd categories.

Example 4.1. One of the most elementary examples of a compact closed Freyd category is (the strict monoidal version of) $J\colon \mathbf{Sets} \underset{\perp}{\rightleftarrows} \mathbf{Rel}\colon \mathcal{P}$. Here J is the identity-on-object functor that maps a function to its graph and $\mathcal{P}$ is the "power set functor" that maps a relation $\mathcal{R} \subseteq A \times B$ to a function $\mathcal{P}(\mathcal{R}) \overset{\text{def}}{=} \{(S_A, S_B) \mid S_B = \{b \mid a \in S_A, a \mathrel{\mathcal{R}} b\}\}$.

A similar, but non-degenerated example (i.e. an example with $A^* \ncong A$) is obtained by replacing sets with posets, functions with monotone functions and relations with downward closed relations. ■

Example 4.2. A more sophisticated example is taken from Laird's game-semantic model of π-calculus [74]. Precisely speaking, the model in [74] itself is not compact closed Freyd, but it is not difficult to define its variant that is compact closed Freyd.[2]

This model is important since it is fully abstract with respect to may-testing equivalence [74, Theorem 1]. Hence our framework has a (syntax-free) model that captures the may-testing equivalence. ■

4.1.3 Interpretation

This section defines the interpretation of π_F-processes in compact closed Freyd categories. The interpretation map $[\![-]\!]$ maps sorts and sort environments to objects as usual, and a well-typed process $\Delta \vdash P : \diamond$ to a morphism $[\![P]\!]\colon [\![\Delta]\!] \to I$ in $\mathcal{K}$.

Figure 4.1 defines the interpretation of sorts and processes. Here the morphism $\pi^{\Delta}_x\colon [\![\Delta]\!] \to [\![T_j]\!]$ is the j-th projection provided that $\Delta = (y_1\colon T_1, \dots, y_n\colon T_n)$ and $x = y_j$. The definition of the interpretation is obtained by formalizing the ideas presented in Section 4.1.1. For example, the interpretation of $!a(\vec{x}).P$ is the abstraction Λ (from the closed Freyd structure) followed by "name allocation" ε (from the compact closed structure).

Example 4.3. In order to better understand how reductions are modeled, let us consider $y : T \vdash (\boldsymbol{\nu}\bar{a}a)(\bar{a}\langle y\rangle \mid !a(x).P) : \diamond$, where $\bar{a}, a, y \notin \mathbf{fn}(P)$ and $a\colon \mathbf{ch}^i[T]$. By (E-Beta) and (E-GC), this process is equal to $P\{y/x\}$. It is natural to expect that the interpretations of the two processes coincide; indeed it does. As the following calculation indicates, our semantics factorizes the reduction into

[2](For readers familiar with game semantics) To obtain a compact closed Freyd model we only need to allow non-negative arenas as objects, and thus the modification does not have any impact.

$\boxed{[\![T]\!]}$

$$[\![\mathbf{ch}^i[T_1,\dots,T_n]]\!] \stackrel{\text{def}}{=} (([\![T_1]\!] \otimes \cdots \otimes [\![T_n]\!]) \Rightarrow I)^*$$
$$[\![\mathbf{ch}^o[T_1,\dots,T_n]]\!] \stackrel{\text{def}}{=} ([\![T_1]\!] \otimes \cdots \otimes [\![T_n]\!]) \Rightarrow I$$

$\boxed{[\![\Delta]\!]}$ $\qquad [\![x_1\colon T_1,\dots,x_n\colon T_n]\!] \stackrel{\text{def}}{=} [\![T_1]\!] \otimes \cdots \otimes [\![T_n]\!]$

$\boxed{[\![\Delta \vdash P : \diamond]\!]}$

$$\begin{aligned}
[\![\Delta \vdash \mathbf{0} : \diamond]\!] &\stackrel{\text{def}}{=} J(!_\Delta)\\
[\![\Delta \vdash !a(\vec{x}).P : \diamond]\!] &\stackrel{\text{def}}{=} J(\langle \pi^\Delta_a, \Lambda_{\Delta,\vec{T},I}([\![\Delta,\vec{x} : \vec{T} \vdash P : \diamond]\!])\rangle); \varepsilon_{\mathbf{ch}[\vec{T}]}\\
[\![\Delta \vdash \bar{a}\langle\vec{x}\rangle : \diamond]\!] &\stackrel{\text{def}}{=} J(\langle \pi^\Delta_{\bar{a}}, \pi^\Delta_{x_1},\dots,\pi^\Delta_{x_n}\rangle); \mathbf{eval}_{\vec{T},I}\\
[\![\Delta \vdash P \mid Q : \diamond]\!] &\stackrel{\text{def}}{=} J(\mathbf{d}_\Delta); ([\![\Delta \vdash P : \diamond]\!] \otimes [\![\Delta \vdash Q : \diamond]\!])\\
[\![\Delta \vdash (\boldsymbol{\nu} xy)P : \diamond]\!] &\stackrel{\text{def}}{=} (\mathrm{id}_\Delta \otimes \eta_T); [\![\Delta, x : T, y : T^\perp \vdash P : \diamond]\!]
\end{aligned}$$

Figure 4.1: Interpretation of types and processes.

two steps: (1) the "transmission" of the closure $\lambda\vec{x}.P$ by the triangle identity of the compact closed structure, and (2) the β-reduction modelled by **eval** of the closed Freyd structure:

$$\begin{aligned}
&[\![y : T \vdash (\boldsymbol{\nu}\bar{a}a)(\bar{a}\langle y\rangle \mid !a(x).P) : \diamond]\!]\\
&= (\mathrm{id}_T \otimes \eta_{\mathbf{ch}^o[T]}); [\![y : T, \bar{a} : \mathbf{ch}^o[T], a : \mathbf{ch}^i[T] \vdash \bar{a}\langle y\rangle \mid !a(x).P : \diamond]\!]\\
&= (\mathrm{id} \otimes \eta); ([\![y : T, \bar{a} : \mathbf{ch}^o[T] \vdash \bar{a}\langle y\rangle : \diamond]\!] \otimes [\![a : \mathbf{ch}^i[T] \vdash !a(x).P : \diamond]\!])\\
&= (\mathrm{id} \otimes \eta); ((c_{T,\mathbf{ch}^o[T]}; \mathbf{eval}_{T,I}) \otimes (\mathrm{id}_{\mathbf{ch}[T]^*} \otimes J(\Lambda([\![x : T \vdash P : \diamond]\!]))); \varepsilon_{T \Rightarrow I}\\
&= (\mathrm{id}_T \otimes J(\Lambda([\![x : T \vdash P : \diamond]\!]))); c_{T,\mathbf{ch}^o[T]}; \mathbf{eval}_{T,I} && \text{(By triangle identity)}\\
&= (J(\Lambda([\![x : T \vdash P : \diamond]\!])) \otimes \mathrm{id}_T); \mathbf{eval}_{T,I}\\
&= [\![x : T \vdash P : \diamond]\!] && \text{(By the universality of } \mathbf{eval}\text{)}\\
&= [\![y : T \vdash P\{y/x\} : \diamond]\!].
\end{aligned}$$

In the above calculation, we implicitly use derived rules for weakening and exchange (See Lemma 4.4). ■

Example 4.4. The interpretation of a forwarder $a : \mathbf{ch}^i[\vec{T}], \bar{b} : \mathbf{ch}^o[\vec{T}] \vdash a \hookrightarrow \bar{b} : \diamond$ is the counit $\varepsilon_{\mathbf{ch}^o[\vec{T}]} \colon [\![\mathbf{ch}^o[\vec{T}]]\!]^* \otimes [\![\mathbf{ch}^o[\vec{T}]]\!] \longrightarrow I$ in $\mathcal{K}$, which is the one-sided form of the identity. In fact, this can be checked as follows (where $S \overset{\text{def}}{=} \mathbf{ch}^o[\vec{T}]$)

$$\begin{aligned}
&[\![a : \mathbf{ch}^i[\vec{T}], \bar{b} : \mathbf{ch}^o[\vec{T}] \vdash a \hookrightarrow \bar{b}]\!] \\
&= [\![a : \mathbf{ch}^i[\vec{T}], \bar{b} : \mathbf{ch}^o[\vec{T}] \vdash !a(\vec{x}).\bar{b}\langle\vec{x}\rangle]\!] \\
&= J((\mathbf{d}_{S^*}) \otimes \mathrm{id}_S); (\mathrm{id}_{S^*} \otimes c_{S^*,S}); (J(\Lambda([\![a : S^\perp, \bar{b} : S, \vec{x} : \vec{T} \vdash \bar{b}\langle\vec{x}\rangle]\!])) \otimes \mathrm{id}_{S^*}); \\
&\qquad c_{S,S^*}; \varepsilon_S \\
&= J((\mathbf{d}_{S^*}) \otimes \mathrm{id}_S); (J(!_{S^*}) \otimes c_{S^*,S}); (J(\Lambda([\![\bar{b} : S, \vec{x} : \vec{T} \vdash \bar{b}\langle\vec{x}\rangle]\!])) \otimes \mathrm{id}_{S^*}); \\
&\qquad c_{S,S^*}; \varepsilon_S \qquad\qquad \text{(Weakening \& naturality of } \Lambda) \\
&= (\mathrm{id}_{\mathbf{ch}^i[\vec{T}]} \otimes J(\Lambda([\![\bar{b} : \mathbf{ch}[\vec{T}], \vec{x} : \vec{T} \vdash \bar{b}\langle\vec{x}\rangle]\!]))); \varepsilon_{\mathbf{ch}^o[\vec{T}]} \\
&\qquad \text{(By naturality of } c\text{, id} = c_{A,B}; c_{B,A} \text{ and } \mathbf{d}_A; (!_A \otimes \mathrm{id}_A) = \mathrm{id}_A) \\
&= (\mathrm{id}_{\mathbf{ch}^i[\vec{T}]} \otimes J(\Lambda(\mathbf{eval}_{\vec{T},I}))); \varepsilon_{\mathbf{ch}^o[\vec{T}]} \\
&= \varepsilon_{\mathbf{ch}[\vec{T}]}. \qquad\qquad \text{(By } \Lambda(\mathbf{eval}) = \mathrm{id})
\end{aligned}$$

Therefore, the interpretation reflects the fact that a forwarder is the identity in every π_F-theory. ■

Remark 4.2. As we mentioned in Remark 3.1, $[\![(\boldsymbol{\nu}\bar{a}a)P]\!] = [\![P]\!]$ may not hold even if $\bar{a}, a \notin \mathbf{fn}(P)$. For example, let us consider the case where $P = \mathbf{0}$. Then $[\![\vdash P : \mathbf{0}]\!] = \mathrm{id}_I$ whereas $[\![\vdash (\boldsymbol{\nu}_T\,\bar{a}a)\mathbf{0} : \diamond]\!] = \eta_{[\![T]\!]}; J(!_{[\![T]\!]\otimes[\![T]\!]^*})$, which is not equal to the identity.

We now state the soundness theorem that says that if $P = Q$ are provable, then their interpretations are the same in all compact closed Freyd categories.

Definition 4.3. We say that an equational judgement $\Delta \vdash P = Q$ is *valid in* J if $[\![\Delta \vdash P : \diamond]\!]_J = [\![\Delta \vdash Q : \diamond]\!]_J$. Given a set Ax of non-logical axioms, J is a *model of* Ax, written $J \models Ax$, if it validates all judgements in Ax. We write $Ax \rhd \Delta \Vdash P = Q$ if $\Delta \vdash P = Q$ is valid in every J such that $J \models Ax$. ■

Theorem 4.3 (Soundness)**.** If $Ax \rhd \Delta \vdash P = Q$, then $Ax \rhd \Delta \Vdash P = Q$. □

4.1.4 Proof of soundness

The soundness is shown by checking that the interpretations is sound with respect to the logical axioms listed in Figure 3.2. The soundness of each logical axiom is proved by lengthy, but (arguably) simple calculation. The presentation of this subsection is rather dry because calculation takes up most of the part. We believe that the intuition behind the interpretation has already been conveyed by the previous subsection.

For readability, in what follows, we may write $[\![\Delta \vdash P]\!]$ or even $[\![P]\!]$ for $[\![\Delta \vdash P : \diamond]\!]$. We may also omit the semantic bracket and write T for $[\![T]\!]$ especially when the object appears as a superscript/subscript.

We start by giving auxiliary lemmas, which are the semantic counterpart of Exchange and Weakening rules, that will be heavily used throughout this section without further comment.

Lemma 4.4 (Exchange and Weakening)**.**

1. If $\Delta, x : S, y : T, \Delta' \vdash P : \diamond$, then $[\![\Delta, x : S, y : T, \Delta' \vdash P]\!] = (\mathrm{id}_\Delta \otimes c_{S,T} \otimes \mathrm{id}_{\Delta'}); [\![\Delta, y : T, x : S, \Delta' \vdash P]\!]$.
2. Let $\Delta, x : T \vdash P : \diamond$ such that $x \notin \mathbf{fn}(P)$. Then $[\![\Delta, x : T \vdash P]\!] = (\mathrm{id}_\Delta \otimes J(!_T)); [\![\Delta \vdash P]\!]$.

Proof. By straight forward induction on the type derivation. The statement (1) is used in the proof of (2). □

Below we proceed by checking the soundness of each logical axiom one by one. The most nontrivial case is the case for (E-Beta) so we will later prove this as an individual lemma.

Lemma 4.5. Suppose that $\Delta \vdash P$ and $\Delta \vdash Q$. If $P \equiv Q$ by an axiom of the structural congruence, then $[\![\Delta \vdash P]\!] = [\![\Delta \vdash Q]\!]$.

Proof. By easy calculation. □

Lemma 4.6. The interpretation is sound with respect to (E-GC), (E-FOut) and (E-Eta).

Proof.
(Proof for (E-GC))

$$\begin{aligned}
&[\![\Delta \vdash (\boldsymbol{\nu}\bar{a}a)!a(\vec{y}).P]\!]\\
&= (\mathrm{id}_\Delta \otimes \eta_{\mathbf{ch}^o[\vec{T}]}); [\![\Delta, \bar{a} : \mathbf{ch}^o[\vec{T}], a : \mathbf{ch}^i[\vec{T}] \vdash !a(\vec{y}).P]\!]\\
&= (\mathrm{id}_\Delta \otimes \eta_{\mathbf{ch}^o[\vec{T}]}); (\mathrm{id}_\Delta \otimes J(!_{\mathbf{ch}^o[\vec{T}]}) \otimes \mathrm{id}_{\mathbf{ch}^o[\vec{T}]^*}); [\![\Delta, a : \mathbf{ch}^i[\vec{T}] \vdash !a(\vec{y}).P]\!]\\
&= (\mathrm{id}_\Delta \otimes \eta_{\mathbf{ch}^o[\vec{T}]}); (\mathrm{id}_\Delta \otimes J(!_{\mathbf{ch}^o[\vec{T}]}) \otimes \mathrm{id}_{\mathbf{ch}^o[\vec{T}]^*}); (\mathrm{id}_\Delta \otimes J(\mathbf{d}_{\mathbf{ch}^o[\vec{T}]^*}));\\
&\qquad (J(\Lambda_{\Delta a:\mathbf{ch}^i[\vec{T}],\vec{T},I}([\![\Delta, a : \mathbf{ch}^i[\vec{T}], \vec{x} : \vec{T} \vdash P]\!])) \otimes \mathrm{id}_{\mathbf{ch}^o[\vec{T}]^*}); c_{\mathbf{ch}^o[\vec{T}],\mathbf{ch}^o[\vec{T}]^*}; \varepsilon_{\mathbf{ch}^o[\vec{T}]}\\
&= (\mathrm{id}_\Delta \otimes \eta_{\mathbf{ch}^o[\vec{T}]}); (\mathrm{id}_\Delta \otimes J(!_{\mathbf{ch}^o[\vec{T}]}) \otimes \mathrm{id}_{\mathbf{ch}^o[\vec{T}]^*}); (\mathrm{id}_\Delta \otimes J(\mathbf{d}_{\mathbf{ch}^o[\vec{T}]^*}));\\
&\qquad (\mathrm{id}_\Delta \otimes J(!_{\mathbf{ch}^o[\vec{T}]^*})); J(\Lambda_{\Delta,\vec{T},I}([\![\Delta, \vec{x} : \vec{T} \vdash P]\!])) \otimes \mathrm{id}_{\mathbf{ch}^o[\vec{T}]^*}); c_{\mathbf{ch}^o[\vec{T}],\mathbf{ch}^o[\vec{T}]^*}; \varepsilon_{\mathbf{ch}^o[\vec{T}]}\\
&\hspace{20em}\text{(Weakening \& naturality of }\Lambda)\\
&= (\mathrm{id}_\Delta \otimes \eta_{\mathbf{ch}^o[\vec{T}]}); (\mathrm{id}_\Delta \otimes J(!_{\mathbf{ch}^o[\vec{T}]}) \otimes \mathrm{id}_{\mathbf{ch}^o[\vec{T}]^*}); J(\Lambda_{\Delta,\vec{T},I}([\![\Delta, \vec{x} : \vec{T} \vdash P]\!])) \otimes \mathrm{id}_{\mathbf{ch}^o[\vec{T}]^*});\\
&\qquad c_{\mathbf{ch}^o[\vec{T}],\mathbf{ch}^o[\vec{T}]^*}; \varepsilon_{\mathbf{ch}^o[\vec{T}]}\\
&= (\eta_{\mathbf{ch}^o[\vec{T}]} \otimes \mathrm{id}_\Delta); (J(!_{\mathbf{ch}^o[\vec{T}]}) \otimes \mathrm{id}_{\mathbf{ch}^o[\vec{T}]^*} \otimes J(\Lambda_{\Delta,\vec{T},I}([\![\Delta, \vec{x} : \vec{T} \vdash P]\!]))); \varepsilon_{\mathbf{ch}^o[\vec{T}]}\\
&\hspace{24em}\text{(Naturality of } c)\\
&= J(\Lambda_{\Delta,\vec{T},I}([\![\Delta, \vec{x} : \vec{T} \vdash P]\!])); J(!_{\mathbf{ch}^o[\vec{T}]}) \hspace{8em}\text{(Triangle identity)}\\
&= J(!_\Delta)\\
&= [\![\Delta \vdash 0]\!].
\end{aligned}$$

((E-FOut) and (E-Eta)) Follows from the fact that forwarders are interpreted as counits of the compact closed category and the triangle identities. The calculation is similar to that in Example 4.4. □

We are left to show that the interpretation is sound with respect to (E-Beta).

To facilitate the calculation to show the soundness of (E-Beta), we prepare two technical lemmas. These lemmas use the trace operator [62]. A trace operator in a symmetrical monoidal category $(\mathcal{C}, \otimes, I, c)$ is a natural family of functions $\mathrm{Tr}^X_{A,B}\mathcal{C}(A \otimes X, B \otimes X) \to \mathcal{C}(A, B)$ that satisfies certain laws (see [47] for the laws). It should be noted that every compact closed category has a canonical trace [62, 48], i.e. a trace operator uniquely exists in every compact closed category. The canonical trace operator is defined by

$$\mathrm{Tr}^X_{A,B}(f) \overset{\mathrm{def}}{=} (\mathrm{id}_A \otimes \eta_X); (f \otimes \mathrm{id}_{X^*}); (\mathrm{id}_B \otimes (c_{X,X^*}; \varepsilon_X)).$$

Using the trace operator instead of directly using units or counits of the compact closed category simplifies the calculation and makes the proof more readable.

Lemma 4.7. Let $\Delta \vdash (\boldsymbol{\nu}\bar{a}a)(Q \mid !a(\vec{x}).P) : \diamond$ be a well-sorted process such that $a \notin \mathbf{fn}(P, Q)$. Suppose that the type of $\bar{a}$ is $S \overset{\text{def}}{=} \mathbf{ch}[\vec{T}]$. Then

$$\begin{aligned}
&[\![\Delta \vdash (\boldsymbol{\nu}\bar{a}a)(Q \mid !a(\vec{x}).P)]\!] \\
&= J(\mathbf{d}_\Delta); (\mathrm{id}_\Delta \otimes \mathrm{Tr}^S_{\Delta,S}((\mathrm{id}_\Delta \otimes J(\mathbf{d}_S)); (c_{\Delta,S} \otimes \mathrm{id}_S); (\mathrm{id}_S \otimes J(\Lambda([\![\Delta, \bar{a} : S, \vec{x} : \vec{T} \vdash P]\!])))); \\
&\quad [\![\Delta, \bar{a} : S \vdash Q]\!].
\end{aligned}$$

Proof. Follows from the definition of $[\![\Delta \vdash (\boldsymbol{\nu}\bar{a}a)(Q \mid !a(\vec{x}).P)]\!]$ and the definition of the canonical trace operator. ☐

Lemma 4.8. Let $J\colon \mathcal{C}\underset{\perp}{\rightleftarrows}\mathcal{K}$ be a compact closed Freyd category, $f\colon A \otimes X \to X$ be a morphism in $\mathcal{C}$ and $f^\dagger \overset{\text{def}}{=} \mathrm{Tr}^X_{A,X}((\mathrm{id}_A \otimes J(\mathbf{d}_X)); (c_{A,X} \otimes \mathrm{id}_X); (\mathrm{id}_X \otimes J(f)))$. Then we have

$$f^\dagger = J(\mathbf{d}_A); (\mathrm{id}_A \otimes f^\dagger); J(f).$$

Proof. Essentially the same as that of [47, Theorem 7.1.1], where a fixed point operator is defined via a trace operator. ☐

The above lemma says that $(-)^\dagger$ is a parameterized-fixed point operator [47, 122]. Intuitively, it means that a "value f" can be copied in certain situations. Together with Lemma 4.7, informally speaking, we can say that the interpretation of $(\boldsymbol{\nu}\bar{a}a)(Q \mid !a(\vec{x}).P)$ is equal to that of $(\boldsymbol{\nu}\bar{a}a)(Q \mid !a(\vec{x}).P \mid (\vec{x}).P \mid (\vec{x}).P \mid \cdots)$ when $a \notin \mathbf{fn}(P, Q)$. These lemmas together with the observation that the π-calculus can be seen as a process passing calculus is the key to show that the interpretation is sound with respect to (E-Beta). In other words, we think that the rule (E-Beta) says that the abstraction $(\vec{x}).P$ is copied and then transmitted to the output action $\bar{a}\langle\vec{x}\rangle$.

We next prove a lemma that is analogous to the standard substitution lemma.

Definition 4.4. If $\Delta, \bar{a} : \mathbf{ch}^o[\vec{T}] \vdash Q : \diamond$ and $\Delta', \vec{x} : \vec{T} \vdash P : \diamond$ are well-sorted processes such that $\mathbf{fn}(\Delta) \cap \mathbf{fn}(\Delta', \vec{x} : \vec{T} \vdash P) = \emptyset$, then we write $Q\{\Delta' \vdash \bar{a} := (\vec{x}).P\}$ for $(\boldsymbol{\nu}\bar{a}a)(Q \mid !a(\vec{x}).P)$, where a is fresh. Note that $Q\{\Delta' \vdash \bar{a} := (\vec{x}).P\}$ is well-sorted under Δ, Δ'. ■

Lemma 4.9 (Substitution Lemma)**.**

1. $[\![\Delta, \Delta' \vdash \bar{a}\langle\vec{y}\rangle\{\Delta' \vdash \bar{a} := (\vec{x}).P\}]\!] = [\![\Delta, \Delta' \vdash P\{\vec{y}/\vec{x}\}]\!]$.
2. If $\bar{a} \notin \mathbf{fn}(Q)$ then $[\![\Delta, \Delta' \vdash Q\{\Delta' \vdash \bar{a} := (\vec{x}).P\}]\!] = [\![\Delta, \Delta' \vdash Q]\!]$.
3. $[\![\Delta, \Delta' \vdash (Q \mid R)\{\Delta' \vdash \bar{a} := (\vec{x}).P\}]\!] = [\![\Delta, \Delta' \vdash Q\{\Delta' \vdash \bar{a} := (\vec{x}).P\} \mid R\{\Delta' \vdash \bar{a} := (\vec{x}).P\}]\!]$.
4. $[\![\Delta, \Delta' \vdash (!b(\vec{y}).Q)\{\Delta' \vdash \bar{a} := (\vec{x}).P\}]\!] = [\![\Delta, \Delta' \vdash !b(\vec{y}).Q\{\Delta' \vdash \bar{a} := (\vec{x}).P\}]\!]$.
5. $[\![\Delta, \Delta' \vdash ((\boldsymbol{\nu}\bar{b}b)Q)\{\Delta' \vdash \bar{a} := (\vec{x}).P\}]\!] = [\![\Delta, \Delta' \vdash (\boldsymbol{\nu}\bar{b}b)(Q\{\Delta' \vdash \bar{a} := (\vec{x}).P\})]\!]$.

Proof. We only show the proof for case (1). The other cases can be proved with similar arguments.

(1) By weakening, Lemma 4.7 and generalized yanking property of the trace operator, we have

$$\begin{aligned}
&[\![\Delta, \Delta' \vdash \bar{a}\langle\vec{y}\rangle\{\Delta' \vdash \bar{a} := (\vec{x}).P\}]\!]\\
&= (\mathrm{id}_\Delta \otimes \mathrm{Tr}^{\mathbf{ch}^o[\vec{T}]}_{\Delta',\mathbf{ch}^o[\vec{T}]}((c_{\Delta',\mathbf{ch}^o[\vec{T}]}; (\mathrm{id}_{\mathbf{ch}^o[\vec{T}]} \otimes J(\Lambda([\![\Delta', \vec{x} : \vec{T} \vdash P]\!]))));\\
&\qquad [\![\Delta, \bar{a} : \mathbf{ch}^o[\vec{T}] \vdash \bar{a}\langle\vec{y}\rangle]\!]\\
&= (\mathrm{id}_\Delta \otimes J(\Lambda_{\Delta',\vec{T},I}([\![\Delta', \vec{x} : \vec{T} \vdash P]\!]))); [\![\Delta, \bar{a} : \mathbf{ch}^o[\vec{T}] \vdash \bar{a}\langle\vec{y}\rangle]\!].
\end{aligned}$$

Now by the definition of $[\![\Delta, \bar{a} : \mathbf{ch}^o[\vec{T}] \vdash \bar{a}\langle\vec{y}\rangle]\!]$ we conclude that

$$\begin{aligned}
&[\![\Delta, \Delta' \vdash \bar{a}\langle\vec{y}\rangle\{\Delta' \vdash \bar{a} := (\vec{x}).P\}]\!]\\
&= (\mathrm{id}_\Delta \otimes J(\Lambda_{\Delta',\vec{T},I}([\![\Delta', \vec{x} : \vec{T} \vdash P]\!]))); [\![\Delta, \bar{a} : \mathbf{ch}^o[\vec{T}] \vdash \bar{a}\langle\vec{y}\rangle]\!]\\
&= c_{\Delta,\Delta'}; (J(\Lambda_{\Delta',\vec{T},I}([\![\Delta', \vec{x} : \vec{T} \vdash P]\!])) \otimes \mathrm{id}_\Delta); [\![\bar{a} \colon \mathbf{ch}^o[\vec{T}], \Delta \vdash \bar{a}\langle\vec{y}\rangle]\!]\\
&\hspace{20em}\text{(by Exchange and naturality of } c)\\
&= c_{\Delta,\Delta'}; (\mathrm{id}_{\Delta'} \otimes J(\langle\pi^\Delta_{y_1}, \dots, \pi^\Delta_{y_n}\rangle));\\
&\qquad (J(\Lambda_{\Delta',\vec{T},I}([\![\Delta', \vec{x} : \vec{T} \vdash P]\!])) \otimes \mathrm{id}_{\vec{T}}); \mathbf{eval}_{\vec{T},I}\\
&= c_{\Delta,\Delta'}; (\mathrm{id}_{\Delta'} \otimes J(\langle\pi^\Delta_{y_1}, \dots, \pi^\Delta_{y_n}\rangle)); [\![\Delta', \vec{x} : \vec{T} \vdash P]\!]\\
&\hspace{20em}\text{(since } (J(\Lambda(f)) \otimes \mathrm{id}); \mathbf{eval} = f)\\
&= c_{\Delta,\Delta'}; (\mathrm{id}_{\Delta'} \otimes J(\langle\pi^\Delta_{y_1}, \dots, \pi^\Delta_{y_n}\rangle)); [\![\Delta', \vec{y} : \vec{T} \vdash P\{\vec{y}/\vec{x}\}]\!]\\
&= [\![\Delta, \Delta' \vdash P\{\vec{y}/\vec{x}\}]\!].
\end{aligned}$$

□

We are now ready to prove the soundness of (E-Beta).

Lemma 4.10. The interpretation is sound with respect to (E-Beta).

Proof. We need to show that

$$[\![\Delta \vdash (\boldsymbol{\nu}\bar{a}a)(!a(\vec{x}).P \mid C[\bar{a}\langle\vec{b}\rangle])]\!] = [\![\Delta \vdash (\boldsymbol{\nu}\bar{a}a)(!a(\vec{x}).P \mid C[P\{\vec{b}/\vec{x}\}])]\!], \tag{4.2}$$

provided that $a \notin \mathbf{fn}(P, C)$ and $\bar{a} \notin \mathbf{bn}(C)$. Since the interpretation is sound with respect to the rule (E-FOut), we may assume that the context C does not contain $\bar{a}$ in object positions of output actions, as in the proof of (1) of Lemma 3.2.

Instead of showing (4.2), we show that

$$\begin{aligned}
&[\![\Delta \vdash (\boldsymbol{\nu}\bar{a}a)(!a(\vec{x}).P \mid D[\bar{a}\langle\vec{b_1}\rangle] \dots [\bar{a}\langle\vec{b_n}\rangle])]\!]\\
&= [\![\Delta \vdash (\boldsymbol{\nu}\bar{a}a)(!a(\vec{x}).P \mid D[P\{\vec{b_1}/\vec{x}\}] \dots [P\{\vec{b_n}/\vec{x}\}])]\!]
\end{aligned}\tag{4.3}$$

where D is a multi-hole context such that $D[\bar{a}\langle\vec{b_1}\rangle] \dots [\bar{a}\langle\vec{b_n}\rangle] = C[\bar{a}\langle\vec{b}\rangle]$ (which means that $\vec{b} = \vec{b_i}$ for some i) and $\bar{a} \notin \mathbf{fn}(D)$. The claim follows from (4.3) because adding a rule $D[\bar{a}\langle\vec{b_1}\rangle] \dots [\bar{a}\langle\vec{b_n}\rangle]) = (\boldsymbol{\nu}\bar{a}a)(!a(\vec{x}).P \mid D[P\{\vec{b_1}/\vec{x}\}] \dots [P\{\vec{b_n}/\vec{x}\}])$ instead of (E-Beta) to the equational theory does not change the power of the equational theory.

We now show that (4.3) holds. By Lemma 4.7 and Lemma 4.8, we can show that

$$\begin{aligned}
&[\![\Delta \vdash (\boldsymbol{\nu}\bar{a}a)(!a(\vec{x}).P \mid D[\bar{a}\langle\vec{b_1}\rangle] \dots [\bar{a}\langle\vec{b_n}\rangle])]\!]\\
&= J(\mathbf{d}_\Delta); (\mathrm{id}_\Delta \otimes \mathrm{Tr}^S_{\Delta,S}((\mathrm{id}_\Delta \otimes J(\mathbf{d}_S)); (c_{\Delta,S} \otimes \mathrm{id}_S); (\mathrm{id}_S \otimes J(\Lambda([\![\Delta, \bar{a} : S, \vec{x} : \vec{T} \vdash P]\!]))));\\
&\qquad [\![\Delta, \bar{a} : S \vdash D[\bar{a}\langle\vec{b_1}\rangle] \dots [\bar{a}\langle\vec{b_n}\rangle]]\!]\\
&= J(\mathbf{d}_\Delta); (\mathrm{id}_\Delta \otimes \mathrm{Tr}^S_{\Delta,S}((\mathrm{id}_\Delta \otimes J(\mathbf{d}_S)); (c_{\Delta,S} \otimes \mathrm{id}_S); (\mathrm{id}_S \otimes J(\Lambda([\![\Delta, \bar{a} : S, \vec{x} : \vec{T} \vdash P]\!]))));\\
&\qquad [\![\Delta, \bar{a} : S \vdash (D[\bar{a}\langle\vec{b_1}\rangle] \dots [\bar{a}\langle\vec{b_n}\rangle])\{\Delta, \bar{a} : S \vdash \bar{a} := (\vec{x}).P\}]\!].
\end{aligned}\tag{4.4}$$

Since $\bar{a}$ does not appear in object positions of output actions, we have

$$\begin{aligned}&[\![\Delta, \bar{a} : S \vdash (D[\bar{a}\langle\vec{b_1}\rangle] \dots [\bar{a}\langle\vec{b_n}\rangle])\{\Delta, \bar{a} : S \vdash \bar{a} := (\vec{x}).P\}]\!]\\ &= [\![\Delta, \bar{a} : S \vdash (D[P\{\vec{b_1}/\vec{x}\}] \dots [P\{\vec{b_n}/\vec{x}\}])]\!]\end{aligned}$$

by Lemma 4.9. Now the claim follows by substituting the above equation to the equation (4.4) and applying Lemma 4.7. $\square$

Theorem 4.3 follows from lemmas stated above since the lemmas say that the interpretation is sound with respect to all the logical axioms of the theory.

4.2 Theory/Model Correspondence

This section shows the correspondence between compact closed Freyd categories and the π_F-theories.

We first establish a correspondence by showing that the categorical semantics not only sound, but *complete*. By complete we mean that $P = Q$ is provable if $[\![P]\!]_J = [\![Q]\!]_J$ for all compact closed Freyd categories J. The proof of the completeness is by the standard term model construction. It should be noted that soundness and completeness results are rather weak criteria for a categorical type theory correspondence.

We then prove a stronger result: the category of models is equivalent to the category of "structure preserving functors" from the term model to (special kinds of) compact closed Freyd categories. This is the standard criterion for a categorical type theory correspondence and thus this result shows that the correspondence between the π_F-calculus and compact closed Freyd category is rigid. However, there is a caveat: a model only corresponds to a functor whose codomain is a *special kind of* compact closed Freyd category. This means that the term model satisfies some additional axioms reflecting some aspects of the π_F-calculus. We shall also discuss this point in this section.

Be warned that the latter part of this section is quite technical (compared to other parts of this thesis) and requires familiarity with categorical type theory.

4.2.1 Term model and completeness

A *term model* is a category freely generated by the syntax, i.e. a category whose objects are type environments and whose morphisms are terms (i.e. processes in this setting). his section gives a construction of the term model, by which we show completeness.

Given a set Ax of axioms, we define the term model $J_{Ax} \colon \mathcal{C}_{Ax} \underset{\leftarrow}{\overset{\rightarrow}{\perp}} \mathcal{K}_{Ax}$, which we also write as $Cl(Ax)$. The definition of $\mathcal{K}_{Ax}$ is standard, except for the fact that (1) composition is given by "parallel composition plus hiding" instead of substitution and (2) identities are defined as forwarders instead of judgements of the form $x : T \vdash x : T$.

Definition 4.5. Given an equational theory over Ax, we define a category $\mathcal{K}_{Ax}$ by the following data. (In the definition below we assume that all the variables appearing in the sort environments are distinct.)

(Objects) Objects are defined as possibly empty list of sorts. Tensor products are defined by list concatenations and the unit object of the monoidal structure is defined as the empty list. The dual of an object $\vec{T}$ is given by $(\vec{T})^* \overset{\text{def}}{=} \vec{T}^{\perp}$.

(Morphisms) A morphism from $\vec{S}$ to $\vec{T}$ is an equivalence class of type judgements of the form $\vec{x} : \vec{S} \vdash P; \vec{y} : \vec{T}$, where the equivalence relation is defined by $(\vec{x} : \vec{S} \vdash P; \vec{y} : \vec{T}) \sim (\vec{x}' : \vec{S} \vdash Q; \vec{y}' : \vec{T})$ just in case $\vec{x}'\sigma = \vec{x}$ and $\vec{y}'\sigma = \vec{y}$ for some substitution σ, and $\vec{x} : \vec{S}, \vec{y} : \vec{T}^{\perp} \vdash P = Q\sigma$. As usual, the equivalence class of a judgement $\Delta \vdash P; \Sigma$ is denoted $[\Delta \vdash P; \Sigma]$.

(Identity) The identity on $\vec{T}$ is given by $[\vec{x} : \vec{T} \vdash \vec{x} \leftrightharpoons \vec{y};\, \vec{y} : \vec{T}]$.

(Composition) The composite of two morphisms $[\Delta \vdash P; \vec{x} : \vec{T}]$ and $[\vec{y} : \vec{T} \vdash Q; \Sigma]$ is defined as $[\Delta \vdash (\boldsymbol{\nu}\vec{x}\vec{y})(P \mid Q);\, \Sigma]$.

(Monoidal Product) The monoidal product of two morphisms $[\Delta \vdash P; \Sigma]$ and $[\Delta' \vdash Q; \Sigma']$ is given by $[\Delta \vdash P; \Sigma] \otimes [\Delta' \vdash Q; \Sigma'] = [\Delta, \Delta' \vdash P \mid Q; \Sigma, \Sigma']$. The symmetry from $\vec{S} \otimes \vec{T}$ to $\vec{T} \otimes \vec{S}$ is defined as $[\vec{x} : \vec{S}, \vec{x}' : \vec{T} \vdash \vec{x} \leftrightharpoons \vec{y} \mid \vec{x}' \leftrightharpoons \vec{y}'; \vec{y}' : \vec{T}, \vec{y} : \vec{S}]$.

(Unit and Counit) The unit and counit for $\vec{T}$ is defined as $[\_ \vdash \vec{x} \leftrightharpoons \vec{y}; \vec{x} : \vec{T}^{\perp}, \vec{y} : \vec{T}]$ and $[\vec{x} : \vec{T}^{\perp}, \vec{y} : \vec{T} \vdash \vec{x} \leftrightharpoons \vec{y}; \_]$, respectively.

Note that the identity, unit, counit on the unit object is defined as $[\_ \vdash \mathbf{0};\ \_]$. ■

The definition of the value category $\mathcal{C}_{Ax}$ is not as straightforward as that of $\mathcal{K}_{Ax}$. This is because there is no notion of *value* in π-calculus.

It is not that hard to come up with the notion of value for π-calculus if we take a look at how values are defined in λ-calculus. In the case of λ-calculus, values are variables and abstractions. As in the case of the λ-calculus, we may attempt to define values for π-calculus by the grammar $V ::= x \mid (\vec{x}).P$, where P is a process and $(\vec{x}).P$ is called an *abstraction*, with the following typing rules:

$$\frac{x : T \in \Delta}{\Delta \vdash x : T} \qquad \frac{\Delta, \vec{x} \colon \vec{T} \vdash P}{\Delta \vdash (\vec{x}).P : \mathbf{ch}^{o}[\vec{T}]}.$$

(To understand the right rule, recall that $[\![\mathbf{ch}^{o}[\vec{T}]]\!] = [\![\vec{T}]\!] \Rightarrow I$.) Then we may define a morphism from $\vec{T}$ to $\vec{S} = (S_1, \ldots, S_n)$ is an n-tuple $(V_1, \ldots, V_n)$ of values of type $\vec{x} \colon \vec{T} \vdash V_i \colon S_i$ for each i (modulo renaming of $\vec{x}$).

This definition, however, is technically inconvenient. First, the set of values is not a subset of the set of processes. Second, the term obtained by substituting a name in a value with a value does not always become a value. For instance, $(x).\bar{a}\langle y\rangle\{(z).Q/y\}$ is not a value.

Therefore, we employ the following definition for values:

Definition 4.6 (Value)**.** *Values*, written $\Delta \vdash_{\mathrm{v}} P; \Sigma$, are inductively defined by the following rules.

$$\frac{}{\Delta \vdash_{\mathrm{v}} \mathbf{0};\ \_}\ (\text{V-Unit}) \qquad \frac{\Delta, \vec{x} : \vec{T} \vdash P \qquad a \notin \mathbf{fn}(\Delta, \vec{x} : \vec{T})}{\Delta \vdash_{\mathrm{v}} !a(\vec{x}).P; a : \mathbf{ch}^{i}[\vec{T}]^{\perp}}\ (\text{V-In})$$

$$\frac{\Delta, \bar{b} : \mathbf{ch}^{o}[\vec{T}] \vdash !a(\vec{x}).\bar{b}\langle\vec{x}\rangle}{\Delta \vdash_{\mathrm{v}} !a(\vec{x}).\bar{b}\langle\vec{x}\rangle; \bar{b} : \mathbf{ch}^{o}[\vec{T}]^{\perp}}\ (\text{V-Star}) \qquad \frac{\Delta \vdash_{\mathrm{v}} P; \Sigma \qquad \Delta \vdash_{\mathrm{v}} Q; \Sigma' \qquad \mathbf{fn}(\Sigma) \cap \mathbf{fn}(\Sigma') = \emptyset}{\Delta \vdash_{\mathrm{v}} P \mid Q; \Sigma, \Sigma'}\ (\text{V-Par})$$ ■

The definition of values $\Delta \vdash_{\mathrm{v}} P; \Sigma$ might seem unusual at first sight, but we can consider them as ordinary values, i.e. variables and abstractions, placed to some names. If we ignore the names that appear in Σ and regard parallel composition as pairing, then the above definition resembles the intuitive definition of values given in the preceding paragraphs. Discarding the name a, the rule

$$
\begin{aligned}
&f;\mathrm{id}_B = f = \mathrm{id}_A; f && \text{(M1)}\\
&f \otimes \mathrm{id}_I = f = \mathrm{id}_I \otimes f && \text{(M2)}\\
&\mathrm{id}_A \otimes \mathrm{id}_B = \mathrm{id}_{A\otimes B} && \text{(M3)}\\
&f;(g;h) = (f;g);h && \text{(M4)}\\
&f \otimes (g \otimes h) = (f \otimes g) \otimes h && \text{(M5)}\\
&f;g \otimes f';g' = (f \otimes g);(f' \otimes g') && \text{(M6)}\\
&c;(f \otimes g) = (g \otimes f);c && \text{(Sym1)}\\
&(c_{A,B} \otimes \mathrm{id}_C);(\mathrm{id}_B \otimes c_{A,C}) = c_{A,B\otimes C} && \text{(Sym2)}\\
&c_{A,B};c_{B,A} = \mathrm{id}_{A\otimes B} && \text{(Sym3)}\\
&(\eta_A \otimes \mathrm{id}_A);(\mathrm{id}_A \otimes \varepsilon_A) = \mathrm{id}_A && \text{(C1)}\\
&(\mathrm{id}_{A^*} \otimes \eta_A);(\varepsilon_A \otimes \mathrm{id}_{A^*}) = \mathrm{id}_{A^*} && \text{(C2)}\\
&(\mathrm{id}_A \otimes \eta_{A^*});(c_{A,A^*};\varepsilon_A \otimes \mathrm{id}_{A^{**}}) = \mathrm{id}_A && \text{(C3)}\\
&(\mathrm{id}_{(A\otimes B)^*} \otimes \eta_A \otimes \eta_B);(\mathrm{id}_{(A\otimes B)^*} \otimes \mathrm{id}_A \otimes c_{A^*,B} \otimes \mathrm{id}_{B^*});(\varepsilon_{A\otimes B} \otimes \mathrm{id}_{A^*\otimes B^*}) &&\\
&= && \text{(C4)}\\
&\mathrm{id}_{(A\otimes B)^*} &&
\end{aligned}
$$

Figure 4.2: Axiomatization of strict compact closed categories.

(V-In) can be read as a rule that creates an abstraction $(\vec{x}).P$ (or $\lambda\vec{x}.P$), so the rule simply says that an abstraction is a value. The rule (V-Unit) and (V-Par) correspond to saying that the empty tuple is a value and pairing of a value is a value respectively. To understand the rule (V-Star), $\Delta \vdash_{\mathrm{v}} !a(\vec{x}).\bar{b}\langle\vec{x}\rangle; \bar{b} : \mathbf{ch}[\vec{T}]^{\perp}$ should be read as $\Delta \vdash_{\mathrm{v}} \bar{b}; \bar{b} : \mathbf{ch}[\vec{T}]^{\perp}$.

This definition is handy because we can define the composition of values by using "parallel composition plus hiding" as in the case for the producer category. It is easy to check that this is well-defined, i.e. "parallel composition plus hiding" of values is again a value, by induction on the definition of value.

Definition 4.7. Given an equational theory over Ax, we define the category $\mathcal{C}_{Ax}$ as the wide subcategory of $\mathcal{K}_{Ax}$, whose morphisms are equivalence classes that contains a value. We often write $[\Delta \vdash_{\mathrm{v}} P;\Sigma]$ instead of $[\Delta \vdash P;\Sigma]$ when we want to emphasize the fact that the equivalence class contains a value.

We write $J_{Ax}\colon \mathcal{C}_{Ax} \to \mathcal{K}_{Ax}$ for the identity-on-object functor that maps $[\Delta \vdash_{\mathrm{v}} P;\Sigma]$ to $[\Delta \vdash P;\Sigma]$. We also write $\mathit{Cl}(Ax)$ for the triple $(\mathcal{C}_{Ax}, \mathcal{K}_{Ax}, J_{Ax})$. ■

Theorem 4.11. *$\mathit{Cl}(Ax)$ is a compact closed Freyd category for every Ax.* □

In the model $\mathit{Cl}(Ax)$, the interpretation of a process $\Delta \vdash P : \diamond$ is the equivalence class that P belongs to. This fact leads to *completeness*.

Theorem 4.12 (Completeness)**.** *If $Ax \rhd \Delta \Vdash P = Q$, then $Ax \rhd \Delta \vdash P = Q$.* □

The rest of this subsection is devoted to the proof of Theorem 4.11. Readers not interested in the proof may skip this part.

Things we need to check are (1) $\mathcal{K}_{Ax}$ is a compact closed category (2) $\mathcal{C}_{Ax}$ is a cartesian category, (3) J_{Ax} is a strict symmetric monoidal functor and (4) J_{Ax} has the right adjoint. Since (3) is trivial and checking (2) can be done by the argument used to check (1), we only prove (1) and (4).

For technical convenience, let us give an axiomatization of (strict) compact closed categories such that satisfies the axiom (**I**) and (**D**). The axiomatization is given in Figure 4.2. We will show that $\mathcal{K}_{Ax}$ is a compact closed category by checking these axioms.

Lemma 4.13. *$\mathcal{K}_{Ax}$ is indeed a compact closed Freyd category.*

Proof. The proof proceeds by checking that $\mathcal{K}_{Ax}$ satisfies all the axioms listed in Fig. 4.2.

(M1) Let $f \stackrel{\text{def}}{=} [\Delta \vdash P; \vec{x} : \vec{T}]$, where $\vec{x} = x_1, \ldots x_n$ and $\vec{T} = T_1, \ldots, T_n$. We show that $f; \mathrm{id} = f$. Recall that the identity over $\vec{T}$ is given by $[\vec{y} : \vec{T} \vdash \vec{y} \leftrightharpoons \vec{z}; \vec{z} : \vec{T}]$ and $\vec{y} \leftrightharpoons \vec{z}$ is defined as $y_1 \leftrightharpoons z_1 \mid \cdots \mid y_n \leftrightharpoons z_n$.

By definition, we have

$$\begin{aligned} f; \mathrm{id} &= [\Delta \vdash P; \vec{x} : \vec{T}]; [\vec{y} : \vec{T} \vdash y_1 \leftrightharpoons z_1 \mid \cdots \mid y_n \leftrightharpoons z_n; \vec{z} : \vec{T}] \\ &= [\Delta \vdash (\boldsymbol{\nu} \vec{x}\vec{y})(P \mid y_1 \leftrightharpoons z_1 \mid \cdots \mid y_n \leftrightharpoons z_n); \vec{z} : \vec{T}]. \end{aligned}$$

So, it suffices to show that

$$\Delta, \vec{z} : \vec{T}^{\perp} \vdash (\boldsymbol{\nu} \vec{x}\vec{y})(P \mid y_1 \leftrightharpoons z_1 \mid \cdots \mid y_n \leftrightharpoons z_n) = P\{\vec{z}/\vec{x}\}.$$

By structural congruence, we have

$$\begin{aligned} &(\boldsymbol{\nu} \vec{x}\vec{y})(P \mid y_1 \leftrightharpoons z_1 \mid \cdots \mid y_n \leftrightharpoons z_n) \\ &= (\boldsymbol{\nu} x_n y_n)(\cdots (\boldsymbol{\nu} x_2 y_2)((\boldsymbol{\nu} x_1 y_1)(P \mid y_1 \leftrightharpoons z_1) \mid y_2 \leftrightharpoons z_2) \cdots \mid y_n \leftrightharpoons z_n). \end{aligned}$$

We now proceed the proof by a case analysis on whether T_1 is a sort for output or input. If T_1 is a sort for output channel then $y_1 \leftrightharpoons z_1 = y_1 \hookrightarrow z_1$ and thus $(\boldsymbol{\nu} x_1 y_1)(P \mid y_1 \leftrightharpoons z_1) = P\{z_1/x_1\}$ by (1) of Lemma 3.2. If T_1 is a sort for input, then $y_1 \leftrightharpoons z_1 = z_1 \hookrightarrow y_1$ and from the (E-Eta) rule it follows that $(\boldsymbol{\nu} x_1 y_1)(P \mid y_1 \leftrightharpoons z_1) = P\{z_1/x_1\}$. So, in both cases we have $(\boldsymbol{\nu} x_1 y_1)(P \mid y_1 \leftrightharpoons z_1) = P\{z_1/x_1\}$.

