



UNIVERSITA'
DI TORINO

UNIVERSITE' PARIS
DIDEROT (Paris 7)



UNIVERSITÀ
I T A L O
FRANCESE

SCUOLA DI DOTTORATO
IN SCIENZE ED
ALTA TECNOLOGIA
INDIRIZZO INFORMATICA

ÉCOLE DOCTORALE
SCIENCES MATHÉMATIQUES
DE PARIS CENTRE
SPÉCIALITÉ INFORMATIQUE

MAURO PICCOLO

Linearity and Beyond in Denotational Semantics

Thèse dirigée par

Pierre Louis CURIEN / Simona RONCHI DELLA ROCCA

Soutenue le 9 Decembre 2009

JURY

M.	Pierre Louis	CURIEN	Université Paris VII
M.	Martin	HOFMANN	LMU, München
Mme	Simona	RONCHI DELLA ROCCA	Università di Torino
Mme	Nobuko	YOSHIDA	Imperial College London

ABSTRACT

In this thesis we propose an exploration of the notion of linearity in its many respects. By the term linearity, we in fact refer to different concepts, more or less related to the ideas underlying Girards Linear Logic. More specifically, in this thesis we present the language $\mathcal{S}\ell\text{PCF}$, a semantically linear programming language. $\mathcal{S}\ell\text{PCF}$ is linear in a denotational sense since it is conceived to be the syntactical counterpart of Linear Stable Functions among Coherence Spaces. We study some denotational models of $\mathcal{S}\ell\text{PCF}$, we show a full abstraction result and we discuss some extensions of this language. We present a syntactical model of $\mathcal{S}\ell\text{PCF}$ by giving a faithful translation of this programming language into a linear process language, called ℓinProc . We continue our study of linearity in process languages by proposing Girard's Ludics as an abstract mathematical tool to study some important properties of processes, like liveness or deadlock-freeness. In particular we give a semantics model of linear π -calculus into a modified version of Ludics, conceived to validate the Mix-rule. Finally, we go beyond linearity by giving a logical characterization of stability using intersection types. We define two intersection type assignment systems for λ -calculus, parametric with respect to a coherence relation between types. We show that such systems give a logical characterization of two interesting classes of λ -calculus models.

RÉSUMÉ

Dans cette thèse, nous proposons une exploration de la notion de linéarité, selon plusieurs aspects. Par le terme linéarité nous entendons les différents concepts, plus au moins liés aux idées qui sont sous-jacentes à la Logique Linéaire de Girard. Plus précisément, dans cette thèse nous présentons le langage \mathcal{SLPCF} , un langage de programmation sémantiquement linéaire. \mathcal{SLPCF} est dénotationnellement linéaire car il est pensé pour être la contre-partie syntaxique des Fonctions Linéaires Stables parmi les Espaces Cohérents. Nous étudions des modèles dénotationnels de \mathcal{SLPCF} , nous montrons un résultat de complète abstraction et nous discutons des extensions de ce langage. Nous présentons un modèle syntaxique de \mathcal{SLPCF} , en donnant un codage fidèle de ce langage de programmation dans un langage de processus appelé *linProc*. Nous continuons notre étude sur la linéarité dans les langages de processus, en proposant la Ludique de Girard comme un outil mathématique abstrait pour étudier des propriétés importantes des processus, comme la "liveness" ou l'absence de deadlock. Nous donnons un modèle sémantique du pi-calcul linéaire dans une version modifiée de la Ludique, pensée pour valider la règle du Mix. Enfin, nous allons au delà de la linéarité en donnant une caractérisation logique de la stabilité, en utilisant les types intersections. Nous définissons deux systèmes d'assignations de type pour le lambda-calcul, qui sont paramétriques sur une relation de cohérence entre types. Nous prouvons que ces systèmes donnent une caractérisation logique de deux classes intéressantes de modèles du λ -calcul.

SOMMARIO

In questa tesi noi proponiamo un'esplorazione della nozione di linearità nei suoi numerosi aspetti. Per il termine linearità, noi ci riferiamo infatti ai differenti concetti, più o meno correlati alle idee soggiacenti la Logica Lineare di Girard. Più specificatamente, in questa tesi noi presentiamo il linguaggio \mathcal{SLPCF} , un linguaggio di programmazione semanticamente lineare. \mathcal{SLPCF} è lineare in senso denotazionale poiché è concepito per essere la controparte sintattica delle Funzioni Lineari Stabili tra Spazi Coerenti. Noi studiamo alcuni modelli denotazionali di \mathcal{SLPCF} , mostriamo un risultato di completa astrazione e discutiamo alcune estensioni di questo linguaggio. Noi presentiamo un modello sintattico di \mathcal{SLPCF} dando una traduzione fedele di questo linguaggio di programmazione in un linguaggio dei processi chiamato $\mathit{linProc}$. Noi continuiamo il nostro studio della linearità nei calcoli dei processi proponendo la Ludica di Girard come uno strumento matematico astratto per studiare alcune proprietà importanti dei processi come la liveness o l'assenza di deadlock. In particolare noi diamo un modello semantico del π -calcolo lineare in una versione modificata della Ludica, concepita per validare la regola del Mix. Infine, noi andiamo oltre la linearità, dando una caratterizzazione logica della stabilità usando i tipi intersezione. Noi definiamo due sistemi di assegnazione di tipo, parametrici rispetto ad una relazione di coerenza fra tipi. Noi mostriamo che questi sistemi danno una caratterizzazione logica di due interessanti classi di modelli del λ -calcolo.