By repeating this argument we have

$$\begin{aligned} &(\boldsymbol{\nu} \vec{x}\vec{y})(P \mid y_1 \leftrightharpoons z_1 \mid \cdots \mid y_n \leftrightharpoons z_n) \\ &= (\boldsymbol{\nu} x_n y_n)(\cdots (\boldsymbol{\nu} x_2 y_2)((\boldsymbol{\nu} x_1 y_1)(P \mid y_1 \leftrightharpoons z_1) \mid y_2 \leftrightharpoons z_2) \cdots \mid y_n \leftrightharpoons z_n) \\ &= (\boldsymbol{\nu} x_n y_n)(\cdots (\boldsymbol{\nu} x_2 y_2)(P\{z_1/x_1\} \mid y_2 \leftrightharpoons z_2) \cdots \mid y_n \leftrightharpoons z_n) \\ &= \cdots \\ &= P\{\vec{z}/\vec{x}\}. \end{aligned}$$

The equation $\mathrm{id}; f = f$ is proved similarly.

(M2) Trivial as $\mathrm{id}_I = [\_ \vdash \mathbf{0}; \_]$ and, for every process P, we have $P = \mathbf{0} \mid P = P \mid \mathbf{0}$ by the structural equivalence.

(M3) Trivial because in $\mathcal{K}_{Ax}$ identities are parallel compositions of forwarders and monoidal product is parallel composition.

(M4) Let $f \stackrel{\text{def}}{=} [\Delta \vdash P; \vec{w} : \vec{S}]$, $g \stackrel{\text{def}}{=} [\vec{x} : \vec{S} \vdash Q; \vec{y} : \vec{T}]$ and $h \stackrel{\text{def}}{=} [\vec{z} : \vec{T} \vdash R; \Sigma]$. The equality follows from the following calculation, which uses the scope extrusion rule and the associativity of the parallel composition.

$$\begin{aligned} f; (g; h) &= [\Delta \vdash P; \vec{w} : \vec{S}]; ([\vec{x} : \vec{S} \vdash Q; \vec{y} : \vec{T}]; [\vec{z} : \vec{T} \vdash R; \Sigma]) \\ &= [\Delta \vdash P; \vec{w} : \vec{S}]; [\vec{x} : \vec{S} \vdash (\boldsymbol{\nu} \vec{y}\vec{z})(Q \mid R); \Sigma] \\ &= [\Delta \vdash (\boldsymbol{\nu} \vec{w}\vec{x})(P \mid (\boldsymbol{\nu} \vec{y}\vec{z})(Q \mid R)); \Sigma] \\ &= [\Delta \vdash (\boldsymbol{\nu} \vec{w}\vec{x})(\boldsymbol{\nu} \vec{y}\vec{z})(P \mid Q \mid R); \Sigma] \qquad (\text{since } \mathbf{fn}(\vec{y}, \vec{z}) \cap \mathbf{fn}(P) = \emptyset) \\ &= [\Delta \vdash (\boldsymbol{\nu} \vec{y}\vec{z})(\boldsymbol{\nu} \vec{w}\vec{x})(P \mid Q \mid R); \Sigma] \\ &= [\Delta \vdash (\boldsymbol{\nu} \vec{y}\vec{z})((\boldsymbol{\nu} \vec{w}\vec{x})(P \mid Q) \mid R); \Sigma] \quad (\text{since } \mathbf{fn}(\vec{w}, \vec{x}) \cap \mathbf{fn}(R) = \emptyset) \\ &= ([\Delta \vdash P; \vec{w} : \vec{S}]; [\vec{x} : \vec{S} \vdash Q; \vec{y} : \vec{T}]); [\vec{z} : \vec{T} \vdash R; \Sigma] \\ &= (f; g); h. \end{aligned}$$

(M5) This case follows from the associativity of parallel compositions.

(M6) Let $f \stackrel{\text{def}}{=} [\Delta \vdash P; \vec{x} : \vec{T}]$, $f' \stackrel{\text{def}}{=} [\Delta' \vdash P'; \vec{x}' : \vec{T}']$, $g \stackrel{\text{def}}{=} [\vec{y} : \vec{T} \vdash Q; \Sigma]$ and $g' \stackrel{\text{def}}{=} [\vec{y}' : \vec{T}' \vdash Q'; \Sigma']$. By definition,

$$\begin{aligned} f; g &= [\Delta \vdash (\boldsymbol{\nu}\vec{x}\vec{y})(P \mid Q); \Sigma] \\ f'; g' &= [\Delta' \vdash (\boldsymbol{\nu}\vec{x}'\vec{y}')(P' \mid Q'); \Sigma'] \\ f \otimes f' &= [\Delta, \Delta' \vdash P \mid P'; \vec{x} : \vec{T}, \vec{x}' : \vec{T}'] \\ g \otimes g' &= [\vec{y} : \vec{T}, \vec{y}' : \vec{T}' \vdash Q \mid Q'; \Sigma, \Sigma']. \end{aligned}$$

Hence, we have

$$\begin{aligned} (f; g) \otimes (f'; g') &= [\Delta \vdash (\boldsymbol{\nu}\vec{x}\vec{y})(P \mid Q); \Sigma] \otimes [\Delta' \vdash (\boldsymbol{\nu}\vec{x}'\vec{y}')(P' \mid Q'); \Sigma'] \\ &= [\Delta, \Delta' \vdash (\boldsymbol{\nu}\vec{x}\vec{y})(P \mid Q) \mid (\boldsymbol{\nu}\vec{x}'\vec{y}')(P' \mid Q'); \Sigma, \Sigma'] \\ &= [\Delta, \Delta' \vdash (\boldsymbol{\nu}\vec{x}\vec{y})(P \mid Q \mid (\boldsymbol{\nu}\vec{x}'\vec{y}')(P' \mid Q')); \Sigma, \Sigma'] \\ &\qquad\qquad (\text{since } \mathbf{fn}(\vec{x}, \vec{y}) \cap \mathbf{fn}((\boldsymbol{\nu}\vec{x}'\vec{y}')(P' \mid Q')) = \emptyset) \\ &= [\Delta, \Delta' \vdash (\boldsymbol{\nu}\vec{x}\vec{y})(\boldsymbol{\nu}\vec{x}'\vec{y}')(P \mid Q \mid P' \mid Q'); \Sigma, \Sigma'] \\ &\qquad\qquad (\text{since } (\mathbf{fn}(\vec{x}', \vec{y}')) \cap \mathbf{fn}(P, Q) = \emptyset) \\ &= [\Delta, \Delta' \vdash P \mid P'; \vec{x} : \vec{T}, \vec{x}' : \vec{T}']; [\vec{y} : \vec{T}, \vec{y}' : \vec{T}' \vdash Q \mid Q'; \Sigma, \Sigma'] \\ &= (f \otimes f'); (g \otimes g'). \end{aligned}$$

(Sym1) Let $f \stackrel{\text{def}}{=} [\vec{x} : \vec{S} \vdash P; \vec{y} : \vec{T}]$ and $g \stackrel{\text{def}}{=} [\vec{x}' : \vec{S}' \vdash Q; \vec{y}' : \vec{T}']$. Then,

$$f \otimes g = [\vec{x} : \vec{S}, \vec{x}' : \vec{S}' \vdash P \mid Q; \vec{y} : \vec{T}, \vec{y}' : \vec{T}']$$

by definition. Recall that a symmetry from $\vec{S}' \otimes \vec{S}$ to $\vec{S} \otimes \vec{S}$ is given by $[\vec{w}' : \vec{S}', \vec{w} : \vec{S} \vdash \vec{z} \leftrightharpoons \vec{w} \mid \vec{z}' \leftrightharpoons \vec{w}'; \vec{z} : \vec{S}, \vec{z}' : \vec{S}']$. By an argument similar to that in the proof of case (M1), we can show that

$$\begin{aligned} & c; (f \otimes g) \\ &= [\vec{w}' : \vec{S}', \vec{w} : \vec{S} \vdash \vec{z}\vec{z}' \leftrightharpoons \vec{w}\vec{w}'; \vec{z} : \vec{S}, \vec{z}' : \vec{S}']; [\vec{x}{:}\vec{S}, \vec{x}'{:}\vec{S}' \vdash P \mid Q; \vec{y}{:}\vec{T}, \vec{y}'{:}\vec{T}'] \\ &= [\vec{w}' : \vec{S}', \vec{w} : \vec{S} \vdash (\boldsymbol{\nu}\vec{z}\vec{x})(\boldsymbol{\nu}\vec{z}'\vec{x}')(\vec{z} \leftrightharpoons \vec{w} \mid \vec{z}' \leftrightharpoons \vec{w}' \mid P \mid Q); \vec{y} : \vec{T}, \vec{y}' : \vec{T}'] \\ &= [\vec{w}' : \vec{S}', \vec{w} : \vec{S} \vdash (P \mid Q)\{\vec{w}'/\vec{x}', \vec{w}/\vec{x}\}; \vec{y} : \vec{T}, \vec{y}' : \vec{T}']. \end{aligned}$$

Similarly, we have

$$(g \otimes f); c = [\vec{x}' : \vec{S}', \vec{x} : \vec{S} \vdash (Q \mid P)\{\vec{z}/\vec{y}, \vec{z}'/\vec{y}'\}; \vec{z} : \vec{T}, \vec{z}' : \vec{T}'].$$

Since

$$\begin{aligned} & [\vec{w}' : \vec{S}', \vec{w} : \vec{S} \vdash (P \mid Q)\{\vec{w}'/\vec{x}', \vec{w}/\vec{x}\}; \vec{y} : \vec{T}, \vec{y}' : \vec{T}'] \\ &= [\vec{x}' : \vec{S}', \vec{x} : \vec{S} \vdash P \mid Q; \vec{y} : \vec{T}, \vec{y}' : \vec{T}'] \\ &= [\vec{x}' : \vec{S}', \vec{x} : \vec{S} \vdash Q \mid P; \vec{y} : \vec{T}, \vec{y}' : \vec{T}'] \\ &= [\vec{x}' : \vec{S}', \vec{x} : \vec{S} \vdash (Q \mid P)\{\vec{z}/\vec{y}, \vec{z}'/\vec{y}'\}; \vec{z} : \vec{T}, \vec{z}' : \vec{T}'], \end{aligned}$$

we conclude that $c; (f \otimes g) = (g \otimes f); c$.

(Sym2) and (Sym3) These cases follow from the definition of id and c in $\mathcal{K}_{Ax}$ and the transitivity of forwarders ((2) of Lemma 3.2). In other words, the proof is similar to the case of (C1), which is given below.

(C1) Suppose that $A = T_1, \ldots, T_n$. Recall that

$$\eta_A = [\_ \vdash \vec{x} \leftrightharpoons \vec{y}; \vec{x} : \vec{T}, \vec{y} : \vec{T}^{\perp}] \qquad \varepsilon_A = [\vec{x}' : \vec{T}^{\perp}, \vec{y}' : \vec{T} \vdash \vec{x} \leftrightharpoons \vec{y}; \_]$$

and thus

$$\begin{aligned}
&\eta_A \otimes \mathrm{id}_A = [\vec{z} : \vec{T} \vdash \vec{x} \leftrightharpoons \vec{y} \mid \vec{w} \leftrightharpoons \vec{z}; \vec{x} : \vec{T}, \vec{y} : \vec{T}^{\perp}, \vec{w} : \vec{T}] \\
&\mathrm{id}_A \otimes \varepsilon_A = [\vec{z}' : \vec{T}, \vec{x}' : \vec{T}^{\perp}, \vec{y}' : \vec{T} \vdash \vec{w}' \leftrightharpoons \vec{z}' \mid \vec{x}' \leftrightharpoons \vec{y}'; \vec{w}' : \vec{T}].
\end{aligned}$$

So, we have

$$\begin{aligned}
&(\mathrm{id}_A \otimes \eta_A); (\varepsilon_A \otimes \mathrm{id}_A) \\
&= [\vec{z} : \vec{T} \vdash (\boldsymbol{\nu} \vec{x}\vec{z}')(\boldsymbol{\nu} \vec{y}\vec{x}')(\boldsymbol{\nu} \vec{w}\vec{y}')(\vec{x} \leftrightharpoons \vec{y} \mid \vec{w} \leftrightharpoons \vec{z} \mid \vec{w}' \leftrightharpoons \vec{z}' \mid \vec{x}' \leftrightharpoons \vec{y}'); \vec{w}' : \vec{T})].
\end{aligned}$$

Thanks to the transitivity of forwarders ((3) of Lemma 3.2), we can check that (C1) holds. More concretely, we have

$$\begin{aligned}
&(\boldsymbol{\nu} \vec{x}\vec{z}')(\boldsymbol{\nu} \vec{y}\vec{x}')(\boldsymbol{\nu} \vec{w}\vec{y}')(\vec{x} \leftrightharpoons \vec{y} \mid \vec{w} \leftrightharpoons \vec{z} \mid \vec{w}' \leftrightharpoons \vec{z}' \mid \vec{x}' \leftrightharpoons \vec{y}') \\
&= (\boldsymbol{\nu} \vec{x}\vec{z}')(\boldsymbol{\nu} \vec{y}\vec{x}')((\boldsymbol{\nu} \vec{w}\vec{y}')(\vec{x}' \leftrightharpoons \vec{y}' \mid \vec{w} \leftrightharpoons \vec{z}) \mid \vec{x} \leftrightharpoons \vec{y} \mid \vec{w}' \leftrightharpoons \vec{z}') \\
&= (\boldsymbol{\nu} \vec{x}\vec{z}')(\boldsymbol{\nu} \vec{y}\vec{x}')(\vec{x}' \leftrightharpoons \vec{z} \mid \vec{x} \leftrightharpoons \vec{y} \mid \vec{w}' \leftrightharpoons \vec{z}') \\
&= (\boldsymbol{\nu} \vec{x}\vec{z}')((\boldsymbol{\nu} \vec{y}\vec{x}')(\vec{x} \leftrightharpoons \vec{y} \mid \vec{x}' \leftrightharpoons \vec{z} \mid \vec{w}' \leftrightharpoons \vec{z}') \\
&= (\boldsymbol{\nu} \vec{x}\vec{z}')(\vec{x} \leftrightharpoons \vec{z} \mid \vec{w}' \leftrightharpoons \vec{z}') \\
&= \vec{w}' \leftrightharpoons \vec{z}.
\end{aligned}$$

Hence, we conclude that $(\mathrm{id}_A \otimes \eta_A); (\varepsilon_A \otimes \mathrm{id}_A) = \mathrm{id}_A$.

(C2), (C3) and (C4) The proof is similar to that of the case (C1).

□

Lemma 4.14. The functor J_{Ax} has the right adjoint that associates an object $\vec{T}$ in $\mathcal{K}_{Ax}$ to an object $\mathbf{ch}^o[\vec{T}^{\perp}]$ in $\mathcal{C}_{Ax}$.

Proof. To show that the J_{Ax} has the right adjoint it suffices to show that there is an arrow $e_{\vec{T}} \colon \mathbf{ch}^o[\vec{T}^{\perp}] \to \vec{T}$ universal from J_{Ax} to $\vec{T}$, for each object $\vec{T}$. In other words, we need to show that for each $f \in \mathcal{K}_{Ax}(J_{Ax}(\vec{S}), \vec{T})$, there is exactly one $g \in \mathcal{C}_{Ax}(\vec{S}, \mathbf{ch}^o[\vec{T}^{\perp}])$ such that $J_{Ax}(g); e_{\vec{T}} = f$. Let us define a map φ from $\mathcal{K}_{Ax}(J_{Ax}(\vec{S}), \vec{T})$ to $\mathcal{C}_{Ax}(\vec{S}, \mathbf{ch}^o[\vec{T}^{\perp}])$ by $\varphi([\Delta \vdash P; \vec{x} : \vec{T}]) \overset{\text{def}}{=} [\Delta \vdash_{\mathrm{v}} !a(\vec{x}).P; a : \mathbf{ch}^o[\vec{T}^{\perp}]]$, where a is a fresh name.

First, we show that for every morphism $f \overset{\text{def}}{=} [\Delta \vdash P; \vec{x} : \vec{T}]$, $J_{Ax}(\varphi(f)); e_{\vec{T}} = f$. This can be confirmed by the following calculation:

$$\begin{aligned}
J_{Ax}(\varphi(f)); e_{\vec{T}} &= [\Delta \vdash !a(\vec{x}).P; a : \mathbf{ch}^o[\vec{T}^{\perp}]]; [\bar{a} : \mathbf{ch}^o[\vec{T}^{\perp}] \vdash \bar{a}\langle \vec{y} \rangle; \vec{y} : \vec{T}] \\
&= [\Delta \vdash (\boldsymbol{\nu} \bar{a}a)(!a(\vec{x}).P \mid \bar{a}\langle \vec{y} \rangle)); \vec{y} : \vec{T}] \\
&= [\Delta \vdash P\{\vec{y}/\vec{x}\}; \vec{y} : \vec{T}] && \text{((E-BETA) and (E-GC))} \\
&= [\Delta \vdash P; \vec{x} : \vec{T}] && \text{(By renaming)} \\
&= f.
\end{aligned}$$

Next, we show that for every $h \overset{\text{def}}{=} [\Delta \vdash_{\mathrm{v}} !a(\vec{x}).P; a : \mathbf{ch}^{o}[\vec{T}^{\perp}]]$ in $\mathcal{C}_{Ax}$, $\varphi(J_{Ax}(h); e_{\vec{T}}) = h$. This is shown by the following calculation:

$$\begin{aligned}
\varphi(J_{Ax}(h); e_{\vec{T}}) &= \varphi([\Delta \vdash (\boldsymbol{\nu}\bar{a}a)(!a(\vec{x}).P \mid \bar{a}\langle\vec{y}\rangle); \vec{y} : \vec{T}]) \\
&= \varphi([\Delta \vdash P\{\vec{y}/\vec{x}\}; \vec{y} : \vec{T}]) \qquad \text{(By (E-Beta) and (E-GC))} \\
&= [\Delta \vdash_{\mathrm{v}} !a(\vec{y}).P\{\vec{y}/\vec{x}\}; a : \mathbf{ch}^{o}[\vec{T}^{\perp}]] \\
&= [\Delta \vdash_{\mathrm{v}} !a(\vec{x}).P; a : \mathbf{ch}^{o}[\vec{T}^{\perp}]] \\
&= h.
\end{aligned}$$

Therefore, if $J_{Ax}(g); e_{\vec{T}} = f$ then $g = \varphi(J_{Ax}(g)); e_{\vec{T}}) = \varphi(f)$. This shows the uniqueness of g and completes the proof of adjointness. $\square$

4.2.2 Classifying category

It is natural to expect that $\mathit{Cl}(Ax)$ is the *classifying category* as in the standard categorical type theory. That is to expect

$$\mathbf{Mod}(Ax, J) \cong \mathbf{CCFC}_s(\mathit{Cl}(Ax), J) \qquad \text{in } \mathbf{Sets}. \tag{4.5}$$

holds, where $\mathbf{Mod}(Ax, J)$ is the set of model of Ax in J and $\mathbf{CCFC}_s(\mathit{Cl}(Ax), J)$ is the set of "structure-preserving functors" from $\mathit{Cl}(Ax)$ to J. The equation (4.5) means that to give a model of Ax in J is equivalent to give a structure-preserving functor from the term model to J. Note that the above equation is given in **Sets** rather than **Cats**. This is because we are only working with the empty signature. (We shall come back to this point at the end of this section.)

The set $\mathbf{Mod}(Ax, J)$ of models of Ax in J is defined as follows. If $J \models Ax$, then $\mathbf{Mod}(Ax, J)$ is a singleton set; otherwise $\mathbf{Mod}(Ax, J)$ is the empty set.

We now define the notion of "structure preserving functors" between compact closed Freyd categories. In Section 2.2 we have already seen what "structure preserving functors" are in the case of cartesian categories and compact closed categories; they are finite product preserving functors (Example 2.3) and compact closed functors (Definition 2.7) respectively. We, therefore, are led to the definition of *compact closed Freyd functors* by combining these two notions.

Definition 4.8 (Map of Adjunctions [80]). Let $F\colon \mathcal{C} \underset{\perp}{\rightleftarrows} \mathcal{D}\colon G$ and $F'\colon \mathcal{C}' \underset{\perp}{\rightleftarrows} \mathcal{D}'\colon G'$ be adjunctions. A *map of adjunctions* is a quadruple (L, K, α, β) of functors $L\colon \mathcal{C} \to \mathcal{C}'$ and $K\colon \mathcal{D} \to \mathcal{D}'$ and natural isomorphisms $\alpha\colon KF \to F'L$ and $\beta\colon LG \to G'K$ such that

$$\begin{tikzcd}
\mathcal{D} \arrow[r, "G"] \arrow[d, "K"] & \mathcal{C} \arrow[r, "F"] \arrow[d, "L"] \arrow[dl, phantom, "\overset{\beta}{\Leftarrow}"] & \mathcal{D} \arrow[d, "K"] \arrow[dl, phantom, "\overset{\alpha}{\Leftarrow}"] \\
\mathcal{D}' \arrow[r, "G'"'] & \mathcal{C}' \arrow[r, "F'"'] & \mathcal{D}'
\end{tikzcd}$$

and such that the diagram of hom-sets and adjunctions commutes

$$\begin{tikzcd}
\mathcal{D}(FC, D) \arrow[rr, "\varphi"] \arrow[d, "K"] & & \mathcal{C}(C, GD) \arrow[d, "L"] \\
\mathcal{D}'(KFC, KD) \arrow[d, "\alpha_C;-"] & & \mathcal{C}'(LC, LGD) \arrow[d, "-;\beta_D"] \\
\mathcal{K}'(F'LC, GD) \arrow[rr, "\varphi'"] & & \mathcal{C}'(LC, G'KD)
\end{tikzcd}$$

for all objects $C \in \mathbf{Obj}(\mathcal{C})$ and $D \in \mathbf{Obj}(\mathcal{D})$, where φ and φ' are the adjunction isomorphisms. We say that (L, K) are *strict map of adjunctions* if the associated natural isomorphisms are identities. ■

Definition 4.9 (Compact closed Freyd functor). A *strong compact closed Freyd functor* between compact closed Freyd categories $J\colon \mathcal{C} \underset{\perp}{\rightleftarrows} \mathcal{K}$ and $J'\colon \mathcal{C}' \underset{\perp}{\rightleftarrows} \mathcal{K}'$ is a pair of functors (Φ, Ψ) such that

- $\Phi\colon \mathcal{C} \to \mathcal{C}'$ is a finite product preserving functor,
- $\Psi\colon \mathcal{K} \to \mathcal{K}'$ is strong compact closed functor and
- (Φ, Ψ) is a map of adjunctions between $J \dashv I \Rightarrow -$ and $J' \dashv I \Rightarrow' -$.

We say that (Φ, Ψ) is a *strict compact closed Freyd functor* if Φ is a strict finite product preserving functor, Ψ is a strict compact closed functor and (Φ, Ψ) is a strict map of adjunctions. ■

Given compact closed Freyd categories J and J', we write $\mathbf{CCFC}_s(J, J')$ for the set of strict compact closed Freyd functors from J to J'; we will work with strict functors for simplicity.

Now that we have defined all the notation appearing in (4.5), we can discuss whether (4.5) holds. Unfortunately, the answer is *no*. More precisely, the left-to-right inclusion does not hold in general. This means that the term model satisfies some additional axioms reflecting some aspects of the π_F-calculus.

The additional axioms reflect the definition of the dual $\vec{T}^*$ in the term model. Recall that $\vec{T}^* \overset{\text{def}}{=} \vec{T}^\perp$ by definition, and thus $\vec{T}^{**} = \vec{T}$ and $(\vec{T} \otimes \vec{S})^* = \vec{T}^* \otimes \vec{S}^*$. In other words, the operator $(-)^*$ is strictly involutive and it strictly distributes over the monoidal products. Since this is not the case in every compact closed Freyd category we require these conditions as additional axioms, which we call (I) and (D):

(I) The canonical isomorphism $A^{**} \to A$ in $\mathcal{K}$ is the identity.

(D) The canonical isomorphism $(A \otimes B)^* \to A^* \otimes B^*$ in $\mathcal{K}$ is the identity.

It might be surprising that we need to add these equations as additional axioms because isomorphisms $A^{**} \cong A$ and $(A \otimes B)^* \cong A^* \otimes B^*$ exist in every compact closed category. One might expect that we can strictify these isomorphisms so that the additional axioms (I) and (D) become redundant. This is not possible, however. The point is that the equations also require the value category $\mathcal{C}$ to have isomorphisms $A^{**} \cong A$ and $(A \otimes B)^* \cong A^* \otimes B^*$ (witnessed by the respective identities), which does not always exist.

Remark 4.2. The nullary version of the axiom (D), i.e. the condition that says the canonical isomorphism from I^* to I is the identity, is derivable from the axioms (I) and (D). We give a sketch of the proof. By the axiom (I), we have

$$I^* = I \otimes I^* = I^{**} \otimes I^* \quad \text{and} \quad I = (I^*)^* = (I^* \otimes I)^*.$$

By the axiom (D), we have

$$I^{**} \otimes I = (I^* \otimes I)^*$$

and thus $I^* = I$. ■

Theorem 4.15. If a compact closed Freyd category $J\colon \mathcal{C} \underset{\perp}{\rightleftarrows} \mathcal{K}\colon I \Rightarrow (-)$ satisfies **(I)** and **(D)**, then we have

$$\mathbf{Mod}(Ax, J) \cong \mathbf{CCFC}_s(\mathit{Cl}(Ax), J) \qquad \text{in } \mathbf{Sets}$$

□

We split the proof into three-steps: (1) we define a functor from the classifying category (2) we show that the functor is indeed a compact closed Freyd functor, and then (3) we show that it is unique. In the sequel we only consider the case where $J \models Ax$ because the proof for the other case is trivial.

Definition of the functor Let us define $[\![\Delta \vdash P; \Sigma]\!]$ as $(\eta_\Sigma \otimes \mathrm{id}_\Delta); (\mathrm{id}_\Sigma \otimes [\![\Sigma^\perp, \Delta \vdash P]\!])$, where $[\![\Delta \vdash P]\!]$ is the interpretation of P in $J\colon \mathcal{C} \underset{\perp}{\rightleftarrows} \mathcal{K}$. We define a map Ψ from $\mathcal{K}_{Ax}$ to $\mathcal{K}$ by

$$\Psi([\Delta \vdash P; \Sigma]) \stackrel{\mathrm{def}}{=} [\![\Delta \vdash P; \Sigma]\!].$$

The map Ψ is well-defined thanks to the soundness of the interpretation.

Since an element of $\mathbf{CCFC}_s$ is a pair of functors one for category of computations and another for the value category, we need to construct a functor for the value category. We define such a functor Φ by induction on the derivation of $\Delta \vdash_{\mathrm{v}} P; \Sigma$.

$$\begin{aligned}
\Phi([\Delta \vdash_{\mathrm{v}} \mathbf{0}; \_]) &\stackrel{\mathrm{def}}{=} !_\Delta \\
\Phi([\Delta \vdash_{\mathrm{v}} !a(\vec{x}).\bar{b}\langle\vec{x}\rangle; \bar{b} : \mathbf{ch}^o[\vec{T}]^\perp]) &\stackrel{\mathrm{def}}{=} \pi_a^\Delta \\
\Phi([\Delta \vdash_{\mathrm{v}} !a(\vec{x}).P; a : \mathbf{ch}^i[\vec{T}]^\perp]) &\stackrel{\mathrm{def}}{=} \Lambda([\![\Delta, \vec{x} : \vec{T} \vdash P]\!]) \\
\Phi([\Delta \vdash_{\mathrm{v}} P \mid Q; \Sigma, \Sigma']) &\stackrel{\mathrm{def}}{=} \langle \Phi([\Delta \vdash_{\mathrm{v}} P; \Sigma]), \Phi([\Delta \vdash_{\mathrm{v}} Q; \Sigma']) \rangle
\end{aligned}$$

It is routine to check that the map Φ is well-defined.

(Φ, Ψ) is a strict compact closed Freyd functor To check that (Φ, Ψ) is a strict compact closed Freyd functor we need to check that (1) Ψ is a strict compact closed functor, (2) Φ is a strict finite product preserving functor and (3) (Φ, Ψ) is a strict map of adjunctions.

Before getting into the proof let us state some lemmas on compact closed Freyd categories that satisfy the axioms **(I)** and **(D)**.

Lemma 4.16. Suppose that $\mathcal{K}$ is a compact closed category that satisfies the axioms **(I)** and **(D)**. Then, for all objects A and B, we have

$$\begin{aligned}
\eta_{A^*} &= \eta_A; c_{A,A^*} \\
\varepsilon_{A^*} &= c_{A,A^*}; \varepsilon_A \\
\eta_{A \otimes B} &= (\eta_A \otimes \eta_B); (\mathrm{id}_A \otimes c_{A^*,B} \otimes \mathrm{id}_{B^*}) \\
\varepsilon_{A \otimes B} &= (\mathrm{id}_{A^*} \otimes c_{A,B^*} \otimes \mathrm{id}_B); (\varepsilon_A \otimes \varepsilon_B)
\end{aligned}$$

□

Remark 4.3. The proof showing that Ψ is a strict compact closed functor relies on this lemma, and thus depends on the two additional axioms. To put it another way, we do not have theory/model correspondence for arbitrary compact closed Freyd category because we cannot construct the structure preserving functor from the classifying category if the category in target does not satisfy the two axioms. ■

We now show that Ψ is a strict compact closed functor and that Φ is a strict finite product preserving functor.

Lemma 4.17. The map Ψ described above is a (strict) compact closed functor.

Proof. We start by checking that Ψ is a functor. Recall that identities in the term model is of the form $[\vec{x} : \vec{T} \vdash \vec{x} \leftrightharpoons \vec{y}; \vec{y} : \vec{T}]$ and a single forwarder is interpreted as counit. Therefore, using Lemma 4.16 it follows that the interpretation of multiple forwarders is also a counit, i.e. $[\![\vec{y} : \vec{T}^\perp, \vec{x} : \vec{T} \vdash \vec{x} \leftrightharpoons \vec{y}]\!] = \varepsilon_{\vec{T}}$. The map Ψ therefore preserves identities because

$$\begin{aligned}\Psi([\vec{x} : \vec{T} \vdash \vec{x} \leftrightharpoons \vec{y}; \vec{y} : \vec{T}]) &= [\![\vec{x} : \vec{T} \vdash \vec{x} \leftrightharpoons \vec{y}; \vec{y} : \vec{T}]\!] \\ &= (\eta_{\vec{T}} \otimes \mathrm{id}_{\vec{T}}); (\mathrm{id}_{\vec{T}} \otimes [\![\vec{y} : \vec{T}^\perp, \vec{x} : \vec{T} \vdash \vec{x} \leftrightharpoons \vec{y}; \_]\!]) \\ &= (\eta_{\vec{T}} \otimes \mathrm{id}_{\vec{T}}); (\mathrm{id}_{\vec{T}} \otimes \varepsilon_{\vec{T}}) \\ &= \mathrm{id}_{\vec{T}}\end{aligned}$$

by the triangle identity. Preservation of compositions can also be checked by unfolding the definition of the interpretation:

$$\begin{aligned}&\Psi([\Delta \vdash P; \vec{x} : \vec{T}]; [\vec{y} : \vec{T} \vdash Q; \Sigma]) \\ &= \Psi([\Delta \vdash (\boldsymbol{\nu}\vec{x}\vec{y})(P \mid Q); \Sigma]) \\ &= (\eta_\Sigma \otimes \mathrm{id}_\Delta); (\mathrm{id}_\Sigma \otimes [\![\Sigma^\perp, \Delta \vdash (\boldsymbol{\nu}\vec{x}\vec{y})(P \mid Q)]\!]) \\ &= (\eta_\Sigma \otimes \mathrm{id}_\Delta \otimes \eta_{\vec{T}^*}); (\mathrm{id}_\Sigma \otimes [\![\Sigma^\perp, \Delta, \vec{x} : \vec{T}^\perp, \vec{y} : \vec{T} \vdash Q \mid P]\!]) \\ &= (\eta_\Sigma \otimes \mathrm{id}_\Delta \otimes \eta_{\vec{T}^*}); (\mathrm{id}_\Sigma \otimes J(\mathbf{d})); \\ &\quad (\mathrm{id}_\Sigma \otimes [\![\Sigma^\perp, \Delta, \vec{x} : \vec{T}^\perp, \vec{y} : \vec{T} \vdash Q]\!] \otimes [\![\Sigma^\perp, \Delta, \vec{x} : \vec{T}^\perp, \vec{y} : \vec{T} \vdash P]\!]) \\ &= (\eta_\Sigma \otimes \mathrm{id}_\Delta \otimes \eta_{\vec{T}^*}; c_{\vec{T}^*, \vec{T}}); (\mathrm{id}_{\Sigma,\Sigma^*} \otimes c_{\Delta, \vec{T}, \vec{T}^*}); (\mathrm{id}_\Sigma \otimes [\![\Sigma^\perp, \vec{y} : \vec{T} \vdash Q]\!] \otimes [\![\vec{x} : \vec{T}^\perp, \Delta \vdash P]\!]) \\ &\qquad\qquad \text{(by weakening \& exchange)} \\ &= (\eta_\Sigma \otimes \mathrm{id}_\Delta \otimes \eta_{\vec{T}}); (\mathrm{id}_{\Sigma,\Sigma^*} \otimes c_{\Delta, \vec{T}, \vec{T}^*}); (\mathrm{id}_\Sigma \otimes [\![\Sigma^\perp, \vec{y} : \vec{T} \vdash Q]\!] \otimes [\![\vec{x} : \vec{T}^\perp, \Delta \vdash P]\!]) \\ &\qquad\qquad \text{(by Lemma 4.16)} \\ &= (\eta_\Sigma \otimes \eta_{\vec{T}} \otimes \mathrm{id}_\Delta); (\mathrm{id}_\Sigma \otimes [\![\Sigma^\perp, \vec{y} : \vec{T} \vdash Q]\!] \otimes [\![\vec{x} : \vec{T}^\perp, \Delta \vdash P]\!]) \\ &= [\![\Delta \vdash P; \vec{x} : \vec{T}]\!]; [\![\vec{y} : \vec{T} \vdash Q; \Sigma]\!].\end{aligned}$$

The functor Ψ is not only a functor, but a strict monoidal functor because

$$\Psi([\Delta \vdash P; \Sigma] \otimes [\Delta' \vdash Q; \Sigma']) = [\![\Delta \vdash P; \Sigma]\!] \otimes [\![\Delta' \vdash Q; \Sigma']\!]$$

can be shown by an argument similar to the one we used for proving the preservation of composition.

The last thing to check is that Ψ strictly preserves chosen units and counits. By the definition of Ψ, we have

$$\begin{aligned}&\Psi([\_ \vdash \vec{x} \leftrightharpoons \vec{y}; \vec{T}, \vec{y} : \vec{T}^\perp]) \\ &= \eta_{\vec{T}, \vec{T}^*}; (\mathrm{id}_{\vec{T}, \vec{T}^*} \otimes [\![\vec{x} : \vec{T}^\perp, \vec{y} : \vec{T} \vdash \vec{x} \leftrightharpoons \vec{y}]\!]) \\ &= \eta_{\vec{T}, \vec{T}^*}; (\mathrm{id}_{\vec{T}, \vec{T}^*} \otimes \varepsilon_{\vec{T}}) \\ &= (\eta_{\vec{T}} \otimes \eta_{\vec{T}^*}); (\mathrm{id}_{\vec{T}} \otimes c_{\vec{T}^*, \vec{T}^*} \otimes \mathrm{id}_{\vec{T}}); (\mathrm{id}_{\vec{T}, \vec{T}^*} \otimes \varepsilon_{\vec{T}}) &&\text{(by Lemma 4.16)} \\ &= \eta_{\vec{T}}; (\mathrm{id}_{\vec{T}} \otimes \eta_{\vec{T}^*} \otimes \mathrm{id}_{\vec{T}^*}); (\mathrm{id}_{\vec{T}} \otimes \mathrm{id}_{\vec{T}^*} \otimes c_{\vec{T}, \vec{T}^*}; \varepsilon_{\vec{T}}) \\ &= \eta_{\vec{T}}; (\mathrm{id}_{\vec{T}} \otimes \eta_{\vec{T}^*} \otimes \mathrm{id}_{\vec{T}^*}); (\mathrm{id}_{\vec{T}} \otimes \mathrm{id}_{\vec{T}^*} \otimes \varepsilon_{\vec{T}^*}) &&\text{(by Lemma 4.16)} \\ &= \eta_{\vec{T}}. &&\text{(by triangle identity)}\end{aligned}$$

Similarly, one can show that Ψ preserves counit on the nose. So we have checked that Ψ is a strict compact closed functor. □

Lemma 4.18. For the maps Φ and Ψ defined above, we have $J_{Ax}; \Psi = \Phi; J$.

Proof. It suffices to show that for each value $\Delta \vdash_{\mathrm{v}} P; \Sigma$, $(J_{Ax}; \Psi)([\Delta \vdash_{\mathrm{v}} P; \Sigma]) = (\Phi; J)([\Delta \vdash_{\mathrm{v}} P; \Sigma])$ by straight forward induction on the derivation of $\Delta \vdash_{\mathrm{v}} P; \Sigma$. It should be noted that Lemma 4.16 is needed for the case (V-Par). □

Lemma 4.19. The map Φ described above is a (strict) finite product preserving functor.

Proof. Before getting into the main thread of the proof we note that

$$\Phi([\Delta \vdash_{\mathrm{v}} !a(\vec{x}).\bar{b}\langle\vec{x}\rangle; a : \mathbf{ch}^i[\vec{T}]^{\perp}]) = \pi^{\Delta}_{\vec{b}}.$$

From this equation and the definition of Φ it follows that

$$\Phi([\Delta \vdash_{\mathrm{v}} x \leftrightharpoons y; y : T]) = \pi^{\Delta}_{x}. \tag{4.6}$$

That is a forwarder is mapped to the projection map regardless of whether T is a sort for outputs or inputs. We will use this equation during the main thread of the proof.

We now check that the map Φ is a functor. Using the above equation, it is easy to see that Φ preserves identities because

$$\begin{aligned}
&\Phi([\vec{y} : \vec{T} \vdash \vec{x} \leftrightharpoons \vec{y}; \vec{x} : \vec{T}]) \\
&= \langle \Phi([\vec{y} : \vec{T} \vdash x_1 \leftrightharpoons y_1; x_1 : T_1]), \ldots, \Phi([\vec{y} : \vec{T} \vdash x_n \leftrightharpoons y_n; x_n : T_n])\rangle \\
&= \langle \pi_{y_1}, \ldots, \pi_{y_n}\rangle && \text{(By (4.6))} \\
&= \mathrm{id}_{\vec{T}}.
\end{aligned}$$

The next thing to check is the preservation of composition. We can show that

$$\Phi([\Delta \vdash_{\mathrm{v}} P; \Theta]; [\Theta \vdash_{\mathrm{v}} Q; \Sigma]) = \Phi([\Delta \vdash_{\mathrm{v}} P; \Theta]); \Phi([\Theta \vdash_{\mathrm{v}} Q; \Sigma])$$

by induction on the derivation of $\Theta \vdash_{\mathrm{v}} Q; \Sigma$ with a case analysis on the last rule applied; Lemma 4.18 is needed to show the case (V-In).

To show that Φ is a strict finite product preserving functor, we show that the chosen terminal map and projections are mapped to the chosen terminal map and projections respectively.

The terminal map in the term model is $[\Delta \vdash \mathbf{0}; \_]$ and this is mapped to $!_{\Delta}$ by the definition of Φ.

Recall that the projections in the term model is defined as $[\vec{x} : \vec{S}, \vec{y} : \vec{T} \vdash \vec{x} \leftrightharpoons \vec{z}; \vec{z} : \vec{S}]$. Since forwarders are mapped to projections, we have

$$\begin{aligned}
&\Phi([\vec{x} : \vec{S}, \vec{y} : \vec{T} \vdash \vec{x} \leftrightharpoons \vec{z}; \vec{z} : \vec{S}]) \\
&= \langle \Phi([\vec{x} : \vec{S}, \vec{y} : \vec{T} \vdash x_1 \leftrightharpoons z_1; z_1 : S_1]), \ldots, \Phi([\vec{x} : \vec{S}, \vec{y} : \vec{T} \vdash x_n \leftrightharpoons z_n; z_n : S_n])\rangle \\
&= \langle \pi_{x_1}, \ldots, \pi_{x_n}\rangle && \text{(By (4.6))} \\
&= \pi_1.
\end{aligned}$$

□

We now check that (Φ, Ψ) is a strict map of adjunctions to conclude that (Φ, Ψ) is a strict compact closed Freyd functor. Since one of the conditions, namely Lemma 4.18, is proved, we are left to check the other two conditions:

Lemma 4.20. The two functors Φ and Ψ satisfy (1) $\Psi; I \Rightarrow (-) = I \Rightarrow_{Ax} (-); \Phi$ and (2) $\Psi; \varphi = \varphi_{Ax}; \Phi$. Here $J_{Ax} \dashv I \Rightarrow_{Ax} (-)$ is the adjunction in the term model and φ and φ_{Ax} are the adjunction isomorphisms for $J \dashv I \Rightarrow (-)$ and $J_{Ax} \dashv I \Rightarrow_{Ax} (-)$.

Proof. First, we prove $\Psi; I \Rightarrow (-) = I \Rightarrow_{Ax} (-); \Phi$. Let $f := [\vec{x} : \vec{S} \vdash P; \vec{y} : \vec{T}]$. By the definition of Ψ, we have

$$I \Rightarrow \Psi f = I \Rightarrow [\![\vec{x} : \vec{S} \vdash P; \vec{y} : \vec{T}]\!].$$

On the other hand,

$$\begin{aligned}\Phi(I \Rightarrow_{Ax} f) &= \Phi([\bar{a} : \mathbf{ch}^o[\vec{S}^\perp], \vec{y} : \vec{T}^\perp \vdash {!b(\vec{y})}.(\boldsymbol{\nu}\vec{x}\vec{z})(P \mid \bar{a}\langle\vec{z}\rangle); b : \mathbf{ch}^i[\vec{T}^\perp]^\perp]) \\ &= \Lambda_{\mathbf{ch}^o[\vec{S}^\perp], \vec{T}^*, I}([\![\bar{a} : \mathbf{ch}^o[\vec{S}^\perp], \vec{y} : \vec{T}^\perp \vdash (\boldsymbol{\nu}\vec{x}\vec{z})(P \mid \bar{a}\langle\vec{z}\rangle)]\!]).\end{aligned}$$

Since the adjunction $J(- \otimes A) \dashv A \Rightarrow (-)$ is defined by the closed structure and the adjunction $J \dashv I \Rightarrow (-)$,

$$\begin{aligned}&\Lambda_{\mathbf{ch}^o[\vec{S}^\perp], \vec{T}^\perp, I}([\![\bar{a} : \mathbf{ch}^o[\vec{S}^\perp], \vec{y} : \vec{T}^\perp \vdash (\boldsymbol{\nu}\vec{x}\vec{z})(P \mid \bar{a}\langle\vec{z}\rangle)]\!]) \\ &\quad = \varphi((\mathrm{id}_{\mathbf{ch}^o[\vec{S}^\perp]} \otimes \eta_{\vec{T}^\perp}); ([\![\bar{a} : \mathbf{ch}^o[\vec{S}^\perp], \vec{y} : \vec{T}^\perp \vdash (\boldsymbol{\nu}\vec{x}\vec{z})(P \mid \bar{a}\langle\vec{z}\rangle)]\!] \otimes \mathrm{id}_{\vec{T}})).\end{aligned}$$

Using the naturality of c, the exchange rule and Lemma 4.16, we can show that

$$\begin{aligned}&(\mathrm{id}_{\mathbf{ch}^o[\vec{S}^\perp]} \otimes \eta_{\vec{T}^*}); ([\![\bar{a} : \mathbf{ch}^o[\vec{S}^\perp], \vec{y} : \vec{T}^\perp \vdash (\boldsymbol{\nu}\vec{x}\vec{z})(P \mid \bar{a}\langle\vec{z}\rangle)]\!] \otimes \mathrm{id}_{\vec{T}}) \\ &\quad = (\eta_{\vec{T}} \otimes \mathrm{id}_{\mathbf{ch}^o[\vec{S}^\perp]}); (\mathrm{id}_{\vec{T}} \otimes [\![\vec{y} : \vec{T}^\perp, \bar{a} : \mathbf{ch}^o[\vec{S}^\perp] \vdash (\boldsymbol{\nu}\vec{x}\vec{z})(P \mid \bar{a}\langle\vec{z}\rangle)]\!]).\end{aligned}$$

By the definition of the interpretation, we can rewrite the above expression as follows:

$$\begin{aligned}&(\eta_{\vec{T}} \otimes \mathrm{id}_{\mathbf{ch}^o[\vec{S}^\perp]}); (\mathrm{id}_{\vec{T}} \otimes [\![\vec{y} : \vec{T}^\perp, \bar{a} : \mathbf{ch}^o[\vec{S}^\perp] \vdash (\boldsymbol{\nu}\vec{x}\vec{z})(P \mid \bar{a}\langle\vec{z}\rangle)]\!]) \\ &\quad = (\mathrm{id}_{\mathbf{ch}^o[\vec{S}^\perp]} \otimes \eta_{\vec{S}}); (c_{\mathbf{ch}^o[\vec{S}^\perp], \vec{S}} \otimes \mathrm{id}_{\vec{S}^*}); (\mathrm{id}_{\vec{S}} \otimes \mathbf{eval}_{\vec{S}^*, I}); [\![\vec{x} : \vec{S} \vdash P; \vec{y} : \vec{T}]\!]\end{aligned}$$

So by substituting this expression, we obtain

$$\begin{aligned}&\Phi(I \Rightarrow_{Ax} f) \\ &\quad = \varphi((\mathrm{id}_{\mathbf{ch}^o[\vec{S}^\perp]} \otimes \eta_{\vec{S}}); (c_{\mathbf{ch}^o[\vec{S}^\perp], \vec{S}} \otimes \mathrm{id}_{\vec{S}^*}); (\mathrm{id}_{\vec{S}} \otimes \mathbf{eval}_{\vec{S}^*, I}); [\![\vec{x} : \vec{S} \vdash P; \vec{y} : \vec{T}]\!]) \\ &\quad = \varphi((\mathrm{id} \otimes \eta); (c \otimes \mathrm{id}); (\mathrm{id} \otimes \mathbf{eval})); I \Rightarrow [\![\vec{x} : \vec{S} \vdash P; \vec{y} : \vec{T}]\!].\end{aligned}$$

It remains to show that $\varphi((\mathrm{id} \otimes \eta); (c \otimes \mathrm{id}); (\mathrm{id} \otimes \mathbf{eval}))$ is the identity morphism. Since $\mathbf{eval}_{\vec{S}^*, I} = (e \otimes \mathrm{id}_{\vec{S}^*}); \varepsilon_{\vec{S}^*}$, where e is the counit of the adjunction $J \dashv I \Rightarrow (-)$, we can show that

$$\begin{aligned}&\varphi((\mathrm{id}_{\mathbf{ch}^o[\vec{S}^\perp]} \otimes \eta_{\vec{S}}); (c_{\mathbf{ch}^o[\vec{S}^\perp], \vec{S}} \otimes \mathrm{id}_{\vec{S}^*}); (\mathrm{id}_{\vec{S}} \otimes \mathbf{eval}_{\vec{S}^*, I})) \\ &\quad = \varphi(\mathrm{Tr}^{\vec{S}}_{\mathbf{ch}^o[\vec{S}^\perp], \vec{S}}(c_{\mathbf{ch}^o[\vec{S}^\perp], \vec{S}}; (\mathrm{id}_{\vec{S}} \otimes e)))\end{aligned}$$

by Lemma 4.16 and the definition of the canonical trace. Now we have

$$\begin{aligned}&\varphi(\mathrm{Tr}^{\vec{S}}_{\mathbf{ch}^o[\vec{S}^\perp], \vec{S}}(c_{\mathbf{ch}^o[\vec{S}^\perp], \vec{S}}; (\mathrm{id}_{\vec{S}} \otimes e))) \\ &= \varphi(e) && \text{(by generalized yanking)} \\ &= \mathrm{id}\end{aligned}$$

as desired.

The other equation clearly holds because we have

$$\begin{aligned}\varphi(\Psi([\Delta \vdash P; \vec{x} : \vec{T}])) &= \varphi([\![\Delta \vdash P; \vec{x} : \vec{T}]\!]) \\ &= \Phi([\Delta \vdash_{\mathrm{v}} !a(\vec{x}).P; a : \mathbf{ch}^i[\vec{T}^{\perp}]^{\perp}]) \\ &= \Phi(\varphi_{Ax}([\Delta \vdash P; \vec{x} : \vec{T}]))\end{aligned}$$

by the definition of Φ and Ψ, and $\Lambda_{A,I,B} = \varphi_{A,B}$. □

Uniqueness of the functor Finally, we show the uniqueness of the (Φ, Ψ), and this will prove Theorem 4.15.

Lemma 4.21. Let (Φ', Ψ') be a strict compact closed Freyd functor from the term model to a compact closed Freyd functor $J \colon \mathcal{C} \underset{\perp}{\rightleftarrows} \mathcal{K}$. Then $(\Phi', \Psi') = (\Phi, \Psi)$, where (Φ, Ψ) is the compact closed Freyd functor defined above.

Proof. The proof is by induction on the derivation of $\Delta \vdash P : \diamond$ and $\Delta \vdash_{\mathrm{v}} P; \Sigma$, with case analysis on the last rule used in the derivation. Briefly speaking, the claim follows because we are using strict functors, i.e. Φ, Φ', Ψ' and Ψ' preserves the chosen structures on the nose.

To show $\Psi' = \Psi$, we only need to consider the morphisms of the form $[\Delta \vdash P; \_]$. This is because

$$\begin{aligned}\Psi'([\Delta \vdash P; \Sigma]) &= \Psi'((\eta_\Delta \otimes \mathrm{id}); (\mathrm{id} \otimes [\Sigma^*, \Gamma \vdash P; \_])) \\ &= (\eta_\Delta \otimes \mathrm{id}); (\mathrm{id} \otimes [\![\Sigma^*, \Gamma \vdash P]\!]) \\ &= [\![\Delta \vdash P; \Sigma]\!]\end{aligned}$$

by the strictness of Ψ' provided that $\Psi'([\Sigma^*, \Delta \vdash P; \_]) = \Psi([\Sigma^*, \Delta \vdash P; \_])$.

Now we do the actual case analysis. We only show the case for (V-In) and (S-In) in detail because these are the most complicated cases; the other cases can be proved by similar arguments.

Case (V-In)**:** Recall that, in the term model, a morphism $[\Delta \vdash_{\mathrm{v}} !a(\vec{x}).P; a : \mathbf{ch}[\vec{T}]]$ can be obtained by applying the adjunction isomorphism Λ_{Ax} to $[\Gamma, \vec{x} : \vec{T} \vdash P; \_]$. So it suffices to show that $\Lambda_{Ax}; \Phi' = \Psi'; \Lambda$, where Λ is the adjunction isomorphism in $J \colon \mathcal{C} \underset{\perp}{\rightleftarrows} \mathcal{K}$. Using this equation we can show that

$$\begin{aligned}\Phi'([\Delta \vdash_{\mathrm{v}} !a(\vec{x}).P; a : \mathbf{ch}[\vec{T}]]) &= \Phi'(\Lambda_{Ax}([\Delta, \vec{x} : \vec{T} \vdash P; \_])) \\ &= \Lambda(\Psi'([\Delta, \vec{x} : \vec{T} \vdash P; \_])) \\ &= \Lambda([\![\Delta, \vec{x} : \vec{T} \vdash P; \_]\!]) \\ &= \Phi([\Delta \vdash_{\mathrm{v}} !a(\vec{x}).P; a : \mathbf{ch}[\vec{T}]])\end{aligned}$$

by the induction hypothesis.

The equation $\Lambda_{Ax}; \Phi' = \Psi'; \Lambda$ follows from the facts that (Φ', Ψ') is a map of adjoints and Ψ' is a strict compact closed functor. Since $\Lambda_{Ax}(f) = \varphi_{Ax}((\mathrm{id} \otimes \eta); (f \otimes \mathrm{id}))$,

$$\begin{aligned}\Phi'(\Lambda_{Ax}(f)) &= \Phi'(\varphi_{Ax}((\mathrm{id} \otimes \eta); (f \otimes \mathrm{id}))) \\ &= \varphi(\Psi'((\mathrm{id} \otimes \eta); (f \otimes \mathrm{id}))) \\ &\qquad \text{(since } (\Phi', \Psi') \text{ is a map of adjoints)} \\ &= \varphi((\mathrm{id} \otimes \eta); (\Psi'(f) \otimes \mathrm{id})) \\ &\qquad \text{(since } \Psi' \text{ is a strict compact closed functor)} \\ &= \Lambda(\Psi'(f))\end{aligned}$$

as desired.

Case (V-Unit)**:** Since Φ' and Φ map the chosen terminal map to the chosen terminal map, we have

$$\Phi'([\Delta \vdash_{\mathrm{v}} \mathbf{0}; \_]) = !_\Gamma = \Phi([\Gamma \vdash_{\mathrm{v}} \mathbf{0}; \_]) = !_\Gamma.$$

Case (V-Star)**:** Similar to the previous case because Φ' and Φ map chosen projections to the chosen projections.

Case (V-Par)**:** This case follows from the induction hypothesis and the fact that Ψ' preserves $\mathbf{d}$ and $f \otimes g$ because Ψ' is a strict finite product preserving functor.

Case (S-In)**:** We first show that $[\Delta \vdash !a(\vec{x}).P; \_]$ is equal to

$$J_{Ax}(\langle \pi_a^\Delta, \mathrm{id}_\Gamma \rangle); (\mathrm{id}_{\mathbf{ch}[\vec{T}]^*} \otimes J_{Ax}(\Lambda'([\Gamma, \vec{x} : \vec{T} \vdash P]))); \varepsilon_{\mathbf{ch}[\vec{T}]}$$

in the term model. The process $!a(\vec{x}).P$ is equal to $(\boldsymbol{\nu}\bar{c}c)(\boldsymbol{\nu}\bar{b}b)(a \hookrightarrow \bar{c} \mid !b(\vec{x}).P \mid c \hookrightarrow \bar{b})$ under the assumption $b, c, \bar{c} \notin \mathbf{n}(!a(\vec{x}).P)$ because

$$\begin{aligned}
&(\boldsymbol{\nu}\bar{c}c)(\boldsymbol{\nu}\bar{b}b)(a \hookrightarrow \bar{c} \mid !b(\vec{x}).P \mid c \hookrightarrow \bar{b}) \\
&= (\boldsymbol{\nu}a\bar{c})(c \hookrightarrow \bar{b} \mid (\boldsymbol{\nu}\bar{b}b)(!b(\vec{x}).P \mid c \hookrightarrow \bar{b})) \\
&= (\boldsymbol{\nu}a\bar{c})(a \hookrightarrow \bar{c} \mid !c(\vec{x}).P) \\
&= !a(\vec{x}).P.
\end{aligned}$$

Therefore,

$$\begin{aligned}
&[\Delta \vdash !a(\vec{x}).P; \_] \\
&= [\Delta \vdash a \hookrightarrow \bar{c} \mid !b(\vec{x}).P; \bar{c} : \mathbf{ch}[\vec{T}]^*, b : \mathbf{ch}[\vec{T}]]; \\
&\quad [c : \mathbf{ch}[\vec{T}]^*, \bar{b} : \mathbf{ch}[\vec{T}] \vdash \bar{b} \hookrightarrow a'; \_]
\end{aligned}$$

and since $[c : \mathbf{ch}[\vec{T}]^*, \bar{b} : \mathbf{ch}[\vec{T}] \vdash \bar{b} \hookrightarrow a'; \_]$ is the counit we have

$$[\Delta \vdash !a(\vec{x}).P; \_] = J_{Ax}(\langle \pi_a, \mathrm{id}_\Delta \rangle); (\mathrm{id}_{\mathbf{ch}[\vec{T}]^*} \otimes [\Delta \vdash !b(\vec{x}).P; b : \mathbf{ch}[\vec{T}]]); \varepsilon_{\mathbf{ch}[\vec{T}]}.$$

It remains to show that $[\Delta \vdash !b(\vec{x}).P; b : \mathbf{ch}[\vec{T}]] = J_{Ax}(\Lambda_{Ax}([\Delta, \vec{x} : \vec{T} \vdash P])$. This follows from the fact that $[\Delta \vdash !b(\vec{x}).P; b : \mathbf{ch}[\vec{T}]]$ is a value because $b \notin \mathbf{fn}(\Delta)$ and that $[\Delta \vdash_{\mathrm{v}} !b(\vec{x}).P; b : \mathbf{ch}[\vec{T}]] = \Lambda_{Ax}([\Delta, \vec{x} : \vec{T} \vdash P])$.

To complete the proof for this case, we show that

$$\Psi'(J_{Ax}(\Lambda_{Ax}([\Delta, \vec{x} : \vec{T} \vdash P]))) = J(\Lambda(\llbracket \Delta, \vec{x} : \vec{T} \vdash P \rrbracket)).$$

This equation holds because

$$\begin{aligned}
&\Psi'(J_{Ax}(\Lambda_{Ax}([\Delta, \vec{x} : \vec{T} \vdash P]))) \\
&= J(\Phi'(\Lambda_{Ax}([\Delta, \vec{x} : \vec{T} \vdash P]))) && \text{(by } J_{Ax}; \Psi' = \Phi'; J\text{)} \\
&= J(\Lambda(\Psi'([\Delta, \vec{x} : \vec{T} \vdash P]))) && \text{(by } \Lambda_{Ax}; \Phi' = \Psi'; \Lambda\text{)} \\
&= J(\Lambda(\llbracket \Delta, \vec{x} : \vec{T} \vdash P \rrbracket)). && \text{(by the induction hypothesis)}
\end{aligned}$$

Hence, $\Psi'([\Delta \vdash !a(\vec{x}).P; \_]) = \llbracket \Delta \vdash !a(\vec{x}).P; \_ \rrbracket$ follows from the above equation and the fact that Ψ' strictly preserves projections (lifted by J_{Ax}) and the counit.

Case (S-Nil)**:** Trivial because Φ' strictly preserves $!\colon A \to I$.

Case (S-Out)**:** We can show Ψ' strictly preserves **eval** by an argument similar to the one we used to show $\Lambda_{Ax}; \Phi' = \Psi'; \Lambda$ in the case for (V-In). From this and the fact that Φ' strictly preserves projections concludes this case.

Case (S-Nu)**:** Again, using the forwarders, it is easy to see that $[\Delta \vdash (\boldsymbol{\nu} xy)P; \_] = (\mathrm{id} \otimes \eta); [\Gamma, x : T, y : T^* \vdash P; \_]$ in the term model. The claim follows by the induction hypothesis and the fact that Ψ' maps the chosen unit to the chosen unit.

Case (S-Par)**:** Similar to the case for (V-Par).

□

Discussions: Strengthening the correspondence We end this chapter by discussing the imperfections in Theorem 4.15 and whether they can be improved. Besides the problem on the axiom **(I)** and **(D)** Theorem 4.15 differs from the standard theory/model correspondence in that (1) "the collection of models" is a set not a category and (2) the structure preserving functor is restricted to the strict ones.

The cause for the first problem lies in the syntax side. As explained, the problem is caused by the fact that we only consider a calculus with the empty signature, i.e. we are considering a calculus with no additional type nor constant. In contrast to the case of the λ-calculus it is not obvious how ground types and constants should be added. This is because it is not clear what the dual of a ground type represents. For instance, it is not clear what $\mathbf{Nat}^{\perp}$ means. Similarly, we are not sure how to express constants such as natural numbers as processes. If we do not need to stick to the syntax of π-calculus, it is possible to give a calculus that corresponds to compact closed Freyd category and can deal with non-empty signatures. In fact, we can use the λ_{ch}-calculus, which we will introduce in Chapter 6. The λ_{ch}-calculus is an instance of the computational λ-calculus, and the λ_{ch}-calculus can be seen as a functional language augmented with communication channels. Since we can add additional constants and types to the computational λ-calculus without any problem, the λ_{ch}-calculus can also deal with non-empty signatures.

We believe that the second point is not an essential problem. The reason why we used strict functors was to facilitate the calculation. One could reasonably expect everything to generalize to the non-strict case by inserting suitable isomorphisms into the calculation we have done. However, checking this seems to require quite an effort.

Chapter 5

π_F-calculus vs. Linear Logic

This chapter discusses logical aspects of the π_F-calculus and compact closed Freyd category.

Section 5.1 introduces a new process representation of linear logic proofs from a semantic consideration. A compact closed Freyd category is a model of *linear logic* (more precisely the multiplicative exponential fragment), as an instance of linear/non-linear model [12]. Therefore, the interpretation of proofs in the term model gives a mapping from proofs to π_F-processes that is correct by construction. The purpose of explicitly showing this interpretation is to clarify the similarities and differences between the π_F-calculus and other π-calculi that are based on linear logic.

Section 5.2 discuss the converse direction, i.e. a proof system to which every π_F-process can be interpreted. Such a logic needs to be expressive enough to capture deadlocks and race conditions. The discovery of a proof system that corresponds to the traditional π-calculus has been an open problem since Abramsky's "proofs as processes" program [3]. We give a reasonable candidate for such a proof system by investigating the relation between compact closed Freyd categories and other categorical models of linear logic.

5.1 Proofs as Processes

5.1.1 Definition of the translation

We explain how MELL formulas and MELL proofs are represented as (lists of) sorts and process of the π_F-calculus respectively. Here we simply focus on the definition of the translation and do not explain how the translation is derived (which will be explained in Section 5.1.3).

We use the standard presentation of classical linear logic by one-sided sequents [43]. The inference rules of MELL are introduced along with the translation, but for more details we refer the readers to the literature [43, 46].

The *formulas* of MELL are given by the following grammar:

$$A ::= 1 \mid \bot \mid A \otimes A \mid A \parr A \mid !A \mid ?A.$$

Note that we do not have propositional variables. This is because we do not have base types in the π_F-calculus. This is a common setting that is used in logical studies of π-calculi (e.g. [11, 54, 19]).

As usual, we introduce negation as a meta-operation on formulas rather than a logical connective. The dual of a formula A, written $A^{\perp}$, is inductively defined

linear logic (formula)	compact closed Freyd category (object)	π_F-calculus (sort environment)
1 $\perp$	I	empty environment
$A \otimes B$ $A \parr B$	$A \otimes B$	$\vec{x} : A, \vec{y} : B$
$!A$	$I \Rightarrow A$	$x : \mathbf{ch}^o[A^\perp]$
$?A$	$(A \Rightarrow I)^*$	$x : \mathbf{ch}^i[A]$

Table 5.1: The categorical and π_F-calculus interpretations of MELL formulas.

as follows:

$$1^\perp \overset{\text{def}}{=} \perp \qquad \perp^\perp \overset{\text{def}}{=} 1$$
$$(A \otimes B)^\perp \overset{\text{def}}{=} A^\perp \parr B^\perp \qquad (A \parr B)^\perp \overset{\text{def}}{=} A^\perp \otimes B^\perp$$
$$(!A)^\perp \overset{\text{def}}{=} ?A^\perp \qquad (?A)^\perp \overset{\text{def}}{=} !A^\perp$$

By definition, the duality is strictly involutive, that is we have $(A^\perp)^\perp = A$.

Formulas are interpreted as list of sorts (or sort environments) rather than sorts. This is because objects of $\mathit{Cl}(\emptyset)$ is a list of sorts, not a single sort. The interpretation of formulas is shown in Table 5.1. For readability, the same metavariables A, B are used to denote formulas, objects and (list of) sorts, but the meaning should be clear. The names appearing in the sorts environment will be used when we translate proofs into process. Note that $A \otimes B$ and $A \parr B$ are translated into the same sort environment. This is because in a compact closed category $A \parr B$ is interpreted as $(A^* \otimes B^*)^*$ which coincides with $A \otimes B$ in the term model. The table also shows the categorical interpretation of formulas in a compact closed Freyd category, but we will explain this later in this section.

Proofs are translated into well-sorted processes. A *sequent* is an expression $\vdash \Gamma$, where a *context* $\Gamma = A_1, \ldots, A_n$ is a finite list of formulas; in what follows we tacitly use the exchange rule and treat a context as if it is a multiset. We write $\vdash \Gamma :: \Pi$ to mean that Π is a *proof* with the sequent $\vdash \Gamma$ as the root. Proofs of the form $\vdash A_1, \ldots, A_n :: \Pi$ are translated to processes in context $\vec{x}_1 : A_1^\perp, \ldots, \vec{x}_n : A_n^\perp \vdash P : \diamond$, and we write $\vdash \Gamma :: \Pi \rightsquigarrow \Gamma^\perp \vdash P : \diamond$ for this translation. Again, by abuse of notation, we are using A to represent formulas and the list of sorts that correspond to the formula A and similarly Γ is used to denote the sort environment corresponding to the context Γ. Note that we take the dual $A^\perp$ when a formula appearing in a context is translated to sorts in a sort judgement.[1] This is because formulas appear in the right-hand side of a sequent whereas sorts appear in the left-hand side of a sort judgement. In what follows, we will often omit the symbol $\diamond$.

Throughout the following paragraphs we introduce how proofs are translated into processes. Rather than giving the whole translation at once, we present and explain how each rule is translated in a step-by-step manner.

Multiplicatives The rule for the tensor $\otimes$ is translated as parallel composition.

$$\otimes\ \frac{\vdash \Gamma, A \qquad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \otimes B} \rightsquigarrow \frac{\Gamma^\perp, (\vec{x}) : A^\perp \vdash P \qquad \Delta^\perp, (\vec{y}) : B^\perp \vdash Q}{\Gamma^\perp, \Delta^\perp, (\vec{x}, \vec{y}) : A^\perp, B^\perp \vdash P \mid Q}$$

[1]The sort $A^\perp$ can either be interpreted as (1) taking the dual of the formula A and then taking the corresponding sort or (2) taking the sort corresponding to the formula A and then considering the dual of that sort. They result in the same sort.

Perhaps surprisingly, the process representation of the introduction rule for $\parr$, which is the dual of $\otimes$, does nothing:

$$\parr\,\frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \parr B} \rightsquigarrow \frac{\Gamma^{\perp}, (\vec{x}) : A^{\perp}, (\vec{y}) : B^{\perp} \vdash P}{\Gamma^{\perp}, (\vec{x}, \vec{y}) : A^{\perp}, B^{\perp} \vdash P}$$

This is because $A \otimes B \cong (A^* \otimes B^*)^* \cong A \parr B$ in compact closed categories and $\otimes$ is interpreted as juxtaposition of sorts in a sort environment.

The rules for multiplicative units are also "silent", i.e. applying the rules for multiplicative units do not have any effect to the translation. This is because 1 and $\perp$ are interpreted as the empty environment in our translation.

Exponential In linear logic weakening and contraction rules do not apply to every formula, but only to a certain class of modal formulas. The modalities are ! and ?. Intuitively, $!A$ represents an arbitrary number of copies of the formula A and ? is the dual of !. The weakening and contraction rules are limited to modal formulas $?A$ on the right-hand side of a sequent.[2]

The introduction rule for ! is translated as follows:

$$!\,\frac{\vdash ?\Gamma, A}{\vdash ?\Gamma, !A} \rightsquigarrow \frac{(?\Gamma)^{\perp}, (\vec{x}) : A^{\perp} \vdash P}{(?\Gamma)^{\perp}, a : \mathbf{ch}^{i}[A^{\perp}] \vdash !a(\vec{x}).P}$$

Here $?\Gamma$ represents a context of the form $?A_1, \ldots, ?A_n$. The ! introduction rule is translated into input prefixing. This translation seems natural because input prefixing blocks the computation, i.e. reduction under prefixing is not allowed, and !-boxes of proof-nets are also used to block computation. An interesting consequence of this translation is that there are no free input names inside $!a(\vec{x}).P$ because $!A = \mathbf{ch}^{o}[A^{\perp}]$ in the term model.

The dereliction rule,[3] which is the rule that introduces ? is translated as follows:

$$\mathrm{D}\,\frac{\vdash \Gamma, A}{\vdash \Gamma, ?A} \rightsquigarrow \frac{\Gamma^{\perp}, (\vec{x}) : A^{\perp} \vdash P}{\Gamma^{\perp}, \bar{a} : \mathbf{ch}^{o}[A] \vdash (\boldsymbol{\nu}\vec{x}\vec{y})(P \mid \bar{a}\langle\vec{y}\rangle)}$$

By using the bounded-output notation, the process in the conclusion can be written as $\bar{a}(\vec{x}).P$. Here $\bar{a}(\vec{x}).P$ means "send fresh names $\vec{x}$ using the channel $\bar{a}$ and then execute P".

The weakening and contraction rules for ? is translated as follows:

$$\mathrm{W}\,\frac{\vdash \Gamma}{\vdash \Gamma, ?A} \rightsquigarrow \frac{\Gamma^{\perp} \vdash P}{\Gamma^{\perp}, \bar{a} : \mathbf{ch}^{o}[A] \vdash P}$$

$$\mathrm{C}\,\frac{\vdash \Gamma, ?A, ?A}{\vdash \Gamma, ?A} \rightsquigarrow \frac{\Gamma^{\perp}, \bar{x} : \mathbf{ch}^{o}[A], \bar{y} : \mathbf{ch}^{o}[A] \vdash P}{\Gamma^{\perp}, \bar{z} : \mathbf{ch}^{o}[A] \vdash P\{\bar{z}/\bar{x}, \bar{z}/\bar{y}\}}$$

Identity and Cut We interpret the axiom as forwarders.

$$\mathrm{AX}\,\frac{}{\vdash A, A^{\perp}} \rightsquigarrow \frac{}{\vec{x} : A^*, \vec{y} : A \vdash \vec{x} \leftrightharpoons \vec{y}}$$

[2]If we use two-sided presentation, then we can also apply weakening and contractions to formulas of the form $!A$ on the left-hand side of a sequent.

[3]It might be easier to understand the intuition behind the dereliction rule when it is written in the two-sided form $\frac{\Gamma, A \vdash \Delta}{\Gamma, !A \vdash \Delta}$, which says that "the linearity information on A is lost".

Interpreting axiom as forwarders is an idea pervasively used in translations from linear logical proofs to π-calculi (e.g. [11, 133, 19]).

The cut rule is represented as "parallel composition + hiding", which is also standard.

$$\text{CUT}\,\frac{\vdash \Gamma, A \qquad \vdash \Delta, A^{\perp}}{\vdash \Gamma, \Delta} \quad \rightsquigarrow \quad \frac{\Gamma^{*}, (\vec{x}) : A^{*} \vdash P \qquad \Delta^{*}, (\vec{y}) : A \vdash P}{\Gamma^{*}, \Delta^{*} \vdash (\boldsymbol{\nu}\vec{x}\vec{y})(P \mid Q)}$$

Remark 5.1. The π_F-calculus can interpret not only MELL proofs, but also *MELL proof-structures* as processes. By proof-structures we mean graphs constructed in the language of proof-nets—a geometrical method of representing proofs of linear logic—that do not necessarily represent a proof in MELL. This is due to the compact closed structure of the π_F-calculus [85]. The interpretation is not given in this thesis since we do not need the interpretation of proof-structures to compare our work with related studies. ■

5.1.2 Cut elimination

We show that the standard proof conversions used in cut elimination procedures correspond to process equivalences. The translation is sound with respect to these proof conversions: if a proof Π is converted to Π' and the process presentation of Π and Π' are P and P', respectively, then we have $P = P'$. This is because our translation is correct by construction (see Corollary 5.2 below). We believe that giving explicit correspondences between proof conversions helps readers understand our translation and also clarifies the difference between our translation and existing translations from linear logic to π-calculi. (Comparison with related work is given in the subsequent section.) Instead of listing all the cut reduction rules and commuting conversion rules, we only present some interesting cases.

The cut reduction for ! and the dereliction rule is one of the most interesting rules to examine since it correspond to communication. The following cut reduction

$$\text{CUT}\,\frac{!\,\dfrac{\vdash ?\Gamma, A :: \Pi}{\vdash ?\Gamma, !A} \qquad \text{D}\,\dfrac{\vdash \Delta, A^{\perp} :: \Pi'}{\vdash \Delta, ?A^{\perp}}}{\vdash ?\Gamma, \Delta} \quad \longrightarrow \quad \text{CUT}\,\frac{\vdash ?\Gamma, A :: \Pi \qquad \vdash \Delta, A^{\perp} :: \Pi'}{\vdash ?\Gamma, \Delta}$$

corresponds to the following equation, where Π and Π' corresponds to P and Q respectively.

$$(\boldsymbol{\nu}\bar{a}a)(!a(\vec{x}).P \mid (\boldsymbol{\nu}\vec{y}\vec{z})\bar{a}\langle\vec{z}\rangle \mid Q) = (\boldsymbol{\nu}\vec{y}\vec{z})(P\{\vec{z}/\vec{x}\} \mid Q)$$

The above equation follows from (E-Beta), (E-GC) and structural equivalence.

On the other hand, the cut reduction between $\otimes$ and $⅋$ is quite boring. The cut reduction for $\otimes$ and $⅋$ is defined as follows:

$$\text{CUT}\,\frac{\otimes\,\dfrac{\vdash \Gamma, A \qquad \vdash \Gamma', B}{\vdash \Gamma, \Gamma', A \otimes B} \qquad ⅋\,\dfrac{\vdash \Delta, A^{\perp}, B^{\perp}}{\vdash \Delta, A^{\perp} ⅋ B^{\perp}}}{\vdash \Gamma, \Gamma', \Delta}$$

$$\longrightarrow \quad \text{CUT}\,\frac{\vdash \Gamma, A \qquad \text{CUT}\,\dfrac{\vdash \Gamma', B \qquad \vdash \Delta, A^{\perp}, B^{\perp}}{\vdash \Gamma', \Delta, A^{\perp}}}{\vdash \Gamma, \Delta}$$

This corresponds to the equation below

$$(\boldsymbol{\nu}\vec{x}_1\vec{x}_2\ \vec{y}_1\vec{y}_2)((P \mid Q) \mid R) = (\boldsymbol{\nu}\vec{x}_1\vec{y}_1)(P \mid (\boldsymbol{\nu}\vec{x}_2\vec{y}_2)(Q \mid R)).$$

The above two processes are simply structurally equivalent, and thus no communications occur to model this cut reduction.

Let us also elaborate on how commuting conversions are modeled. Commuting conversions are conversions that "pushes the cut inside", which corresponds to pushing the $\boldsymbol{\nu}$ operator inside. Most of the commuting conversion rules corresponds to the scope extrusion rule $(\boldsymbol{\nu}\bar{a}a)(P \mid Q) \equiv (\boldsymbol{\nu}\bar{a}a)P \mid Q$, where $\bar{a}, a \notin \mathbf{fn}(Q)$. The commuting conversion for ! is a little tricky from the π_F-calculus perspective because it requires the commutation between input prefixing and $\boldsymbol{\nu}$. The commuting conversion for ! and its process representation is given as

$$\textsc{cut}\ \dfrac{!\ \dfrac{\vdash ?\Gamma, A, ?B :: \Pi}{\vdash ?\Gamma, !A, ?B} \qquad \vdash ?\Delta, !B^{\perp} :: \Pi'}{\vdash ?\Gamma, ?\Delta, !A}$$

$$\longrightarrow\quad !\ \dfrac{\textsc{Cut}\ \dfrac{\vdash ?\Gamma, A, ?B :: \Pi \qquad \vdash ?\Delta, !B^{\perp} :: \Pi'}{\vdash ?\Gamma, ?\Delta, A}}{\vdash ?\Gamma, ?\Delta, !A}$$

and

$$(\boldsymbol{\nu}\bar{b}b)(!a(\vec{x}).P \mid Q) = !a(\vec{x}).(\boldsymbol{\nu}\bar{b}b)(P \mid Q).$$

where P corresponds to Π and Q corresponds to Π'.The above equation is valid because the rules of MELL and the definition of the translation ensures that Q is equivalent to a process of the form $!b(\vec{w}).Q'$.Since $!b(\vec{w}).Q'$ works as a substitution, we can equate the above two processes, i.e. we can equate the processes by (E-Beta) and (E-GC).

5.1.3 Semantics-directed translation

As mentioned in the beginning of this chapter, our translation is *derived* using our semantic framework. The translation is defined by interpreting the MELL proofs using the term model $Cl(\emptyset)$ and this ensures the soundness of our syntactic translation.

Here we briefly explain the relationship between compact closed Freyd categories and models of linear logic and how linear logic proofs are interpreted in these model. For more details on categorical semantics of linear logic, the reader may consult a survey written by Melliès [84].

Definition 5.1 (Linear/non-linear model [12, 9])**.** A *linear/non-linear model* (LNL model) is a symmetric monoidal adjunction $F\colon \mathcal{C} \underset{\perp}{\rightleftarrows} \mathcal{M}\colon G$, where $\mathcal{C}$ is a cartesian category and $\mathcal{M}$ is a symmetric monoidal closed category. A LNL model is a categorical model of intuitionistic multiplicative linear logic with exponentials. If the symmetric monoidal closed category $\mathcal{M}$ is a $*$-autonomous category [10], then we say that the adjunction is a *LNL model of MELL.* ■

Remark 5.2. Our definition of the LNL model differs from the original definition given by Benton [12] in that we only require $\mathcal{C}$ to be cartesian instead of requiring $\mathcal{C}$ to be cartesian closed. ■

Compact closed Freyd categories are LNL models of MELL. The functor $J\colon \mathcal{C} \to \mathcal{K}$ of a compact closed Freyd category is a (strong) symmetric monoidal functor and the right adjoint $I \Rightarrow -$ is also a monoidal functor because the right adjoint of a strong monoidal functor lifts to a monoidal functor [84, Proposition 14]. The category $\mathcal{C}$ is cartesian and the category $\mathcal{K}$ is $*$-autonomous because every compact closed category is a $*$-autonomous category.

We briefly explain how formulas and proofs are interpreted in LNL models. Let $F\colon \mathcal{C} \underset{\perp}{\rightleftarrows} \mathcal{M}\colon G$ be a LNL model of MELL. As the notation suggests the multiplicative $\otimes$ and the unit 1 are interpreted by monoidal products and the monoidal unit in $\mathcal{M}$. Each $*$-autonomous category is equipped by the dualization functor $(-)^*\colon \mathcal{M}^{\mathrm{op}} \to \mathcal{M}$ and this determines the interpretation of the remaining multiplicative connectives. That is, $[\![A \parr B]\!] \overset{\mathrm{def}}{=} ([\![A]\!]^* \otimes [\![B]\!]^*)^*$ and $[\![\perp]\!] \overset{\mathrm{def}}{=} I^*$. [4] The ! modality is modeled by the comonad GF induced by the monoidal adjunction and $?A$ is defined as $(GFA^*)^*$. So, in a compact closed Freyd category $J\colon \mathcal{C} \underset{\perp}{\rightleftarrows} \mathcal{K}\colon I \Rightarrow (-)$, the modality ! is modeled by the comonad $J(I \Rightarrow -)$ as we summarized in Table 5.1. A proof $\vdash A_1, \dots, A_n :: \Pi$ is interpreted as a morphism $[\![\Pi]\!]\colon I \to [\![A_1 \parr \cdots \parr A_n]\!]$ in $\mathcal{M}$. See [84] for the definition of the translation.

The important point is that the interpretation of MELL proofs in every LNL model is known to be *sound*, i.e. the interpretation is invariant modulo cut-elimination. It is easy to check that the translations we defined in Section 5.1.1 correctly specifies a representative of the interpretation in the term model $\mathit{Cl}(\emptyset)$ and thus it follows that the syntactic translation is also sound *without the need for an extra proof.*

Theorem 5.1. Let $\vdash \Gamma :: \Pi$ be an MELL proof and assume $\vdash \Gamma :: \Pi \rightsquigarrow \Gamma^\perp \vdash P : \diamond$. Then the interpretation of Π in the term model $\mathrm{Cl}(\emptyset)$ is $[\_ \vdash P; \Gamma]$, which is a process typed under $\Gamma^\perp$.

Proof. By simple calculation. $\square$

Corollary 5.2. Assume $\vdash \Gamma :: \Pi \rightsquigarrow \Gamma^\perp \vdash P : \diamond$ and $\vdash \Gamma :: \Pi' \rightsquigarrow \Gamma^\perp \vdash P : \diamond$ and suppose that $\Pi \longrightarrow \Pi'$ in the cut-elimination process. Then we have $\emptyset \rhd \Gamma^\perp \vdash P = P'$. $\square$

5.1.4 Comparison with related studies

On the translation of formulas and proofs We discuss the similarities and the differences between our translation and other studies that have investigated π-calculus interpretation of linear logic proofs. As mentioned, axioms and cuts are translated as forwarders and "parallel composition + hiding" in most studies. Hence, we only discuss how the rules for exponential modalities and multiplicative connectives are translated.

The way we translate the rules for exponential modalities are very similar to that of the translation from linear logic to session-typed π-calculi [133, 19]. In fact, our translation for the exponential rules and Wadler's translation for the exponential rules coincide. More concretely, the introduction rule for ! is represented as replicated input prefixing and the dereliction rule is represented as bound output in both translations. (Weakening and contraction rules are also represented in the same way.) Our translation is not directly comparable to πCLL given by Caires et al. [19] since they use a dyadic system [6], i.e. a system with

[4] Note that, in general, $(A^* \otimes B^*)^*$ is not isomorphic to $A \otimes B$ and I^* is not isomorphic to I in a $*$-autonomous category $\mathcal{C}$. As repeatedly mentioned, they are isomorphic if $\mathcal{C}$ is compact closed.

dual contexts, instead of the standard one-sided formalism of linear logic with single context. Nevertheless, there are similarities in the translation for the rules for exponential modalities. The introduction rule for !, i.e. the (T!) rule in [19], corresponds to replicated inputs and the (Tcopy) rule, a rule that can be seen as a decomposition of the dereliction rule, corresponds to bound output.

The Abramsky translation [11] for the exponentials are more complicated than ours. This is because the translation tries to encode the cut reduction between ! introduction and weakening and contraction for ? using communications rather than process equivalence. Moreover, due to this complexity, it is hard to consider their system as a type assignment system; there is no natural interpretation for the formula $!A$.

However, when the translations for multiplicative connectives are compared, our translation is conceptually closer to the Abramsky translation than the translation to session-typed π-calculi.

Abramsky translation interprets $A \otimes B$ as $\mathbf{ch}^o[A, B]$ and $A \invamp B$ as $\mathbf{ch}^i[A, B]$.[5] The $\invamp$ introduction rule is interpreted as follows

$$\frac{P \vdash \Gamma, x : A, y : B}{z(x, y).P \vdash \Gamma, z : A \invamp B} \tag{5.1}$$

The difference compared to our translation is that Abramsky translation adds an input prefixing so that the cut reduction for $\otimes$ and $\invamp$ corresponds to communication. However, like our translation, the idea that (1) names represent “communication ports” and (2) process describes how the ports are connected can be seen in this translation. In the words of Bellin and Scott, “communications in π-calculus is about the pluggings in proof structures”.

In linear logic based session-typed π-calculi, $A \otimes B$ is interpreted as a type for sessions that first outputs a value of type A and then behaves according to B. Dually, $A \invamp B$ is interpreted as a type for sessions that input A and behave as B. Thus the type does no longer represent the “pluggings in proof structures”. For example the $\invamp$ introduction rule given by Wadler is as follows:

$$\frac{P \vdash \Gamma, x : A, B : y}{x(y).P \vdash \Gamma, x : A \invamp B}$$

Note that the input only receives one name in contrast to (5.1).

To our knowledge, the work most similar to ours is the correspondence between a local π-calculus and polarized proof-nets discovered by Honda and Laurent [54]. The polarized logic used in their work is presented in focalized form [44] and the set of formulas is given by the following grammar:

$$P ::= \bigotimes_{1 \le i \le n} !N_i \qquad\qquad N ::= \bigparr_{1 \le i \le n} !P_i$$

As in the other translations ! and ? (on the right-hand side of a sequent) correspond to input action and output actions respectively. The interesting point is that connectives $\otimes$ and $\invamp$ are used to express polyadic communication action, i.e. receiving or sending n names. Note that each $\invamp$ connective appears under the !-modality. This allows one to consider $!(P_1 \invamp \cdots \invamp P_n)$ as a type for input channels receiving n-arguments. This is exactly the way how sorts and formulas correspond in the π_F-calculus. However, a crucial difference compared to our

[5]It is more precise to interpret $A \otimes B$ as a linear output channel [69] because the name assigned to the type $\mathbf{ch}^o[A, B]$ is linearly used.

work is that processes in their calculus only receives input names. This condition is called the locality constraint [88, 136] in the context of π-calculus and reflects the fact that the formulas are polarized.

On cut-elimination A possible criticism on our translation is that it does not always map cut reductions to communications. As we have seen, the cut reduction for $\otimes$ and $\parr$ is modeled by structural equivalence in our translation. This is in contrast to the other studies where the correspondences between cut reductions and communications were emphasized.

However, we think that the requirement that every cut reduction should correspond to reduction/communication is too strong. When one tries to fulfill this strong requirement one often ends up with a process calculus whose reduction relation is distant from the conventional reduction. For example, the π-calculus used by Bellin and Scott admits prefix commutations (e.g. $a(x).b(y).P$ and $b(y).a(x).P$ are equivalent), which is not allowed in conventional π-calculus. Wadler's CP also suffers from this problem. The session typed calculi introduced by Caires et al. [19] do not suffer from this problem because they relate some of the cut reduction rules to process equivalence instead of reduction; the operational validity of these equivalences has also been checked [103]. We think it is not that peculiar to model the cut reduction for $\otimes$ and $\parr$ using structural equivalence once we forgo the idea that cut reduction should correspond to communication.

Also there is no strong semantic reason to interpret $\otimes$ as outputs and $\parr$ as inputs. For example, Bellin and Scott mentions that the choice of treating $\otimes$ as senders and $\parr$ as receivers is arbitrary [11]; having said that, we note that they gave a detailed observation based on information flow to support their choice. Another example is the work on multiparty session types [20] where $\otimes$ is interpreted as inputs and $\parr$ as outputs. Thus, the fact that $\otimes$ does not correspond to neither inputs nor outputs in our translation is not unnatural from the semantic perspective.

5.2 Processes as Proofs

Here we discuss the converse direction, i.e. a variant of linear logic to which every π_F-process can be interpreted. It should be a degenerate system in the sense that $A \otimes B$ and $A \parr B$ must coincide because $A \otimes B$ and $A \parr B$ are isomorphic in any compact closed category. It seems that we also need weakening and contraction rules for formulas of the form $?A$ appearing in the left-hand side of the sequent, because input channels can be replicated and discarded in the π_F-calculus.

Instead of making guesses at the rules we need based only on syntactic observations, we use our categorical semantics as a guide. We start from a model of linear logic and look for additional data and axioms that are sufficient to turn that model into a compact closed Freyd category using some categorical construction.

We use linear categories as our starting point. It is well-known that a linear category that is also $*$-autonomous category is a model of MELL. Although LNL models and linear categories are both sound and complete models of linear logic, linear categories have better match with sequent style formulation of linear logic. The internal language of LNL models is a kind of adjoint logic and does not exactly correspond to linear logic [81].

Definition 5.2 ([13, 59]). A *linear category* is a symmetric monoidal closed category $(\mathcal{L}, \otimes, I)$ with a symmetric monoidal comonad $((!, m^0, m^2), \delta, \varepsilon)$ with two monoidal natural transformations d and e such that for all $A \in \mathbf{Obj}(\mathcal{L})$

1. $d_A \colon !A \to !A \otimes !A$ and $e_A \colon !A \to I$ form a commutative comonoid A, d_A, e_A
2. d_A and e_A are coalgebra morphisms
3. the morphism δ_A is a comonoid morphism.

The symmetric monoidal comonad ! is called the *linear exponential comonad.* ■

Since the proof system we are seeking for needs to have the compact closed structure, it is natural to start from a linear category that is compact closed. The logic corresponding to linear categories that are also compact closed should be a proper extension of linear logic. For example, the following rules are invalid in linear logic but admissible in compact closed categories:

$$\frac{\vdash \Gamma \qquad \vdash \Delta}{\vdash \Gamma, \Delta} \qquad \frac{\vdash \Gamma, A, B \qquad \vdash \Delta, A^{\perp}, B^{\perp}}{\vdash \Gamma, \Delta} \qquad \frac{\vdash \Gamma, A, A^{\perp}}{\vdash \Gamma}.$$

These rules, especially the second rule called *multicut*, were often studied in concurrency theory; see the work by Abramsky et al. [5] for their relevance to concurrency.

Now the following is the question we want to answer:

> Can we obtain a compact closed Freyd category from a linear category that is compact closed?

Since compact closed Freyd categories are monoidal adjunctions (with additional properties) between cartesian category and monoidal category, we need to construct these structures.

Fortunately there is a well-known recipe to construct a cartesian category from a linear category. Linear categories are instances of LNL models because the Eilenberg-Moore category of the linear exponential comonad is cartesian.

Proposition 5.3 ([84, Proposition 28])**.** Let $(\mathcal{L}, \otimes, I)$ be a linear category whose linear exponential comonad is !. The monoidal structure inherited from $\mathcal{L}$ is cartesian in its Eilenberg Moore category $\mathcal{L}^!$. Therefore, the $U \colon \mathcal{L}^! \rightleftarrows \mathcal{L} \colon F$ ($\perp$) is a LNL model, where U and F are the forgetful and the free functor respectively. □

Be warned that the above adjunction is not a compact closed Freyd category as U is not identity-on-object. Our aim is to construct a Freyd category from the above adjunction.

From the monoidal adjunction $U \dashv F$ in the above proposition, we can construct a monad FU over $\mathcal{L}^!$. When we have a monad there is a somewhat canonical way to obtain an identity-on-object functor: taking the Kleisli adjunction of the monad. However, the Kleisli category $(\mathcal{L}^!)_{FU}$ is not a compact closed category in general even if $\mathcal{L}$ is compact closed. Thus taking the Kleisli adjunction does not always give a compact closed Freyd category. The following proposition characterizes the condition for $(\mathcal{L}^!)_{FU}$ to be a compact closed category.

Proposition 5.4. Let $(\mathcal{L}, \otimes, I)$ be a compact closed category, where the dualization functor is given by $(-)^*$, together with a linear exponential comonad $(((!, m^0, m^2), \delta, \varepsilon)$. Then the adjunction $U \colon \mathcal{L}^! \rightleftarrows \mathcal{L} \colon F$ ($\perp$) induces a monad FU. The Kleisli category of this monad $(\mathcal{L}^!)_{FU}$ is a compact closed Freyd category if and only if $\mathcal{L}$ satisfies the following condition:

(R) for all coalgebra $(A, A \xrightarrow{h} !A)$ there exists a coalgebra $(A^*, A^* \xrightarrow{h^\bullet} !(A^*))$

Proof. First observe that $(\mathcal{L}^!)_{FU}$ is equivalent to a category $\mathcal{L}^!_U$ whose objects are the same as $\mathcal{L}^!$ and morphisms are defined as

$$\begin{aligned}\mathcal{L}^!_U((A, A \xrightarrow{h} !A), (B, B \xrightarrow{h'} !B)) &\overset{\text{def}}{=} \mathcal{L}(U(A, A \xrightarrow{h} !A), U(B, B \xrightarrow{h'} !B)) \\ &= \mathcal{L}(A, B)\end{aligned}$$

This category is clearly a monoidal category since $\mathcal{L}^!$ is equipped by with the symmetric monoidal functor structure induced by $\mathcal{L}$.

Now we prove the only if direction. Since $(\mathcal{L}^!)_{FU}$ is compact closed by assumption, the category $\mathcal{L}^!_U$ is also compact closed. Thus, there is a dual for every object $(A, A \xrightarrow{h} !A)$, which we write $(A^\bullet, A \xrightarrow{h^\bullet} !A^\bullet)$. It follows that $A^\bullet$ is the dual of A in $\mathcal{L}$ and thus $A^\bullet \cong A^*$. It is easy to check that $(A^*, A^* \cong A^\bullet \xrightarrow{h^\bullet} !A^\bullet \cong !A^*)$ is a coalgebra. Hence, $\mathcal{L}$ satisfies the condition (R).

The proof for the if direction is easy. It suffices to show that $\mathcal{L}^!_U$ is a compact closed category. We can define the dual objects by $(A, A \xrightarrow{h} !A)^* \overset{\text{def}}{=} (A^*, A^* \xrightarrow{h^\bullet} !A^*)$. The unit and counit is given by the unit $\eta_A \colon I \to A^* \otimes A$ and counit $\varepsilon_A \colon A \otimes A^* \to I$ inherited from $\mathcal{L}$. □

Since formulas of the form $!A$ are coalgebras in MELL, the above proposition leads us to consider the following axiom:

$$R\;\dfrac{}{\vdash\, !?A, (?A)^\perp} \qquad \left(\text{or an equivalent axiom in a two-sided form} \quad \dfrac{}{?A \vdash\, !?A}\right)$$

with the following conversion rules.

$$\text{Cut}\;\dfrac{\text{R}\;\dfrac{}{\vdash\, !?A, (?A)^\perp} \qquad \text{D}\;\dfrac{\text{Ax}\;\dfrac{}{\vdash ?A, (?A)^\perp}}{\vdash ?A, ?(?A)^\perp}}{\vdash ?A, (?A)^\perp} \quad\longrightarrow\quad \text{Ax}\;\dfrac{}{\vdash ?A, (?A)^\perp}$$

$$\text{Cut}\;\dfrac{\text{R}\;\dfrac{}{\vdash\, !?A, (?A)^\perp} \qquad !\;\dfrac{\text{D}\;\dfrac{\text{R}\;\dfrac{}{\vdash\, !?A, (?A)^\perp}}{\vdash\, !?A, ?(?A)^\perp}}{\vdash\, !!?A, ?(?A)^\perp}}{\vdash\, !!?A, (?A)^\perp} \quad\longrightarrow\quad \text{Cut}\;\dfrac{\text{R}\;\dfrac{}{\vdash\, !?A, (?A)^\perp} \qquad !\;\dfrac{\text{Ax}\;\dfrac{}{\vdash\, !?A, (!?A)^\perp}}{\vdash\, !!?A, ?(?A)^\perp}}{\vdash\, !!?A, (?A)^\perp}$$

The axiom in the two-sided form corresponds to requiring the existence of a morphism $(!A)^* \to !(!A)^*$ for every A; note that $?A = (!A^\perp)^\perp$. The two conversion rules correspond to the axioms that coalgebras need to satisfy, which are sometimes called the unit triangle and the multiplication square.

With this axiom, the weakening rule and the contraction rule for ! (on the right-hand side of a sequent) becomes derivable. For example, the weakening rule is given as follows:

$$\text{Cut}\;\dfrac{\text{R}\;\dfrac{}{\vdash\, !?A^\perp, !A} \qquad \text{w}\;\dfrac{\vdash \Gamma}{\vdash \Gamma, ?!A}}{\vdash \Gamma, !A}$$

Categorically speaking, this means that $?A$ is a comonoid object. From the π-calculus point of view, this means that input channels can be duplicated and discarded. Therefore, races are introduced by this axiom.

We conjecture that MELL together with the multicut rule, the axiom $?A \vdash\, !?A$ and the above two proof conversion rules induces a category initial among linear categories that are compact closed and satisfy the condition (R). If this is the case then we can derive a syntactic translation from π_F-calculus to this proof system by applying Proposition 5.4 to the linear category constructed from this proof system. Checking this conjecture is left for future work.

Note that the proofs in this proof system are also representable by π_F-processes. This is because the term model $Cl(\emptyset)$ give rise to a linear category that satisfies (R) by taking $! \overset{\text{def}}{=} J_{Ax}(I \Rightarrow_{Ax} -)$ as the linear exponential comonad. So, if the conjecture holds, we may say that π_F-calculus corresponds to "MELL + multicut + axiom (R)". The correspondence is, however, quite loose in the sense that it is not a direct correspondence. We need some categorical constructions, like taking the Kleisli category, for the mutual interpretation.

To our knowledge, the axiom $?A \vdash !?A$, has not been considered in the literature. However, a similar rule was considered when Atkey et al. [7] tried to introduce race into the session typed calculus CP [133]. They identified the formula $?A$ with $!A$ and showed that this yields access points, a rendezvous mechanism for initiating session typed communication that may cause race conditions. Our axiom is weaker than their requirement in the sense that $?A \vdash !A$ is not derivable in general, i.e. we do not have back-and-forth translations between $?A$ and $!A$. Another difference is that their syntactic identification of ! and ? is not well understood from the semantic viewpoint. In fact, it is not clear what it means to identify a linear exponential comonad ! with a monad ? defined by $?A \overset{\text{def}}{=} (!A^*)^*$. To the contrary, our axiom $?A \vdash !?A$ originates from a semantic consideration. However, the proof theoretic and process theoretic understanding against this axiom is still missing. This is left for future work. In particular, we are interested in whether we can define a variant of π-calculus that directly corresponds to "MELL + multicut + the axiom $?A \vdash !?A$". Such a calculus is interesting because we may be able to introduce linear types [69] to the calculus. This is because types that are not duplicable nor discardable can exist in such a calculus.

Chapter 6

Comparing First-Order and Higher-Order Calculi

In order to demonstrate the relevance of our semantic framework, this chapter tries to give a semantic understanding against the various existing translations from higher-order calculi, such as the λ-calculus, to the π-calculus. Our primary tool to analyze these translations is a higher-order calculus called the *λ_{ch}-calculus* that also corresponds to compact closed Freyd categories. The λ_{ch}-calculus is an instance of the computational λ-calculus and is obtained by a straightforward extension of the coincidence between the computational λ-calculus and closed Freyd categories. Since the λ_{ch}-calculus is sound and complete with respect to compact closed Freyd categories, there are translations between the λ_{ch}-calculus and the π_F-calculus. We show that these translations can be seen as (an extended version of) the translations between the higher-order π-calculus and the first-order π-calculus [115, 118], and use them to study other translations from higher-order calculi to the π-calculus.

6.1 The λ_{ch}-calculus

The *λ_{ch}-calculus*[1] is a computational λ-calculus with additional constructors dealing with channels. This section introduces and explains the calculus.

The situation is nicely expressed by the following intuitive equation:

$$\frac{\lambda_{ch}}{\lambda_c} \qquad \approx \qquad \frac{(\text{compact closed Freyd category} + \mathbf{I} + \mathbf{D})}{(\text{closed Freyd category})}.$$

The base calculus λ_c-calculus is the *computational λ-calculus*, which corresponds to closed Freyd category [97, 111]. It is a call-by-value higher-order programming language, given in Fig. 6.1(a).[2] Our calculus λ_{ch}-calculus is obtained by adding type and term constructors originating from the compact closed structure, which λ_c-calculus does not have.

Remark 6.1. It is possible to define an instance of the computational λ-calculus that corresponds to compact closed Freyd categories that do not necessary satisfy conditions **(I)** and **(D)**. However, we wanted λ_{ch}-calculus to correspond to the π_F-calculus, so we made it to correspond to "compact closed Freyd category + **(I)** + **(D)**". ■

[1]There is another calculus also called the λ_{ch}-calculus in the literature [38, 37], which is very similar to ours. This is just a coincidence and our calculus was developed independently of that of [38, 37]. (See Section 6.4 for details.)

[2]The syntax of λ_c-calculus is adapted to the setting of this thesis.

$\sigma ::= \tau \to \tau' \qquad \xi ::= \sigma \qquad \tau ::= (\xi_1, \ldots, \xi_n)$

$V ::= x \mid \lambda\langle\vec{x}\rangle.M$

$M ::= \langle\vec{V}\rangle \mid V\ \langle\vec{V}\rangle \mid \mathbf{let}\ \langle\vec{x}\rangle = M\ \mathbf{in}\ M'$

(a) λ_c-calculus

$\xi ::= \cdots \mid \sigma^*$

$V ::= \cdots \mid \mathbf{channel}_\sigma \mid \mathbf{send}_\sigma$

(b) λ_{ch} (difference from λ_c)

Figure 6.1: Syntax of types and terms of the λ_c-calculus and λ_{ch}-calculus.

6.1.1 Syntax

As for types, λ_{ch} has a new constructor coming from the dual object A^*. Normalizing occurrences of the dual A^* using the axioms **(I)** $A^{**} = A$ and **(D)** $(A \otimes B)^* = A^* \otimes B^*$, we obtain the following grammar of types:

$$\sigma ::= \tau \to \tau' \qquad \xi ::= \sigma \mid \sigma^* \qquad \tau ::= (\xi_1, \ldots, \xi_n)$$

where $n \geq 0$ and $(\xi_1, \ldots, \xi_n)$ is an alternative notation for $\xi_1 \otimes \cdots \otimes \xi_n$. Compared with λ_c, the only new type is the dual type σ^* of a function type σ.

As for terms, λ_{ch} has constructors corresponding to the unit and counit

$$\eta_A : I \longrightarrow A \otimes A^* \qquad \varepsilon_A : A^* \otimes A \longrightarrow I \qquad \text{(for each object } A\text{)}$$

of the compact closed structure. We simply add these morphisms as constants:

$$\frac{}{\Delta \vdash \mathbf{channel}_\sigma : () \to (\sigma, \sigma^*)} \quad \text{and} \quad \frac{}{\Delta \vdash \mathbf{send}_\sigma : (\sigma^*, \sigma) \to ()}.$$

We shall often omit the subscript σ.

Type environments are of the form $\Delta = x_1 : \xi_1, \ldots, x_n : \xi_n$ and the type judgement is of the form $\Delta \vdash M : \tau$. We omit the typing rules since they are standard.

In summary, we obtain the syntax of λ_{ch}-calculus shown in Figure 6.1. Interestingly, λ_{ch}-calculus can be seen as a very core of Concurrent ML [112], a practical higher-order concurrent language, even though λ_{ch}-calculus is developed from purely semantic considerations.

6.1.2 Semantics

Let us first discuss the intuitive meanings of the new constructors. The type σ^* is for *output channels*; **channel** $\langle\rangle$ creates and returns a pair of an input channel and an output channel that are connected; and **send** $\langle\alpha, V\rangle$ sends the value V via the output channel α. The following points are worth noting.

- λ_{ch} has no type constructor for *input channels*. The type system does not distinguish between input channels for type σ and values of type σ.
- λ_{ch} has no *receive* constructor. Receiving operation is implicit and on demand, delayed as much as possible.
- The send operator broadcasts a value via a channel. Several receivers may receive the same value from the same channel.

The first two points reflect the asynchrony of the π_F-calculus, and the last point reflects the absence of non-replicated input (cf. Section 6.2).

$$\begin{aligned}
\mathbf{let}\,\langle\vec{x}\rangle = M\,\mathbf{in}\,\langle\vec{x}\rangle &= M\\
\mathbf{let}\,\langle\vec{x}\rangle = \langle\vec{V}\rangle\,\mathbf{in}\,M &= M\{\vec{V}/\vec{x}\}\\
\mathbf{let}\,\langle\vec{x}\rangle = (\mathbf{let}\,\langle\vec{y}\rangle = M\,\mathbf{in}\,N)\,\mathbf{in}\,L &= \mathbf{let}\,\langle\vec{y}\rangle = M\,\mathbf{in}\,\mathbf{let}\,\langle\vec{x}\rangle = N\,\mathbf{in}\,L\\
(\lambda\langle\vec{x}\rangle.M)\;\langle\vec{V}\rangle &= \mathbf{let}\,\vec{x} = \langle\vec{V}\rangle\,\mathbf{in}\,M\\
(\lambda\langle\vec{x}\rangle.y\langle\vec{x}\rangle) &= y
\end{aligned}$$

(a) Rules for λ_c-calculus

$$\mathbf{let}\,\langle\vec{x}\rangle = M\,\mathbf{in}\,\mathbf{let}\,\langle\vec{y}\rangle = N\,\mathbf{in}\,L = \mathbf{let}\,\langle\vec{y}\rangle = N\,\mathbf{in}\,\mathbf{let}\,\langle\vec{x}\rangle = M\,\mathbf{in}\,L$$

(b) Rules for concurrent λ_c-calculus

$$\begin{aligned}
(\boldsymbol{\nu} x\bar{x})(\mathbf{send}\,\langle\bar{x},V\rangle \parallel M) &= (\boldsymbol{\nu} x\bar{x})(\mathbf{send}\,\langle\bar{x},V\rangle \parallel M\{V/x\}) \quad (\bar{x}\notin \mathbf{fv}(V)\cup\mathbf{fv}(M))\\
(\boldsymbol{\nu} x\bar{x})(\mathbf{send}\,\langle\bar{x},V\rangle) &= \langle\rangle \qquad (x,\bar{x}\notin \mathbf{fv}(V))\\
(\boldsymbol{\nu} y\bar{y})(\mathbf{send}\,\langle\bar{z},y\rangle \parallel N) &= N\{\bar{z}/\bar{y}\} \qquad (y\notin\mathbf{fv}(N) \text{ and } \bar{z}\neq\bar{y})
\end{aligned}$$

(c) Rules for λ_{ch}-calculus

Figure 6.2: Equational rules for the λ_{ch}-calculus.

Based on this intuition, we develop the axiomatic and categorical semantics of the λ_{ch}-calculus. We shall use the following abbreviations:

$$(\boldsymbol{\nu} xy)M \overset{\text{def}}{=} \mathbf{let}\,\langle x,y\rangle = \mathbf{channel}\,\langle\rangle\,\mathbf{in}\,M \qquad M\parallel N \overset{\text{def}}{=} \mathbf{let}\,\langle\rangle = M\,\mathbf{in}\,N.$$

These abbreviations help us clarify the similarity among λ_{ch}-calculus, π_F-calculus and the higher-order π-calculus.

Axiomatic semantics The inference rules of the equational logic for the λ_{ch}-calculus are listed in Figure 6.1.2. The rules are equations-in-context; each rule has implicit assumptions that the both sides of the equation are well-typed terms. The first five are the standard rules for computational λ-calculus [78, 47]. The sixth one is the rule of concurrent evaluation that reflects the bifunctoriality of monoidal products. The last three rules are the β- and η-rules for channels and a GC rule that reflect the compact closed structure.

Categorical semantics One can interpret λ_{ch}-terms in a compact closed Freyd category with **(I)** and **(D)**. Here we define the interpretation $[\![-]\!]_J$, where J is a compact closed Freyd category. The interpretation of types and terms are listed in Figure 6.1.2 where a type environment $x_1\colon \xi_1,\dots,x_n\colon\xi_n$ is interpreted as $[\![\xi_1]\!]\otimes\cdots\otimes[\![\xi_n]\!]$. The interpretation of the λ_c-calculus part is standard [111, 78]. Interpretations of constants $\mathbf{channel}_\sigma$ and $\mathbf{send}_\sigma$) are the part that is specific to the λ_{ch}-calculus. They are interpreted as the "closure" whose body is η_σ and ε_σ, respectively, as expected.

The categorical semantics is sound and complete with respect to the equational theory of the λ_{ch}-calculus. The completeness is shown by constructing a term model using types and terms of the λ_{ch}-calculus. We omit the proofs for soundness and completeness since they are analogous to that for the π_F-calculus; actually, soundness and completeness for the λ_{ch}-calculus is easier to prove because we can reuse some results on the λ_c-calculus. There is a subtle issue in

$$\llbracket \tau \to \tau' \rrbracket \overset{\text{def}}{=} \llbracket \tau \rrbracket \Rightarrow \llbracket \tau' \rrbracket \quad \llbracket \sigma^* \rrbracket \overset{\text{def}}{=} \llbracket \sigma \rrbracket^* \quad \llbracket (\xi_1, \dots, \xi_n) \rrbracket \overset{\text{def}}{=} \llbracket \xi_1 \rrbracket \otimes \cdots \otimes \llbracket \xi_n \rrbracket$$

$$\begin{aligned}
\llbracket \Delta \vdash x : \xi \rrbracket &\overset{\text{def}}{=} J(\pi^{\Delta}_{x}) \\
\llbracket \Delta \vdash \langle V_1, \dots, V_n \rangle : (\xi_1, \dots, \xi_n) \rrbracket &\overset{\text{def}}{=} J(\langle \mathrm{id}, \dots, \mathrm{id} \rangle); (\llbracket \Delta \vdash V_1 : \xi_1 \rrbracket \otimes \cdots \otimes \llbracket \Delta \vdash V_n : \xi_n \rrbracket) \\
\llbracket \Delta \vdash \lambda \langle \vec{x} \rangle . M : (\vec{\xi}) \to \tau \rrbracket &\overset{\text{def}}{=} J(\Lambda_{\Delta, \vec{\xi}, \tau}(\llbracket \Gamma, \vec{x} \colon \vec{\xi} \vdash M : \tau \rrbracket)) \\
\llbracket \Delta \vdash V \ \langle \vec{W} \rangle : \tau' \rrbracket &\overset{\text{def}}{=} J(\mathbf{d}_{\Delta}); (\llbracket \Delta \vdash V : \tau \to \tau' \rrbracket \otimes \llbracket \Delta \vdash \langle \vec{W} \rangle : \tau \rrbracket); \mathbf{eval}_{\tau, \tau'} \\
\llbracket \Delta \vdash \mathbf{let}\, \vec{x} = M \,\mathbf{in}\, N \rrbracket &\overset{\text{def}}{=} J(\mathbf{d}_{\Delta}); (\mathrm{id}_{\Delta} \otimes \llbracket \Delta \vdash M : (\vec{\xi}) \rrbracket); \llbracket \Delta, \vec{x} : \vec{\xi} \vdash N : \tau \rrbracket \\
\llbracket \Delta \vdash \mathbf{channel}_{\sigma} : () \to (\sigma, \sigma^*) \rrbracket &\overset{\text{def}}{=} J(!_{\Delta}; \Lambda_{I, I, \sigma \otimes \sigma^*}(\eta_{\sigma})) \\
\llbracket \Delta \vdash \mathbf{send}_{\sigma} : (\sigma^*, \sigma) \to () \rrbracket &\overset{\text{def}}{=} J(!_{\Delta}; \Lambda_{I, \sigma \otimes \sigma^*, I}(\varepsilon_{\sigma})).
\end{aligned}$$

Figure 6.3: Interpretations of the types and λ_{ch}-terms.

$$\{\!| \mathbf{ch}^{o}[\vec{T}] |\!\} \overset{\text{def}}{=} \{\!| \vec{T} |\!\} \to () \quad \{\!| \mathbf{ch}^{i}[\vec{T}] |\!\} \overset{\text{def}}{=} (\{\!| \vec{T} |\!\} \to ())^* \quad \{\!| (T_1, \dots, T_n) |\!\} \overset{\text{def}}{=} (\{\!| T_1 |\!\}, \dots, \{\!| T_n |\!\})$$

$$\{\!| \mathbf{0} |\!\} \overset{\text{def}}{=} \langle\rangle \qquad \{\!| P \mid Q |\!\} \overset{\text{def}}{=} \{\!| P |\!\} \parallel \{\!| Q |\!\} \qquad \{\!| (\boldsymbol{\nu} xy) P |\!\} \overset{\text{def}}{=} (\boldsymbol{\nu} xy) \{\!| P |\!\}$$

$$\{\!| \bar{a}\langle \vec{x} \rangle |\!\} \overset{\text{def}}{=} \bar{a} \ \langle \vec{x} \rangle \qquad \{\!| !a(\vec{x}).P |\!\} \overset{\text{def}}{=} \mathbf{send}\, \langle a, \lambda(\vec{x}).\{\!| P |\!\} \rangle$$

Figure 6.4: Translation from π_F-calculus to λ_{ch}-calculus.

the definition of the term model that is worth mentioning: we have different definitions of the right adjoint $I \Rightarrow (-)$, which are of course equivalent but do not coincide on the nose. Our choice here is $I \Rightarrow \langle \vec{\xi} \rangle \overset{\text{def}}{=} (\vec{\xi}^{\perp}) \to ()$.

6.2 Translations between λ_{ch} and π_F

The higher-order calculus λ_{ch}-calculus is equivalent to π_F-calculus. This is because both calculi correspond to the same class of categories, namely, the class of compact closed Freyd categories with **(I)** and **(D)**, i.e.,

$$(\lambda_{ch}\text{-calculus}) \ \approx \ (\text{compact closed Freyd category} + \mathbf{I} + \mathbf{D}) \ \approx \ (\pi_F\text{-calculus})$$

This section studies translations derived from this semantic correspondence.

The translations are defined by the interpretations in the term models. For example, the translation $(\!| - |\!)$ from λ_{ch}-calculus to π_F-calculus is induced by the interpretation of λ_{ch}-terms in the term model $\mathit{Cl}(\emptyset)$. The interpretation $\llbracket M \rrbracket_{\mathit{Cl}(\emptyset)}$ of a λ_{ch}-term M is an equivalence class of π_F-processes, since a morphism in $\mathit{Cl}(\emptyset)$ is an equivalence class of π_F-processes. The translation $(\!| M |\!)$ is defined by choosing a representative of the equivalence class. The other direction $\{\!| - |\!\}$ is obtained by the interpretation of π_F-processes in the term model of λ_{ch}-calculus.

Figures 6.4 and 6.5 are concrete definitions of the translations defined by a natural choice of representatives. The representatives are chosen so that there is no redundant forwarders or redexes.

Let us discuss the translations in more details. The translation from π_F-calculus to λ_{ch}-calculus (Fig. 6.4) should be easy to understand. It directly

$$(\!|\tau_1 \to \tau_2|\!) \overset{\text{def}}{=} \mathbf{ch}^o[(\!|\tau_1|\!), (\!|\tau_2|\!)^\perp] \quad (\!|\sigma^*|\!) \overset{\text{def}}{=} (\!|\sigma|\!)^\perp \quad (\!|(\tau_1, \dots, \tau_n)|\!) \overset{\text{def}}{=} ((\!|\tau_1|\!), \dots, (\!|\tau_n|\!))$$

$$(\!|x|\!)_p \overset{\text{def}}{=} (p \leftrightharpoons x) \qquad (\!|\lambda\vec{x}.M|\!)_p \overset{\text{def}}{=} !p(\vec{x}, \vec{q}).(\!|M|\!)_{\vec{q}} \qquad (\!|\langle\vec{V}\rangle|\!)_{\vec{p}} \overset{\text{def}}{=} (\!|V_1|\!)_{p_1} \mid \cdots \mid (\!|V_n|\!)_{p_n}$$

$$(\!|V\ \langle\vec{W}\rangle|\!)_{\vec{p}} \overset{\text{def}}{=} (\boldsymbol{\nu} a\bar{a})(\boldsymbol{\nu}\vec{r}\vec{s})((\!|V|\!)_a \mid (\!|\langle\vec{W}\rangle|\!)_{\vec{s}} \mid \bar{a}\langle\vec{r}, \vec{p}\rangle)$$

$$(\!|\mathbf{let}\,\langle\vec{x}\rangle = M\,\mathbf{in}\,N|\!)_{\vec{p}} \overset{\text{def}}{=} (\boldsymbol{\nu}\vec{x}\vec{q})((\!|M|\!)_{\vec{q}} \mid (\!|N|\!)_{\vec{p}})$$

$$(\!|\mathbf{channel}|\!)_p \overset{\text{def}}{=} !p(x, y).x \hookrightarrow y \qquad (\!|\mathbf{send}|\!)_p \overset{\text{def}}{=} !p(x, y).x \hookrightarrow y$$

Figure 6.5: Translation from λ_{ch}-calculus to π_F-calculus.

expresses the higher-order view of the first-order π-calculus that has been repeatedly discussed in this thesis. For example, an output action is mapped to an application and an input-prefixing $!a(\vec{x}).P$ to a send operation of the value $\lambda\langle\vec{x}\rangle.P$ via the channel a.

An interesting (and perhaps confusing) phenomenon is that an input channel in the π_F-calculus is mapped to an output channel in λ_{ch}-calculus. This can be explained as follows. In the name-passing viewpoint, the reduction

$$(\boldsymbol{\nu} xy)(!y(\vec{z}).P \mid x\langle\vec{u}\rangle) \quad\longrightarrow\quad (\boldsymbol{\nu} xy)(!y(\vec{z}).P \mid P\{\vec{u}/\vec{z}\})$$

sends $\vec{u}$ to the process $!y(\vec{z}).P$, and thus x is output and y is input. In the process-passing viewpoint, the abstraction $(\vec{z}).P$ is sent to the location of x, and thus y is the output and x is the input.

Next, we explain the translation from the λ_{ch}-calculus to the π_F-calculus shown in Figure 6.5). Let us first examine the translation of types. The most non-trivial part is the translation of a function type $\tau_1 \to \tau_2$. A key to understand the translation is the isomorphism $\tau_1 \to \tau_2 \cong \tau_1 \otimes \tau_2^\perp \to ()$. The latter form of function type corresponds to an output channel type in the π_F-calculus. Hence a function is understood as a process additionally taking channels to which the return values are passed.

The translation $(\!|M|\!)_{\vec{p}}$ of a λ_{ch}-term $\Delta \vdash M : (\xi_1, \dots, \xi_n)$ takes extra parameters $\vec{p} = p_1, \dots, p_n$ to which the values should be placed. This is a consequence of the definition in the term model of the π_F-calculus that a morphism $\vec{T} \longrightarrow \vec{S}$ is a process $\vec{x}\colon \vec{T}$, $\vec{y}\colon \vec{S}^\perp \vdash P : \diamond$ where there is no "return value". Here $\vec{p}$ corresponds to $\vec{y}$, Γ to $\vec{x}\colon \vec{T}$ and $\vec{\xi}$ to $\vec{S}$.

Now it should be easier to understand the interpretations of the λ_{ch}-terms. For example, the abstraction $(\!|\lambda\langle\vec{x}\rangle.M|\!)_p$ is mapped to an abstraction $(\vec{x}, \vec{q}).(\!|M|\!)_{\vec{q}}$ placed at p, which takes additional channels $\vec{q}$ to which the results of the evaluation of M should be sent.

Example 6.1. As an example, let us see how a β-redex $(\lambda\bar{x}.M)\ N$is translated and how reductions are expressed. Since we cannot directly write $(\lambda\bar{x}.M)\ N$ in λ_{ch}-calculus we will use an equivalent expression that uses the let-expression.

$$\begin{aligned}
&(\!|\mathbf{let}\,\bar{y} = N\,\mathbf{in}\,(\lambda\bar{x}.M)\ \bar{y}|\!)_p \\
&= (\boldsymbol{\nu}\bar{y}y)((\!|N|\!)_y \mid (\boldsymbol{\nu}\bar{a}a)(\boldsymbol{\nu}\bar{b}b)(!a(\bar{x}, q).(\!|M|\!)_q \mid b \hookrightarrow \bar{y} \mid \bar{a}\langle\bar{b}, p\rangle) \\
&= (\boldsymbol{\nu}\bar{y}y)((\!|N|\!)_y \mid (\boldsymbol{\nu}\bar{a}a)(!a(\bar{x}, q).(\!|M|\!)_q \mid \bar{a}\langle\bar{y}, p\rangle) \qquad \text{((E-FOUT) and (E-GC))} \\
&\longrightarrow (\boldsymbol{\nu}\bar{y}y)((\!|N|\!)_y \mid (\boldsymbol{\nu}\bar{a}a)(!a(\bar{x}, q).(\!|M|\!)_q \mid (\!|M|\!)_p\{\bar{y}/\bar{x}\}) \\
&= (\boldsymbol{\nu}\bar{y}y)((\!|N|\!)_y \mid (\!|M|\!)_p\{\bar{y}/\bar{x}\}) \qquad \text{(E-GC)}
\end{aligned}$$

Note that the above process allows to reduce the abstraction body M and the argument N in parallel sharing the private channel $\bar{y}$ and y. Intuitively, this happens because $(\lambda x.M)N = \mathbf{let}\, x = N\, \mathbf{in}\ M$ and terms N and M can be reduced in parallel in λ_{ch}-calculus. A similar observation has been given by Toninho et al. [126] when they gave an encoding of the simply-typed λ-calculus in session-typed π-calculus by employing Girard's call-by-value translation from intuitionistic logic to linear logic. They observed that the call-by-value Girard translation only captures a reduction strategy in which sub-expressions are evaluated in parallel. We note that we are observing the same problem in different type systems because the way we interpret the arrow type is the call-by-value Girard translation. That is we have $A \Rightarrow B \cong J(I \Rightarrow A^* \otimes B) \cong\ !(A \multimap B)$.

This means that the translation from λ_{ch}-calculus to π_F-calculus is too loose as an encoding of the standard call-by-value (or call-by-name) λ-calculus. In Section 6.3, we show that this problem can be avoided by restricting the evaluation strategy using CPS translations, which is the idea that is often used in the literature (e.g. [90, 118, 17]). ■

It might be surprising that the interpretations of **channel** and **send** coincide. This is because of the one-sided formulation of the π_F-calculus. In the two-sided formulation, the unit η and counit ε of the compact closed structure, corresponding to **channel** and **send**, can be written as logical inference rules

$$\frac{\Delta, A, A^{\perp} \vdash \Delta'}{\Delta \vdash \Delta'} \qquad \text{and} \qquad \frac{\Delta \vdash A^{\perp}, A, \Delta'}{\Delta \vdash \Delta'},$$

which are different. In the one-sided formulation, however, they become

$$\frac{\Delta, A, A^{\perp}, (\Delta')^{\perp} \vdash}{\Delta, (\Delta')^{\perp} \vdash}.$$

Hence η and ε (or **channel** and **send**) cannot be distinguished in the π_F-calculus.

The translation $(\!|-|\!)$ must be the inverse of $[\![-]\!]$ because both the term models are the initial compact closed Freyd category with **(I)** and **(D)**. That means, $\emptyset \rhd \Delta \vdash P = (\!|[\![P]\!]|\!)$ and $\emptyset \rhd \Delta \vdash M = [\![(\!|M|\!)]\!]$ are provable for every P and M. This result is independent of the choice of representatives.

6.2.1 λ_{ch}-calculus vs. HOπ

We show that a variant of a higher-order π-calculus can be seen as a subcalculus of the λ_{ch}-calculus. This fact together with the mutual translation between the λ_{ch}-calculus and the π_F-calculus gives a semantic reconstruction of the translations by Sangiorgi [118] (see also [119]) between *asynchronous higher-order π-calculus* (*AHOπ* for short) and *asynchronous local π-calculus* ($L\pi$ for short).

A variant of AHOπ can be seen as a fragment of the λ_{ch}-calculus. The syntax of AHOπ and their representation by λ_{ch}-terms are given in Figure 6.6. The first rows of the BNF notations are the grammar for the AHOπ and the corresponding λ_{ch}-terms are written below. We can thus consider the constructs of AHOπ as macros defined over the λ_{ch}-calculus. The syntax of AHOπ presented here slightly differs from that of the original one because $\boldsymbol{\nu}$ binds a pair of names and non-replicated inputs are forbidden.

This fragment is nicely described as a limitation on types:

$$\sigma ::= (\vec{\sigma}) \to () \qquad \xi ::= \sigma \mid \sigma^* \qquad \tau ::= ().$$

$$\begin{array}{rclclclclclcl}
v, w & ::= & x & \mid & (\vec{x}).P \\
 & & x & & \lambda\langle\vec{x}\rangle.P \\[1em]
P, Q & ::= & \mathbf{0} & \mid & (P \mid Q) & \mid & (\boldsymbol{\nu} xy)P & \mid & !x\,v & \mid & v\langle\vec{w}\rangle \\
 & & \langle\rangle & & P \parallel Q & & (\boldsymbol{\nu} xy)P & & \mathbf{send}\,\langle x, v\rangle & & v\ \langle\vec{w}\rangle.
\end{array}$$

Figure 6.6: AHOπ processes as λ_{ch}-terms.

$$\llparenthesis\mathbf{0}\rrparenthesis \overset{\text{def}}{=} \mathbf{0} \qquad \llparenthesis P \mid Q\rrparenthesis \overset{\text{def}}{=} \llparenthesis P\rrparenthesis \mid \llparenthesis Q\rrparenthesis \qquad \llparenthesis(\boldsymbol{\nu} xy)P\rrparenthesis \overset{\text{def}}{=} (\boldsymbol{\nu} xy)\llparenthesis P\rrparenthesis \qquad \llparenthesis\,!x\,v\rrparenthesis \overset{\text{def}}{=} \llparenthesis v\rrparenthesis_x$$

$$\llparenthesis v\langle w_1, \dots, w_n\rangle\rrparenthesis \overset{\text{def}}{=} (\boldsymbol{\nu}\bar{a}a)(\boldsymbol{\nu}\bar{b}_1 b_1)\dots(\boldsymbol{\nu}\bar{b}_n b_n)(\llparenthesis v\rrparenthesis_a \mid \llparenthesis w_1\rrparenthesis_{b_1} \mid \cdots \mid \llparenthesis w_n\rrparenthesis_{b_n} \mid \bar{a}\langle\bar{b}_1, \dots, \bar{b}_n\rangle)$$

$$\llparenthesis x\rrparenthesis_a \overset{\text{def}}{=} (a \hookrightarrow x) \qquad \llparenthesis(\vec{x}).P\rrparenthesis_a \overset{\text{def}}{=} !a(\vec{x}).\llparenthesis P\rrparenthesis$$

Figure 6.7: Translation from AHOπ to π_F-calculus.

Recall that σ is a type for abstractions, ξ is a type for variables, and τ is a type for terms. This limitation means that (1) an abstraction cannot take a channel as an argument, and (2) a term M must be of the unit type, i.e. a process.

Once regarding AHOπ as a fragment of λ_{ch}-calculus, the translation from AHOπ to π_F-calculus is obtained by restricting $\llparenthesis - \rrparenthesis$ to AHOπ-processes. The resulting translation is in Fig. 6.7. As mentioned, the translation is the same as that of Sangiorgi [118] except for minor differences due to the slight change of the syntax.

Sangiorgi also gave a translation in the opposite direction, from Lπ to AHOπ in the same paper. The calculus Lπ, a variant of the local π-calculus [88, 136]), is a fragment of the π-calculus in which only output channels can be passed. The **i**/**o**-separation of the π_F-calculus allows us to characterize the local version of the π_F-calculus by a limitation on types. In the local variant, the output channel type is restricted to $T ::= \mathbf{ch}^o[\vec{T}]$, expressing that only output channels can be passed via an output channel. Then the definition of type environment should be changed accordingly: $\Delta ::= \cdot \mid \Delta, x : T \mid \Delta, x : T^{\perp}$ (since the syntactic class represented by T is not closed under the dual $(-)^{\perp}$ in the local setting).

Interestingly the limitation on types in AHOπ coincides with that in Lπ, when one identify $\mathbf{ch}^o[\vec{T}]$ with $(\vec{T}) \to ()$ (as we have done in many places). In other words, the syntactic restrictions of AHOπ and Lπ are the same semantic conditions described in different syntax. As a consequence, the image of Lπ by $\llbracket - \rrbracket$ is indeed in AHOπ.

6.3 Combining CPS transformation

6.3.1 From $\lambda V/\lambda N$ to π_F

In another paper [117] (see also [119]), Sangiorgi observed that call-by-value and call-by-name encoding of λ-calculus into π-calculus can be factorized into *CPS transformation* and the compilation from HOπ to the first order π-calculus. The situation is nicely summarized in Figure 6.8. The translations $\mathcal{V}$ and $\mathcal{N}$ are the call-by-value and call-by-name encoding of the λ-calculus, respectively. The maps $\mathcal{C}_V$ and $\mathcal{C}_N$ are call-by-value and call-by-name CPS translations that translates

$$\begin{tikzcd}
\lambda V \arrow[dr, "\mathcal{C}_V"] \arrow[dddrr, bend right, "\mathcal{V}"'] & & & & \lambda N \arrow[dl, "\mathcal{C}_N"'] \arrow[dddll, bend left, "\mathcal{N}"] \\
& \mathrm{CPS_v} \arrow[dr, "\mathcal{H}"] & & \mathrm{CPS_N} \arrow[dl, "\mathcal{H}"'] & \\
& & \mathrm{HO}\pi \arrow[d, "(\!|-|\!)"] & & \\
& & \pi & &
\end{tikzcd}$$

Figure 6.8: Decomposition of the π-calculus encoding of λV and λN. The figure is taken from [117].

λ-terms to CPS-calculus; roughly speaking, CPS-calculus is a subcalculus of the λ-calculus that is defined as the image of the CPS translation. The translation $\mathcal{H}$ is the injection from the CPS-calculus to the higher-order π-calculus. It is the identity except for the fact that it does some uncurrying such as $\lambda x.\lambda k.M$ to $\lambda(x,k).M$.

We give semantic analysis to these translations. We show that encoding of the call-by-value and call-by-name λ-calculus can be understood as a call-by-value and call-by-name monad translation [131, 98] for a certain continuation monad.

Our starting point is the observation that a continuation monad can be obtained from a tensorial negation [87]. A *tensorial negation* on a symmetric monoidal category $(\mathcal{C}, \otimes I)$ is a functor $\neg\colon \mathcal{C} \to \mathcal{C}^{\mathrm{op}}$ together with a natural family of bijections

$$\varphi_{A,B,C}\colon \mathcal{C}(A\otimes B, \neg C) \cong \mathcal{C}(A, \neg(B\otimes C))$$

that satisfy a coherent condition with respect to the associativity of the monoidal product (see [87] for details). From the definition it easily follows that $\neg \dashv \neg^{\mathrm{op}}$, i.e. $\neg$ is a self-adjunction. Thus $\neg^{\mathrm{op}}\neg$ is a monad, called the *continuation monad of the negation*, and this can be considered as a "double-negation monad".[3]

Now let us apply this observation to compact closed Freyd category. Let $J\colon \mathcal{C}\underset{\perp}{\rightleftarrows}\mathcal{K}\colon I \Rightarrow (-)$ be a compact closed Freyd category. Since $(-)^*\colon \mathcal{K}\to\mathcal{K}^{op}$ is a self adjoint, we have the following three adjunctions:

$$\begin{tikzcd}
\mathcal{C} \arrow[r, bend left, "J"] \arrow[r, phantom, "\perp"] & \mathcal{K} \arrow[l, bend left, "I\Rightarrow-"] \arrow[r, bend left, "(-)^*"] \arrow[r, phantom, "\perp"] & \mathcal{K}^{\mathrm{op}} \arrow[l, bend left, "((-)^*)^{\mathrm{op}}"] \arrow[r, bend left, "(I\Rightarrow-)^{\mathrm{op}}"] \arrow[r, phantom, "\perp"] & \mathcal{C}^{\mathrm{op}} \arrow[l, bend left, "J^{\mathrm{op}}"]
\end{tikzcd}$$

We define $\neg \overset{\mathrm{def}}{=} J;(-)^*;(I\Rightarrow -)^{\mathrm{op}}$. Then it is easy to check that $\neg$ is also a tensorial negation. By abuse of notation, we also write $\neg$ for the opposite functor of $\neg\colon \mathcal{C}\to\mathcal{C}^{\mathrm{op}}$. Then $\neg\neg\colon \mathcal{C}\to\mathcal{C}$ is a strong monad on $\mathcal{C}$. Moreover, we have a Kleisli exponential for $(\mathcal{C},\neg\neg)$, which is obtained from the following isomorphisms:

$$\begin{aligned}
\mathcal{C}(A\otimes B, \neg\neg C) &= \mathcal{C}(A\otimes B, I\Rightarrow J(I\Rightarrow J(C)^*)^*)\\
&\cong \mathcal{K}(J(A\otimes B), J(I\Rightarrow J(C)^*)^*)\\
&\cong \mathcal{K}(J(A), J(B)^*\otimes J(I\Rightarrow J(C)^*)^*)\\
&\cong \mathcal{C}(A, I\Rightarrow J(B)^*\otimes J(I\Rightarrow J(C)^*)^*).
\end{aligned}$$

[3]It should be noted that the relationship between self-adjunction and continuation was already explored in Thielecke's PhD thesis [125].

$$()^\circ \overset{\text{def}}{=} () \qquad (\tau \to \sigma)^\circ \overset{\text{def}}{=} (\tau^\circ, \sigma^\circ \to ()) \to ()$$

$$\begin{aligned} x^\circ &\overset{\text{def}}{=} \lambda k.k\ x \\ (\lambda x.M)^\circ &\overset{\text{def}}{=} \lambda k.k\ (\lambda(x,v).M^\circ\ v) \\ (M\ N)^\circ &\overset{\text{def}}{=} \lambda k.M^\circ\ (\lambda v.N^\circ\ (\lambda w.v\ \langle w,k\rangle)) \end{aligned}$$

$$(x_1 : \tau_1, \dots x_n : \tau_n \vdash M : \sigma)^\circ \overset{\text{def}}{=} x_1 : \tau_1^\circ, \dots, x_n : \tau_n^\circ \vdash M^\circ : (\sigma^\circ \to ()) \to ()$$

Figure 6.9: Translation from the call-by-value λ-calculus to the λ_{ch}-calculus.

If we ignore J (recall that J is identity-on-object) then the Kleisli exponential with respect to $\neg\neg$ is given by $B \Rightarrow_{Kl} C \overset{\text{def}}{=} I \Rightarrow B^* \otimes (I \Rightarrow C^*)^*$ which is isomorphic to $B \otimes (C \Rightarrow I) \Rightarrow I$. Since the category $\mathcal{C}$ is a cartesian category with a strong monad $\neg\neg$ that has Kleisli exponentials, $(\mathcal{C}, \neg\neg)$ is a λ_c-model [98].[4]

We show that the translation from call-by-value λ-calculus can be obtained by interpreting the call-by-value λ-calculus in the Kleisli category $(\mathcal{C}, \neg\neg)$ constructed from the term model. Instead of directly transforming the λ-calculus into the π_F-calculus, we first translate it into the λ_{ch}-calculus, so that we can better compare our translation with the translation given by Sangiorgi.

The result is summarized in Figure 6.3.1. Here we are considering a λ-calculus whose only base type is $()$; furthermore, we assume that all the variables appearing in a type environment have type that is not $()$.[5] This translation is exactly the same as the translation from call-by-value λ-calculus to the HOπ given by Sangiorgi; the translation $(-)^\circ$ coincides with $\mathcal{H}$ composed with $\mathcal{C}_V$. This translation can also be seen as a variant of the call-by-value CPS transformation given by Plotkin [107].

As the final step we interpret the term as the morphism in the category of computation by means of the following natural bijection

$$\begin{aligned} \mathcal{C}(A_1 \otimes \cdots \otimes A_n, I \Rightarrow B) &\cong \mathcal{K}(J(A_1) \otimes \cdots \otimes J(A_n), B) \\ &\cong \mathcal{K}(J(A_1) \otimes \cdots \otimes J(A_n) \otimes B^*, I) \end{aligned}$$

and then apply the compilation from λ_{ch}-calculus to π_F-calculus. Concretely speaking, if a type judgement of a λ-calculus $x_1 : \tau_1, \dots, x_n : \tau_n \vdash M : \sigma$ is given, then we translate this judgement as $x_1 : (\!|\tau_1^\circ|\!), \dots, x_n : (\!|\tau_n^\circ|\!), \bar{p} : (\!|\sigma^\circ \to ()|\!) \vdash (\!|M^\circ p|\!) : \diamond$. By simplifying the result using the equational rules, the translation from call-by-value λ-calculus to π_F-calculus can be summarized as in Figure 6.3.1. The result is essentially the same as the translation $\mathcal{V}$ given by Sangiorgi [117]. A minor difference is that he uses non-replicated inputs to model the additional λ-abstractions introduced by the CPS translation reflecting the fact that they are linearly used.

Remark 6.2. Note that we did not use arguments specific to the compact closed structure in the above argument. The same argument applies if $\mathcal{K}$ is a symmetric

[4]Strictly speaking, $(\mathcal{C}, \neg\neg)$ is not a λ_c-model since the mono-requirement [98] is not satisfied. However, this does not give any affect to the following arguments.

[5]This assumption is not essential. We can still define the translation even if there is a variable of type $()$. However, with this assumption, the derived translation becomes simpler since we do not have to deal with the equality $A \otimes I = A$.

$$[\![()]\!]^{\mathrm{v}} \overset{\mathrm{def}}{=} () \qquad [\![(\tau \to \sigma)]\!]^{\mathrm{v}} \overset{\mathrm{def}}{=} \mathbf{ch}^{o}[[\![\tau]\!]^{\mathrm{v}}, \mathbf{ch}^{o}[[\![\sigma]\!]^{\mathrm{v}}]]$$

$$\begin{aligned}
[\![x]\!]^{\mathrm{v}}_{\bar{p}} &\overset{\mathrm{def}}{=} \bar{p}\langle x\rangle \\
[\![(\lambda x.M)]\!]^{\mathrm{v}}_{\bar{p}} &\overset{\mathrm{def}}{=} (\boldsymbol{\nu}\bar{y}y)(\bar{p}\langle\bar{y}\rangle \mid !y(x,\bar{q}).[\![M]\!]^{\mathrm{v}}_{\bar{q}}) \\
[\![(M\ N)]\!]^{\mathrm{v}}_{\bar{p}} &\overset{\mathrm{def}}{=} (\boldsymbol{\nu}\bar{a}a)([\![M]\!]^{\mathrm{v}}_{\bar{a}} \mid !a(\bar{x}).(\boldsymbol{\nu}\bar{b}b)([\![N]\!]^{\mathrm{v}}_{\bar{b}} \mid !b(\bar{y}).\bar{x}\langle\bar{y},\bar{p}\rangle)
\end{aligned}$$

$$[\![(x_1:\tau_1,\dots x_n:\tau_n \vdash M:\sigma)]\!]^{\mathrm{v}} \overset{\mathrm{def}}{=} x_1:[\![\tau_1]\!]^{\mathrm{v}},\dots,x_n:[\![\tau_n]\!]^{\mathrm{v}},\bar{p}:\mathbf{ch}^{o}[[\![\sigma]\!]^{\mathrm{v}}] \vdash [\![M]\!]^{\mathrm{v}}_{\bar{p}}:\diamond$$

Figure 6.10: Translation from call-by-value λ-calculus to π_F-calculus.

$$()^{\bullet} \overset{\mathrm{def}}{=} () \qquad (\tau \to \sigma)^{\bullet} \overset{\mathrm{def}}{=} (\tau^{\bullet\neg\neg}, \sigma^{\bullet} \to ()) \to () \qquad \tau^{\bullet\neg\neg} \overset{\mathrm{def}}{=} (\tau^{\bullet} \to ()) \to ()$$

$$\begin{aligned}
x^{\bullet} &\overset{\mathrm{def}}{=} \lambda k.x\ k \\
(\lambda x.M)^{\bullet} &\overset{\mathrm{def}}{=} \lambda k.k\ (\lambda(x,v).M^{\bullet}\ v) \\
(M\ N)^{\bullet} &\overset{\mathrm{def}}{=} \lambda k.M^{\bullet}\ (\lambda v.v\ \langle N^{\bullet}, k\rangle)
\end{aligned}$$

$$(x_1:\tau_1,\dots x_n:\tau_n \vdash M:\sigma)^{\bullet} \overset{\mathrm{def}}{=} x_1:\tau_1^{\bullet\neg\neg},\dots,x_n:\tau_n^{\bullet\neg\neg} \vdash M^{\bullet}:\sigma^{\bullet\neg\neg}$$

Figure 6.11: Translation from the call-by-name λ-calculus to the λ_{ch}-calculus.

monoidal category with a tensorial negation. This is not a surprise because λ-calculus is a sequential language. The expressive power of compact closed categories are not needed to describe the "communication topology" of λ-calculus.

We also could have used the functor $(-) \multimap R$ instead of $(-)^*$ to define the negation, i.e. there is no need to restrict the answer type to I. However, we chose to use $(-)^*$ so that the induced translation coincides with the translation given by Sangiorgi [117]. ■

The encoding of call-by-name λ-calculus can be obtained in a similar manner by using the call-by-name monad transformation. That is we interpret $\sigma \to \tau$ as the object $\neg\neg[\![\sigma]\!] \Rightarrow_{Kl} [\![\tau]\!]$and a judgement $x_1:\tau_1,\dots,x_n:\tau_n \vdash M:\sigma$ as a morphism $[\![M]\!]\colon \neg\neg[\![\tau_1]\!] \otimes \cdots \otimes \neg\neg[\![\tau_n]\!] \to \neg\neg[\![\sigma]\!]$. By interpreting the λ-term in the term model we obtain the translations given in Figure 6.11 and Figure 6.12. Again the translation to λ_{ch}-calculus is essentially the same as $\mathcal{H}$ composed with $\mathcal{C}_N$ and the translation to the π_F-calculus is the translation $\mathcal{N}$.

6.3.2 Discussions

Relation to linear logic Using our semantic framework we briefly relate the translation from λ-calculus to π_F-calculus and translations from classical logic to linear logic.

Firstly we observe that the encoding of call-by-value λ-calculus is related to the Q-translation [28] that transforms a proof of classical sequent calculus (LK) to a proof of linear logic (LL). In Q translation the implication is translated by $(\sigma \to \tau)^Q \overset{\mathrm{def}}{=} !(\sigma^Q \multimap ?\tau^Q)$. The translation of sequents is $(A_1,\dots,A_m \vdash$

$$[\![()]\!]^{\mathrm{n}} \overset{\mathrm{def}}{=} () \qquad [\![(\tau \to \sigma)]\!]^{\mathrm{n}} \overset{\mathrm{def}}{=} \mathbf{ch}^o[[\![\tau]\!]^{\mathrm{n}\neg\neg}, \mathbf{ch}^o[[\![\sigma]\!]^{\mathrm{n}}]] \qquad [\![\tau]\!]^{\mathrm{n}\neg\neg} \overset{\mathrm{def}}{=} \mathbf{ch}^o[\mathbf{ch}^o[[\![\tau]\!]^{\mathrm{n}}]]$$

$$\begin{aligned} [\![x]\!]^{\mathrm{n}}_{\bar{p}} &\overset{\mathrm{def}}{=} x\langle \bar{p}\rangle \\ [\![(\lambda x.M)]\!]^{\mathrm{n}}_{\bar{p}} &\overset{\mathrm{def}}{=} (\boldsymbol{\nu}\bar{y}y)(\bar{p}\langle\bar{y}\rangle \mid !y(x,\bar{q}).[\![M]\!]^{\mathrm{n}}_{\bar{q}}) \\ [\![(M\ N)]\!]^{\mathrm{n}}_{\bar{p}} &\overset{\mathrm{def}}{=} (\boldsymbol{\nu}\bar{a}a)([\![M]\!]^{\mathrm{n}}_{\bar{a}} \mid !a(\bar{x}).(\boldsymbol{\nu}\bar{b}b)(!b(\bar{y}).[\![N]\!]^{\mathrm{n}}_{\bar{y}} \mid \bar{x}\langle\bar{b},\bar{p}\rangle)) \end{aligned}$$

$$[\![(x_1\!:\!\tau_1,\dots x_n\!:\!\tau_n \vdash M\!:\!\sigma)]\!]^{\mathrm{n}} \overset{\mathrm{def}}{=} x_1 : [\![\tau_1]\!]^{\mathrm{n}\neg\neg},\dots,x_n : [\![\tau_n]\!]^{\mathrm{n}\neg\neg}, \bar{p} : \mathbf{ch}^o[[\![\sigma]\!]^{\mathrm{n}}] \vdash [\![M]\!]^{\mathrm{n}}_{\bar{p}} : \diamond$$

Figure 6.12: Translation from the call-by-name λ-calculus to the π_F-calculus.

$B_1,\dots,B_n)^Q \overset{\mathrm{def}}{=} A_1^Q,\dots,A_n^Q \vdash ?B_1^Q,\dots,?B_m^Q$; note that if we only consider the intuitionistic fragment, then there must be at most one conclusion. This is exactly how we translated types and judgements in the translation from call-by-value λ-calculus to the π_F-calculus. Recall that $[\![\sigma \to \tau]\!]^{\mathrm{v}} = \mathbf{ch}^o[[\![\sigma]\!]^{\mathrm{v}}, \mathbf{ch}^o[[\![\tau]\!]^{\mathrm{v}}]]$ and thus the interpretation of $[\![\sigma \to \tau]\!]^{\mathrm{v}}$ can be read as follows (for readability we write τ^{v} for $[\![\tau]\!]^{\mathrm{v}}$)

$$\begin{aligned} [\![(\sigma \to \tau)^{\mathrm{v}}]\!] &= [\![\mathbf{ch}^o[\sigma^{\mathrm{v}}, \mathbf{ch}^o[\tau^{\mathrm{v}}]]]\!] \\ &= [\![\sigma^{\mathrm{v}}]\!] \otimes ([\![\tau^{\mathrm{v}}]\!] \Rightarrow I) \Rightarrow I \\ &\cong J(I \Rightarrow [\![\sigma^{\mathrm{v}}]\!]^* \otimes ([\![\tau^{\mathrm{v}}]\!] \Rightarrow I)^*) \\ &\cong !([\![\sigma^{\mathrm{v}}]\!]^* \otimes ([\![\tau^{\mathrm{v}}]\!] \Rightarrow I)^*) \\ &\cong !([\![\sigma^{\mathrm{v}}]\!] \multimap (![\![\tau^{\mathrm{v}}]\!]^*)^*) \\ &\cong !([\![\sigma^{\mathrm{v}}]\!] \multimap ?[\![\tau^{\mathrm{v}}]\!]) \end{aligned}$$

Here we considered $J(I \Rightarrow -)$ as $!$ because this is the exponential comonad and used the isomorphisms $?A \cong (!A^*)^*$ and $A^* \otimes B \cong A \multimap B$. As for judgements, if we move the $\bar{p} : \mathbf{ch}^o[[\![\sigma]\!]^{\mathrm{v}}]$ in

$$x_1 : [\![\tau_1]\!]^{\mathrm{v}},\dots,x_n : [\![\tau_n]\!]^{\mathrm{v}}, \bar{p} : \mathbf{ch}^o[[\![\sigma]\!]^{\mathrm{v}}] \vdash [\![M]\!]^{\mathrm{v}}_{\bar{p}} : \diamond$$

to the right-hand side, we have

$$x_1 : [\![\tau_1]\!]^{\mathrm{v}},\dots,x_n : [\![\tau_n]\!]^{\mathrm{v}}, \bar{p} : \mathbf{ch}^o[[\![\sigma]\!]^{\mathrm{v}}] \vdash [\![M]\!]^{\mathrm{v}}_{\bar{p}} : \bar{p} : \mathbf{ch}^o[[\![\sigma]\!]^{\mathrm{v}}]^{\perp}$$

and by considering $\mathbf{ch}^o[[\![\sigma]\!]^{\mathrm{v}}]^{\perp}$ as $?[\![\sigma]\!]^{\mathrm{v}}$, the translation of types becomes identical to that of the Q-translation. We conjecture that we can show a correspondence at the level of proofs as well. That is we conjecture that we can define a variant of Q-translation that transforms a proof of NJ to LL and interpreting the translated proof using π_F-processes will coincide with the call-by-value translation from λ-calculus to π_F-calculus.

Similarly, call-by-name encoding of the λ-calculus is related to the translation that maps $\tau \to \sigma$ to $!(!?\tau \multimap ?\sigma)$.[6] Danos et al. [28] have rectified this translation and defined T-translation that maps $\sigma \to \tau$ to $?(!\sigma \multimap \tau)$ which makes less use of exponential modalities. It would be interesting to investigate whether an encoding of call-by-name λ-calculus to π_F-calculus that corresponds to the

[6]This translation does not have a name, but is known to correspond to Plotkin's call-by-value CPS-translation [107]. Sometimes this translation is described as a translation that maps $\sigma \to \tau$ to $!?!\tau \to ?!\sigma$. These translations are the same translations in different presentations.

T-translation can be defined and, once defined, study its faithfulness from operational perspective. Note that the image of such a translation would not be included in the local fragment of the π_F-calculus because $?(!\sigma \multimap \tau)$ corresponds to $\mathbf{ch}^i[\mathbf{ch}^i[\sigma^\perp], \tau^\perp]$.

Encoding of $\lambda\mu$-calculus Since CPS translations or the Q-translation are used to interpret classical proofs in intuitionistic logic or linear logic, it would be natural to expect that we can interpret classical proofs in the π-calculus by using these translations.

With hindsight, this seems to be done by Honda et al. [57]. They gave an encoding of the call-by-value $\lambda\mu$-calculus [100] into a variant of **i**/**o**-typed asynchronous π-calculus, which is similar to the π_F-calculus.[7] Though it is not argued in the paper, their encoding seems to be equivalent to a combination of a variant of call-by-value CPS transformation from $\lambda\mu$-calculus to HOπ and the compilation from HOπ to the first order π-calculus.

We use λ_{ch}-calculus to explain this translation. The call-by-value CPS translation from $\lambda\mu$-calculus to the λ_{ch}-calculus is given as follows:

$$\bot^\circ \overset{\text{def}}{=} 0 \quad \text{where } 0 \text{ is a type such that } 0 \to () = ()$$

$$(\mu\alpha.M)^\circ \overset{\text{def}}{=} \lambda\alpha.M^\circ\ \langle\rangle \qquad ([\alpha]M)^\circ \overset{\text{def}}{=} \lambda k.M^\circ\alpha$$

(For the descriptions on the constructs of the $\lambda\mu$-calculus, please consult [100, 121].) Here 0 is a type called *zero* [39] such that $0 \to ()$ is terminal. (For the ease of presentation we identified $0 \to ()$ with $()$, but in general they are just isomorphic.) We only listed the difference from the CPS translation for the call-by-value λ-calculus (Figure 6.3.1), but the λ-calculus part needs some modification as well. This is because the shape of $(\sigma \to \tau)^\circ$ changes depending on whether τ is equal to 0 or not.

By applying the compilation from λ_{ch}-calculus to π_F-calculus we get

$$[\![\mu\alpha.M]\!]^{\mathrm{v}}_{\bar{p}} \overset{\text{def}}{=} [\![M]\!]^{\mathrm{v}}_{\bar{q}}\{\bar{p}/\alpha\} \qquad [\![[\alpha]M]\!]^{\mathrm{v}}_{\bar{p}} \overset{\text{def}}{=} [\![M]\!]^{\mathrm{v}}_{\alpha}$$

This is essentially the same as the way Honda et al. translates $\lambda\mu$-terms. The translation for the λ-calculus part is also the same, except for the fact that they are using bounded output instead of free outputs. If we rewrite the bounded outputs using free outputs and name creation, then the translation is almost identical to our translation given in Figure 6.3.1.

However, the categorical/denotational understanding against this translation is still premature, at least not as successful as the case of the encoding of λ-calculus. The problem is that we do not have a zero object[8] in the category of values, i.e. we do not have an object 0 that satisfies $\neg 0 \cong I$. If we do have a zero object, we can apply the result of Führmann and Thielecke [39] which says that a *response category* with zero is a model of call-by-value language with a control operator. Since a cartesian category with a tensorial negation is essentially the same as their definition of response category, the only thing we are lacking to

[7]Their type system has additional features to ensure properties like deadlock-freedom and the uniqueness of a server (replicated) process. These properties were used to prove the definability of the encoding and the decoding of processes to $\lambda\mu$-terms. Nevertheless, for discussing the encoding, a simple **i**/**o**-type will be sufficient.

[8]By zero object we mean an object that corresponds to zero type. Do not confuse with the object that is both initial and terminal. A initial object is a zero object, but the requirement of being zero is weaker than being an initial object.

apply their result is the zero object. However, even if we had a zero object whether (the continuation category of the) response category models the call-by-value $\lambda\mu$-calculus is not that clear. This is because the language used in the work of Führmann and Thielecke [39] is not the $\lambda\mu$-calculus, but a λ-calculus with Felleisen's $\mathcal{C}$ operator [35]. Further investigation is left for future work.

6.4 Related Work

Since this chapter is somewhat independent of other chapters and there exists a number of studies that relate to this chapter, comparison with related studies is made here.

Translating higher-order languages to π-calculus A number of translations from higher-order languages to the π-calculus have been developed [130, 115, 116, 118, 126] since Milner [90] presented the encodings of the λ-calculus into the π-calculus. The basic idea shared by these studies is to transform $\lambda x.M$ to a process $!a(x,p).P$ that receives the argument x together with a name p where the rest of the computation will be transmitted. In our framework, this idea is described as the isomorphism $A \Rightarrow B \cong A \otimes B^* \Rightarrow I$.

Among others, the translation from AHOπ to Lπ [118] is the closest to our translation from the λ_{ch}-calculus to the π_F-calculus. Although the translation from higher-order π-calculus to (first-order) π-calculus has already been given by Sangiorgi [115] himself, the use of Lπ allowed him to simplify the translation. He showed that his translation is fully abstract with respect to both strong and weak barbed congruence. Our translation from λ_{ch}-calculus to π_F-calculus can be seen as a semantic reconstruction of Sangiorgi's translation, as well as an extension because we do not restrict our translation to local π-calculus. However, it should be noted that the full-abstraction result with respect to barbed congruence cannot be obtained from a semantic consideration. The property that the categorical semantics guarantee is that the translation is sound and complete with respect to the axiomatic semantics of the π_F-calculus and the λ_{ch}-calculus.

The relationship between continuation passing and translation from higher-order languages to the π-calculus was already noticed since Milner's encoding of the λ-calculus [90]. The relation between CPS transformation and Milner's encoding was first partly formalized by Boudol [17] and Thielecke [125]. Thielecke [125] introduced a variant of a CPS calculus, which does not have functions as primitives, and showed the similarity between the CPS calculus and the π-calculus. Moreover, he showed that if Plotkin's CPS transformations [107] are formulated in his CPS calculus, their translations into the π-calculus yield an encoding similar to Milner's encoding. Actually, the main concern of Thielecke's work was the categorical semantics of the CPS calculus, and he observed the importance of the self-adjunction to model such a calculus. This observation led us to define our continuation monad for the π_F-calculus. Later, Sangiorgi [117] observed that Milner's translation can be factorized using CPS translation and the compilation of AHOπ to the π-calculus. We showed that Sangiorgi's translations can be understood as call-by-value and call-by-name monad interpretations using a certain continuation monad. This may not be a surprise because the relationship between call-by-name and call-by-value CPS translation and monadic interpretations were already studied by Wadler [131] and Hatcliff and Danvy [49] in the case of functional languages. However, a fully semantic explanation against monadic interpretation and translation from λ-calculus to π-calculus would not have been possible without the correspondence between π_F-calculus and compact closed Freyd categories.

Higher-order calculi with channels Besides the λ_{ch}-calculus, there are numbers of functional languages augmented by communication channels, from theoretical ones [41, 127, 133, 79, 38] to practical languages [112, 105].

On the practical side, Concurrent ML (CML) [112], among others, is a well-developed higher-order concurrent language. CML has primitives to create channels and threads, and primitives to send and accept values through channels. Since our λ_{ch}-calculus can create (non-linear) channels and send values via channels, the λ_{ch}-calculus can be seen as a core calculus of CML despite its origin in categorical semantics. The major difference between CML and the λ_{ch}-calculus is that communications in CML are synchronous whereas communications in the λ_{ch}-calculus are asynchronous.

On the theoretical side, session-typed functional languages have been actively studied [41, 127, 133, 79]. Notably, some of these languages [127, 133, 79] are built upon the Curry-Howard foundation between linear logic and session-typed processes. It might be interesting to investigate whether we can relate these languages and the λ_{ch}-calculus through the lens of Curry-Howard-Lambek correspondence.

Last but not least, we compare our λ_{ch}-calculus to the λ_{ch}-calculus given by Fowler et al. [38, 37]. To avoid confusion we write λ^S_{ch} for our calculus and λ^F_{ch} for their calculus. The λ^F_{ch} is also a concurrent λ-calculus extended with asynchronous channels that was developed independently of λ^S_{ch}. It is based on fine-grain call-by-value λ-calculus [78], which is a calculus that corresponds to the closed Freyd categories. Therefore, it has constructs such as let expressions, λ-abstractions and function applications as is the case for λ^S_{ch}-calculus. The λ^F_{ch} calculus have a special type ChanRef(A) that is for a channel that transmits values of type A. Operations related to ChanRef(A) is as follows:

$$\frac{\Gamma \vdash V : A \qquad \Gamma \vdash W : \mathrm{ChanRef}(A)}{\Gamma \vdash \mathbf{give}\, V W : 1} \qquad \frac{}{\Gamma \vdash \mathbf{newCh} : \mathrm{ChanRef}(A)}$$

$$\frac{\Gamma \vdash V : \mathrm{ChanRef}(A)}{\Gamma \vdash \mathbf{take}(V) : A}$$

The operators (and their typing rules) **give** and **newCh** are very similar to **send** and **channel** of λ^S_{ch}. However, λ^F_{ch} does not adopt **i**/**o**-separation unlike λ^S_{ch}. This difference also explains why an explicit receive operator **take** is needed in λ^F_{ch}, but not in λ^S_{ch}.

The λ^F_{ch} does not have a denotational semantics and the study on λ^F_{ch} is purely operational. The evaluation strategy is slightly different from λ^S_{ch} as well. In the λ^S_{ch}-calculus, almost all the expressions are evaluated in parallel as the let expression cannot be used for sequentializing the evaluation. On the other hand, λ^F_{ch}-calculus is basically a call-by-value language with a **fork**(M) operation that spawns a new process to evaluate term M. Probably, the call-by-value strategy was taken because λ^F_{ch} was introduced as a core language for describing realistic channel-based languages. More specifically, λ^F_{ch} was introduced to study the relationship between channel-based and actor-based programming languages. We think the fact that λ^S_{ch} and λ^F_{ch} are very similar despite their difference in the motivations supports the design of these languages.

Chapter 7

Redesigning Operational Semantics from Semantic Viewpoints

So far we have seen the semantic or logical aspects of the π_F-calculus, but the main concern of this chapter is operational semantics. The purpose of this chapter is to provide an operational semantics that harmonizes with categorical semantics.

Recall that we have shown that asynchronous π-calculus modulo most of the behavioral equivalences do not form categories (Theorem 3.5). The root cause of this problem is that most of the behavioral equivalences do not satisfy the rule (E-Eta), which is the rule that says forwarders are the left-identities.

Therefore, if a process calculus based on the asynchronous π-calculus were to have some categorical foundation, their operational behavior must be different from conventional behavior. This chapter introduces a novel reduction semantics to the π_F-calculus and show that weak barbed congruence, defined on top of the new reduction semantics is a π_F-theory. In other words, we show that there is a compact closed Freyd model that is fully abstract with respect to the barbed congruence defined upon the new reduction semantics.

Let us briefly review the problem with (E-Eta), which says that forwarders work as substitution operations for input names (cf. Section 3.2.1). A forwarder $!a(x).\bar{b}\langle x\rangle$ under the standard behavioral interpretation does not work as substitution. This is because a forwarder $!a(x).\bar{b}\langle x\rangle$ receives a message from a, *possibly waits as long as it wants or needs*, and then sends the message to the receiver. Hence the process $(\boldsymbol{\nu}\bar{b}b)(!a(x).\bar{b}\langle x\rangle \mid P)$ can immediately receive a message from a and keep it until P actually requires a message from b. On the other hand, $P\{a/b\}$ does not receive a message from a unless P actually requires it. This difference is significant in the presence of race condition, and thus (E-Eta) fails for weak barbed congruence.

This chapter discusses a new operational semantics on processes in which forwarders are delayless. The new operational semantics introduced in this chapter is a reduction semantics that forces output actions to happen as soon as they get unguarded. In the new reduction semantics, when a forwarder $!a(x).\bar{b}\langle x\rangle$ receives a message m from a, it *must immediately send* the message to a receiver b. In other words, the following two transitions are atomic

$$!a(x).\bar{b}\langle x\rangle \quad \xrightarrow{a(m)} \quad \underline{!a(x).\bar{b}\langle x\rangle \mid \bar{b}\langle m\rangle} \quad \xrightarrow{\bar{b}\langle m\rangle} \quad !a(x).\bar{b}\langle x\rangle,$$

and the process cannot stop at the underlined intermediate step since it has an unguarded output action. So a one-step reduction of the new semantics corresponds to a multi-step reduction in the conventional operational semantics. We may consider that the new behavior expresses a synchronous communication since a message m now cannot be kept in a communication medium $\bar{a}\langle m\rangle$.

We also discuss the relationship between the standard reduction relation and the new reduction relation. We show that the π_F-calculus with the standard reduction semantics can be embedded into the π_F-calculus with the new reduction semantics by using a special constant τ that explicitly represents the existence of a delay. The translation replaces each output action $\bar{a}\langle m\rangle$ with $\tau.\bar{a}\langle m\rangle$, making explicit the delay of the output action in the standard reduction. Note that the following reduction, which simulates the behavior of a conventional forwarder, is valid because the output action is guarded.

$$!a(x).\tau.\bar{b}\langle x\rangle \xrightarrow{a(m)} !a(x).\bar{b}\langle x\rangle \mid \tau.\bar{b}\langle m\rangle$$

The results in this chapter are not meant to say that we should change the operational semantics of the π-calculus. The standard reduction semantics has a good match with labeled transition semantics and are very useful for reasoning processes. Our intention is to clarify the mismatch between the standard reduction semantics and categorical semantics.

Technically the new operational semantics is quite complicated because its one-step reduction is a multi-step reduction with a certain condition in the conventional calculus. To overcome the difficulty in reasoning about such a complicated calculus, we develop an intersection type system, or equivalently a system of linear approximations [128, 83], that captures the behavior of a process. We think that the system would be of independent technical interest.

7.1 Reduction with Undelayed Output

This section makes formal the idea that a one-step reduction in the new reduction relation corresponds to a multi-step reduction in the standard reduction.

7.1.1 Definitions and the main result

We first extend the syntax of the π_F-calculus. The grammar of π_F-processes and contexts (originally defined in Section 3.1) are extended as follows:

$$P ::= \cdots \mid \tau.P \qquad\qquad C ::= \cdots \mid \tau.C$$

The operation $\tau.P$ may be regarded as an internal action as usual, but in this thesis, we emphasize the view that τ is a guard that freezes the continuation P. In other words, the continuation P in $\tau.P$ is delayed until the guard is explicitly taken off.

The sort assignment rule is also extended by the following rule:

$$\frac{(\tau : \mathbf{ch}^i[]) \in \Delta \qquad \Delta \vdash P}{\Delta \vdash \tau.P} \tag{S-Tau}$$

We allow bindings of the form $(\tau : \mathbf{ch}^i[])$ to appear in a sort environment Δ.

We redefine the reduction relation for technical convenience. The *standard reduction relation* $\xrightarrow{\ell}$ ($\ell = \tau$ or 0) is defined by the base rules

$$\begin{array}{rcl} (\boldsymbol{\nu}\vec{w}\vec{z})(\boldsymbol{\nu}\bar{a}a)(!a(\vec{x}).P \mid \bar{a}\langle\vec{y}\rangle \mid Q) & \xrightarrow{0} & (\boldsymbol{\nu}\vec{w}\vec{z})(\boldsymbol{\nu}\bar{a}a)(!a(\vec{x}).P \mid P\{\vec{y}/\vec{x}\} \mid Q) \\ (\boldsymbol{\nu}\vec{w}\vec{z})(\tau.P \mid Q) & \xrightarrow{\tau} & (\boldsymbol{\nu}\vec{w}\vec{z})(P \mid Q) \end{array}$$

together with the structural rule which concludes $P \xrightarrow{\ell} Q$ from $P \Rrightarrow P' \xrightarrow{\ell} Q' \Rrightarrow Q$ for some P' and Q'. Note that we are using the structural precongruence $\Rrightarrow$

(cf. Section 3.2.1) rather than structural congruence $\equiv$ in the above definition. We write $P \to Q$ if the label is not important.

We now formally define $\Rightarrow$. A process P has an *unguarded output action* if $P \equiv (\boldsymbol{\nu}\vec{w}\vec{z})(\bar{a}\langle\vec{x}\rangle \mid Q)$ for some Q. A process with an unguarded output action is regarded as an incomplete, intermediate state that needs to perform further actions to complete an "atomic operation". We say that P is *settled* if P has no unguarded output action. We write $P \Rightarrow Q$ if $P \xrightarrow{\tau} (\xrightarrow{0})^* Q$ and Q is settled.

The notion of *barbed congruence* can be easily adapted to this setting.

Definition 7.1 (Barbed bisimulation and barbed congruence)**.** Let $\mathcal{R}$ be a binary relation on settled processes. We say that $\mathcal{R}$ is a *barbed bisimulation with respect to Trans* if whenever $P \mathrel{\mathcal{R}} Q$,

1. $P{\downarrow}^{\tau}_{\bar{a}}$ if and only if $Q{\downarrow}^{\tau}_{\bar{a}}$
2. $P \Rightarrow P'$ implies $Q \Rightarrow Q'$ and $P' \mathrel{\mathcal{R}} Q'$ for some process Q'
3. $Q \Rightarrow Q'$ implies $P \Rightarrow P'$ and $P' \mathrel{\mathcal{R}} Q'$ for some process P',

where the notation $P{\downarrow}^{\tau}_{\bar{a}}$ means that $P \xrightarrow{\tau} (\xrightarrow{0})^* \equiv (\boldsymbol{\nu}\vec{x}\vec{y})(\bar{a}\langle\vec{z}\rangle \mid P')$ and $\bar{a}$ is a free name of P.

The *barbed bisimilarity with respect to* $\Rightarrow$, which we write $\overset{\bullet}{\sim}_{\tau}$, is the largest barbed bisimulation with respect to $\Rightarrow$. Processes P and Q are *barbed congruent*, written $P \simeq^{c}_{\tau} Q$, if $\tau.C[P] \overset{\bullet}{\sim} \tau.C[Q]$ for all context C. (The additional τ-prefixing is to ensure that the processes are settled.) ■

The main result of this chapter is that there exists a categorical model that is fully abstract with respect to $\simeq^{c}_{\tau}$.

Theorem 7.1. The relation $\simeq^{c}_{\tau}$ is a π_F-theory, which means that $Cl(\simeq^{c}_{\tau})$ is a compact closed Freyd category. Hence, there exists a compact closed Freyd category that is fully abstract with respect to $\simeq^{c}_{\tau}$. □

The proof can be easily adapted to prove a similar claim for any other congruence that subsumes $\simeq^{c}_{\tau}$, such as weak barbed congruence (for $\Rightarrow$).

7.1.2 Relationship to the standard semantics

We have introduced two reduction relations to the π_F-calculus, namely $\to$ and $\Rightarrow$. There exists an embedding of the π_F-calculus with $\to$ to that with $\Rightarrow$.

The translation, which we write $(-)^{\dagger}$, is a homomorphism on all constructs except output for which we have

$$(\bar{a}\langle\vec{x}\rangle)^{\dagger} \overset{\text{def}}{=} \tau.\bar{a}\langle\vec{x}\rangle.$$

Intuitively, this transformation adds an additional guard before every output action reflecting the fact that an output action in the standard semantics can be delayed.

The translation $(-)^{\dagger}$ preserves the semantics in the following sense.

Proposition 7.2. Let P be π_F-process such that $\Delta \vdash P$ for some Δ. Then (i) $P \to Q$ implies $(P)^{\dagger} \Rightarrow (Q)^{\dagger}$, (ii) $(P)^{\dagger} \Rightarrow Q'$ implies $Q' = (Q)^{\dagger}$ and $P \to Q$ for some Q, and (iii) $P{\downarrow}_{\bar{a}}$ iff $(P)^{\dagger}{\downarrow}^{\tau}_{\bar{a}}$.

Proof. By a simple induction on the definition of the reduction relation. We also use the following facts: (1) $(P)^{\dagger} \Rightarrow P'$ implies either $(P)^{\dagger} \xrightarrow{\tau} P'$ or $(P)^{\dagger} \xrightarrow{\tau}\xrightarrow{0} P'$ and (2) and $(P)^{\dagger} \equiv (Q)^{\dagger}$ if and only if $P \equiv Q$. □

From this proposition and the compositionality of $(-)^\dagger$, we obtain the following result. Recall that $\simeq^c$ is the (conventional) strong barbed congruence for π_F-processes (Definition 3.2).

Theorem 7.3. If $\Delta \vdash P$, $\Delta \vdash Q$ and $(P)^\dagger \simeq^c_\tau (Q)^\dagger$, then $P \simeq^c Q$.

Proof sketch. Assume that $P \not\simeq^c Q$. Then there is a context C such that $C[P] \stackrel{\bullet}{\not\sim} C[Q]$. Since $(C[P])^\dagger = (C)^\dagger[(P)^\dagger]$ and the same holds for Q, we obtain $(C)^\dagger[(P)^\dagger] \stackrel{\bullet}{\not\sim}_\tau (C)^\dagger[(Q)^\dagger]$ with the help of Proposition 7.2. Thus we conclude $(P)^\dagger \not\simeq^c_\tau (Q)^\dagger$. □

This translation, however, is *not* fully abstract with respect to barbed congruence. Contexts that are not in the image of the translation $(-)^\dagger$ give additional observational power.

7.2 Technical Overview

To prove our main theorem, i.e. to show that $\simeq^c_\tau$ is a π_F-theory, we need to show that barbed congruence with respect to $\Rrightarrow$ is a congruence that satisfies the logical axioms listed in Figure 3.2. Since barbed congruence is a congruence by definition, it suffices to check that barbed congruence satisfies the axioms.

However, checking the required axioms directly using the definition of $\Rrightarrow$ given in the previous section does not seem tractable. Recall that $P \Rrightarrow Q$ is indeed a reduction sequence $P \xrightarrow{\tau} P_1 \xrightarrow{0} \dots \xrightarrow{0} P_n \xrightarrow{0} Q$. The problem is that $P_i \xrightarrow{0} P_{i+1}$ is defined in terms of the structure of P_i, which may be quite different from that of P. A representation of reduction sequence defined by structural induction on P, without directly referring to P_i, would be desirable.

We thus utilize the following correspondence [128, 83]:

$$\begin{aligned}\text{(Reduction sequences)} &\cong \text{(Derivations in an intersection type system)}\\ &\cong \text{(Linear approximations)}.\end{aligned}$$

An example of a linear approximation is $(a_1.\tau_1.\bar{a}_2 \parallel a_2.\bot) \sqsubset\, !a.\tau.\bar{a}$ where the green part is the linear approximation of the right-hand side. A linear approximation is *linear* in the sense that each name is used exactly once and all inputs are non-replicated; it is an *approximation* in the sense that some part of the original process is discarded (e.g. $\bot \sqsubset \tau.\bar{a}$ and replicated inputs are replaced by a finite number of its copies (e.g. $(a_1.\tau_1.\bar{a}_2 \parallel a_2.\bot) \sqsubset\, !a.\tau.\bar{a}$).[1]

To see how a linear approximation corresponds to a reduction sequence, let us consider the following linear approximation:

$$\begin{aligned}&(\boldsymbol{\nu}[\langle\bar{a}_1, a_1\rangle\langle\bar{a}_2, a_2\rangle\langle\bar{a}_3, a_3\rangle])((a_1.\tau_1.(\bar{a}_2 \mid \bar{a}_3) \parallel a_2.\bot) \;\mid\; \tau_2.\bar{a}_1 \;\mid\; a_3.\bot)\\ &\qquad\qquad\qquad\qquad \sqsubset \quad (\boldsymbol{\nu}\bar{a}a)(!a.\tau.(\bar{a} \mid \bar{a}) \;\mid\; \tau.\bar{a} \;\mid\; !a.\tau.\bar{b}).\end{aligned}$$

Because of linearity, a linear approximation is race-free; hence it induces an essentially unique reduction sequence. For example,

$$\begin{aligned}&(\boldsymbol{\nu}[\langle\bar{a}_1, a_1\rangle\langle\bar{a}_2, a_2\rangle\langle\bar{a}_3, a_3\rangle])((a_1.\tau_1.(\bar{a}_2 \mid \bar{a}_3) \parallel a_2.\bot) \mid \tau_2.\bar{a}_1 \mid a_3.\bot) \qquad (7.1)\\ \xrightarrow{\tau_2}\xrightarrow{0}\; &(\boldsymbol{\nu}[\langle\bar{a}_2, a_2\rangle\langle\bar{a}_3, a_3\rangle])(a_2.\bot \mid \tau_1.(\bar{a}_2 \mid \bar{a}_3) \mid a_3.\bot)\\ \xrightarrow{\tau_1}\xrightarrow{0}\; &(\boldsymbol{\nu}[\langle\bar{a}_2, a_2\rangle])(a_2.\bot \mid (\bar{a}_2 \mid \bot) \mid \bot) \quad \xrightarrow{0} \quad (\boldsymbol{\nu}[])(\bot \mid (\bot \mid \bot) \mid \bot).\end{aligned}$$

[1] In the approximation, $\mid$ represents parallel-composition coming from the original process, whereas $p \parallel q$ means that p and q originate from the same replicated (sub)process. The operational semantics does not distinguish between $\mid$ and $\parallel$.

Importantly a reduction sequence of an approximation canonically induces that of the approximated process: the reduction sequence corresponding to (7.1) is

$$\begin{aligned}(\boldsymbol{\nu}\bar{a}a)(!a.\tau.(\bar{a} \mid \bar{a}) \mid \tau.\bar{a} \mid !a.\tau.\bar{b}) &\xrightarrow{\tau}\xrightarrow{0}(\boldsymbol{\nu}\bar{a}a)(!a.\tau.(\bar{a} \mid \bar{a}) \mid \tau.(\bar{a} \mid \bar{a}) \mid !a.\tau.\bar{b}) \\ &\xrightarrow{\tau}\xrightarrow{0}(\boldsymbol{\nu}\bar{a}a)(!a.\tau.(\bar{a} \mid \bar{a}) \mid (\bar{a} \mid \tau.\bar{b}) \mid !a.\tau.\bar{b}) \\ &\xrightarrow{0}(\boldsymbol{\nu}\bar{a}a)(!a.\tau.(\bar{a} \mid \bar{a}) \mid (\tau.(\bar{a} \mid \bar{a}) \mid \tau.\bar{b}) \mid !a.\tau.\bar{b}).\end{aligned} \tag{7.2}$$

Via the three-way correspondence mentioned above, this phenomenon can be understood as subject reduction of the intersection type system.

The fact that output actions cannot be delayed/discarded is also achieved by tweaking the definition of linear approximations. In the definition of the linear approximation we allow an output action to be approximated by a linear output, but we disallow an output action to be approximated by $\bot$, i.e. $\bar{a}_1\langle x\rangle \sqsubset \bar{a}\langle x\rangle$ is valid, but $\bot \sqsubset \bar{a}\langle x\rangle$ is not.

So far, we have discussed a relationship between $\{\, Q \mid P \rightarrow^* Q \,\}$ and $\{\, p \mid p \sqsubset P \,\}$. This relation can be seen as a bisimulation, by appropriately introducing a relation to $\{\, p \mid p \sqsubset P \,\}$. Note that such a relation is not the reduction, since $p \rightarrow q$ changes the subject, i.e. $q \sqsubset Q$ for some Q with $P \longrightarrow Q$ but not $q \sqsubset P$. Instead, we introduce an "ordering" over approximations of P. The idea is that a longer reduction sequence corresponds to a larger approximation. We write $p_1 \trianglelefteq p_2$ if p_1 is obtained by discarding some (sub)processes of p_2. For example, the second step of (7.3) corresponds to

$$\begin{aligned}&(\boldsymbol{\nu}[\langle\bar{a}_1, a_1\rangle])((a_1.\bot) \ \mid\ \tau_2.\bar{a}_1 \ \mid\ \bot) \\ &\trianglelefteq (\boldsymbol{\nu}[\langle\bar{a}_1, a_1\rangle\langle\bar{a}_3, a_3\rangle])(a_1.\tau_1.(\bot \mid \bar{a}_3) \ \mid\ \tau_2.\bar{a}_1 \ \mid\ a_3.\bot),\end{aligned} \tag{7.3}$$

that means, the third process in (7.3) is obtained by performing the actions corresponding to τ_1 and $\bar{a}_3$. It should be emphasized that *two linear approximations appearing in* (7.3.1) *are defined according to the structure of* P.

The bisimilarity gives us a characterization of the behavior of a process P in terms of linear approximations (or intersection type derivations) for P. Then checking that $\simeq^c_\tau$ satisfies the non-logical axioms can be proved by "proof manipulation". For example, the proof for (E-Beta), i.e. $(\boldsymbol{\nu}\bar{a}a)(!a(\vec{x}).P \mid C[\bar{a}\langle\vec{y}\rangle]) = (\boldsymbol{\nu}\bar{a}a)(!a(\vec{x}).P \mid C[P\{\vec{y}/\vec{x}\}])$, resembles to the proof of the substitution lemma in a typical type system.

The reset of this chapter is devoted to the proof of the main theorem. We first formally define the notion of linear approximation and then proceed to the proof of the main theorem.

7.3 Linear Approximation and Execution Sequence

We introduce *linear processes* by which executions of processes can be described.

7.3.1 Linear processes and intersection types

We start by defining linear processes. Although the definition of linear processes depends on the definition of intersection types because processes are annotated by types, we defer defining types for the sake of presentation.

Definition 7.2 (Linear processes)**.** A *linear name* is an object of the form x_i where x is an ordinary name and i is a natural number. Similarly, a *linear term*, denoted by t, is either a linear name or a constant of the form τ_i.

Linear processes are defined by the following grammar:

$$\begin{aligned} p, q ::= \mathbf{0} \mid x_i\langle\lambda_1, \dots, \lambda_n\rangle \mid x_i(\mu_1, \dots, \mu_n).p \mid \tau_i.p \\ \mid (p \mid q) \mid (p_1 \parallel \cdots \parallel p_n) \mid (\boldsymbol{\nu}[\langle x_{i_1}, y_{i_1}\rangle_{\rho_1}, \dots, \langle x_{i_n}, y_{i_n}\rangle_{\rho_n}])p \\ \mu ::= \langle x_{i_1}, \dots, x_{i_n}\rangle \qquad \lambda ::= \langle \varphi_1 \bullet x_{i_1}, \dots, \varphi_n \bullet x_{i_n}\rangle \end{aligned}$$

Here name restriction is annotated with types ρ_i, and the argument of an output action is annotated with witnesses of type isomorphisms φ_i. (The notion of types and type isomorphisms are introduced below and thus can be ignored for the moment.) In the above definition, n may be 0; for example, $(\boldsymbol{\nu}[])p$ and $\langle\rangle$ are valid process and list, respectively. We require that each linear term of a linear process appears *exactly once*. ■

The informal meanings of the constructs are almost the same as that of the ordinary processes. The linear processes $\mathbf{0}$, $x_i\langle\lambda_1, \dots, \lambda_n\rangle$ and $x_i(\mu_1, \dots, \mu_n).p$ are nil process, output action and input prefixing, respectively. An important difference from the ordinary process is that, in linear processes, the output and input take lists of variables as arguments. When a list of linear names is received each element of a list must be used exactly once. There are two types of parallel composition $p \mid q$ and $p \parallel q$. The former is the conventional parallel composition and the latter is used when a replicated process is approximated by finite parallel compositions.[2] We use $\Pi_i p_i$ as a shorthand notation of $p_1 \parallel \cdots \parallel p_n$ and write the nullary composition of $\parallel$ as $\perp$. The approximation relation defined later (Section 7.3.2) may also help the readers to understand the intuitive meaning of linear processes.

We identify linear processes with "similar structure". The *strong structural congruence*, written $p \equiv_0 q$ over linear processes is the smallest congruence relation that satisfies:

$$p \parallel q \equiv_0 q \parallel p \quad (p \parallel q) \parallel r \equiv_0 p \parallel (q \parallel r)$$
$$(\boldsymbol{\nu}[\langle x_1, y_1\rangle, \dots, \langle x_n, y_n\rangle])p \equiv_0 (\boldsymbol{\nu}[\langle x_{\sigma(1)}, y_{\sigma(1)}\rangle, \dots, \langle x_{\sigma(n)}, y_{\sigma(n)}\rangle])p,$$

where σ is a permutation over $[n]$. Since we do not care how the pairs are ordered in $\boldsymbol{\nu}[\langle x_{i_1}, y_{i_1}\rangle, \dots, \langle x_{i_n}, y_{i_n}\rangle]$, we will write $\boldsymbol{\nu}[\langle x_i, y_i\rangle]_{i\in I}$, where $I = \{i_1, \dots, i_n\}$, to represent this binding.

We now define the intersection types. The syntax of *raw types* and *raw (indexed) intersection types* are given by the following grammar:

$$\begin{aligned} &\text{(Raw types)} & \rho ::= \mathbf{ch}^o_\alpha[\theta_1, \dots, \theta_m] \mid \mathbf{ch}^i_\alpha[\theta_1, \dots, \theta_n] \\ &\text{(Raw intersection types)} & \theta ::= \bigwedge_{i\in I}(i, \rho_i) \end{aligned}$$

where $I \subseteq_{\mathrm{fin}} \mathbf{Nat}$ and α ranges over the set of *levels* $(\mathcal{A}, \leq)$, a universal poset in which any finite poset can be embedded into. In the above grammar, an intersection $\bigwedge_{i\in I}(i, \rho_i)$ is a map $i \mapsto \rho_i$ from I to types. The intuitive meaning of $\bigwedge_{i\in I}(i, \rho_i)$ is the intersection $\rho_{i_1} \wedge \rho_{i_2} \wedge \cdots \wedge \rho_{i_n}$ provided that $I = \{i_1 < i_2 < \cdots < i_n\}$.

Levels represents "timing information" and only raw types with appropriate "timing information" are considered as valid types. Let us write $\overline{\mathbf{lv}}(\rho)$ and $\overline{\mathbf{lv}}(\theta)$

[2](For readers familiar with resource calculi) Although the intuitive meaning of $p \parallel q$ is the parallel composition of p and q, this process should be thought of as an analogous to the bag in the resource λ-calculi [16, 34].

for the set of levels that appear in ρ and θ, respectively. Then *types* and *intersection types* are inductively defined as follows: $\mathbf{ch}^o_\alpha[\theta_1,\dots,\theta_n]$ is a type if θ_i is an intersection type for all $i \in [n]$ and $\alpha \le \gamma$ for all $\gamma \in \overline{\mathbf{lv}}(\theta_1,\dots,\theta_n)$ (similar for $\mathbf{ch}^i_\alpha[\theta_1,\dots,\theta_n]$) and $\bigwedge_{i\in I}(i,\rho_i)$ is an intersection type if ρ_i is a type for all $i \in I$. *In what follows, we use the metavariables ρ and θ to range over types and intersection types, respectively.*

Notations: A special symbol $\bullet$ is introduced to mean undefined type of sort T; now an intersection type θ can be also be represented by a (total) function from $\mathbf{Nat}$ to the union of the set of types and $\{\bullet\}$. We write $(i_1,\rho_{i_1}) \wedge \cdots \wedge (i_n,\rho_{i_n})$ for the intersection type θ such that $\mathrm{dom}(\theta) = \{i_1 < \cdots < i_n\}$ and $\theta(i_j) = \rho_{i_j}$ for every $j \in [n]$. We also write $\top$ for the empty intersection, i.e. θ such that $\theta(i) = \bullet$ for all $i \in \mathbf{Nat}$. The *dual* $\rho^\perp$ of type ρ is defined by $\mathbf{ch}^o_\alpha[\vec{\theta}]^\perp \overset{\mathrm{def}}{=} \mathbf{ch}^i_\alpha[\vec{\theta}]$ and $\mathbf{ch}^i_\alpha[\vec{\theta}]^\perp_\alpha \overset{\mathrm{def}}{=} \mathbf{ch}^o_\alpha[\vec{\theta}]$. We also define $\theta^\perp$ by $\theta^\perp(i) = (\theta(i))^\perp$.

The type $\mathbf{ch}^i_\alpha[\theta_1,\dots,\theta_n]$ is for a channel that is used to receive n lists, where the i-th list has type θ_i and the type $\mathbf{ch}^o_\alpha[\vec{\theta}]$ is for output channels. If the i-th list has type $(i_1,\rho_1) \wedge \cdots \wedge (i_m,\rho_m)$, it means that the j-the element of the list has type ρ_j. For example, $a_i(\langle x_1,x_2\rangle,\langle \bar{y}_1\rangle).x_1().x_2().\bar{y}_1\langle\langle\rangle\rangle$ is well-typed if a_i has type $\mathbf{ch}^i_\alpha[(0,\mathbf{ch}^i_\beta[]) \wedge (1,\mathbf{ch}^i_\gamma[]),\ (0,\mathbf{ch}^o_\gamma[\top])]$ with $\alpha \le \beta \le \gamma$. The levels are used to describe the timing of actions. In the above example, the level γ tells us that the second element of the first argument, namely x_2, and the first element of the second argument, namely $\bar{y}_1$, must be used at the same timing. Levels also describe the fact that x_1 must be used before x_2 and $\bar{y}_1$ are used.

Although the intersection types are non-commutative in the sense that $(0,\rho) \wedge (1,\rho') \ne (0,\rho') \wedge (1,\rho)$, we consider that they are isomorphic. Intuitively, this means that we do not mind that much about the order of elements in a list. For example, we consider that $\bar{a}_0\langle\langle x_0,x_1\rangle\rangle$ and $\bar{a}_0\langle\langle x_1,x_0\rangle\rangle$ are almost identical.[3]Without this identification, we face a technical problem: an approximation of a forwarder $a_0(\langle y_0,y_1\rangle).\bar{b}_0\langle\langle y_1,y_0\rangle\rangle$ cannot be seen as an "identity" because $(\boldsymbol{\nu}[\langle \bar{a}_0,a_0\rangle])(a_0(\langle y_0,y_1\rangle).\bar{b}_0\langle\langle y_1,y_0\rangle\rangle \mid \bar{a}_0\langle\langle x_0,x_1\rangle\rangle)$ "reduces to" $\bar{b}_0\langle\langle x_1,x_0\rangle\rangle$.

Definition 7.3 (Type isomorphism)**.** We write $\varphi\colon \rho \cong \rho'$ (resp. $\varphi\colon \theta \cong \theta'$) to mean that ρ and ρ' (resp. θ and θ') are *isomorphic* and that φ is the *witness* of this isomorphism. This relation is defined by the rules below:[4]

$$\frac{}{\mathrm{id}_\bullet\colon \bullet \cong \bullet}$$

$$\frac{\varphi_i\colon \theta_i \cong \theta'_i \qquad (\text{for } i \in [n])}{\mathbf{ch}^o_\alpha[\varphi_1,\dots,\varphi_n]\colon \mathbf{ch}^o_\alpha[\theta'_1,\dots,\theta'_n] \cong \mathbf{ch}^o_\alpha[\theta_1,\dots,\theta_n]}$$

$$\frac{\varphi_i\colon \theta_i \cong \theta'_i \qquad (\text{for } i \in [n])}{\mathbf{ch}^i_\alpha[\varphi_1,\dots,\varphi_n]\colon \mathbf{ch}^i_\alpha[\theta_1,\dots,\theta_n] \cong \mathbf{ch}^i_\alpha[\theta'_1,\dots,\theta'_n]}$$

$$\frac{\sigma\colon \mathbf{Nat} \overset{\cong}{\to} \mathbf{Nat} \qquad \varphi_i\colon \rho_i \cong \rho'_{\sigma(i)} \qquad (\text{for } i \in \mathbf{Nat})}{(\sigma,(\varphi_i)_{i\in\mathbf{Nat}})\colon \bigwedge_{i\in\mathbf{Nat}}(i,\rho_i) \cong \bigwedge_{i\in\mathbf{Nat}}(i,\rho'_i)}$$

Note that $\mathbf{ch}^o_\alpha[-]$ is a contravariant operator, whereas $\mathbf{ch}^i_\alpha[-]$ is a covariant operator. ■

[3]The reason for annotating arguments of free outputs with φ is slightly more technical. This is needed to ensure that the η-rule for the linear process is valid. (cf. Remark 7.3)

[4]Here, $\bigwedge_{i\in I}(i,\rho_i)$ is considered as a total map $\bigwedge_{i\in\mathbf{Nat}}(i,\rho_i)$ in which $\rho_i \overset{\mathrm{def}}{=} \bullet$ if $i \notin I$.

$$\frac{}{\emptyset \vdash_\alpha \mathbf{0}} \quad \text{(TNIL)}$$

$$\frac{\varphi_i \colon \theta_i \cong \theta_i' \qquad \varphi_i \bullet x^i = \lambda_i \quad (\text{for } i \in [n]) \quad \alpha \le \overline{\mathbf{lv}}(\theta_1, \ldots, \theta_n)}{x^1 : \theta_1 \sqcap \cdots \sqcap x^n : \theta_n, \bar{a} : (i, \mathbf{ch}^o_\alpha[\theta_1', \ldots, \theta_n']) \vdash_\alpha \bar{a}_i\langle \lambda_1, \ldots, \lambda_n \rangle} \quad \text{(TOUT)}$$

$$\frac{\Gamma, x^1 : \theta_1, \ldots, x^n : \theta_n \vdash_\beta p \quad \mathrm{id}_{\theta_i} \bullet x^i = \mu_i \quad \alpha \le \beta}{\Gamma \sqcap a : (i, \mathbf{ch}^i_\beta[\theta_1, \ldots, \theta_n]) \vdash_\alpha a_i(\mu_1, \ldots, \mu_n).p} \quad \text{(TIN)}$$

$$\frac{\Gamma \vdash_\beta p \quad \alpha \le \beta}{\Gamma \sqcap \tau : (i, \mathbf{ch}^i_\beta[]) \vdash_\alpha \tau_i.p} \quad \text{(TTAU)} \qquad \frac{\Gamma_1 \vdash_\alpha p_1 \qquad \Gamma_2 \vdash_\alpha p_2}{\Gamma_1 \sqcap \Gamma_2 \vdash_\alpha p_1 \mid p_2} \quad \text{(TPAR)}$$

$$\frac{\Gamma_i \vdash_\alpha p_i \quad (1 \le i \le n)}{\Gamma_1 \sqcap \cdots \sqcap \Gamma_n \vdash_\alpha p_1 \parallel \cdots \parallel p_n} \quad \text{(TREP)} \qquad \frac{\Gamma, \bar{a} : \theta, a : \theta^\perp \vdash_\alpha p}{\Gamma \vdash_\alpha (\boldsymbol{\nu}[\langle \bar{a}_i, a_i \rangle]_{i \in \mathrm{dom}(\theta)})p} \quad \text{(TNU)}$$

Figure 7.1: Typing rules for the intersection type system.

We define yet another operator $\theta_1 \sqcap \theta_2$ for intersection types which "coalesces" the two intersection. It is defined by

$$(\theta_1 \sqcap \theta_2)(i) \stackrel{\text{def}}{=} \begin{cases} \theta_1(i) & (\text{if } i \in \mathrm{dom}(\theta_1)) \\ \theta_2(i) & (\text{if } i \in \mathrm{dom}(\theta_2)) \\ \bullet & (\text{otherwise}) \end{cases}$$

provided that $\mathrm{dom}(\theta_1) \cap \mathrm{dom}(\theta_2) = \emptyset$; otherwise $\theta_1 \sqcap \theta_2$ is undefined.

We now present the typing rules for the intersection type system. Type judgments are of the form $\Gamma \vdash_\alpha p$ and the typing rules are given in Figure 7.1. We stipulate that the deduction is allowed only if the result of the $\sqcap$ operation in the conclusion is defined. The operation $\varphi \bullet x$ used in the above definition is defined by

$$(\sigma, (\varphi_i)_{i \in \mathbf{Nat}}) \bullet x \stackrel{\text{def}}{=} \langle \varphi_{\sigma^{-1}(i_1)} \bullet x_{\sigma^{-1}(i_1)}, \ldots, \varphi_{\sigma^{-1}(i_n)} \bullet x_{\sigma^{-1}(i_n)} \rangle,$$

where $(\sigma, (\varphi_i)_{i \in \mathbf{Nat}}) \colon \theta \cong \theta'$ and $\mathrm{dom}(\theta') = \{i_1 < \cdots < i_n\}$; similarly $\mathrm{id}_\theta \bullet x$ is defined as $\langle x_{i_1}, \ldots, x_{i_n} \rangle$ when $\mathrm{dom}(\theta) = \{i_1 < \cdots < i_n\}$.

Let us explain how the subscript α of $\vdash_\alpha$ is used; the other parts of the typing rule should be easy to understand. The intuitive meaning of the subscript α of $\vdash_\alpha$ is the "current time". The typing rule for output actions ensures that the "level of $\bar{a}_i$" is the "current time". In other words, the rule ensures that the output cannot be delayed. On the other hand, we may delay an input or a τ action. For example, in the rule (TIN), the "level of a_i" can be greater than α meaning that we can delay the use of a_i. The rule (TIN) also says that the "level of a_i" must be equal to the level assigned to $\Gamma, x^1 : \theta_1, \ldots, x^n : \theta_n \vdash_\beta p$. This expresses the fact that the unguarded outputs in p must be used as soon as a_i is used, i.e. there cannot be any delay between an input and an output.

7.3.2 Approximation

In this subsection we show how sorts are refined by intersection types and processes are approximated by linear processes.

Given a sort T, the *refinement relation* $\rho \sqsubset T$ (resp. $\theta \sqsubset T$), meaning that the type ρ (resp. the intersection type θ) refines the sort T, is defined by the following rules:

$$\frac{\theta_i \sqsubset T_i \quad (i \in [n]) \qquad m \in \{i, o\}}{\mathbf{ch}^m_\alpha[\theta_1, \ldots, \theta_n] \sqsubset \mathbf{ch}^m[T_1, \ldots, T_n]} \qquad \frac{\rho_i \sqsubset T \quad (i \in I \subseteq_{\text{fin}} \mathbf{Nat})}{\bigwedge_{i \in I} (i, \rho_i) \sqsubset T}.$$

$$\frac{}{x : \{i_1, \dots, i_n\} \vdash \langle \varphi_1 \bullet x_{i_1}, \dots, \varphi_n \bullet x_{i_n} \rangle \sqsubset x} \qquad \frac{}{\vdash \mathbf{0} \sqsubset \mathbf{0}} \qquad \frac{}{\vdash \bot \sqsubset \tau.P}$$

$$\frac{X_i \vdash \lambda_i \sqsubset x_i \quad (\text{for } i \in [n])}{X_1 \sqcap \cdots \sqcap X_n \sqcap \bar{a} : \{i\} \vdash \bar{a}_i\langle \lambda_1, \dots, \lambda_n \rangle \sqsubset \bar{a}\langle x_1, \dots, x_n \rangle} \qquad \frac{X \vdash p \sqsubset P}{X \sqcap \tau : \{i\} \vdash \tau_i.p \sqsubset \tau.P}$$

$$\frac{X, x_1 : S_1, \dots, x_n : S_n \vdash p \sqsubset P \qquad x_i : S_i \vdash \mu_i \sqsubset x_i \quad (\text{for } i \in [n])}{X \sqcap a : \{i\} \vdash a_i(\mu_1, \dots, \mu_n).p \sqsubset a(x_1, \dots, x_n).P}$$

$$\frac{X_1 \vdash p \sqsubset P \qquad X_1 \vdash q \sqsubset Q}{X_1 \sqcap X_2 \vdash p \mid q \sqsubset P \mid Q} \qquad \frac{X_i \vdash p_i \sqsubset P \quad (\text{for } i \in [n])}{X_1 \sqcap \cdots \sqcap X_n \vdash p_1 \parallel \cdots \parallel p_n \sqsubset !P}$$

$$\frac{X, x : S, y : S \vdash p \sqsubset P \qquad S = \{i_1, \dots, i_n\} \qquad \rho_i \sqsubset T \quad (\text{for } i \in S)}{X \vdash (\boldsymbol{\nu}[\langle x_{i_1}, y_{i_1} \rangle_{\rho_{i_1}}, \dots, \langle x_{i_n}, y_{i_n} \rangle_{\rho_{i_n}}])p \sqsubset (\boldsymbol{\nu}_T xy)P}$$

Figure 7.2: Rules for approximation relation. We stipulate that the deduction is allowed only if the result of the $\sqcap$ operation in the conclusion is defined.

We write $\Gamma \sqsubset \Delta$ if $(x : \theta) \in \Gamma$ implies that $(x : T) \in \Delta$ for some T and $\theta \sqsubset T$.

Next we show how processes are approximated by linear processes.

A *term refinement* X is a finite set of the form $t_1 : S_1, \dots, t_n : S_n$ such that $S_i \subseteq_{\text{fin}} \mathbf{Nat}$ and $i \neq j$ implies $t_i \neq t_j$. Notations $X(t)$ and $X_1 \sqcap X_2$ are defined analogous to $\Gamma(t)$ and $\Gamma_1 \sqcap \Gamma_2$. There is a canonical way to obtain a term refinement from a type environment: given a type environment Γ, we define $\Gamma^\natural$ as $\{(t : \mathrm{dom}(\Gamma(t))) \mid t \in \mathrm{dom}(\Gamma)\}$.

An *approximation judgement* is of the form $X \vdash p \sqsubset P$ and inference rules for judgments are given in Figure 7.2. It should be emphasized that we do not allow $\bot \sqsubset \bar{a}\langle \vec{x} \rangle$, that is, we ensure that all the output actions are used. Note that we can discard an output action that is guarded by τ, i.e. $\bot \sqsubset \tau.\bar{a}\langle \vec{x} \rangle$, and this is why the translation $(-)^\dagger$ defined in Section 7.1.2 allows us to relate the reduction $\longrightarrow$ with $\Rightarrow$.

7.3.3 Reduction

Here we define the reduction relation for linear processes. We also show that every reduction sequence from P has a representation by a linear process p such that $X \vdash p \sqsubset P$.

The reduction relation of linear processes is almost the same as that of ordinary processes. We define the base step of the reduction, where the substitution of (linear) names occur, and the structural rule for the reduction.

However, substitutions for linear processes is more complex than that for ordinary processes. We need to take actions of type isomorphisms to linear processes into account. The action of φ to linear processes is defined by the rules in Figure 7.3. It is defined via the action of type isomorphisms on subject names and operation $\{\varphi \bullet y/x\}$, which substitutes $\varphi \bullet y$ to x. The substitution $\{\varphi \bullet y/x\}$ works as the standard substitution, except for the fact the action of φ is performed after the substitution. The witness $\varphi_2 \circ \varphi_1 \colon \rho_1 \cong \rho_3$ is the *composition* of $\varphi_1 \colon \rho_1 \cong \rho_2$ and $\varphi_2 \colon \rho_2 \cong \rho_3$ defined below. For readability, given $\lambda \overset{\text{def}}{=} \langle \varphi_1 \bullet y_1, \dots, \varphi_n \bullet y_n \rangle$ and $\mu \overset{\text{def}}{=} \langle x_1, \dots, x_n \rangle$, we write $\{\lambda/\mu\}$ to denote $\{\varphi_1 \bullet y_1/x_1, \dots, \varphi_n \bullet y_n/x_n\}$.

We complete the definition of $\{\varphi \bullet y/x\}$ by giving the definition of $\varphi_2 \circ \varphi_1$.

$$\begin{aligned}
&\langle \varphi_1 \centerdot x_{i_1}, \ldots, \varphi_i \centerdot (\varphi' \centerdot y), \ldots, \varphi_n \centerdot x_{i_n} \rangle \stackrel{\text{def}}{=} \langle \varphi_1 \centerdot x_{i_1}, \ldots, (\varphi_i \circ \varphi') \centerdot y, \ldots, \varphi_n \centerdot x_{i_n} \rangle \\
&(\mathbf{ch}^o[\varphi] \centerdot \bar{a}_i)\langle\langle \varphi'_{i_1} \centerdot x_{i_1}, \ldots, \varphi'_{i_n} \centerdot x_{i_n} \rangle\rangle \stackrel{\text{def}}{=} \bar{a}_i\langle\langle (\varphi_{i_1} \circ \varphi'_{\sigma(i_1)}) \centerdot x_{\sigma(i_1)}, \ldots, (\varphi_{i_n} \circ \varphi'_{\sigma(i_n)}) \centerdot x_{\sigma(i_n)} \rangle\rangle \\
&(\mathbf{ch}^i[\varphi] \centerdot a_i)(\langle x_{i_1}, \ldots, x_{i_n} \rangle).p \stackrel{\text{def}}{=} a_i(\langle x_{i_1}, \ldots, x_{i_n} \rangle).p\{\varphi_{\sigma^{-1}(j)} \centerdot x_{\sigma^{-1}(j)}/x_j\}_{j \in \{i_1, \ldots, i_n\}}
\end{aligned}$$

Figure 7.3: Action of isomorphisms on linear (monadic) processes where $\varphi = (\sigma, (\varphi_i))$ in the last two equations; the action on polyadic processes is defined similarly.

Composition of witnesses are defined by:

$$\begin{aligned}
\mathbf{ch}^o_\alpha[\varphi'_1, \ldots, \varphi'_n] \circ \mathbf{ch}^o_\alpha[\varphi_1, \ldots, \varphi_n] &\stackrel{\text{def}}{=} \mathbf{ch}^o_\alpha[\varphi_1 \circ \varphi'_1, \ldots, \varphi_n \circ \varphi'_n] \\
\mathbf{ch}^i_\alpha[\varphi'_1, \ldots, \varphi'_n] \circ \mathbf{ch}^i_\alpha[\varphi_1, \ldots, \varphi_n] &\stackrel{\text{def}}{=} \mathbf{ch}^i_\alpha[\varphi'_1 \circ \varphi_1, \ldots, \varphi'_n \circ \varphi_n] \\
(\sigma_2, (\varphi'_i)_{i \in \mathbf{Nat}}) \circ (\sigma_1, (\varphi_i)_{i \in \mathbf{Nat}}) &\stackrel{\text{def}}{=} (\sigma_2\sigma_1, (\varphi'_{\sigma_1(i)} \circ \varphi_i)_{i \in \mathbf{Nat}})
\end{aligned}$$

Types and type isomorphisms forms a groupoid. That is, we can define the inverse operator $(-)^{-1}$ for witnesses of type isomorphisms and show that there is an identity $\mathrm{id}_\rho \colon \rho \cong \rho$ for every type ρ. The inverse operator $(-)^{-1}$ is defined by:

$$\begin{aligned}
(\mathbf{ch}^m_\alpha[\varphi_1, \ldots, \varphi_n])^{-1} &\stackrel{\text{def}}{=} \mathbf{ch}^m_\alpha[\varphi_1^{-1}, \ldots, \varphi_n^{-1}] \quad (\text{for } m \in \{i, o\}) \\
(\sigma, (\varphi_i)_{i \in \mathbf{Nat}})^{-1} &\stackrel{\text{def}}{=} (\sigma^{-1}, (\varphi^{-1}_{\sigma^{-1}(i)})_{i \in \mathbf{Nat}}).
\end{aligned}$$

As is the case of standard substitution, $\{\varphi \centerdot y/x\}$ satisfies (a modified version of) the substitution lemma.

Lemma 7.4 (Substitution Lemma)**.** Suppose that $\Gamma \sqcap x : (i, \rho) \vdash_\alpha p$, $\varphi \colon \rho' \cong \rho$ and $\Gamma \sqcap y : (j, \rho')$ is defined. Then $\Gamma \sqcap y : (j, \rho') \vdash_\alpha p\{\varphi \centerdot y_j/x_i\}$.

Proof. By induction on the structure of p. The proof of this lemma is similar to that of the conventional substitution lemma, except for the fact that we need to take group actions into account. □

Lemma 7.5. Let $\Gamma \sqcap x : (i, \rho) \vdash_\alpha p$, $\varphi_1 \colon \rho \cong \rho'$, $\varphi_2 \colon \rho' \cong \rho''$ and assume that $y_j, z_k \notin \mathbf{fn}(p)$. Then $p\{\varphi_1 \centerdot y_j/x_i\}\{\varphi_2 \centerdot z_k/y_j\} = p\{(\varphi_2 \circ \varphi_1) \centerdot z_k/x_i\}$

Proof. By induction on the structure of p. □

Once we defined the notion of substituion, it is straightforward to define the reduction relation. The *structural precongruence* $\Rrightarrow$ over linear process is the smallest *precongruence* relation that contains $\equiv_0$, contains α-equivalence and satisfies:

$$\begin{aligned}
&\mathbf{0} \mid p \Lleftarrow\!\!\Rrightarrow p \quad p \mid q \Lleftarrow\!\!\Rrightarrow q \mid p \quad (p \mid q) \mid r \Lleftarrow\!\!\Rrightarrow p \mid (q \mid r) \\
&(\boldsymbol{\nu}[\langle \vec{w}, \vec{z} \rangle])(\boldsymbol{\nu}[\langle \vec{y}, \vec{z} \rangle])p \Lleftarrow\!\!\Rrightarrow (\boldsymbol{\nu}[\langle \vec{y}, \vec{z} \rangle])(\boldsymbol{\nu}[\langle \vec{w}, \vec{x} \rangle])p \quad (\mathbf{fn}(\vec{w}, \vec{x}) \cap \mathbf{fn}(\vec{y}, \vec{z}) = \emptyset) \\
&(\boldsymbol{\nu}[\langle x_1, y_1 \rangle, \ldots, \langle x_n, y_n \rangle])p \mid q \Rrightarrow (\boldsymbol{\nu}[\langle x_1, y_1 \rangle, \ldots, \langle x_n, y_n \rangle])(p \mid q) \quad (\vec{x}, \vec{y} \notin \mathbf{fn}(q))
\end{aligned}$$

where $p \Lleftarrow\!\!\Rrightarrow q$ means $p \Rrightarrow q$ and $q \Rrightarrow p$. The *structural congruence* $\equiv$ for linear processes is defined as symmetric closure of $\Rrightarrow$.

We define the *one-step reduction relation* over well-typed linear processes by the base rule

$$\begin{aligned}&(\boldsymbol{\nu}\vec{\xi})(\boldsymbol{\nu}[\langle \bar{a}_i, a_i\rangle]_{i\in J})(\Pi_{i\in I} a_i(\mu_{i1},\dots,\mu_{in}).p_i \mid \bar{a}_m\langle \lambda_1,\dots,\lambda_n\rangle \mid q) \xrightarrow{0}\\ &(\boldsymbol{\nu}\vec{\xi})(\boldsymbol{\nu}[\langle \bar{a}_i, a_i\rangle]_{i\in J'})(\Pi_{i\in I'} a_i(\mu_{i1},\dots,\mu_{in}).p_i \mid p_m\{\lambda_1/\mu_{m1},\dots,\lambda_n/\mu_{mn}\} \mid q)\end{aligned}$$

where $(\boldsymbol{\nu}\vec{\xi})$ is a sequence of name restrictions, $m \in I \subseteq J$, $J' = J \setminus \{m\}$ and $I' = I \setminus \{m\}$, and the structural rule which concludes $p \xrightarrow{0} q$ from $p \Rrightarrow p'$ and $p' \xrightarrow{0} q$.

The relation $\xrightarrow{\tau_i}$ is obtained by replacing the base rule of the $\xrightarrow{0}$ with $(\boldsymbol{\nu}\vec{\xi})(\tau_i.p \mid q) \xrightarrow{\tau_i} (\boldsymbol{\nu}\vec{\xi})(p \mid q)$. We write $p \xrightarrow{\tau} q$ if $p \xrightarrow{\tau_i} q$ for some index i.

Remark 7.1. We use $\Rrightarrow$ instead of $\equiv$ in the definition of reduction because $X \vdash p \sqsubset P$ and $p \equiv q$ does not ensure the existence of Q such that $X \vdash q \sqsubset Q$ and $P \equiv Q$. For instance, if $P \overset{\text{def}}{=} (\boldsymbol{\nu}\bar{a}a)(!a(x).R \mid \tau.\bar{a}\langle y\rangle)$ then $(\boldsymbol{\nu}[])(\bot \mid \bot)$ approximates P and this linear process is structurally congruent to $(\boldsymbol{\nu}[])\bot \mid \bot$, but there is no Q such that $(\boldsymbol{\nu}[])\bot \mid \bot \sqsubset Q$ and $P \equiv Q$. ■

We now show the correspondence between execution sequences and linear approximations, which we briefly explained in Section 7.2. Let us write $P \xrightarrow{\pi} Q$ if there exists a sequence $P = P_0 \xrightarrow{l_1} P_1 \xrightarrow{l_2} \cdots \xrightarrow{l_n} P_n = Q$, where each l_i is either 0 or τ, and $\pi = l_1 l_2 \dots l_n$; $p \xrightarrow{\pi} q$ is defined similarly. We write $(p \xrightarrow{\pi} q) \sqsubset (P \xrightarrow{\pi} Q)$ if there exists $p = p_0 \xrightarrow{l_1} \cdots \xrightarrow{l_n} p_n = q$ and $P = P_0 \xrightarrow{l_1} \cdots \xrightarrow{l_n} P_n = Q$ such that $X_i \vdash p_i \sqsubset P_i$ for some X_i for each $i \in \{0,\dots,n\}$ and $\pi = l_1 \cdots l_n$.

Proposition 7.6. Let $\tau : \mathbf{ch}^i[] \vdash P$, i.e. let P be a process that does not have any free names.

1. Suppose that $\Gamma \vdash_\alpha p$ and $\Gamma^\natural \vdash p \sqsubset P$. If $p \xrightarrow{\pi} q$ then we have $(p \xrightarrow{\pi} q) \sqsubset (P \xrightarrow{\pi} Q)$ for some Q.
2. Assume $P \xrightarrow{\pi} Q$, $\Gamma \vdash_\alpha q$ and $\Gamma^\natural \vdash q \sqsubset Q$. Then we have $(p \xrightarrow{\pi} q) \sqsubset (P \xrightarrow{\pi} Q)$ for some p.

□

The above proposition is a consequence of the subject reduction/expansion property of the type system. We thus prove the above theorem after proving subject reduction/expansion.

The subject reduction lemma, and a similar lemma for the τ-reduction can be stated as follows:

Lemma 7.7 (Subject reduction)**.** Assume that $\Gamma \vdash_\alpha p$ and $p \xrightarrow{0} q$. Then we have $\Gamma \vdash_\alpha q$. Moreover, if $\Gamma^\natural \vdash p \sqsubset P$ then there exists Q such that $\Gamma^\natural \vdash q \sqsubset Q$ and $P \longrightarrow Q$. □

Lemma 7.8. Suppose that $\Gamma \sqcap \tau : (i, \mathbf{ch}^i_\beta[]) \vdash_\alpha p$, $\beta \leq \gamma$ for all $\gamma \in \overline{\mathbf{lv}}(\Gamma)$ and $p \xrightarrow{\tau_i} q$. Then we have $\Gamma \vdash_\beta q$. Moreover, if $({}^\natural\Gamma) \sqcap \tau : \{i\} \vdash p \sqsubset P$, then there exists Q such that $\Gamma^\natural \vdash q \sqsubset Q$ and $P \xrightarrow{\tau} Q$. □

We omit the proofs as they can be shown by standard arguments.

Since we are using an intersection type system, the subject expansion property also holds.

Lemma 7.9 (Subject expansion)**.** Suppose that $P \longrightarrow P'$, $\Gamma \vdash_\alpha p'$ and $\Gamma^\natural \vdash p' \sqsubset P'$. Then there exists p such that $\Gamma \vdash_\alpha p$, $\Gamma^\natural \vdash p \sqsubset P$ and $p \xrightarrow{0} p'$.

Proof. We only consider the base case of the reduction; proving the correspondence between $\Rrightarrow$ and $\Rrightarrow$ is easy. Furthermore, for simplicity, we consider the case where the arity of the channel is one. That is we consider the case where

$$P = (\boldsymbol{\nu}\bar{a}a)(!a(x).Q \mid \bar{a}\langle y\rangle \mid R)$$
$$P' = (\boldsymbol{\nu}\bar{a}a)(!a(x).Q \mid Q\{y/x\} \mid R).$$

(We also omitted the outermost ν operators.)

Since $\Gamma^\natural \vdash p' \sqsubset P'$ and $\Gamma \vdash_\alpha p'$, p' is of the form

$$(\boldsymbol{\nu}[\langle \bar{a}_i, a_i\rangle]_{i\in\mathrm{dom}(\theta)})(\Pi_{i\in I} a_i(\mu_i).q_i \mid q' \mid r)$$

and the derivation of $\Gamma \vdash_\alpha p'$ must be of the following form

$$\dfrac{\dfrac{\pi_1 :: \Gamma_1 \vdash \Pi_{i\in I} a_i(\mu_i).q_i \quad \pi_2 :: \Gamma_2 \vdash_\alpha q' \quad \pi_3 :: \Gamma_3 \vdash_\alpha r}{\Gamma, \bar{a} : \theta, a : \theta^\perp \vdash_\alpha (\Pi_{i\in I} a_i(\mu_i).q_i \mid q' \mid r)}}{\Gamma \vdash_\alpha p'}$$

where $\Gamma_1 \sqcap \Gamma_2 \sqcap \Gamma_3 = \Gamma, \bar{a} : \theta, a : \theta^\perp$ and $I \subseteq \mathrm{dom}(\theta)$. Note that we also have $\Gamma_1^\natural \vdash \Pi_{i\in I} a_i(\mu_i).q_i \sqsubset\, !a(x).Q$, $\Gamma_2^\natural \vdash q' \sqsubset Q\{y/x\}$ and $\Gamma_3^\natural \vdash r \sqsubset R$. Since we know the shape of Q and $Q\{y/x\}$, we know which occurrences of y in $Q\{y/x\}$ are due to the substitution $\{y/x\}$. Thus, we can split Γ_2 as $\Gamma_2' \sqcap y : \theta_y$ such that y_i $(i \in \mathrm{dom}(\theta_y))$ approximates the occurrence of y that originates from the substitution. Let us write q_k for $q'\{x_i/y_i\}_{i\in\mathrm{dom}(\theta_y)}$ under the assumption that $k \notin \mathrm{dom}(\theta)$. Then define p as

$$(\boldsymbol{\nu}[\langle \bar{a}_i, a_i\rangle]_{i\in\mathrm{dom}(\theta')})(\Pi_{i\in I\cup\{k\}} a_i(\mu_i).q_i \mid \bar{a}_k\langle\lambda\rangle \mid r)$$

where $\theta' = \theta \wedge (k, \mathbf{ch}^o_\alpha[\theta_y])$, $\mu_k = \mathrm{id}_{\theta_y} \bullet x$ and $\lambda = \mathrm{id}_{\theta_y} \bullet y$. It is clear that this linear process can reduce to p'.

We now show that p is typed under Γ. By substitution lemma (Lemma 7.4), we have $\Gamma_2', x : \theta_y \vdash_\alpha q_k$. Applying the typing rule for input prefixing to this derivation, we obtain a derivation π_2' for $\Gamma_2' \sqcap a : (k, \mathbf{ch}^i_\alpha[\theta_y]) \vdash_\alpha a_k(\mu_k).q_k$. We also have a type derivation for $\bar{a}_k\langle\lambda\rangle$, namely

$$\dfrac{}{\bar{a} : (k, \mathbf{ch}^o_\alpha[\theta_y]), y : \theta_y \vdash_\alpha \bar{a}_k\langle\lambda\rangle}$$

Therefore, we can construct a derivation for p:

$$\dfrac{\dfrac{\dfrac{\pi_1 \qquad \pi_2'}{\Gamma_1 \sqcap \Gamma_2' \vdash \Pi_{i\in I\cup\{k\}} a_i(\mu_i).q_i} \quad \dfrac{}{\bar{a} : (k, \mathbf{ch}^o_\alpha[\theta_y]), y : \theta_y \vdash_\alpha \bar{a}_k\langle\lambda\rangle} \quad \dfrac{\pi_3}{\Gamma_3 \vdash_\alpha r}}{\Gamma, \bar{a} : \theta', a : \theta'^\perp \vdash_\alpha (\Pi_{i\in I\cup\{k\}} a_i(\mu_i).q_i \mid \bar{a}_k\langle\lambda\rangle)}}{\Gamma \vdash_\alpha p}$$

Checking that $\Gamma^\natural \vdash p \sqsubset P$ is easy. $\square$

Lemma 7.10. Suppose that $P \xrightarrow{\tau} Q$, $\Gamma \vdash_\alpha q$ and $\Gamma^\natural \vdash q \sqsubset Q$. For all $i \notin \mathrm{dom}(\Gamma(\tau))$, there exists p and β such that $\Gamma \sqcap \tau : (i, \mathbf{ch}^i_\beta[]) \vdash_\alpha p$, $\Gamma^\natural \sqcap \tau : \{i\} \vdash p \sqsubset P$ and $p \xrightarrow{\tau_i} q$.

Proof. Similar to that of the previous lemma. □

We are now ready to prove Proposition 7.6.

Proof of Prop. 7.6. (Proof of 1.) Let us write $\mathbf{ch}^i_{\alpha_i}[]$ for $\Gamma(\tau)(i)$ when $i \in \mathrm{dom}(\Gamma(\tau))$. By the assumption that $p \xrightarrow{\pi} q$, there exists a sequence $p = p_0 \xrightarrow{l_1} \cdots \xrightarrow{l_n} p_n = q$. Let $\tau_{i_1} \ldots \tau_{i_k}$ be the subword of π that is obtained by deleting 0 from π. Without loss of generality, we may assume that $\alpha_{i_1} < \cdots < \alpha_{i_k}$ and $\alpha_{i_k} < \alpha$ for all $\alpha \in \{\alpha_i \mid i \in \mathrm{dom}(\Gamma(\tau))\} \setminus \{\alpha_{i_1}, \ldots, \alpha_{i_k}\}$; if not we can always reannotate the levels appearing in Γ and use that type environment instead of Γ. Now suppose that $l_1 = \tau_{i_1}$. Then we can apply Lemma 7.8 to obtain P_1 such that $P_0 \xrightarrow{\tau} P_1$ and $\Gamma_1^\natural \vdash p_1 \sqsubset P_1$, where Γ_1 is the type environment that satisfy $\Gamma_1 \vdash_{\alpha_{i_1}} p_1$. If $l_1 = 0$ we can use Lemma 7.7 instead. By repeating this argument we obtain a sequence $P = P_0 \xrightarrow{l_1} \cdots \xrightarrow{l_n} P_n = Q$ that can be used to show $(p \xrightarrow{\pi} q) \sqsubset (P \xrightarrow{\pi} Q)$.

(Proof of 2.) Since $P \xrightarrow{\pi} Q$, we have $P = P_0 \xrightarrow{l_1} \cdots \xrightarrow{l_n} P_n = Q$, where $\pi = l_1 \ldots l_n$. Let us consider the case where $l_n = \tau$. In this case we can appeal to Lemma 7.10 (if $l_n = 0$ we use Lemma 7.9). Let i be an index such that $i \notin \mathrm{dom}(\Gamma(\tau))$ and β be a level such that $\beta < \gamma$ for all $\gamma \in \overline{\mathbf{lv}}(\Gamma)$. Then by Lemma 7.10, we have p_{n-1} such that $\Gamma \sqcap \tau : (i, \mathbf{ch}^i_\beta[]) \vdash_\beta p_{n-1}$ and $\Gamma^\natural \sqcap \tau : \{i\} \vdash p_{n-1} \sqsubset P_{n-1}$. By repeating the argument we obtain $p \xrightarrow{\pi} q$ with the desired property. □

7.4 LTS Based on Linear Approximation

Using the notion of linear processes, we introduce a labelled transition system (LTS) for processes to describe the behaviour of processes in which outputs cannot be delayed. Intuitively, the LTS that describes the behaviour of P is given as an LTS whose states are linear approximations of P and transition relation is the extension relation $\trianglelefteq$, which we briefly explained in Section 7.2. This LTS will be presented as a presheaf following the view that presheaves can be regarded as transition systems [23, 135].

7.4.1 Extension relation

We now define an ordering $p' \trianglelefteq p$ over linear processes, which may be read as "p extends p'". Giving a larger linear approximation corresponds to extending an execution sequence.

Before we define the extension relation on linear processes, we define the extension relation over types.

Definition 7.4. Let A be a set of levels. *Restriction of types and intersection types* are inductively defined by:

$$\mathbf{ch}^o_\alpha[\theta_1, \ldots, \theta_n]{\upharpoonright_A} \overset{\text{def}}{=} \begin{cases} \mathbf{ch}^o_\alpha[\theta_1{\upharpoonright_A}, \ldots, \theta_n{\upharpoonright_A}] & (\text{if } \alpha \in A) \\ \bullet & (\text{otherwise}) \end{cases}$$

$$\mathbf{ch}^i_\alpha[\theta_1, \ldots, \theta_n]{\upharpoonright_A} \overset{\text{def}}{=} \begin{cases} \mathbf{ch}^i_\alpha[\theta_1{\upharpoonright_A}, \ldots, \theta_n{\upharpoonright_A}] & (\text{if } \alpha \in A) \\ \bullet & (\text{otherwise}) \end{cases}$$

$$(\theta{\upharpoonright_A})(i) \overset{\text{def}}{=} \theta(i){\upharpoonright_A}.$$

Similarly, *restriction of type isomorphisms* is defined by:

$$\mathbf{ch}^o_\alpha[\varphi_1, \dots, \varphi_n] \overset{\text{def}}{=} \begin{cases} \mathbf{ch}^o_\alpha[\varphi_1\restriction_A, \dots, \varphi_n\restriction_A] & (\text{if } \alpha \in A) \\ \mathrm{id}_\bullet & (\text{otherwise}) \end{cases}$$

$$\mathbf{ch}^i_\alpha[\varphi_1, \dots, \varphi_n] \overset{\text{def}}{=} \begin{cases} \mathbf{ch}^i_\alpha[\varphi_1\restriction_A, \dots, \varphi_n\restriction_A] & (\text{if } \alpha \in A) \\ \mathrm{id}_\bullet & (\text{otherwise}) \end{cases}$$

$$(\sigma, (\varphi_i)_{i \in \mathbf{Nat}})\restriction_A \overset{\text{def}}{=} (\sigma, (\varphi_i\restriction_A)_{i \in \mathbf{Nat}})$$

We write $\rho' <: \rho$ (resp. $\theta' <: \theta$) if $\rho' = \rho\restriction_A$ (resp. $\theta' = \theta\restriction_A$) for some $A \subseteq \mathcal{A}$ and $\varphi' <: \varphi$ if $\varphi' = \varphi\restriction_A$ for some $A \subseteq \mathcal{A}$. ■

For $\rho' <: \rho$, there may be multiple set of levels A that satisfies $\rho = \rho\restriction_A$. We say that A is the *standard witness* of $\rho' <: \rho$ if and only if $A = \bigcap\{A' \subseteq \mathcal{A} \mid \rho' = \rho\restriction_{A'}\}$; the notion of standard witness for intersection types and type isomorphisms are defined similarly.

We list some basic properties of type restriction. These properties are only used in the proofs of the forthcoming propositions and can be ignored for the moment. We recommend the readers to come back when they find the use of these lemmas. All of the lemmas given below are easy to prove; most of the proofs are by straightforward induction on the structure of type or intersection type.

Lemma 7.11.

1. If $\rho\restriction_A = \rho'$ (resp. $\theta\restriction_A = \theta'$) then $\overline{\mathbf{lv}}(\rho') \subseteq A$ (resp. $\overline{\mathbf{lv}}(\theta') \subseteq A$).
2. Let A be the standard witness of $\rho' <: \rho$ (resp. $\theta <: \theta'$). Then $A = \overline{\mathbf{lv}}(\rho')$ (resp. $A = \overline{\mathbf{lv}}(\theta')$).

□

Lemma 7.12. Let $A \subseteq \mathcal{A}$ be a downward-closed set and let ρ be a type. If $\alpha \in A$ and $\alpha \in \overline{\mathbf{lv}}(\rho)$ then $\alpha \in \overline{\mathbf{lv}}(\rho\restriction_A)$. □

Lemma 7.13. If $\varphi\colon \rho \cong \rho'$ (resp. $\varphi\colon \theta \cong \theta'$) then $\varphi\restriction_A\colon \rho\restriction_A \cong \rho'\restriction_A$ (resp. $\varphi\restriction_A\colon \theta\restriction_A \cong \theta'\restriction_A$). □

Lemma 7.14.

1. Let $\rho_1 = \rho_1'\restriction_A$ and $\varphi\colon \rho_1 \cong \rho_2$. Then there exist a type ρ_2' and a type isomorphism $\varphi'\colon \rho_1' \cong \rho_2'$ such that $\rho_2 = \rho_2'\restriction_A$ and $\varphi = \varphi_2'\restriction_A$.
2. Let $\theta_1 = \theta_1'\restriction_A$ and $\varphi\colon \theta_1 \cong \theta_2$. Then there exist an intersection type θ_2' and a type isomorphism $\varphi'\colon \theta_1' \cong \theta_2'$ such that $\theta_2 = \theta_2'\restriction_A$ and $\varphi = \varphi_2'\restriction_A$.

□

The *extension relation* on linear processes, written $p' \trianglelefteq p$, is inductively defined by the rules in Figure 7.4. For example, $a_1(\langle\rangle).\bot \trianglelefteq a_1(\langle x_1 \rangle).\tau_1.x_1\langle\rangle$ holds and this intuitively means that $!a(x).x\langle\rangle \xrightarrow{a(x)} !a(x).x\langle\rangle \mid x\langle\rangle$ can be extended to $!a(x).x\langle\rangle \xrightarrow{a(x)} !a(x).x\langle\rangle \mid x\langle\rangle \xrightarrow{x\langle\rangle} !a(x).x\langle\rangle \mid \mathbf{0}$ (under the assumption that both of the linear process approximates $!a(x).x\langle\rangle$).

Extending a linear process does not precisely correspond to extending an execution sequence: there are cases where an execution sequence cannot be extended even if the corresponding linear process can be extended. This problem

$$\frac{I = \{i_1 < \cdots < i_m\} \qquad J = \{j_1 < \cdots < j_n\} \qquad J \subseteq I \qquad \varphi'_i <: \varphi_i \quad (\text{for } i \in J)}{\langle \varphi'_{j_1} \mathbin{\scriptstyle\blacksquare} x_{j_1}, \dots, \varphi'_{j_n} \mathbin{\scriptstyle\blacksquare} x_{j_n} \rangle \trianglelefteq \langle \varphi_{i_1} \mathbin{\scriptstyle\blacksquare} x_{i_1}, \dots, \varphi_{i_m} \mathbin{\scriptstyle\blacksquare} x_{i_m} \rangle}$$

$$\frac{}{\mathbf{0} \trianglelefteq \mathbf{0}} \qquad \frac{}{\bot \trianglelefteq \tau_i.p} \qquad \frac{p \trianglelefteq q}{\tau_i.p \trianglelefteq \tau_i.q} \qquad \frac{J \subseteq I \qquad \rho'_i <: \rho_i \quad (\text{for } i \in J) \qquad p \trianglelefteq q}{(\boldsymbol{\nu}[\langle \bar{a}_i, a_i \rangle_{\rho'_i}]_{i \in J})q \trianglelefteq (\boldsymbol{\nu}[\langle \bar{a}_i, a_i \rangle_{\rho_i}]_{i \in I})p}$$

$$\frac{\lambda'_i \trianglelefteq \lambda_i \quad (\text{for } i \in [n])}{\bar{a}_i\langle \lambda'_1, \dots, \lambda'_n \rangle \trianglelefteq \bar{a}_i\langle \lambda_1, \dots, \lambda_n \rangle} \qquad \frac{p \trianglelefteq q \qquad \mu'_i \trianglelefteq \mu_i \quad (\text{for } i \in [n])}{a_i(\mu'_1, \dots, \mu'_n).p \trianglelefteq a_i(\mu_1, \dots, \mu_n).q}$$

$$\frac{}{\bot \trianglelefteq a_i(\vec{\mu}).q} \qquad \frac{p' \trianglelefteq p \qquad q' \trianglelefteq q}{p' \mid q' \trianglelefteq p \mid q} \qquad \frac{m \leq n \qquad p'_i \trianglelefteq p_i \quad p'_i \neq \bot \quad (\text{for } i \in [m])}{p'_1 \parallel \cdots \parallel p'_m \trianglelefteq p_1 \parallel \cdots \parallel p_n}$$

Figure 7.4: Rules for extension relation. Here we identify processes up to $\equiv_0$.

is due to the existence of deadlocks. For instance, we have $(\boldsymbol{\nu}[])(\boldsymbol{\nu}[])(\bot \mid \bot) \trianglelefteq (\boldsymbol{\nu}[\langle \bar{a}_1, a_1 \rangle])(\boldsymbol{\nu}[\langle \bar{b}_1, b_1 \rangle])(a_1.\tau_1.\bar{b}_1 \mid b_1.\tau_2.\bar{a}_1)$, but both of the linear processes are not reducible. To exclude linear processes that may create a deadlock, we introduce the notion of terminable processes:

Definition 7.5. A linear process p is *idle* if it has no action (input, output nor τ), i.e. consisting of 0, $\bot$, $\mid$ and $\boldsymbol{\nu}[]$. A linear process p is *terminable* if $(\boldsymbol{\nu}\vec{\xi}).(p \mid q) \xrightarrow{0}{}^* r$ for some $\vec{\xi}$, q and idle r. ■

Only terminable processes will be used as the states of the LTS.

In case p and p' correspond to executions that only consists of $\xrightarrow{0}$ and $\xrightarrow{\tau}$ the intuition that $p \trianglelefteq p'$ corresponds to "extending execution sequences" can be formalised as follows:

Proposition 7.15. Let $\tau : \mathbf{ch}^i[] \vdash P$ and let $\mathcal{R}$ be a relation between execution sequences starting from P and well-typed terminable linear approximations of P such that $(P \xrightarrow{\pi} Q) \mathrel{\mathcal{R}} p$ if and only if $(p \xrightarrow{\pi} q) \sqsubset (P \xrightarrow{\pi} Q)$ for a process q that is typed under the empty environment. Then if $(P \xrightarrow{\pi} Q) \mathrel{\mathcal{R}} p$

1. $Q \xrightarrow{\pi'} Q'$ implies that $(P \xrightarrow{\pi} Q \xrightarrow{\pi'} Q') \mathrel{\mathcal{R}} p'$ and $p \trianglelefteq p'$ for some p'.
2. if $p \trianglelefteq p'$ for some terminable well-typed linear process p' that approximates P, then there is an execution $Q \xrightarrow{\pi'} Q'$ such that $(P \xrightarrow{\pi} Q \xrightarrow{\pi'} Q') \mathrel{\mathcal{R}} p'$.

Lemmas used to prove Proposition 7.15 is given below. The first lemma says that substitution $\{\varphi \mathbin{\scriptstyle\blacksquare} y/x\}$ is monotonic with respect to $\trianglelefteq$. The second and the third lemmas are variants of the subject reduction lemmas and the last two lemmas are variants of the subject expansion lemmas; the monotonicity of substitution is used to prove the second and the fourth lemma. If we ignore the details, the first two lemmas say that if $p \trianglelefteq p'$ and $p \xrightarrow{l} q$ then $p' \xrightarrow{l} q'$ for some q' such that $q \trianglelefteq q'$. We omit the proofs for these lemmas because they are similar to that of the subject reduction/expansion lemmas.

Lemma 7.16 (Monotonicity of substitution)**.** Suppose that $q \trianglelefteq p$, $\varphi' <: \varphi$ and assume that p, q, $p\{\varphi \mathbin{\scriptstyle\blacksquare} y_j/x_i\}$ and $q\{\varphi' \mathbin{\scriptstyle\blacksquare} y_j/x_i\}$ are appropriately typed. Then $q\{\varphi' \mathbin{\scriptstyle\blacksquare} y_j/x_i\} \trianglelefteq p\{\varphi \mathbin{\scriptstyle\blacksquare} y_j/x_i\}$.

Proof. By induction on the structure of p. □

Lemma 7.17. Let $\Gamma \vdash_\alpha p$ and $\Gamma \vdash_\alpha q$ be well-typed linear processes such that $p \xrightarrow{0} q$, $\Gamma^\natural \vdash p \sqsubset P$ and $\Gamma^\natural \vdash q \sqsubset Q$. Suppose that there exists p' such that $\Gamma' \vdash_\beta p'$, $(\Gamma')^\natural \vdash p' \sqsubset P$. Then there is a linear process q' such that $p' \xrightarrow{0} q'$, $\Gamma' \vdash_\beta q'$ and $(\Gamma')^\natural \vdash q' \sqsubset Q$. $\square$

Lemma 7.18. Let $\Gamma \sqcap \tau : (i, \mathbf{ch}^i_\beta) \vdash_\alpha p$ and $\Gamma \vdash_\alpha q$ be well-typed linear processes such that $p \xrightarrow{\tau_i} q$, $\Gamma^\natural \sqcap \tau : \{i\} \vdash p \sqsubset P$ and $\Gamma^\natural \vdash q \sqsubset Q$. Suppose that there exists p' such that $\Gamma' \sqcap \tau : (i, \mathbf{ch}^i_\delta[]) \vdash_\gamma p'$, $(\Gamma')^\natural \sqcap \tau : \{i\} \vdash p' \sqsubset P$. Then there is a linear process q' such that $p' \xrightarrow{\tau_i} q'$, $\Gamma' \vdash_\gamma q'$ and $(\Gamma')^\natural \vdash q' \sqsubset Q$. $\square$

Lemma 7.19. Let $\Gamma \vdash_\alpha p$ and $\Gamma \vdash_\alpha q$ be processes such that $p \xrightarrow{0} q$, $\Gamma^\natural \vdash p \sqsubset P$ and $\Gamma^\natural \vdash q \sqsubset Q$. Assume that there exists q' such that $q \trianglelefteq q'$, $\Gamma' \vdash_\beta q'$ and $(\Gamma')^\natural \vdash q' \sqsubset Q$, where $\Gamma(t)(i) <: \Gamma'(t)(i)$ for all term t and index i. Then there exists p' such that $p \trianglelefteq p'$, $\Gamma' \vdash_\beta p'$, $(\Gamma')^\natural \vdash p' \sqsubset P$ and $p' \xrightarrow{0} q'$. $\square$

Lemma 7.20. Let $\Gamma \sqcap \tau : (i, \mathbf{ch}^i_\beta[]) \vdash_\alpha p$ and $\Gamma \vdash_\beta q$ be processes such that $p \xrightarrow{\tau_i} q$, $\Gamma^\natural \sqcap \tau : \{i\} \vdash p \sqsubset P$ and $\Gamma^\natural \vdash q \sqsubset Q$. Assume that there exists q' such that $q \trianglelefteq q'$, $\Gamma' \vdash_\gamma q'$ and $(\Gamma')^\natural \vdash q'$, where $\Gamma(t)(i) <: \Gamma'(t)(t)$, for all term t and index i, and $\Gamma'(\tau)(i) = \bullet$. Then there exists p' such that $\Gamma' \sqcap \tau : (i, \mathbf{ch}^i_\gamma[]) \vdash_\gamma p'$, $(\Gamma')^\natural \sqcap \tau : \{i\} \vdash p' \sqsubset P$ and $p' \xrightarrow{\tau_i} q'$. $\square$

Using these lemmas, we prove that there is a bisimulation between execution sequences and the extension relation.

Proof. (Proof of Proposition 7.15)

(Proof of 1.) By the assumption that $(p \xrightarrow{\pi} q) \sqsubset (P \xrightarrow{\pi} Q)$ and $P \xrightarrow{\pi} Q \xrightarrow{\pi'} Q'$, we have

$$\begin{array}{ccccccccccc}
P & = & Q_0 & \xrightarrow{l_1} & Q_1 & \xrightarrow{l_2} \cdots \xrightarrow{l_n} & Q_n & = & Q & \xrightarrow{\pi'} & Q' \\
\sqcup & & \sqcup & & \sqcup & & \sqcup & & \sqcup & & \\
p & = & q_0 & \xrightarrow{l_1} & q_1 & \xrightarrow{l_2} \cdots \xrightarrow{l_n} & q_n & = & q & &
\end{array}$$

where $\pi = l_1 \dots l_m$ and $\pi' = l'_1 \dots l'_n$. Let q' be a linear process that approximates Q' and is typed under the empty environment. By Proposition 7.6, there exists a sequence $q'_n \xrightarrow{\pi} q'$ such that $(q'_n \xrightarrow{\pi} q') \sqsubset (Q \xrightarrow{\pi'} Q')$. Since q'_n is a linear process that approximates $Q(= Q_n)$ and q_n is the minimum approximation (with respect to $\trianglelefteq$), we have $q_n \trianglelefteq q'_n$. Now let q_{n-1} be the process that approximates Q_{n-1} and appears in the execution $p \xrightarrow{\pi} q$. Then, by Lemma 7.19 (or Lemma 7.20) there is a well-typed linear process q'_{n-1} that approximates Q_{n-1} such that $q'_{n-1} \xrightarrow{l_n} q'_n$ and $q_{n-1} \trianglelefteq q'_{n-1}$. By applying Lemma 7.19 (or 7.20) repeatedly along the reduction sequence $p \xrightarrow{\pi} q$ (in reverse order), we obtain a well-typed terminable linear process p' that approximates P and satisfies $p' \trianglelefteq p$ and $(p' \xrightarrow{\pi\pi'} q') \sqsubset (P \xrightarrow{\pi} Q \xrightarrow{\pi'} Q')$.

(Proof of 2.) By the assumption $(p \xrightarrow{\pi} q) \sqsubset (P \xrightarrow{\pi} Q)$, we have

$$\begin{array}{ccccccccc}
P & = & Q_0 & \xrightarrow{l_1} & Q_1 & \xrightarrow{l_2} \cdots \xrightarrow{l_n} & Q_n & = & Q \\
\sqcup & & \sqcup & & \sqcup & & \sqcup & & \sqcup \\
p & = & q_0 & \xrightarrow{l_1} & q_1 & \xrightarrow{l_2} \cdots \xrightarrow{l_n} & q_n & = & q
\end{array}$$

where $\pi = l_1 \dots l_n$. By the assumption we also have a well-typed terminable linear process p' such that $p \trianglelefteq p'$. From Lemma 7.17 (or Lemma 7.18) we obtain

q_1' such that $p' \xrightarrow{l_1} q_1'$ and $q_1 \trianglelefteq q_1'$. By repeating this argument we obtain $(p' \xrightarrow{\pi} q') \sqsubset (P \xrightarrow{\pi} Q)$ for some q' such that $q \trianglelefteq q'$. Since p' is terminable and p' is closed it must reduce to an idle process.Therefore, we have $q' \xrightarrow{\pi'} r$ for an idle r, and by the definition of idle process r is typed under the empty environment. Since the reduction of the linear process induces that of (non-linear) processes, i.e. by Lemma 7.6 there exists an execution sequence, $Q \xrightarrow{\pi'} R$ that satisfy $(p' \xrightarrow{\pi} q' \xrightarrow{\pi'} r) \sqsubset (P \xrightarrow{\pi} Q \xrightarrow{\pi'} R)$ as desired. □

Remark 7.2. Let $\mathcal{R}$ be the relation defined in Proposition 7.15 and let $(P \xrightarrow{\pi} Q)\ \mathcal{R}\ p$. When we take a larger terminable linear process p', we can always extend $P \xrightarrow{\pi} Q$ according to p' until it reaches a settled process, i.e. there exists Q' that is settled and $P \xrightarrow{\pi} Q \xrightarrow{\pi'} Q'\ \mathcal{R}\ p'$. This is because p' can be reduced until it becomes an idle process q' and if $\emptyset \vdash q' \sqsubset Q'$ then Q' must be idle. (Recall that an unguarded output cannot be replaced by $\perp$.) ■

7.4.2 Presheaf semantics

We define the LTS that describes the behaviour of $\Delta \vdash P$ as a presheaf $[\![P]\!] : \mathcal{E}_\Delta \to$ **Sets**. Roughly speaking, the path category $\mathcal{E}_\Delta$ is a category whose objects are type environments and morphisms are extension relations and $[\![P]\!]$ maps a type environment Γ to the set of approximations of P that is typed under Γ.

Actually, the objects of the path category are not only type environments, but the pair of type environments and the "current time".

Definition 7.6. We say that (Γ, α) *extends* (Γ', α) and write $(\Gamma', \alpha) <: (\Gamma, \alpha)$ if there exists a *witness* $A \subseteq \overline{\mathbf{lv}}(\Gamma) \cup \{\alpha\}$ that satisfies

1. $\mathrm{dom}(\Gamma') \subseteq \mathrm{dom}(\Gamma)$ and $\Gamma'(t) = \Gamma(t){\restriction_A}$, for $t \in \mathrm{dom}(\Gamma)$,
2. $\alpha \in A$ and
3. A is downward-closed: for every β, γ in $\overline{\mathbf{lv}}(\Gamma)$, $\beta \leq \gamma$ and $\gamma \in A$ implies $\beta \in A$.

■

We define the *category of type environments* $\mathcal{E}_\Delta$ to be a category whose objects are (Γ, α) such that $\Gamma \sqsubset \Delta$ and whose morphisms are given by the relation $(\Gamma', \alpha) <: (\Gamma, \alpha)$.

Now we define the presheaf $[\![P]\!]$. Given $\Delta \vdash P$ and $\Gamma \sqsubset \Delta$, the set $[\![P]\!](\Gamma, \alpha)$ is defined by $[\![P]\!](\Gamma, \alpha) \overset{\text{def}}{=} \{p \mid \Gamma^\natural \vdash p \sqsubset P, \Gamma \vdash_\alpha p \text{ and } p \text{ is terminable}\}$. (Here we are identifying linear processes up to $\equiv_0$.)

The key lemma to construct a presheaf is given below. The proof of this lemma will be given after we have defined the presheaf.

Lemma 7.21. Assume that $\Gamma \vdash_\alpha p$, p is terminable and $(\Gamma', \alpha) <: (\Gamma, \alpha)$. Then there is a unique (up to $\equiv_0$) linear process that satisfy $q \trianglelefteq p$ and $\Gamma' \vdash_\alpha q$.

By Lemma 7.21 there is a map $[\![P]\!](-,-)$ that maps an extension relation $(\Gamma', \alpha) <: (\Gamma, \alpha)$ to a function from $[\![P]\!](\Gamma, \alpha)$ to $[\![P]\!](\Gamma', \alpha)$ that maps $p \in [\![P]\!](\Gamma, \alpha)$ to q such that $q \trianglelefteq p$ and $\Gamma' \vdash_\alpha q$; we will write $p{\restriction_{\Gamma', \alpha}}$ for the unique process q.

Theorem 7.22. Let $\Delta \vdash P$. Then $[\![P]\!](-,-)$ is a functor from $\mathcal{E}_\Delta$ to **Sets**.

Proof. We need to check that $[\![P]\!](-,-)$ preserves identities and composition. The fact that $[\![P]\!](-,-)$ preserves identities follows from Lemma 7.21.

So it remains to show that $[\![P]\!](-,-)$ preserves composition. This also follows from Lemma 7.21. Suppose that $(\Gamma_1,\alpha) <: (\Gamma_2,\alpha) <: (\Gamma_3,\alpha)$. Let $p \in [\![P]\!](\Gamma_3,\alpha)$ and let us write f for $[\![P]\!]((\Gamma_2,\alpha) <: (\Gamma_3,\alpha))$, g for $[\![P]\!]((\Gamma_1,\alpha) <: (\Gamma_2,\alpha))$ and h for $[\![P]\!]((\Gamma_1,\alpha) <: (\Gamma_3,\alpha))$. Then we have $g(f(p)) \trianglelefteq f(p)$ and $f(p) \trianglelefteq p$, and hence $g(f(p)) \trianglelefteq p$ because $\trianglelefteq$ is transitive. We also have $h(p) \trianglelefteq p$. Since $g(f(p))$ and $h(p)$ are both typed under Γ_1, by Lemma 7.21, we have $g(f(p)) \equiv_0 h(p)$ as desired. □

We now prove the key lemma (Lemma 7.21). The proof of the lemma proceeds by induction on the structure of the derivation of $\Gamma \vdash_\alpha p$.

The nontrivial case is the case of ν-restriction and to handle this case we use the following lemmas.

Lemma 7.23. Suppose that $\Gamma \vdash_\alpha (\boldsymbol{\nu}[\langle \bar{a}_i, a_i\rangle]_{i\in\mathrm{dom}(\theta)})p$ and that $(\boldsymbol{\nu}[\langle \bar{a}_i, a_i\rangle]_{i\in\mathrm{dom}(\theta)})p$ is terminable. Then $\overline{\mathbf{lv}}(\theta) \subseteq \overline{\mathbf{lv}}(\Gamma) \cup \{\alpha\}$. □

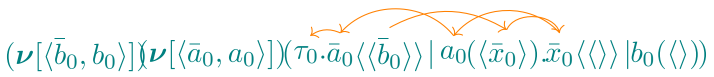
Lemma 7.24. Let $(\boldsymbol{\nu}[\langle \bar{a}_i, a_i\rangle]_{i\in\mathrm{dom}(\theta)})p$ be a terminable process such that $\Gamma \vdash_\alpha (\boldsymbol{\nu}[\langle \bar{a}_i, a_i\rangle]_{i\in\mathrm{dom}(\theta)})p$. Suppose that $(\boldsymbol{\nu}[\langle \bar{a}_i, a_i\rangle]_{i\in\mathrm{dom}(\theta')})q \trianglelefteq (\boldsymbol{\nu}[\langle \bar{a}_i, a_i\rangle]_{i\in\mathrm{dom}(\theta)})p$ and $\Gamma' \vdash_\alpha (\boldsymbol{\nu}[\langle \bar{a}_i, a_i\rangle]_{i\in\mathrm{dom}(\theta')})q$, where Γ' satisfies $\Gamma'(t)(i) <: \Gamma(t)(i)$ for all term t and index i.If there is a level β such that $\beta \in \overline{\mathbf{lv}}(\theta)$ but $\beta \notin \overline{\mathbf{lv}}(\theta')$, then there is a term t and an index i such that $\beta \in \Gamma(t)(i)$ and $\beta \notin \Gamma'(t)(i)$.

Instead of giving a detailed proof of these lemmas, we explain why this holds by giving an example.

Example 7.1. Let us consider a well typed linear process

$$\tau : (0, \mathbf{ch}^i_\beta[]), \vdash_\gamma (\boldsymbol{\nu}[\langle \bar{b}_0, b_0\rangle_{\rho_b}])(\boldsymbol{\nu}[\langle \bar{a}_0, a_0\rangle_{\rho_a}])(\tau_0.\bar{a}_0\langle\langle b_0\rangle\rangle \mid a_0(\langle \bar{x}_0\rangle).\bar{x}_0\langle\langle\rangle\rangle \mid b_0(\langle\rangle))$$

where $\rho_b = \mathbf{ch}^o_\beta[]$ and $\rho_a = \mathbf{ch}^o_\alpha[(0,\rho_\beta)]$. The following figure shows the way to point a free name (or a constant τ_i) whose type contains the level $\beta \in \overline{\mathbf{lv}}(\rho_b)$. (In this case we can tell that the type for τ_0 contains β.)


$(\boldsymbol{\nu}[\langle \bar{b}_0, b_0\rangle])(\boldsymbol{\nu}[\langle \bar{a}_0, a_0\rangle])(\tau_0.\bar{a}_0\langle\langle \bar{b}_0\rangle\rangle \mid a_0(\langle \bar{x}_0\rangle).\bar{x}_0\langle\langle\rangle\rangle \mid b_0(\langle\rangle))$

Let us explain what the pointers mean. A pointer points to a name that must be "executed at the same time" with the name placed at the source of the pointer. We start from $\bar{b}_0$ because that is the name with type ρ_b. Since $\bar{b}_0$ is in an object position of an output via the name $\bar{a}_0$ and $\bar{a}_0$ is bound, we first look for the name that communicates with $\bar{a}_0$, which is a_0 in this case. Because $\bar{x}_0$ is the argument that corresponds to $\bar{b}_0$, the type for $\bar{x}_0$ must have the level β for its "outermost level". So now we have another name $\bar{x}_0$ whose type has β as the "outermost level", and the link from $\bar{b}_0$ to $\bar{x}_0$ is used to expresses this fact. Now we look for the place where $\bar{x}_0$ is actually used, this is expressed by the second link. Since $\bar{x}_0$ is guarded by a_0 we now know that a_0 must be executed at the same time as $\bar{x}_0$. Because $\bar{a}_0$ communicates with a_0, we know that $\bar{a}_0$ and a_0 must be executed simultaneously and thus we have a pointer from a_0 to $\bar{a}_0$. The output $\bar{a}_0$ is guarded by τ_0, so we know that τ_0 also happens at the same time. Because τ_0 is a constant we conclude that β appears in the type environment.

Lemma 7.23 can be proved by formalising the notion of pointer and generalising the above procedure.

Lemma 7.24 can be proved by showing that $\trianglelefteq$ does not create any "dangling pointer". That is if $q \trianglelefteq p$ and linear terms t_j, t_i appearing in p are linked by a pointer, then either t_j and t_i both appears in q or t_j and t_i do not appear in q. This follows from the definition of $\trianglelefteq$ and the way we add pointers. For example, let us consider the case where p is the linear process depicted above. The only process q such that $\bar{b}_0$ does not appear in q and $q \trianglelefteq p$ is $(\boldsymbol{\nu}[])(\boldsymbol{\nu}[])(\bot \mid \bot \mid \bot)$. ■

Now we are ready to prove the key lemma.

Proof of Lemma 7.21. Let A be the smallest witness of $(\Gamma', \alpha) <: (\Gamma, \alpha)$. The proof proceeds by induction on the derivation of $\Gamma \vdash_\alpha p$.

(Case (TNIL)**)** Trivial.

(Case (TOUT)**)** In this case, $\Gamma \vdash_\alpha p$ is of the form $x^1 : \theta_1 \sqcap \cdots \sqcap x^n : \theta_n, \bar{a} : (i, \mathbf{ch}^o_\alpha[\theta'_1, \ldots, \theta'_n]) \vdash_\alpha \bar{a}_i\langle\lambda_1, \ldots, \lambda_n\rangle$, where $\lambda_i = \varphi_i \bullet x^i$ for some type isomorphim $\varphi_i \colon \theta_i \cong \theta'_i$ for each $i \in \{1, \ldots, n\}$. Recall that by the definition of witness, we have $\alpha \in A$. Therefore, Γ' is $x^1 : \theta_1{\restriction_A} \sqcap \cdots \sqcap x^n : \theta_n{\restriction_A}, \bar{a} : (i, \mathbf{ch}^o_\alpha[\theta'_1{\restriction_A}, \ldots, \theta'_n{\restriction_A}])$. We also have $\varphi_i{\restriction_A} \colon \theta_1{\restriction_A} \cong \theta'{\restriction_A}$ by Lemma 7.13. So we have $\Gamma' \vdash_\alpha \bar{a}_i\langle\lambda'_1, \ldots, \lambda'_2\rangle$, where $\lambda'_i = \varphi_i{\restriction_A} \bullet x^i$. Since $\lambda'_i \trianglelefteq \lambda_i$ by the definition of $\varphi_i{\restriction_A}$ and $\varphi_i{\restriction_A} \bullet x^i$, we can take $\bar{a}_i\langle\lambda'_1, \ldots, \lambda'_2\rangle$ as q.

To show the uniqueness of q it suffices to show that

$$\text{if } \varphi'_i <: \varphi_i \text{ and } \varphi'_i \colon \theta_i{\restriction_A} \cong \theta'_i{\restriction_A} \text{ then } \varphi'_i = \varphi_i{\restriction_A}.$$

This can be shown by a straightforward induction on the structure of φ_i.

(Case (TIN)**)** The derivation for $\Gamma \vdash_\alpha p$ must be of the following form

$$\frac{\Gamma_0, x^1 : \theta_1, \ldots, x^n : \theta_n \vdash_\beta p_0}{\Gamma_0 \sqcap b : (i, \mathbf{ch}^i_\beta[\theta_1, \ldots, \theta_n]) \vdash_\alpha b_i(\mu_1, \ldots, \mu_n).p_0}$$

where $\mu_i = id_{\theta_i} \bullet x^i$ for $i \in \{1, \ldots, n\}$ and $\alpha \leq \beta$. Let us consider the case where $\beta \in A$; the other case is easy.

We first show the existence of q. Let Γ'_0 be the type environment that satisfies $\Gamma'_0 \sqcap b : (i, \mathbf{ch}^o_\beta[\theta_1, \ldots, \theta_n]{\restriction_A}) = \Gamma'$. Then we have $A \colon (\Gamma'_0, x^1 : \theta_1{\restriction_A}, \ldots, x^n : \theta_n{\restriction_A}, \beta) <: (\Gamma_0, x^1 : \theta_1, \ldots, x^n : \theta_n, \beta)$. Hence, by the induction hypothesis, there exists q_0 such that $\Gamma'_0, x^1 : \theta_1{\restriction_A}, \ldots, x^n : \theta_n{\restriction_A} \vdash_\beta q_0$ and $q_0 \trianglelefteq p_0$. By applying (TIN), we have $\Gamma' \vdash_\alpha b_i(\mu'_1, \ldots, \mu'_n).q_0$, where $\mu'_i = \mathrm{id}_{\theta_i{\restriction_A}} \bullet x^i$, and we can take this process as q.

We now prove the uniqueness of q. Assume that there exists q' such that $\Gamma' \vdash_\alpha q'$ and $q' \trianglelefteq p$. Then by the definition of $\trianglelefteq$, q' must be of the form $b_i(\mu''_1, \ldots, \mu''_n).q'_0$ with $q'_0 \trianglelefteq p_0$. Therefore the last rule applied to the derivation $\Gamma' \vdash_\beta q'$ must be (TIN) and the derivation must be the following form

$$\frac{\Gamma_0, x^1 : \theta_1{\restriction_A}, \ldots, x^n : \theta_n{\restriction_A} \vdash_\beta q'_0}{\Gamma'_0 \sqcap b : (i, \mathbf{ch}^i_\beta[\theta_1{\restriction_A}, \ldots, \theta_n{\restriction_A}]) \vdash_\alpha b_i(\mu''_1, \ldots, \mu''_n).q'_0}$$

where $\mu''_i = \mathrm{id}_{\theta_i{\restriction_A}} \bullet x^i = \mu'_i$ for all $i \in \{1, \ldots, n\}$. By the uniqueness of q_0 we have $q_0 \equiv_0 q'_0$, and thus $q' \equiv_0 q$ as desired.

(Case (TTAU)**)** Similar to the case for (TIN).

(Case (TPar)**)** In this case, the derivation for $\Gamma \vdash_\alpha p$ is of the following form:

$$\frac{\Gamma_1 \vdash_\alpha p_1 \qquad \Gamma_2 \vdash_\alpha p_2}{\Gamma_1 \sqcap \Gamma_2 \vdash_\alpha p_1 \mid p_2}$$

Let Γ'_i be a type environment defined by $\Gamma'_i(t) \overset{\text{def}}{=} \Gamma_i(t)\restriction_A$ for $i \in \{1,2\}$. Then we have $A\colon (\Gamma'_i, \alpha) <: (\Gamma_i, \alpha)$ for $i \in \{1,2\}$. By the induction hypothesis there exists q_i such that $\Gamma'_i \vdash_\alpha q_i$ and $q_i \trianglelefteq p_i$. Since $\Gamma' = \Gamma'_1 \sqcap \Gamma'_2$ we can take $q_1 \mid q_2$ as q.

For the uniqueness part, assume that there exists q' such that $\Gamma' \vdash_\alpha q'$ and $q' \trianglelefteq p$. Then by the definition of $\trianglelefteq$, $q' = q'_1 \mid q'_2$ for some q'_1 and q'_2 such that $q_i \trianglelefteq p_i$ $(i \in \{1,2\})$. The type derivation for $\Gamma' \vdash_\alpha q'_1 \mid q'_2$ must have the form

$$\frac{\Gamma''_1 \vdash_\alpha q'_1 \qquad \Gamma''_2 \vdash_\alpha q'_2}{\Gamma' \vdash_\alpha q'_1 \mid q'_2}$$

where $\Gamma' = \Gamma''_1 \sqcap \Gamma''_2$. It suffices to show that $\Gamma''_i = \Gamma'_i$ for $i \in \{1,2\}$ because then we can conclude that $q'_i \equiv_0 q_i$ by the induction hypothesis. Since $\Gamma''_i(t)(j) \neq \bullet$ iff $t_j \in \mathbf{fn}(q'_i) \subseteq \mathbf{fn}(p_i)$ and $\Gamma''_i(t)(j) = \Gamma'(t)(j) = \Gamma(t)(j)\restriction_A$ when $\Gamma''_i(t)(j) \neq \bullet$, we have $\Gamma'_i(t)(j) = \Gamma''_i(t)(j)$. We therefore conclude that $q' \equiv_0 q$.

(Case (TRep)**)** Similar to the case for (TPar).

(Case (TNu)**)** In this case the derivation must have the following form:

$$\frac{\Gamma, \bar{a} : \theta, a : \theta^\perp \vdash_\alpha p_0}{\Gamma \vdash_\alpha (\boldsymbol{\nu}[\langle \bar{a}_i, a_i \rangle]_{i \in \mathrm{dom}(\theta)}) p_0}$$

By Lemma 7.23, we have $A \subseteq \overline{\mathbf{lv}}(\Gamma, \bar{a} : \theta, a : \theta^\perp)$ and thus $A\colon (\Gamma', \bar{a} : \theta\restriction_A, a : \theta^\perp\restriction_A, \alpha) <: (\Gamma, \bar{a} : \theta, a : \theta^\perp, \alpha)$. By the induction hypothesis, there exists a terminable q_0 such that $q_0 \trianglelefteq p_0$ and $\Gamma', \bar{a} : \theta\restriction_A, a : \theta^\perp\restriction_A \vdash_\alpha q_0$. Thus we can take $(\boldsymbol{\nu}[\langle \bar{a}_i, a_i \rangle]_{i \in \mathrm{dom}(\theta\restriction_A)}) q_0$ as q.

Now we show the uniqueness of q. Since $q \trianglelefteq p$, the derivation of $\Gamma' \vdash_\alpha q$ is of the following form

$$\frac{\Gamma', \bar{a} : \theta', a : \theta'^\perp \vdash_\alpha q'_0}{\Gamma' \vdash_\alpha (\boldsymbol{\nu}[\langle \bar{a}_i, a_i \rangle]_{i \in \mathrm{dom}(\theta')}) q'_0}$$

Our goal is to show that θ' must be $\theta\restriction_A$. Then we can conclude the proof by applying the induction hypothesis. By the definition of $\trianglelefteq$, we have $\theta'(i) <: \theta(i)$ for each $i \in \mathrm{dom}(\theta') \subseteq \mathrm{dom}(\theta)$. Let B_i be the standard witness of $\theta'(i) <: \theta(i)$ and let A_i be the standard witness of $(\theta\restriction_A)(i) <: \theta(i)$ for each $i \in \mathrm{dom}(\theta)$. It suffices to show that $A_i = B_i$. We first show that $B_i \subseteq A_i$. Take $\beta \in B_i$. Note that we have $\beta \in \overline{\mathbf{lv}}(\theta'(i)) \subseteq \overline{\mathbf{lv}}(\theta(i))$ by Lemma 7.11. By Lemma 7.23, we have $\beta \in \overline{\mathbf{lv}}(\Gamma')$. Since A is the smallest set such that $A\colon (\Gamma', \alpha) <: (\Gamma, \alpha)$, we have $\beta \in A$. Because A is downward-closed, we have $\beta \in \overline{\mathbf{lv}}(\theta(i)\restriction_A)$ by Lemma 7.12. Hence, $\beta \in A_i$. Now we show $A_i \subseteq B_i$. Assume that there exists $\beta \in A_i$ such that $\beta \notin B_i$; we show that this leads to a contradiction. Since $\beta \notin B_i$, we have $\beta \notin \overline{\mathbf{lv}}(\theta'(i))$ by Lemma 7.11. We also have $\beta \in \overline{\mathbf{lv}}(\theta(i))$ because $\beta \in A_i$. This means that there exists a term t and index j such that $\beta \in \overline{\mathbf{lv}}(\Gamma(t)(j))$, but $\beta \notin$

$(\overline{\mathbf{lv}}\Gamma'(t)(j))$ by Lemma 7.24. However, this is a contradiction because by applying Lemma 7.12 to the fact that $\Gamma'(t)(j) = \Gamma(t)(j)\restriction_A$, $\beta \in A$ and $\beta \in \overline{\mathbf{lv}}(\Gamma)(t)(j)$, we have $\beta \in \overline{\mathbf{lv}}(\Gamma'(t)(j))$.

□

7.5 Proof of the Main Theorem

Now that we have set up all the technical tools, we are ready to prove the main theorem (Theorem 7.1), which says that $\simeq^c_\tau$ is a π_F-theory.

7.5.1 Main line

Here we prove the main theorem modulo some propositions that will be proved in the subsequent subsection.

To prove Theorem 7.1, it suffices to show that (i) $\simeq^c_\tau$ satisfies the axioms such as (E-Eta) and (E-Beta) (defined in Chapter 3), and (ii) that barbed congruence is a congruence relation, which trivially holds. Instead of directly proving (i), we define a yet another equivalence $\sim$ and show that $\sim$ is a congruence relation that satisfies the axioms and $\sim \subseteq \simeq^c_\tau$. These are relatively easier to show than to directly prove (i). The main difficulty of directly proving (i) is that we need to deal with arbitrary context. By introducing $\sim$, we can avoid this difficulty.

The equivalence $\sim$ is defined using the notion of open map bisimulation [61]. We write $P \sim Q$ if and only if $[\![P]\!]$ and $[\![Q]\!]$ are open map bisimilar. [5] This means that there is a span $[\![P]\!] \xleftarrow{f} X \xrightarrow{g} [\![Q]\!]$, where f and g are open maps. A map $f\colon X \to Y$ is an *open map* if and only if whenever, for $m\colon y(\Gamma_1,\alpha_1) \to y(\Gamma,\alpha_2)$, the square

$$\begin{tikzcd}
y(\Gamma_1,\alpha_1) \arrow[r, "p"] \arrow[d, "m"] & X \arrow[d, "f"] \\
y(\Gamma_2,\alpha_2) \arrow[r, "q"] & Y
\end{tikzcd} \tag{7.4}$$

commutes there is a diagonal map $d\colon y(\Gamma_2,\alpha_2) \to X$

$$\begin{tikzcd}
y(\Gamma_1,\alpha_1) \arrow[r, "p"] \arrow[d, "m"] & X \arrow[d, "f"] \\
y(\Gamma_2,\alpha_2) \arrow[r, "q"] \arrow[ur, "d"] & Y
\end{tikzcd} \tag{7.5}$$

making the two triangles commute. Here y is the Yoneda embedding, i.e. $y(\Gamma,\alpha) = \mathcal{E}_\Delta(-,(\Gamma,\alpha))\colon \mathcal{E}_\Delta^{\mathrm{op}} \to \mathbf{Sets}$

To show that $[\![P]\!]$ and $[\![Q]\!]$ are open map bisimilar, it suffices to show that there is an open map $r\colon [\![P]\!] \to [\![Q]\!]$. Intuitively, this is analogous to giving a functional bisimulation (indexed by (Γ,α)) between $[\![P]\!](\Gamma,\alpha)$ and $[\![Q]\!](\Gamma,\alpha)$.

The fact that $\sim$ satisfy the rules such as (E-Beta), can be proved by "proof manipulation". As explained, to show that $P \sim Q$, it suffices to give a functional bisimulation between $[\![P]\!](\Gamma,\alpha)$ and $[\![Q]\!](\Gamma,\alpha)$. As a special case, let us consider the case where $P = (\boldsymbol{\nu}\bar{a}a)(!a(x).P \mid \bar{a}\langle y\rangle)$ and $Q = (\boldsymbol{\nu}\bar{a}a)(!a(x).P \mid P\{y/x\})$. In this case, a functional bisimulation r can be defined by $r(p) \overset{\text{def}}{=} q$, where $p \xrightarrow{0} q$, and the proof that this r is a bisimulation is similar to that of subject reduction/expansion. Hence, we have

[5]To be more specific, we define the open-map bisimulation in the setting where $y\mathcal{E}_\Delta$ (the Yoneda embedding of $\mathcal{E}_\Delta$) is the path category and $[\mathcal{E}_\Delta^{\mathrm{op}}, \mathbf{Sets}]$ is the category of models.

Lemma 7.25. The relation $\sim$ satisfies the axioms listed in Figure 3.2. $\square$

We can also show that $\sim$ is a congruence relation.

Lemma 7.26. The relation $\sim$ is a congruence. $\square$

Checking that $\sim$ is a congruence is not that difficult, thanks to the fact that $\trianglelefteq$ is defined according to the structure of a process.

From the above two lemmas (Lemma 7.25 and Lemma 7.26) we have

Theorem 7.27. The relation $\sim$ is a π_F-theory. Therefore, $\mathit{Cl}(\sim)$ is a compact closed Freyd category. $\square$

Since our main interest is the main theorem, we postpone the proof of this theorem (and the proof of Lemma 7.25 and Lemma 7.26). The proofs are given at the end of this chapter (Section 7.5.2).

Now we show that $\sim \subseteq \simeq^c_\tau$. From this and the above theorem (Theorem 7.27) we obtain our main theorem (Theorem 7.1), which states the existence of a compact closed Freyd model that is fully abstract with respect to $\simeq^c_\tau$.

To show that $\sim \subseteq \simeq^c_\tau$ we first show that $\sim$ is a barbed bisimulation (with respect to $\Rightarrow$).

Lemma 7.28. Let $\Delta \overset{\text{def}}{=} \tau : \mathbf{ch}^i[], \bar{o}_1 : \mathbf{ch}^o[], \dots, \bar{o}_n : \mathbf{ch}^o[]$ and P and Q be settled process such that $\Delta \vdash P$ and $\Delta \vdash Q$. Then $P \sim Q$ implies $P \overset{\bullet}{\sim}_\tau Q$.

Before giving the proof, let us prepare a notation that will be used in the proof.

Definition 7.7. Let $\Delta \vdash P$ and $\Delta \vdash Q$ be processes such that $P \sim Q$. For $p \in [\![P]\!](\Gamma, \alpha)$ and $q \in [\![Q]\!](\Gamma, \alpha)$, we write $p \sim q$ if there exists a span of open maps $[\![P]\!] \overset{f}{\leftarrow} X \overset{g}{\rightarrow} [\![Q]\!]$ and an element x of $X(\Gamma, \alpha)$ such that $f_{\Gamma,\alpha}(x) = p$ and $g_{\Gamma,\alpha}(x) = q$. $\blacksquare$

Proof of Lemma 7.28. Let $\mathcal{S}$ be a relation defined as

$$\left\{ (P_k, Q_k) \;\middle|\; \begin{array}{l} (P = P_0 \Rightarrow P_1 \Rightarrow \cdots \Rightarrow P_k) \mathrel{\mathcal{R}} p_0^k \\ (Q = Q_0 \Rightarrow Q_1 \Rightarrow \cdots \Rightarrow Q_k) \mathrel{\mathcal{R}} q_0^k \\ p_0^k \sim q_0^k \\ \text{for some sequences } P = P_0 \Rightarrow P_1 \Rightarrow \cdots \Rightarrow P_k \text{ and} \\ Q = Q_0 \Rightarrow Q_1 \Rightarrow \cdots \Rightarrow Q_k \text{ and} \\ \text{some linear approximations } p_0^k \text{ and } q_0^k \end{array} \right\}$$

where $\mathcal{R}$ is the relation used in Proposition 7.15. (Strictly speaking, we cannot directly use Proposition 7.15 because P and Q have free outputs. However, the same argument can be applied to the current situation.) Clearly, we have $P \mathrel{\mathcal{S}} Q$.

Our goal is to show that $\mathcal{S}$ is a barbed bisimulation. Suppose that $P_k \mathrel{\mathcal{S}} Q_k$. First we show that a transition $P_k \Rightarrow P_{k+1}$ can be mimicked by Q_k. By the definition of $\mathcal{S}$, we have $(P = P_0 \Rightarrow P_1 \Rightarrow \cdots \Rightarrow P_k) \mathrel{\mathcal{R}} p_0^k$ for some reduction sequence and p_0^k, an approximation of P. By the definition of $\mathcal{R}$ we have $(P = P_0 \Rightarrow P_1 \Rightarrow \cdots \Rightarrow P_k \Rightarrow P_{k+1}) \mathrel{\mathcal{R}} p_0^{k+1}$ for some p_0^{k+1} that approximates P such that $p_0^k \trianglelefteq p_0^{k+1}$. Moreover, we have $\tau : (i_1, \mathbf{ch}^i_{\alpha_1}[]) \wedge \cdots \wedge (i_{k+1}, \mathbf{ch}^i_{\alpha_{k+1}}[]) \vdash_\beta p_0^{k+1}$. Without loss of generality, we may assume that $\alpha_1 < \cdots < \alpha_{k+1}$. Since $p_0^k \sim q_0^k$, there exists a terminable linear process q_0^{k+1} such that $q_0^k \trianglelefteq q_0^{k+1}$, $\tau : (i_1, \mathbf{ch}^i_{\alpha_1}[]) \wedge \cdots \wedge (i_{k+1}, \mathbf{ch}^i_{\alpha_{k+1}}[]) \vdash_\beta q_0^{k+1}$, and $p_{k+1}^0 \sim q_{k+1}^0$. Thanks to

Proposition 7.15, we know that there is an execution sequence $Q = Q_0 \Rightarrow Q_1 \Rightarrow \cdots \Rightarrow Q_k \xrightarrow{\pi} Q_{k+1}$ that is related to p_0^{k+1} by $\mathcal{R}$. Moreover, we can assume that Q_{k+1} is settled (See Remark 7.2). It remains to show that the word π is of the form $\tau 0^*$. Let $q_0^{k+1} \xrightarrow{\pi'} q_k^1 \xrightarrow{\pi} q_{k+1}^0$ be the execution that corresponds to the execution $Q = Q_0 \Rightarrow Q_1 \Rightarrow \cdots \Rightarrow Q_k \xrightarrow{\pi} Q_{k+1}$. Now observe that we have $\tau : (i_{k+1}, \mathbf{ch}^i_{\alpha_{k+1}}) \vdash_{\alpha_k} q_k^1$. Since q_0^{k+1} is terminable and closed, we must have $q_k^1 (\xrightarrow{0})^* \xrightarrow{\tau_{i_{k+1}}} (\xrightarrow{0})^* q_{k+1}^0$ (Recall that q_{k+1}^0 is idle). Note that we have $q_k^1 \not\xrightarrow{0}$; if $q_k^1 \xrightarrow{0} q'$ for some q', then there is a process Q' such that $Q_k \longrightarrow Q'$, but this contradicts to the fact that Q_k is settled. Therefore, we must have $q_k^1 \xrightarrow{\tau} (\xrightarrow{0})^* q_{k+1}^0$, that is π is of the form $\tau 0^*$.

Checking that the transition $Q_k \Rightarrow Q_{k+1}$ is matched by a transition from P can be done in a similar fashion.

Similarly we can show that $P_k {\downarrow}^{\tau}_{\bar{a}}$ if and only if $Q_k {\downarrow}^{\tau}_{\bar{a}}$. $\square$

Note that we only proved $P \sim Q$ implies $P \overset{\bullet}{\sim}_\tau Q$ in case P and Q are sorted under particular sort environments. This does not cause a problem due to the following observation.

Observation 1. When we want to prove that P and Q are barbed congruent, we do not need to consider arbitrary contexts. First, we can assume that $C[P]$ does not have free input name. If there is a free input name a in $C[P]$, we can consider the context $(\boldsymbol{\nu} \bar{a} a)(C[-] \mid !a(\vec{x}).\bar{a}\langle \vec{x} \rangle)$, where $\bar{a}$ is fresh, instead of C. Because the output action $\bar{a}$ can never happen, the behaviour of $C[P]$ and $(\boldsymbol{\nu} \bar{a} a)(C[P] \mid !a(\vec{x}).\bar{a}\langle \vec{x} \rangle)$ are the same. Second, we can restrict the types for free outputs. If $C[P]$ has a free output $\bar{a}$ with type $\mathbf{ch}^o[\vec{T}]$, then we can consider the context $(\boldsymbol{\nu} \bar{a} a)(C[-] \mid !a(\vec{x}).\bar{b}\langle\rangle)$, where a and $\bar{b}$ are fresh and $\bar{b}$ has type $\mathbf{ch}^o[]$, instead of C. Note that we have $C[P] {\downarrow}^{\tau}_{\bar{a}}$ if and only if $(\boldsymbol{\nu} \bar{a} a)(C[P] \mid !a(\vec{x}).\bar{b}\langle\rangle) {\downarrow}^{\tau}_{\bar{b}}$. $\square$

The above lemma immediately implies that $\sim \subseteq \simeq^c_\tau$.

Lemma 7.29. If $P \sim Q$ then $P \simeq^c_\tau Q$.

Proof. Suppose that $P \sim Q$ and let C be a context. Without loss of generality, we may assume that there are no free input names in $C[P]$ and that all the free outputs in $C[P]$ have type $\mathbf{ch}^o[]$ by Observation 1. Since $\sim$ is a congruence by Lemma 7.26, we have $\tau.C[P] \sim \tau.C[Q]$. By Lemma 7.28, we have $\tau.C[P] \overset{\bullet}{\sim}_\tau \tau.C[Q]$ and thus $P \simeq^c_\tau Q$. $\square$

Finally, we have our main theorem.

Theorem 7.1. The relation $\simeq^c_\tau$ is a π_F-theory, which means that $Cl(\simeq^c_\tau)$ is a compact closed Freyd category. Hence, there exists a compact closed Freyd category that is fully abstract with respect to $\simeq^c_\tau$. $\square$

Proof. Since $\sim \subseteq \simeq^c_\tau$ by Lemma 7.29 and $\sim$ satisfies the axioms listed in Figure 3.2, $\simeq^c_\tau$ also satisfies these axioms. This concludes that $\simeq^c_\tau$ is a π_F-theory because it is a congruence. $\square$

7.5.2 Remaining proofs

This section skeches the proof of Theorem 7.27, which says that $\sim$ is a π_F-theory. We do not give the full proof to keep the proof understandable; the proof requires a lot of case analysis which can be proved by the same technique and proving all the cases will just make the proof longer. Recall that to prove $\sim$ is a π_F-theory it suffices to show that (1) $\sim$ satisfies the axioms listed in Figure 3.2 and (2) $\sim$ is a congruence. We start by showing (1).

Proof of Lemma 7.25

As explained, the fact that $\sim$ satisfies the axioms can be shown by "proof manipulation". Here we only prove the case for (E-ETA) because this is the most nontrivial case; the proof for the other axioms are similar. The proof for the η-rule also clarifies why witnesses of type isomorphisms needs to appear in the syntax of linear processes.

Proposition 7.30. Let P be a well-sorted process such that $b, \bar{a} \notin \mathbf{fn}(P)$. Then there is an open-map $r\colon [\![(\boldsymbol{\nu}\bar{a}a)(b \hookrightarrow \bar{a} \mid P)]\!] \to [\![P\{b/a\}]\!]$.

Proof. Let $p_1 \in [\![(\boldsymbol{\nu}\bar{a}a)(b \hookrightarrow \bar{a} \mid P)]\!](\Gamma, \alpha)$ and let $\theta = \Gamma(b)$. Then p_1 must be of the following form

$$(\boldsymbol{\nu}[\langle \bar{a}_j, a_j\rangle]_{j \in \mathrm{dom}(\theta')})(\Pi_{i \in \mathrm{dom}(\theta)} b_i(\mu_{i1}, \dots, \mu_{im}).\bar{a}_{\sigma(i)}\langle \lambda_{\sigma(i)1}, \dots, \lambda_{\sigma(i)m}\rangle \mid p). \quad (7.6)$$

where $\sigma\colon \mathbf{Nat} \to \mathbf{Nat}$ is a bijection such that $\sigma\restriction_{\mathrm{dom}(\theta)}$ is a bijection from $\mathrm{dom}(\theta)$ to $\mathrm{dom}(\theta')$. Note that $\theta(i)$ and $\theta'^{\perp}(\sigma(i))$ must be isomorphic for each i. In fact, the witness of the type isomorphism between $\theta(i)$ and $\theta'^{\perp}(\sigma(i))$ can be explicitly given by inspecting the linear process p_1. We write φ_i for the witness of $\theta(i) \cong \theta'^{\perp}(\sigma(i))$.

The function $r_{\Gamma,\alpha}$ is defined by $r_{\Gamma,\alpha}(p_1) \stackrel{\mathrm{def}}{=} p\{\varphi_i \bullet b_i/a_{\sigma(i)}\}_{i \in \mathrm{dom}(\theta)}$. To see that this is well-defined, we need to check that $r_{\Gamma,\alpha}(p_1) \in [\![P\{b/a\}]\!](\Gamma, \alpha)$. Since $\Gamma \vdash_\alpha p_1$, we have a derivation of the following form:

$$\dfrac{\dfrac{\dfrac{\pi_{1i} :: \Gamma_{1i} \vdash_\alpha b_i(\mu_{i1}, \dots, \mu_{im}).\bar{a}_{\sigma(i)}\langle \lambda_{\sigma(i)1}, \dots, \lambda_{\sigma(i)m}\rangle}{(b:\theta, \bar{a}:\theta') \vdash_\alpha \Pi_{i \in \mathrm{dom}(\theta)} b_i(\mu_{i1}, \dots, \mu_{im}).\bar{a}_{\sigma(i)}\langle \lambda_{\sigma(i)1}, \dots, \lambda_{\sigma(i)m}\rangle} \quad \pi_2 :: \Gamma_2, a:\theta'^{\perp} \vdash_\alpha p}{(b:\theta, \bar{a}:\theta') \sqcap (\Gamma_2, a:\theta'^{\perp}) \vdash_\alpha (\Pi_{i \in \mathrm{dom}(\theta)} b_i(\mu_{i1}, \dots, \mu_{im}).\bar{a}_{\sigma(i)}\langle \lambda_{\sigma(i)1}, \dots, \lambda_{\sigma(i)m}\rangle \mid p)}}{\Gamma \vdash_\alpha p_1}$$

where $\bigsqcap_i \Gamma_{1i} = b:\theta, \bar{a}:\theta'$ and $\Gamma = b:\theta, \Gamma_2$. By applying Lemma 7.4 (Substitution lemma) to the derivation π_2, we obtain a type derivation for $\Gamma_2, b:\theta \vdash_\alpha p\{\varphi_i \bullet b_i/a_{\sigma(i)}\}_{i \in \mathrm{dom}(\theta)}$ with $\Gamma_2, b:\theta = \Gamma$ as desired.

We next see that r is a natural transformation. Let $p_1 \in [\![(\boldsymbol{\nu}\bar{a}a)(b \hookrightarrow \bar{a} \mid P)]\!](\Gamma, \alpha)$ and suppose that $A\colon (\Gamma', \alpha') <: (\Gamma, \alpha)$. It suffices to show that $r(p_1\restriction_{\Gamma',\alpha'}) \trianglelefteq r(p_1)$. Recall that p_1 is of the form of (7.6). Therefore, by (the proof of) Lemma 7.21, $p_1\restriction_{\Gamma',\alpha'}$ is of the following form:

$$(\boldsymbol{\nu}[\langle \bar{a}_j, a_j\rangle]_{j \in \mathrm{dom}(\theta'\restriction_A)})(\Pi_{i \in \mathrm{dom}(\theta\restriction_A)} b_i(\mu'_{i1}, \dots, \mu'_{im}).\bar{a}_{\sigma(i)}\langle \lambda'_{\sigma(i)1}, \dots, \lambda'_{\sigma(i)m}\rangle \mid p').$$

where

$$b_i(\mu'_{i1}, \dots, \mu'_{im}).\bar{a}_{\sigma(i)}\langle \lambda'_{\sigma(i)1}, \dots, \lambda'_{\sigma(i)m}\rangle \trianglelefteq b_i(\mu'_{i1}, \dots, \mu'_{im}).\bar{a}_{\sigma(i)}\langle \lambda'_{\sigma(i)1}, \dots, \lambda'_{\sigma(i)m}\rangle \quad (7.7)$$

for $i \in \mathrm{dom}(\theta'\restriction_A)$ and $p' \trianglelefteq p$. It is easy to check that the witness of type isomorphism between $\theta\restriction_A(i)$ and $(\theta'\restriction_A)^{\perp}(\sigma(i))$ that is constructed from the relation (7.7) is given by $\varphi_i\restriction_A$.Hence,

$$r(p_1\restriction_{\Gamma',\alpha'}) = p'\{\varphi\restriction_A \bullet b_i/a_{\sigma(i)}\}_{i \in \mathrm{dom}(\theta\restriction_A)}$$

and thus $r(p_1\restriction_{\Gamma',\alpha'}) \trianglelefteq r(p_1)$, that is

$$p'\{\varphi\restriction_A \bullet b_i/a_{\sigma(i)}\}_{i \in \mathrm{dom}(\theta\restriction_A)} \trianglelefteq p\{\varphi \bullet b_i/a_{\sigma(i)}\}_{i \in \mathrm{dom}(\theta)}$$

by the monotonicity of substitution with respect to $\trianglelefteq$ (Lemma 7.16).

Our remaining task is to verify that r is an open map. To this end, we construct the diagonal map d by defining $p_2 \overset{\text{def}}{=} d_{\Gamma_2,\alpha_2}(\mathrm{id})$ provided that the square (7.4), with X, Y and f replaced by $[\![(\boldsymbol{\nu}\bar{a}a)(b \hookrightarrow \bar{a} \mid P)]\!]$, $[\![P\{b/a\}]\!]$ and r respectively, commutes. Let $p_1 \overset{\text{def}}{=} p_{\Gamma_1,\alpha_1}(\mathrm{id}) \in [\![(\boldsymbol{\nu}\bar{a}a)(b \hookrightarrow \bar{a} \mid P)]\!](\Gamma_1,\alpha_1)$ and $q \overset{\text{def}}{=} q_{\Gamma_2,\alpha_2}(\mathrm{id}) \in [\![P\{b/a\}]\!](\Gamma_2,\alpha_2)$. Again p_1 is of the following form

$$(\boldsymbol{\nu}[\langle \bar{a}_j, a_j\rangle]_{j\in\mathrm{dom}(\theta_{\bar{a}})})(\Pi_{i\in\mathrm{dom}(\theta_b)} b_i(\mu_{i1},\dots,\mu_{im}).\bar{a}_{\sigma(i)}\langle \lambda_{\sigma(i)1},\dots,\lambda_{\sigma(i)m}\rangle \mid p).$$

where $\theta_b = \Gamma_1(b)$. It should be noted that, by the naturality of q, we have $r(p_1) = p\{\varphi_i \centerdot b_i/a_{\sigma(i)}\}_{i\in\mathrm{dom}(\theta_b)} = q{\restriction}_{\Gamma_1,\alpha_1}$.This means that $p = q{\restriction}_{\Gamma_1,\alpha_1}\{\varphi_i^{-1} \centerdot a_{\sigma(i)}/b_i\}_{i\in\mathrm{dom}(\theta_b)}$ by Lemma 7.5. Our strategy is to extend the process p_1 according to how the type environment Γ_2 is extended from Γ_1. For each $i \in \mathrm{dom}(\Gamma_1(b))$, let us first pick a type isomorphism φ'_i that extends $\varphi_i \colon \Gamma_1(b)(i) \cong \theta_a^{\perp}(\sigma(i))$ to a type isomorphism from $\Gamma_2(b)(i)$ to some ρ'_i, which is an extension of $\theta_a^{\perp}(\sigma(i))$. Such an extension always exists by Lemma 7.14. We define p_2 by

$$p_2 \overset{\text{def}}{=} (\boldsymbol{\nu}[\langle \bar{a}_j, a_j\rangle]_{j\in\mathrm{dom}(\theta'')}) \begin{pmatrix} \Pi_{i\in\mathrm{dom}(\theta)} b_i(\mu'_{i1},\dots,\mu'_{il}).\bar{a}_j\langle \lambda'_{\sigma(i)1},\dots,\lambda'_{\sigma(i)l}\rangle \\ \parallel \Pi_{j\in J} b_i(\mu_{j1},\dots,\mu_{jm}).\bar{a}_j\langle \mu_{j1},\dots,\mu_{jm}\rangle \\ \mid q\{(\varphi'_i)^{-1} \centerdot a_i/b_i\}_{i\in\mathrm{dom}(\theta)}\{a_j/b_j\}_{j\in J} \end{pmatrix}$$

where

- $J \overset{\text{def}}{=} \mathrm{dom}(\Gamma_2(b)) \setminus \mathrm{dom}(\theta)$
- θ'_a is the intersection type such that $(\theta'_a)^{\perp}(\sigma(i)) = \rho'_i$ for $i \in \mathrm{dom}(\theta)$ and $(\theta'_a)^{\perp}(i) = \Gamma_2(b)(i)$ for $i \notin \mathrm{dom}(\theta)$
- for $j \in J, k \in \{1,\dots,m\}$, $\mu_{jk} \overset{\text{def}}{=} \mathrm{id}_{\theta_{jk}} \centerdot x^k$ if the type of the k-th argument of $\Gamma_2(b)(j)$ is θ_{jk}
- for $i \in \mathrm{dom}(\theta), k \in \{1,\dots,m\}$, $\mu'_{ik} \overset{\text{def}}{=} id_{\theta_{ik}} \centerdot x^k$ if the type of the k-th argument of $\Gamma_2(b)(i)$ is θ_{ik}
- for $i \in \mathrm{dom}(\theta), k \in \{1,\dots,m\}$, $\lambda'_{ik} \overset{\text{def}}{=} \varphi'_{ik} \centerdot x^k$, where $\varphi'_i = \mathbf{ch}^i[\varphi'_{i1},\dots,\varphi'_{i1}]$

Intuitively, p_2 is obtained by replacing $q{\restriction}_{\Gamma_1,\alpha}$ with q and "padding linear variables" to p according to the type $\Gamma_2(b)$. Clearly $r(p_2) = q$ because

$$r(p_2) = q\{(\varphi'_i)^{-1} \centerdot a_{\sigma(i)}/b_i\}_{i\in\mathrm{dom}(\theta)}\{a_j/b_j\}_{j\in J}\{\varphi'_i \centerdot b_i/a_{\sigma(i)}\}_{i\in\mathrm{dom}(\theta)}\{b_j/a_j\}_{j\in J},$$

which is equal to q by Lemma 7.5. Together with the naturality of r, this means that the lower triangle of (7.5) commutes. To show the commutativity of the upper triangle, it suffices to show that $p_2{\restriction}_{\Gamma_1,\alpha_1} = p_1$. This follows from the construction of p_2 and the monotonicity of substitution with respect to $\trianglelefteq$ (Lemma 7.16). $\square$

Witnesses of type isomorphisms play an important role in the above proof. To see why let us consider a linear calculus without type isomorphisms. That is a calculus identical to the one defined in Section 7.3.1, except for the fact that all the outputs are of the form $\bar{a}\langle \mu_1,\dots,\mu_n\rangle$ instead of $\bar{a}\langle \lambda_1,\dots,\lambda_n\rangle$. The typing rule for the output action is also modified as follows:

$$\frac{\mathrm{id} \centerdot x^i = \mu_i \quad (\text{for } i \in [n]) \quad \alpha \le \overline{\mathbf{lv}}(\theta_1,\dots,\theta_n)}{x^1 : \theta_1 \sqcap \dots \sqcap x^n : \theta_n, \bar{a} : (i, \mathbf{ch}^o_\alpha[\theta_1,\dots,\theta_n]) \vdash_\alpha \bar{a}_i\langle \mu_1,\dots,\mu_n\rangle} \text{(TOUT')}$$

Now let $P \overset{\text{def}}{=} (\boldsymbol{\nu}\bar{a}a)(!b(x).\bar{a}\langle x\rangle \mid \bar{c}\langle a\rangle)$ and $Q \overset{\text{def}}{=} \bar{c}\langle b\rangle$ be processes typed under a sort environment $\Delta \overset{\text{def}}{=} b : \mathbf{ch}^i[\mathbf{ch}^o[]], \bar{c} : \mathbf{ch}^o[\mathbf{ch}^i[\mathbf{ch}^o[]]]$. Note that P and Q can be equated by the η-rule. Let us consider a type environment $\Gamma \overset{\text{def}}{=} b : (0, \mathbf{ch}^i_\beta[(0,\rho_0) \wedge (1,\rho_1)]), \bar{c} : (0, \mathbf{ch}^o_\gamma[\mathbf{ch}^i_\beta[(1,\rho_1) \wedge (0,\rho_0)]])$, where ρ_0 and ρ_1 are refinements of $\mathbf{ch}^o[]$, Γ is a valid refinement of Δ and a linear process $p \overset{\text{def}}{=} (\boldsymbol{\nu}\langle\bar{a}_0, a_0\rangle)(b_0(\langle x_0, x_1\rangle).\bar{a}_0\langle\langle x_1, x_0\rangle\rangle \mid \bar{c}_0\langle\langle a_0\rangle\rangle)$ is a valid approximation of P, i.e. $\Gamma^\natural \vdash p \sqsubset P$. Clearly, $\Gamma \vdash_\gamma p$. However, there is no q such that $\Gamma \vdash_\gamma q$ and $\Gamma^\natural \vdash q \sqsubset Q$. We want to take $\bar{c}_0\langle\langle b_0\rangle\rangle$ for q, but this is not possible because the type of b_0 is $\mathbf{ch}^o_\beta[(0,\rho_0) \wedge (1,\rho_1)]$ rather than $\mathbf{ch}^o_\beta[(1,\rho_1) \wedge (0,\rho_0)]$. In other words, a strict non-commutative intersection type system does not meet our technical requirement.

Remark 7.3. The notion of type isomorphism was taken from the rigid intersection type system given by Tsukada et al. [128], but in their calculus, witnesses do not appear in the syntax. This is so because all the (raw) terms in their resource calculus are assumed to be in η-long form. (See [128] for details.)

Similarly, we may remove witnesses of type isomorphisms from our linear calculus, if there is a way to convert a linear process p to an "equivalent" process p' that does not contain any free outputs. A possible way to do this is to transform a free output to a "bound output + forwarder". That is to (recursively) transform a free output $\bar{a}_0\langle\langle \bar{b}_0\rangle\rangle$ into $(\boldsymbol{\nu}[\langle\bar{c}_0, c_0\rangle])(\bar{a}_0\langle\langle\bar{c}_0\rangle\rangle \mid b(\mu).\bar{c}_0\langle\mu\rangle)$. We chose to keep free outputs in the syntax of the linear process because by doing so it is easier to see the correspondence between a (non-linear) process P, which may contain free outputs, and its linear approximation p. Note that we cannot assume that (non-linear) processes do not contain free outputs because the validity of the transformation $\bar{a}\langle b\rangle = (\boldsymbol{\nu}\bar{c}c)(\bar{a}\langle\bar{c}\rangle \mid c \hookrightarrow \bar{b})$ is not something that is taken for granted (even if the forwarder does not introduce any delay). Indeed the purpose of Lemma 7.30 was to check whether such kind of translations are valid. ■

Proof that $\sim$ is a congruence

Now we prove the remaining lemma which says that $\sim$ is a congruence (Lemma 7.26).

Proof of Lemma 7.26. Assume that $P \sim Q$. Then there exists a span of open map $[\![P]\!] \leftarrow X \rightarrow [\![Q]\!]$. For all context C, we construct a functor $C[X]$ and show that there is a span of open maps from $[\![C[P]]\!]$ to $[\![C[Q]]\!]$ of the form $[\![C[P]]\!] \leftarrow C[X] \rightarrow [\![C[Q]]\!]$. The proof proceeds by induction on the structure of C. We only prove the case for the parallel composition in detail; we only sketch the remaining cases because they can be proved by a similar reasoning.

(Case $C = [\,]$) Trivial because we can take X as $C[X]$.

(Case $C = C' \mid R$) We define $C[X]$ by

$$C[X](\Gamma,\alpha) \overset{\text{def}}{=} \{(x,r) \mid x \in C'[X](\Gamma_1,\alpha),\ r \in [\![R]\!](\Gamma_2,\alpha),\ \Gamma = \Gamma_1 \sqcap \Gamma_2\}.$$

The action over morphisms is defined by

$$C[X]((\Gamma,\alpha) <: (\Gamma',\alpha'))(x',r') \overset{\text{def}}{=} (x,r)$$

where

$$C'[X]((\Gamma_1,\alpha) <: (\Gamma'_1,\alpha'))(x') = x$$
$$[\![R]\!]((\Gamma_2,\alpha) <: (\Gamma'_2,\alpha'))(r') = r.$$

Here Γ'_2 is the unique type environment such that $\Gamma'_2 \vdash_{\alpha'} r$ and $\Gamma'_2 \subseteq \Gamma$, which can be inferred from r, and Γ'_1 is the type environment such that $\Gamma' = \Gamma'_1 \sqcap \Gamma'_2$. Type environments Γ_1 and Γ_2 are the restrictions of Γ'_1 and Γ'_2, respectively, such that $\Gamma = \Gamma_1 \sqcap \Gamma_2$.

We next construct an open map f from $C[X]$ to $[\![C[P]]\!]$. Let us first choose an open map f' from $C'[X]$ to $[\![C'[P]]\!]$, which exists by the induction hypothesis. Each component $f_{\Gamma,\alpha}$ of f is defined as a map that maps (x, r) to a linear process $p \mid r$ where $p = f'_{\Gamma_1,\alpha}(x)$. Here the type environment Γ_1 is the type environment that satisfies $\Gamma = \Gamma_1 \sqcap \Gamma_2$, where Γ_2 is the type environment determined by r. Naturality of f is inherited from that of f'.

We now verify that f is an open map. That is we show that there is a diagonal map $d\colon y(\Gamma_2, \alpha_2) \to C[X]$ whenever the square

$$\begin{tikzcd}
y(\Gamma_1, \alpha_1) \arrow[r, "p"] \arrow[d, "m"] & C[X] \arrow[d, "f"] \\
y(\Gamma_2, \alpha_2) \arrow[r, "q"] & {[\![C[P]]\!]}
\end{tikzcd}$$

commutes. To define the diagonal map d, we use a diagonal map that is associated with the open map $f'\colon C'[X] \to [\![C'[P]]\!]$. By "dropping the information that is related to the linear approximation of R" from the above square we get

$$\begin{tikzcd}
y(\Gamma_{11}, \alpha_1) \arrow[r, "p'"] \arrow[d, "m'"] & C'[X] \arrow[d, "f'"] \\
y(\Gamma_{21}, \alpha_2) \arrow[r, "q'"] \arrow[ur, "d'"] & {[\![C'[P]]\!]}
\end{tikzcd} \qquad (7.8)$$

Formally speaking, Γ_{11} is the type environment such that $\Gamma_1 = \Gamma_{11} \sqcap \Gamma_{12}$, where Γ_{12} is the type environment that satisfies $\Gamma_{12} \vdash_{\alpha_1} r_1$, provided that $(x_1, r_1) = p_{\Gamma_1,\alpha_1}(\mathrm{id}_{\Gamma_1,\alpha_1})$. The natural transformation p' is the one that satisfies $p_{\Gamma \sqcap \Gamma_{12}}(*) = (p'_\Gamma(*), r_2)$. (Similarly, Γ_{21} and q' are determined by $p_2 \mid r_2 = q_{\Gamma_2,\alpha_2}(\mathrm{id})$.) Let us now construct the (Γ, α) component of d. We want the two triangles in the following square to commute:

$$\begin{tikzcd}
\mathcal{E}((\Gamma,\alpha),(\Gamma_1,\alpha_1)) \arrow[r, "p_{\Gamma,\alpha}"] \arrow[d, "m"] & C[X](\Gamma,\alpha) \arrow[d, "f_{\Gamma,\alpha}"] \\
\mathcal{E}((\Gamma,\alpha),(\Gamma_2,\alpha_2)) \arrow[r, "q_{\Gamma,\alpha}"] \arrow[ur, "d_{\Gamma,\alpha}"] & {[\![C[P]]\!](\Gamma,\alpha)}
\end{tikzcd}$$

We only consider the case where $\mathcal{E}((\Gamma,\alpha),(\Gamma_2,\alpha_2))$ is a singleton set $\{*\}$; the other case, namely the case where $\mathcal{E}((\Gamma,\alpha),(\Gamma_2,\alpha_2))$ is empty, is trivial because $d_{\Gamma,\alpha}$ must be the empty map. Now let $p \mid r \overset{\text{def}}{=} q_{\Gamma,\alpha}(*)$. Then the map $d_{\Gamma,\alpha}$ is defined by $d_{\Gamma,\alpha}(*) \overset{\text{def}}{=} (d'(*), r)$. The lower triangle commutes because

$$\begin{aligned}
f_{\Gamma,\alpha}(d_{\Gamma,\alpha}(*)) &= f_{\Gamma,\alpha}(d'_{\Gamma',\alpha}(*), r) \\
&= p' \mid r \qquad (\text{where } p' = f'_{\Gamma',\alpha}(d'_{\Gamma',\alpha}(*)) = p) \\
&= q_{\Gamma,\alpha}(*).
\end{aligned}$$

for some suitable Γ'. Now we check the commutativity of the upper triangle. Suppose that the $\mathcal{E}((\Gamma,\alpha),(\Gamma_1,\alpha_1)) = \{*\}$; if it is empty the commutativity of the triangle is trivial. Let $(x_1, r_1) \overset{\text{def}}{=} p_{\Gamma,\alpha}(*)$. Since the outer square commutes, we have $f_{\Gamma,\alpha}(x_1, r_1) = p \mid r$, and hence $r_1 = r$. We also have $x_1 = d'_{\Gamma',\alpha'}(*)$ because $x_1 = p'_{\Gamma',\alpha}(*)$ by definition of p' and $p'_{\Gamma',\alpha}(*) = d'_{\Gamma',\alpha'}$ by (7.8). Therefore, $p_{\Gamma,\alpha}(*) = d_{\Gamma,\alpha}(*)$ as desired. It remains to check the naturality of d, but this is immediate from the naturality of d'.

The open map from $C[X]$ to $[\![C[Q]]\!]$ can be defined in a similar manner.

(Case $C = (\boldsymbol{\nu}\bar{a}a)C'$) Let $[\![C'[P]]\!] \xleftarrow{f'} C'[X] \xrightarrow{g'} [\![C'[Q]]\!]$ be a span of open map, which exists by the induction hypothesis. We define $C[X]$ by

$$C[X](\Gamma,\alpha) \overset{\text{def}}{=} \left\{ (x, (\bar{a}:\theta, a:\theta^{\perp})) \;\middle|\; \begin{array}{c} \bar{a}, a \notin \mathrm{dom}(\Gamma), \overline{\mathbf{lv}}(\theta) \subseteq \overline{\mathbf{lv}}(\Gamma) \cup \{\alpha\} \\ x \in C'[X](\Gamma, \bar{a}:\theta, a:\theta^{\perp}, \alpha) \\ (\boldsymbol{\nu}[\langle \bar{a}_i, a_i\rangle]_{i\in\mathrm{dom}(\theta)})p \text{ is terminable,} \\ \text{where } p = f'_{\Gamma,\bar{a}:\theta,a:\theta^{\perp},\alpha}(x) \end{array} \right\}$$

The action over morphisms is defined by

$$C[X]((\Gamma,\alpha) <: (\Gamma',\alpha))(x', (\bar{a}:\theta', a:\theta'^{\perp})) \overset{\text{def}}{=} (x, (\bar{a}:\theta'\!\restriction_A, a:\theta'^{\perp}\!\restriction_A))$$

where

$$x = C'[X]((\Gamma, \bar{a}:\theta'\!\restriction_A, a:\theta'^{\perp}\!\restriction_A, \alpha) <: (\Gamma, \bar{a}:\theta, a:\theta^{\perp}, \alpha))(x')$$

and A is a witness of $(\Gamma,\alpha) <: (\Gamma',\alpha)$.

The open map g from $C[X]$ to $[\![C[Q]]\!]$ is given by

$$g_{\Gamma,\alpha}(x, (\bar{a}:\theta, a:\theta^{\perp})) \overset{\text{def}}{=} (\boldsymbol{\nu}[\langle \bar{a}_i, a_i\rangle]_{i\in\mathrm{dom}(\theta)})q,$$

where $q = g_{\Gamma,\bar{a}:\theta,a:\theta^{\perp},\alpha}$. It is not hard to check that g is indeed an open map. (The open map from $C[X]$ to $[\![C[P]]\!]$ is defined similarly.)

(Case $C = a(\vec{x}).C'$) and (Case $C = \tau.C'$)] We only sketch the case of input prefixing. Let T be the sort of a. We define $C[X]$ by

$$C[X](\Gamma,\alpha) \overset{\text{def}}{=}$$
$$\left\{ ((i, \mathbf{ch}^i_\beta[\vec{\theta}]), (y^1,\ldots,y^n), x) \;\middle|\; \begin{array}{r} \mathbf{ch}^i_\beta[\theta_1,\ldots,\theta_n] \sqsubset T, \\ x \in C'[X](\Gamma', y^1:\theta_1,\ldots,y^n:\theta_n,\beta), \\ y^1,\ldots y^n \notin \mathrm{dom}(\Gamma'), \\ \Gamma = \Gamma' \sqcap a:(i,\mathbf{ch}^i_\beta[\vec{\theta}]),\ \alpha \le \beta \end{array} \right\}$$

(The action on morphisms is defined accordingly)

The open map $f\colon C[X] \to [\![C[P]]\!]$ is given by

$$f_{\Gamma,\alpha}((i, \mathbf{ch}^i_\beta[\vec{\theta}]), (y^1,\ldots,y^n), x) \overset{\text{def}}{=} a_i(\mu_1,\ldots,\mu_n).p$$

where $\mu_i = \mathrm{id} \centerdot y^i$, $p = f'_{\Gamma',\vec{y}:\vec{\theta},\beta}(x)$ and Γ' is the type environment such that $\Gamma = \Gamma' \sqcap a:\mathbf{ch}^i_\beta[\vec{\theta}]$.

□

Chapter 8

Related Work

In the chapters so far, we have already commented on relationships of our results to other work. These comments were mostly technical. Here we discuss related work from a broader perspective.

8.1 Logical Studies of π-calculi

There is a considerable amount of studies on connections between process calculi and linear logic. Here we divide these studies into two classes. One is the studies that investigates the connection between π-calculus and proof-nets (or proof-structures[1]) of linear logic. In these work names are related to ports of a proof-net and types are considered as the specification of a port. The other class is the studies that is based on the Curry-Howard correspondence between *session typed* π-calculi and linear logic. Our work is more closely related to the former than the latter, but some interesting coincidence to the latter kind of studies can also be found.

Proof nets and π-calculus The former class of research dates back to the work by Abramsky [3] and Bellin and Scott [11], where they discovered that π-calculus processes can encode proof-nets of classical linear logic. Later, Abramsky et al. [5] introduced the *interaction categories* to give a semantic description of a CCS-like process calculus. In their work, they observed that the compact closed structure is important to capture the strong expressive power of process calculi.

A tighter connection between π-calculus and proof-nets was recently presented by Honda and Laurent [54]. They showed that an $\mathbf{i}/\mathbf{o}$-typed π-calculus corresponds to *polarized proof-nets*, and introduced the notion of *extended reduction* for the π-calculus to simulate cut-elimination. The π-calculus used in this work is very similar to the π_F-calculus in terms of syntax and reduction. Their calculus is asynchronous, does not allow non-replicated inputs, and requires $\mathbf{i}/\mathbf{o}$-separation. Furthermore, the extended reduction is almost the same as the rules (E-Beta) and (E-GC) except for the side conditions. A significant difference compared to our work is that their calculus is *local* [88, 136], reflecting the fact that the corresponding logic is polarized.

Our work is inspired by these studies. The idea of $\mathbf{i}/\mathbf{o}$-separation can already be found in the work by Bellin and Scott and the use of compact closed category is motivated by the study of interaction category. However, none of these studies (explicitly) provide a categorical type theoretic correspondence like we did.

[1]By proof-structures we mean a proof-net that does not necessarily satisfy the correctness criterion.

Linear logic and session-typed π-calculus The latter approach started with the Curry-Howard correspondences between session-typed π-calculi and linear logic established by Caires, Pfenning and [18, 19] and subsequently by Wadler [132, 133]. The key difference between the above class of studies and these studies lies in the interpretation of $\otimes$ (and $\parr$). For example in CP [133], the connective $\otimes$ is interpreted as

$$\frac{P \vdash \Gamma, y : A \qquad Q \vdash \Delta, x : B}{(\boldsymbol{\nu} y)x\langle y\rangle.(P \mid Q) \vdash \Gamma, \Delta x : A \otimes B}$$

where $A \otimes B$ is the type of a channel which outputs data of type A and then behaves as B rather than "a parallel connection" of A and B. Note that the constructor for output always takes two continuations P and Q whereas in process calculi, only one continuation is typically necessary. Recently this discrepancy was solved by introducing hyperenvironments to linear logic and considering a calculus, called HCP, that corresponds to this logic [70].

These correspondences are exact in the sense that every process has a corresponding proof, and vice versa. As a consequence of the exact correspondence, processes of the calculi inherit good properties of linear logic proofs such as termination and confluence of cut-elimination. In terms of process calculi, process of these calculi do not fall into deadlock or race condition. This can be seen as a serious restriction of expressive power as pointed out by various researchers (e.g. [133, 7, 82]).

Several extensions to increase the expressiveness of these calculi have been proposed and studied. Interestingly, ideas behind some of these extensions are related to our work, in particular to Section 5.2 discussing the multicut rule [5] and the axiom that relates ! with ?. Atkey et al. [7] studied CP [133] with the multicut rule and rules that identify ! with ? and discussed how these extensions increase the expressiveness of the calculus, at the cost of losing some good properties of CP. Dardha and Gay [29] studied another extension of CP with multicut, keeping the calculus deadlock-free by an elaborated type system. Their type system is inspired by type systems introduced by Kobayashi [65, 66], where types are annotated by "tags" to ensure the absence of deadlock. Besides introducing multicut rule, other approaches are also taken to increase the expressiveness of (logically founded) session-typed π-calculi. Balzer and Pfenning [8] proposed a session-typed calculus with shared (mutable) resources, inspired by linear-non-linear adjunction [12]. Kokke et al. [71] also introduced the notion of shared channels to HCP so that some amount of races can be expressed. In their work, types of a shared channel tracks how many times it is used, much like the modalities in bounded linear logic [45], and this type system guarantees deadlock-freedom.

8.2 Categorical Semantics of π-calculi

The idea of using a closed Freyd category to model the π-calculus is strongly inspired by Laird [74]. He introduced the *distributive-closed Freyd category* to describe abstract properties of a game-semantic model of the asynchronous π-calculus and showed that distributive-closed Freyd categories with some additional structures suffice to interpret the asynchronous π-calculus. The additional structures are specific to his game model and not completely axiomatized. Our notion of compact closed Freyd category might be seen as a rectification of his idea, obtained by filtering out some structures difficult to axiomatize and by strengthening some others to make axioms simpler. A significant difference is that our categorical model does not deal with non-replicated inputs, which we think is essential for a simple axiomatization.

Before Laird's work, Freyd categories have already been used to model a process calculus called *action calculus* [91] introduced by Milner. In fact, Power [109] showed that the *elementary control structure*, which is essentially a (monoidal) Freyd category, corresponds to a moderately special case of the action calculus. Later, Gardner and Hasegawa [40] showed that extensions of action calculus correspond to special Freyd categories: higher-order action calculus [92] and reflexive action calculus were shown to correspond to closed Freyd category and traced Freyd category, respectively. Since the introduction of the action calculus, it was known that action calculus could encode π-calculus by adding specific rules and constants [91]. However, the relationship between π-calculus and closed Freyd category was not investigated in depth by this time.

Another approach for categorical semantics of the π-calculus has been the presheaf based approach [123, 36, 22]. These studies gave particular categories that nicely handles the nominal aspects of the π-calculus; these studies, however, do not aim for a correspondence between a categorical structure and the π-calculus. Among others, the work by Cattani et al. [22] shares similarity with our presheaf semantics (given in Chapter 7), because they define the equality over processes using the notion of open map bisimilarity [61]. However, the definition of the path category is significantly different. The path category used in the work by Cattani et al. is indexed by the category of finite name sets and injective maps so that it can treat fresh names. On the other hand, our path category is simply the category of type environments of an intersection type system.

8.3 Linear Approximation and π-calculus

We are not the first one to apply the notion of linear approximation to the π-calculus. A non-idempotent intersection type system for a variant of the π-calculus that comes from the notion of linear approximations has also been introduced by Dal Lago et al. [72]; it should be noted that they did not introduce a "linear calculus" that corresponds to the derivation of the intersection type system as we did. The connection between linear approximations and intersection types [83] was applied to the encoding of π-calculus to proof-nets to derive the basis of an intersection type system for a fragment of the local π-calculus [136, 88] called *hyperlocalized π-calculus*. They showed that the type system obtained this way characterizes some "good behavior", such as deadlock-freedom, of hyperlocalized processes. In contrast to our work, they use intersection types to guarantee that typable processes are "well-behaved", rather than to define "operational semantics" of the calculus.

8.4 On the Delay of Forwarders

In Chapter 7, we observed that the delay that a forwarder introduces is a cause for the mismatch between behavioral equivalences and categorical semantics. The delays that forwarders add has also been an issue in the field of game semantics. In game semantics, forwarders correspond to copycat strategies and the delay copycat strategies introduce were an obstacle to model synchronous computations using game semantics. When giving a game semantics of a synchronous session typed π-calculus, Castellan and Yoshida [21] observed that the traditional copycat strategy does not behave as identity due to the delay it introduces. To avoid this problem, they introduced a copycat strategy that does not introduce any delay and proved that this "delayless copycat strategy" works as the identity. Whereas [21] added delayless forwarders as semantics elements that processes cannot represent, Chapter 7 of this thesis discusses a new operational semantics on processes with respect to which forwarders are delayless. A "delayless copycat strategy" has also been considered by Melliès in a framework called template games [86], but the relation to π-calculus is not clear. Although these game semantic work are apparently different from ours, we believe that they are relevant to our work given that there is a tight relationship between game semantics and linear approximations [129]; detailed comparisons are left for future work.

Chapter 9

Conclusion and Future Work

We have developed the π_F-calculus, a variant of an **i/o**-typed π-calculus which has a corresponding categorical structure. This gives a categorical foundation to π-calculi that are not session-typed.

The corresponding categorical structure we introduced is the compact closed Freyd category, which is defined as a combination of closed Freyd category and compact closed category. The correspondence was established based on the observation that the π-calculus can be considered as a higher-order programming language [115]. Inputs and outputs were considered λ-abstractions (that is located at a certain address) and applications, respectively, and these were modeled by the closed Freyd structure. Another characteristic of π_F-calculus is the **i/o**-separation, which ensures that each type has its dual. This allowed channels to be modeled by the compact closed structures. The correspondence is quite solid: we proved that the categorical semantics is sound and complete, and the term model is the classifying category. We also discussed that, to achieve this correspondence, it was necessary to consider the η-rule, which is awkward from the operational semantic viewpoint.

The π_F-calculus and compact closed Freyd structure capture the strong expressive power of the π-calculus. The compact closed structure allows us to connect ports in an arbitrary way, in return for the possibility of deadlocks. The Freyd structure allows us to duplicate objects, and duplication of input channels introduces the possibility of race conditions. Hence, the π_F-calculus is more expressive than linear logic based session-typed π-calculi [19, 133] in the sense that it can express deadlocks and race conditions. However, even in the π_F-calculus, it seems impossible to encode reference cells and locks due to the lack of non-replicated inputs.

In Chapter 5, we discussed the relationship between the π_F-calculus and linear logic. First, we showed that every linear logic proof can be interpreted as a π_F-process. Then we discussed the other direction, i.e. a proof system that can interpret π_F-processes as proofs. We presented an extension of linear logic with multicut rule [5] and an axiom $?A \vdash !?A$ as a candidate of a logical system that can interpret π_F-processes. We think that this gives a new observation against the open problem that asks whether there is a proof system that corresponds to the traditional π-calculus.

To illustrate the usefulness of the π_F-calculus and compact closed Freyd category, we gave a semantic analysis against the relationship between higher-order languages and π-calculus (Chapter 6). We introduced the λ_{ch}-calculus, a computational λ-calculus augmented with communication channels that corresponds to π_F-calculus, and showed that translations between these calculi are essentially the same as the translations between the higher-order π-calculus and the

π-calculus [118]. We also showed that encoding of call-by-value and call-by-name λ-calculus to π-calculus [90], can be understood as call-by-value and call-by-name monadic interpretation using a continuation monad defined over compact closed Freyd categories. It should be emphasized that these translations were rediscovered without any elaborated analysis/calculation but by a simple semantic considerations thanks to the correspondence between the π_F-calculus and compact closed Freyd categories.

Lastly, in an effort to bridge the gap between behavioral and categorical semantics of the π_F-calculus, we developed a new operational semantics that is compatible with the categorical semantics (Chapter 7). The novel operational semantics is based on a simple idea: unguarded output actions must be immediately consumed. We showed that barbed congruence defined upon this new operational semantics is a π_F-theory. The novel operational semantics was related to the conventional operational semantics by defining a translation from π_F-calculus with conventional operational semantics to the π_F-calculus with the new semantics. We showed that this translation is sound, but incomplete with respect to barbed congruence.

Future Work An interesting direction of future work is to add linear channels to π_F-calculus. In order to add linear channels to π_F-calculus, we need to give up using Freyd categories. This is because requiring an identity-on-object functor from a cartesian category makes every object copyable. A hopeful solution is to consider linear categories (or LNL models) with additional axioms as we did in Chapter 5. In these models, we conjecture that linear channels can be added as resource modalities [87].

Adding non-replicated inputs (for non-linear channels) to the π_F-calculus seems much harder than adding linear channels. As mentioned, we think it is similar to adding reference cells to λ-calculus, which is known to be hard to give denotational models. It would be interesting to give a formal relationship between reference cells and non-replicated inputs. For example, we may develop mutual encoding between "π_F-calculus + reference cells" and "π_F-calculus + non-replicated inputs" and study whether it is fully-abstract. Constructing a specific denotational model (e.g. game model) for π-calculus with non-replicated inputs by adopting the techniques used to model reference cells [99, 73] is also a technically interesting future work.

From a broader perspective, we would like to apply our semantic framework to the study of type systems. There are two directions that we want to explore. One is to design a type system for π-calculus by logical or semantic techniques. For instance, we are interested in using bounded linear logic [45] to design a type system that ensures deadlock-freedom. The other is to give semantic understanding to the sophisticated type systems of π-calculus [67, 60, 68, 102] that are based on **i**/**o**-types and are used to analyze properties such as deadlocks and races.

Semantic understanding of the relationship between **i**/**o**-typed π-calculi and session typed π-calculi is also left for future work. It is well known that session-types can be encoded into **i**/**o**-type by "continuation passing" [30, 67]. We are interested in whether this can be understood as "model translation", i.e. whether we can understand this encoding as a construction of a model of linear logic from the model of π_F-calculus. To do such a semantic analysis, we may first need to add linear channels to π_F-calculus for capturing the linear usage of continuations.

Revealing the relationship between locality and the η-rule is another important problem. The local π-calculus behaves well with respect to the "(π_F-theory)

$-$ (η-rule)". For instance, (1) the compilation from asynchronous higher-order π-calculus to the local fragment of the π_F-calculus preserves and reflects the provability without using the η-rules and (2) in local π-calculus we can replace free outputs with bound outputs without using the η-rule. Our semantic framework cannot (at least at the moment) explain why local π-calculus behaves well. Further investigation on the relationship between local π-calculus and polarized linear logic [54] might provide some insights to this phenomenon.

Finally, we would also like to continue the study on λ_{ch}-calculus, which we introduced in Chapter 6. Though the λ_{ch}-calculus is equivalent to π_F-calculus, we think λ_{ch}-calculus is attracting because it is (1) closer to practical programming languages and (2) an instance of λ_c-calculus, which has been well-studied. Primarily, the operational properties of the λ_{ch}-calculus and its relation to the equational theory needs further investigation. We are also interested in combining communication with various computational effects. Restructuring communication in the form of an algebraic effect [108] may help improve our understanding against this issue since algebraic effects can be naturally combined. Applying sophisticated typing systems studied for λ-calculus, such as effect [42, 124] and coeffect [104] systems, to λ_{ch}-calculus to analyze concurrent programs might also be possible. It would be interesting to consider whether we can port these typing systems to the π-calculus by using the relationship between λ_{ch} and π_F as in the case of session-typed π-calculus [101].

References

[1] Erlang Programming Language. `https://www.erlang.org/`. Accessed: 2020-12-04.

[2] The Go Programming Language. `https://golang.org/`. Accessed: 2020-12-04.

[3] Samson Abramsky. Proofs as processes. *Theor. Comput. Sci.*, 135(1):5–9, 1994.

[4] Samson Abramsky and Bob Coecke. Abstract physical traces. *Theory and Applications of Categories [electronic only]*, 14:111–124, 2005.

[5] Samson Abramsky, Simon J. Gay, and Rajagopal Nagarajan. Interaction categories and the foundations of typed concurrent programming. In *Proceedings of the NATO Advanced Study Institute on Deductive Program Design, Marktoberdorf, Germany*, pages 35–113, 1996.

[6] Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *J. Log. Comput.*, 2(3):297–347, 1992.

[7] Robert Atkey, Sam Lindley, and J. Garrett Morris. Conflation confers concurrency. In *A List of Successes That Can Change the World - Essays Dedicated to Philip Wadler on the Occasion of His 60th Birthday*, pages 32–55, 2016.

[8] Stephanie Balzer and Frank Pfenning. Manifest sharing with session types. *PACMPL*, 1(ICFP):37:1–37:29, 2017.

[9] Andrew Barber. Dual intuitionistic linear logic. Technical Report LFCS-96-347, University of Edinburgh, 1997.

[10] Michael Barr. **- Autonomous Categories*, volume 752 of *Lecture Notes in Mathematics*. Springer-Verlag Berlin Heidelberg, 1979.

[11] Gianluigi Bellin and Philip J. Scott. On the π-calculus and linear logic. *Theor. Comput. Sci.*, 135(1):11–65, 1994.

[12] P. N. Benton. A mixed linear and non-linear logic: Proofs, terms and models. In *Computer Science Logic, 8th International Workshop, CSL '94, Kazimierz, Poland, September 25-30, 1994, Selected Papers*, pages 121–135, 1995.

[13] Gavin M. Bierman. What is a categorical model of intuitionistic linear logic? In *Typed Lambda Calculi and Applications, Second International Conference on Typed Lambda Calculi and Applications, TLCA '95, Edinburgh, UK, April 10-12, 1995, Proceedings*, pages 78–93, 1995.

[14] Michele Boreale. On the expressiveness of internal mobility in name-passing calculi. *Theor. Comput. Sci.*, 195(2):205–226, 1998.

[15] Michele Boreale and Rocco De Nicola. Testing equivalence for mobile processes. *Inf. Comput.*, 120(2):279–303, 1995.

[16] Gérard Boudol. The lambda-calculus with multiplicities (abstract). In Eike Best, editor, *CONCUR '93, 4th International Conference on Concurrency Theory, Hildesheim, Germany, August 23-26, 1993, Proceedings*, volume 715 of *Lecture Notes in Computer Science*, pages 1–6. Springer, 1993.

[17] Gérard Boudol. The pi-calculus in direct style. In Peter Lee, Fritz Henglein, and Neil D. Jones, editors, *Conference Record of POPL'97: The 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Papers Presented at the Symposium, Paris, France, 15-17 January 1997*, pages 228–241. ACM Press, 1997.

[18] Luís Caires and Frank Pfenning. Session types as intuitionistic linear propositions. In *CONCUR 2010 - Concurrency Theory, 21th International Conference, CONCUR 2010, Paris, France, August 31-September 3, 2010. Proceedings*, pages 222–236, 2010.

[19] Luís Caires, Frank Pfenning, and Bernardo Toninho. Linear logic propositions as session types. *Mathematical Structures in Computer Science*, 26(3):367–423, 2016.

[20] Marco Carbone, Fabrizio Montesi, Carsten Schürmann, and Nobuko Yoshida. Multiparty session types as coherence proofs. *Acta Inf.*, 54(3):243–269, 2017.

[21] Simon Castellan and Nobuko Yoshida. Two sides of the same coin: session types and game semantics: a synchronous side and an asynchronous side. *PACMPL*, 3(POPL):27:1–27:29, 2019.

[22] Gian Luca Cattani, Ian Stark, and Glynn Winskel. Presheaf models for the pi-calculus. In *Category Theory and Computer Science, 7th International Conference, CTCS '97, Santa Margherita Ligure, Italy, September 4-6, 1997, Proceedings*, pages 106–126, 1997.

[23] Gian Luca Cattani and Glynn Winskel. Presheaf models for concurrency. In *Computer Science Logic, 10th International Workshop, CSL '96, Annual Conference of the EACSL, Utrecht, The Netherlands, September 21-27, 1996, Selected Papers*, pages 58–75, 1996.

[24] Pierre Clairambault and Marc de Visme. Full abstraction for the quantum lambda-calculus. *Proc. ACM Program. Lang.*, 4(POPL):63:1–63:28, 2020.

[25] JRB Cockett, M Hasegawa, and RAG Seely. Coherence of the double involution on *-autonomous categories. *Theory and Applications of Categories*, 17(2):17–29, 2006.

[26] Rust Community. Rust Programming Language. `https://www.rust-lang.org/`. Accessed: 2020-12-04.

[27] Haskell Curry and Robert Feys. *Combinatory Logic. I.* North Holland Publishing Company, 1958.

[28] Vincent Danos, Jean-Baptiste Joinet, and Harold Schellinx. Lkq and lkt: sequent calculi for second order logic based upon dual linear decompositions of classical implication. *Advances in Linear Logic*, 222:211–224, 1995.

[29] Ornela Dardha and Simon J. Gay. A new linear logic for deadlock-free session-typed processes. In *Foundations of Software Science and Computation Structures - 21st International Conference, FOSSACS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings*, pages 91–109, 2018.

[30] Ornela Dardha, Elena Giachino, and Davide Sangiorgi. Session types revisited. *Inf. Comput.*, 256:253–286, 2017.

[31] Rowan Davies and Frank Pfenning. A modal analysis of staged computation. *J. ACM*, 48(3):555–604, 2001.

[32] Rocco De Nicola and Matthew Hennessy. Testing equivalence for processes. In *Automata, Languages and Programming, 10th Colloquium, Barcelona, Spain, July 18-22, 1983, Proceedings*, pages 548–560, 1983.

[33] Rocco De Nicola and Matthew Hennessy. Testing equivalences for processes. *Theor. Comput. Sci.*, 34:83–133, 1984.

[34] Thomas Ehrhard and Laurent Regnier. Uniformity and the taylor expansion of ordinary lambda-terms. *Theor. Comput. Sci.*, 403(2-3):347–372, 2008.

[35] Matthias Felleisen, Daniel P. Friedman, Eugene E. Kohlbecker, and Bruce F. Duba. A syntactic theory of sequential control. *Theor. Comput. Sci.*, 52:205–237, 1987.

[36] Marcelo P. Fiore, Eugenio Moggi, and Davide Sangiorgi. A fully abstract model for the π-calculus. *Inf. Comput.*, 179(1):76–117, 2002.

[37] Simon Fowler. *Typed concurrent functional programming with channels, actors and sessions.* PhD thesis, University of Edinburgh, UK, 2019.

[38] Simon Fowler, Sam Lindley, and Philip Wadler. Mixing metaphors: Actors as channels and channels as actors. In Peter Müller, editor, *31st European Conference on Object-Oriented Programming, ECOOP 2017, June 19-23, 2017, Barcelona, Spain*, volume 74 of *LIPIcs*, pages 11:1–11:28. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[39] Carsten Führmann and Hayo Thielecke. On the call-by-value CPS transform and its semantics. *Inf. Comput.*, 188(2):241–283, 2004.

[40] Philippa Gardner and Masahito Hasegawa. Types and models for higher-order action calculi. In *Theoretical Aspects of Computer Software, Third International Symposium, TACS '97, Sendai, Japan, September 23-26, 1997, Proceedings*, pages 583–603, 1997.

[41] Simon J. Gay and Vasco Thudichum Vasconcelos. Linear type theory for asynchronous session types. *J. Funct. Program.*, 20(1):19–50, 2010.

[42] David K. Gifford and John M. Lucassen. Integrating functional and imperative programming. In *Proceedings of the 1986 ACM Conference on LISP and Functional Programming, LFP 1986, August 4-6, 1986, Cambridge, Massachusetts, USA*, pages 28–38. ACM, 1986.

[43] Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987.

[44] Jean-Yves Girard. Locus solum: From the rules of logic to the logic of rules. *Math. Struct. Comput. Sci.*, 11(3):301–506, 2001.

[45] Jean-Yves Girard, Andre Scedrov, and Philip J. Scott. Bounded linear logic: A modular approach to polynomial-time computability. *Theor. Comput. Sci.*, 97(1):1–66, 1992.

[46] Jean-Yves Girard, Paul Taylor, and Yves Lafont. *Proofs and types*, volume 7. Cambridge university press Cambridge, 1989.

[47] Masahito Hasegawa. *Models of sharing graphs : a categorical semantics of let and letrec.* PhD thesis, University of Edinburgh, UK, 1997.

[48] Masahito Hasegawa. On traced monoidal closed categories. *Mathematical Structures in Computer Science*, 19(2):217–244, 2009.

[49] John Hatcliff and Olivier Danvy. A generic account of continuation-passing styles. In Hans-Juergen Boehm, Bernard Lang, and Daniel M. Yellin, editors, *Conference Record of POPL'94: 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Portland, Oregon, USA, January 17-21, 1994*, pages 458–471. ACM Press, 1994.

[50] Susumu Hayashi. Adjunction of semifunctors: Categorical structures in nonextensional lambda calculus. *Theor. Comput. Sci.*, 41:95–104, 1985.

[51] Daniel Hirschkoff, Enguerrand Prebet, and Davide Sangiorgi. On the representation of references in the pi-calculus. In Igor Konnov and Laura Kovács, editors, *31st International Conference on Concurrency Theory, CONCUR 2020, September 1-4, 2020, Vienna, Austria (Virtual Conference)*, volume 171 of *LIPIcs*, pages 34:1–34:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[52] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

[53] Kohei Honda. Types for dyadic interaction. In *CONCUR '93, 4th International Conference on Concurrency Theory, Hildesheim, Germany, August 23-26, 1993, Proceedings*, pages 509–523, 1993.

[54] Kohei Honda and Olivier Laurent. An exact correspondence between a typed pi-calculus and polarised proof-nets. *Theor. Comput. Sci.*, 411(22-24):2223–2238, 2010.

[55] Kohei Honda and Mario Tokoro. An object calculus for asynchronous communication. In Pierre America, editor, *ECOOP'91 European Conference on Object-Oriented Programming, Geneva, Switzerland, July 15-19, 1991, Proceedings*, volume 512 of *Lecture Notes in Computer Science*, pages 133–147. Springer, 1991.

[56] Kohei Honda, Vasco Thudichum Vasconcelos, and Makoto Kubo. Language primitives and type discipline for structured communication-based programming. In *Programming Languages and Systems - ESOP'98, 7th European Symposium on Programming, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'98, Lisbon, Portugal, March 28 - April 4, 1998, Proceedings*, pages 122–138, 1998.

[57] Kohei Honda, Nobuko Yoshida, and Martin Berger. Process types as a descriptive tool for interaction - control and the pi-calculus. In *Rewriting and Typed Lambda Calculi - Joint International Conference, RTA-TLCA 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, pages 1–20, 2014.

[58] William A Howard. The formulae-as-types notion of construction. *To HB Curry: essays on combinatory logic, lambda calculus and formalism*, 44:479–490, 1980.

[59] Martin Hyland and Andrea Schalk. Glueing and orthogonality for models of linear logic. *Theor. Comput. Sci.*, 294(1/2):183–231, 2003.

[60] Atsushi Igarashi and Naoki Kobayashi. A generic type system for the pi-calculus. *Theor. Comput. Sci.*, 311(1-3):121–163, 2004.

[61] André Joyal, Mogens Nielsen, and Glynn Winskel. Bisimulation from open maps. *Inf. Comput.*, 127(2):164–185, 1996.

[62] André Joyal, Ross Street, and Dominic Verity. Traced monoidal categories. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 119, pages 447–468. Cambridge Univ Press, 1996.

[63] Gregory M Kelly and Miguel L Laplaza. Coherence for compact closed categories. *Journal of Pure and Applied Algebra*, 19:193–213, 1980.

[64] Aleks Kissinger. *Pictures of processes : automated graph rewriting for monoidal categories and applications to quantum computing.* PhD thesis, University of Oxford, UK, 2011.

[65] Naoki Kobayashi. A partially deadlock-free typed process calculus. *ACM Trans. Program. Lang. Syst.*, 20(2):436–482, 1998.

[66] Naoki Kobayashi. A type system for lock-free processes. *Inf. Comput.*, 177(2):122–159, 2002.

[67] Naoki Kobayashi. Type systems for concurrent programs. In *Formal Methods at the Crossroads. From Panacea to Foundational Support, 10th Anniversary Colloquium of UNU/IIST, the International Institute for Software Technology of The United Nations University, Lisbon, Portugal, March 18-20, 2002, Revised Papers*, pages 439–453, 2002.

[68] Naoki Kobayashi. A new type system for deadlock-free processes. In Christel Baier and Holger Hermanns, editors, *CONCUR 2006 - Concurrency Theory, 17th International Conference, CONCUR 2006, Bonn, Germany, August 27-30, 2006, Proceedings*, volume 4137 of *Lecture Notes in Computer Science*, pages 233–247. Springer, 2006.

[69] Naoki Kobayashi, Benjamin C. Pierce, and David N. Turner. Linearity and the pi-calculus. *ACM Trans. Program. Lang. Syst.*, 21(5):914–947, 1999.

[70] Wen Kokke, Fabrizio Montesi, and Marco Peressotti. Better late than never: a fully-abstract semantics for classical processes. *Proc. ACM Program. Lang.*, 3(POPL):24:1–24:29, 2019.

[71] Wen Kokke, J. Garrett Morris, and Philip Wadler. Towards races in linear logic. In Hanne Riis Nielson and Emilio Tuosto, editors, *Coordination Models and Languages - 21st IFIP WG 6.1 International Conference, COORDINATION 2019, Held as Part of the 14th International Federated Conference on Distributed Computing Techniques, DisCoTec 2019, Kongens Lyngby, Denmark, June 17-21, 2019, Proceedings*, volume 11533 of *Lecture Notes in Computer Science*, pages 37–53. Springer, 2019.

[72] Ugo Dal Lago, Marc de Visme, Damiano Mazza, and Akira Yoshimizu. Intersection types and runtime errors in the pi-calculus. *PACMPL*, 3(POPL):7:1–7:29, 2019.

[73] James Laird. A categorical semantics of higher order store. *Electr. Notes Theor. Comput. Sci.*, 69:209–226, 2002.

[74] Jim Laird. A game semantics of the asynchronous π-calculus. In *CONCUR 2005 - Concurrency Theory, 16th International Conference*, pages 51–65, 2005.

[75] Joachim Lambek. Deductive systems and categories i. syntactic calculus and residuated categories. *Math. Syst. Theory*, 2(4):287–318, 1968.

[76] Joachim Lambek. From lambda-calculus to cartesian closed categories. *To HB Curry: essays on combinatory logic, lambda calculus and formalism*, pages 375–402, 1980.

[77] Joachim Lambek and Philip J Scott. *Introduction to higher-order categorical logic*, volume 7. Cambridge University Press, 1988.

[78] Paul Blain Levy, John Power, and Hayo Thielecke. Modelling environments in call-by-value programming languages. *Inf. Comput.*, 185(2):182–210, 2003.

[79] Sam Lindley and J. Garrett Morris. A semantics for propositions as sessions. In *Programming Languages and Systems - 24th European Symposium on Programming, ESOP 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, pages 560–584, 2015.

[80] Saunders Mac Lane. *Categories for the working mathematician*, volume 5. Springer Science & Business Media, 1978.

[81] Maria Emilia Maietti, Paola Maneggia, Valeria de Paiva, and Eike Ritter. Relating categorical semantics for intuitionistic linear logic. *Appl. Categorical Struct.*, 13(1):1–36, 2005.

[82] Damiano Mazza. The true concurrency of differential interaction nets. *Mathematical Structures in Computer Science*, 28(7):1097–1125, 2018.

[83] Damiano Mazza, Luc Pellissier, and Pierre Vial. Polyadic approximations, fibrations and intersection types. *PACMPL*, 2(POPL):6:1–6:28, 2018.

[84] Paul-André Melliès. Categorical semantics of linear logic. *Panoramas et syntheses*, 27:15–215, 2009.

[85] Paul-André Melliès. Ribbon tensorial logic. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 689–698. ACM, 2018.

[86] Paul-André Melliès. Categorical combinatorics of scheduling and synchronization in game semantics. *PACMPL*, 3(POPL):23:1–23:30, 2019.

[87] Paul-André Melliès and Nicolas Tabareau. Resource modalities in tensor logic. *Ann. Pure Appl. Log.*, 161(5):632–653, 2010.

[88] Massimo Merro. *Locality in the π-calculus and applications to distributed objects.* PhD thesis, École Nationale Supérieure des Mines de Paris, 2000.

[89] Robin Milner. *Communication and concurrency.* PHI Series in computer science. Prentice Hall, 1989.

[90] Robin Milner. Functions as processes. *Mathematical Structures in Computer Science*, 2(2):119–141, 1992.

[91] Robin Milner. Action calculi, or syntactic action structures. In Andrzej M. Borzyszkowski and Stefan Sokolowski, editors, *Mathematical Foundations of Computer Science 1993, 18th International Symposium, MFCS'93, Gdansk, Poland, August 30 - September 3, 1993, Proceedings*, volume 711 of *Lecture Notes in Computer Science*, pages 105–121. Springer, 1993.

[92] Robin Milner. Higher-order action calculi. In *Computer Science Logic, 7th Workshop, CSL '93, Swansea, United Kingdom, September 13-17, 1993, Selected Papers*, pages 238–260, 1993.

[93] Robin Milner. The polyadic π-calculus: a tutorial. In FriedrichL. Bauer, Wilfried Brauer, and Helmut Schwichtenberg, editors, *Logic and Algebra of Specification*, volume 94 of *NATO ASI Series*, pages 203–246. Springer Berlin Heidelberg, 1993.

[94] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, I. *Inf. Comput.*, 100(1):1–40, 1992.

[95] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, II. *Inf. Comput.*, 100(1):41–77, 1992.

[96] Robin Milner and Davide Sangiorgi. Barbed bisimulation. In *Automata, Languages and Programming, 19th International Colloquium, ICALP92, Vienna, Austria, July 13-17, 1992, Proceedings*, pages 685–695, 1992.

[97] Eugenio Moggi. Computational lambda-calculus and monads. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science (LICS '89), Pacific Grove, California, USA, June 5-8, 1989*, pages 14–23, 1989.

[98] Eugenio Moggi. Notions of computation and monads. *Inf. Comput.*, 93(1):55–92, 1991.

[99] Andrzej S. Murawski and Nikos Tzevelekos. Nominal game semantics. *Foundations and Trends in Programming Languages*, 2(4):191–269, 2016.

[100] C.-H. Luke Ong and Charles A. Stewart. A curry-howard foundation for functional computation with control. In Peter Lee, Fritz Henglein, and Neil D. Jones, editors, *Conference Record of POPL'97: The 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Papers Presented at the Symposium, Paris, France, 15-17 January 1997*, pages 215–227. ACM Press, 1997.

[101] Dominic A. Orchard and Nobuko Yoshida. Effects as sessions, sessions as effects. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 568–581, 2016.

[102] Luca Padovani. Deadlock and lock freedom in the linear π-calculus. In Thomas A. Henzinger and Dale Miller, editors, *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, pages 72:1–72:10. ACM, 2014.

[103] Jorge A. Pérez, Luís Caires, Frank Pfenning, and Bernardo Toninho. Linear logical relations for session-based concurrency. In Helmut Seidl, editor, *Programming Languages and Systems - 21st European Symposium on Programming, ESOP 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, volume 7211 of *Lecture Notes in Computer Science*, pages 539–558. Springer, 2012.

[104] Tomas Petricek, Dominic A. Orchard, and Alan Mycroft. Coeffects: a calculus of context-dependent computation. In Johan Jeuring and Manuel M. T. Chakravarty, editors, *Proceedings of the 19th ACM SIGPLAN international conference on Functional programming, Gothenburg, Sweden, September 1-3, 2014*, pages 123–135. ACM, 2014.

[105] Simon L. Peyton Jones, Andrew D. Gordon, and Sigbjorn Finne. Concurrent Haskell. In *Conference Record of POPL'96: The 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Papers Presented at the Symposium, St. Petersburg Beach, Florida, USA, January 21-24, 1996*, pages 295–308, 1996.

[106] Benjamin C. Pierce and Davide Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–453, 1996.

[107] Gordon D. Plotkin. Call-by-name, call-by-value and the lambda-calculus. *Theor. Comput. Sci.*, 1(2):125–159, 1975.

[108] Gordon D. Plotkin and John Power. Algebraic operations and generic effects. *Applied Categorical Structures*, 11(1):69–94, 2003.

[109] John Power. Elementary control structures. In *CONCUR '96, Concurrency Theory, 7th International Conference, Pisa, Italy, August 26-29, 1996, Proceedings*, pages 115–130, 1996.

[110] John Power and Edmund Robinson. Premonoidal categories and notions of computation. *Mathematical Structures in Computer Science*, 7(5):453–468, 1997.

[111] John Power and Hayo Thielecke. Closed Freyd- and kappa-categories. In *Automata, Languages and Programming, 26th International Colloquium, ICALP'99*, pages 625–634, 1999.

[112] John H. Reppy. CML: A higher-order concurrent language. In *Proceedings of the ACM SIGPLAN'91 Conference on Programming Language Design and Implementation (PLDI), Toronto, Ontario, Canada, June 26-28, 1991*, pages 293–305, 1991.

[113] Emily Riehl. *Category Theory in Context.* Aurora: Modern Math Originals. Dover Publications, 2016.

[114] Ken Sakayori and Takeshi Tsukada. A categorical model of an **i**/**o**-typed π-calculus. In *Programming Languages and Systems - 28th European Symposium on Programming, ESOP 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings*, pages 640–667, 2019.

[115] Davide Sangiorgi. *Expressing mobility in process algebras : first-order and higher-order paradigms.* PhD thesis, University of Edinburgh, UK, 1993.

[116] Davide Sangiorgi. π-Calculus, internal mobility, and agent-passing calculi. *Theor. Comput. Sci.*, 167(1&2):235–274, 1996.

[117] Davide Sangiorgi. From λ to π; or, Rediscovering continuations. *Mathematical Structures in Computer Science*, 9(4):367–401, 1999.

[118] Davide Sangiorgi. Asynchronous process calculi: the first- and higher-order paradigms. *Theor. Comput. Sci.*, 253(2):311–350, 2001.

[119] Davide Sangiorgi and David Walker. *The π-calculus—A Theory of Mobile Processes.* Cambridge University Press, 2001.

[120] R.A.G. Seely. Linear logic, *-autonomous categories and cofree coalgebras. In *In Categories in Computer Science and Logic*, pages 371–382. American Mathematical Society, 1989.

[121] Peter Selinger. Control categories and duality: on the categorical semantics of the lambda-mu calculus. *Mathematical Structures in Computer Science*, 11(2):207–260, 2001.

[122] Alex K. Simpson and Gordon D. Plotkin. Complete axioms for categorical fixed-point operators. In *15th Annual IEEE Symposium on Logic in Computer Science, Santa Barbara, California, USA, June 26-29, 2000*, pages 30–41. IEEE Computer Society, 2000.

[123] Ian Stark. A fully abstract domain model for the π-calculus. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, USA, July 27-30, 1996*, pages 36–42, 1996.

[124] Jean-Pierre Talpin and Pierre Jouvelot. The type and effect discipline. *Inf. Comput.*, 111(2):245–296, 1994.

[125] Hayo Thielecke. *Categorical structure of continuation passing style.* PhD thesis, University of Edinburgh, UK, 1997.

[126] Bernardo Toninho, Luís Caires, and Frank Pfenning. Functions as session-typed processes. In *Foundations of Software Science and Computational Structures - 15th International Conference, FOSSACS 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, pages 346–360, 2012.

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

[128] Takeshi Tsukada, Kazuyuki Asada, and C.-H. Luke Ong. Generalised species of rigid resource terms. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12, 2017.

[129] Takeshi Tsukada and C.-H. Luke Ong. Plays as resource terms via non-idempotent intersection types. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 237–246, 2016.

[130] David N. Turner. *The polymorphic Pi-calculus : theory and implementation.* PhD thesis, University of Edinburgh, UK, 1996.

[131] Philip Wadler. Comprehending monads. *Math. Struct. Comput. Sci.*, 2(4):461–493, 1992.

[132] Philip Wadler. Propositions as sessions. In Peter Thiemann and Robby Bruce Findler, editors, *ACM SIGPLAN International Conference on Functional Programming, ICFP'12, Copenhagen, Denmark, September 9-15, 2012*, pages 273–286. ACM, 2012.

[133] Philip Wadler. Propositions as sessions. *J. Funct. Program.*, 24(2-3):384–418, 2014.

[134] David Walker. Objects in the pi-calculus. *Inf. Comput.*, 116(2):253–271, 1995.

[135] Glynn Winskel and Mogens Nielsen. Presheaves as transition systems. In *Partial Order Methods in Verification, Proceedings of a DIMACS Workshop, Princeton, New Jersey, USA, July 24-26, 1996*, pages 129–140, 1996.

[136] Nobuko Yoshida. Minimality and separation results on asynchronous mobile processes – representability theorems by concurrent combinators. *Theor. Comput. Sci.*, 274(1-2):231–276, 2002.

Index