Ringraziamenti

Ringrazio tutti coloro che mi hanno aiutato e sostenuto in tutti questi tre anni.

Ringrazio Simona Ronchi Della Rocca che ha accettato di essere la mia direttrice di tesi italiana. La sua guida e suoi insegnamenti sono stati per me continua fonte di ispirazione. Ringrazio Pierre Louis Curien che ha accettato di essere il mio direttore di tesi francese e che mi ha accolto all'interno del laboratorio PPS, di cui lui all'epoca era direttore.

Vorrei ringraziare Martin Hofmann e Nobuko Yoshida per aver accettato di fare i revisori di questa tesi.

Vorrei ringraziare tutti i membri del gruppo di λ -calcolo dell'Università di Torino. In modo particolare ringrazio Stefano Berardi, Ugo de' Liguoro, Mariangiola Dezani Ciancaglini, Mario Coppo, Luca Roversi per le numerose chiacchierate che ho avuto con loro dalle quali ho appreso molto; ringrazio inoltre Viviana Bono, Sara Capecchi, Ferruccio Damiani, Luca Fossati, Marco Gaboardi, Elena Giachino (con cui ho condiviso l'esperienza parigina), Luca Roversi, Angelo Troina, Luca Vercelli, Michele Pagani, Alexis Saurin (che mi ha aiutato per le traduzioni in francese) e tutti gli altri membri del gruppo λ .

Vorrei ringraziare tutti i membri del laboratorio PPS di Parigi e in modo particolare Claudia Faggian e Daniele Varacca.

Un ringraziamento particolare va a Luca Paolini, per le numerose discussioni scientifiche così come per le serate passate davanti a una bottiglia di vino a discutere di λ -calcolo. Nonostante le discussioni che abbiamo avuto siano state talvolta faticose, le acute osservazioni che mi ha prodigato sono state estremamente proficue nella mia attività.

Infine ringrazio tutti i miei compagni dottorandi che ho incontrato al Dipartimento di Informatica: in particolare Dino Ienco, Elena Roglia, Serena Villata, Stefano Grenna, Roberto Furnari e tutti gli altri dottorandi del mio ciclo.

Contents

1	Introduction	9
1.1	The linear big bang	10
1.2	Linearity	12
1.3	... and beyond	14
1.4	Structure of the thesis	15
2	Technical Preliminaries	17
2.1	Category Theory	17
2.1.1	Categories, Functors and Natural Transformations	17
2.1.2	Adjonctions and (Co)-Monads	21
2.1.3	Cartesian Closed Categories	24
2.1.4	Monoidal categories	27
2.2	Functional programming languages and calculi	30
2.2.1	λ -calculi and simply typed λ -calculi	31
2.2.2	The programming language PCF	33
2.3	Denotational Models	35
2.3.1	Categorical Model	35
2.3.2	Scott Domains	40
2.3.3	Coherence Spaces	43
I	Linearity in Denotational Semantics	47
3	Semantically Linear PCF	49
3.1	Semantically Linear λ -calculus	51
3.1.1	Term rewriting system	51
3.1.2	Categorical model of $\mathcal{S}\ell\lambda$ -calculus	54
3.2	Semantically Linear PCF	65
3.3	A Scott-Domain semantics of $\mathcal{S}\ell\text{PCF}$	69
3.3.1	The category StrictBcdom	69
3.3.2	Adequacy and correctness	71
3.4	A coherence space semantics for $\mathcal{S}\ell\text{PCF}$	73
3.4.1	More on Coherence Spaces	74
3.4.2	LinCoh is a model for $\mathcal{S}\ell\text{PCF}$	75
3.4.3	Adequacy and Correctness	78
3.4.4	Token Definability	79

3.5	Linear Extensions of $\mathcal{S}\ell\text{PCF}$	80
3.5.1	Second Order Gustave Or	81
3.5.2	The <i>which?</i> -operator	86
3.5.3	Towards finite definability	90
3.6	Conclusions	93
4	A Process Model for Linear Programs	96
4.1	A Linear Process Calculus	98
4.1.1	Typing System	100
4.1.2	<i>linProc</i> Reductions and Congruences	102
4.2	Processing Programs	106
4.3	Soundness and correctness	110
4.4	Conclusion	112
5	Ludics Strategies and the π-calculus	114
5.1	Event Structures	116
5.1.1	A category of event structures	117
5.1.2	Construction on event structures	118
5.1.3	A cpo of event structures	118
5.1.4	Confusion free event structures	119
5.2	Ludics	120
5.2.1	Loci, actions and designs	120
5.2.2	Normalisation	123
5.2.3	A category of multidesigns	128
5.2.4	Analytical Theorems	130
5.2.5	Technical characterisation of parallel composition	133
5.3	Finitary linear π -calculus	139
5.3.1	Typing system	141
5.3.2	Observational equivalence	143
5.4	Interpreting finitary linear π -calculus	143
5.4.1	Defining the model	143
5.4.2	Adequacy and correctness	145
5.4.3	Completeness and full abstraction	147
5.5	Extending the finitary π -calculus	148
5.5.1	Recursion and linearity	150
5.5.2	Encoding data and functions	151
5.6	Interpreting linear π -calculus	155
5.6.1	Defining the model	155
5.7	Conclusion	157
II	Behind Linearity using Intersection Types	158
6	Logical Semantics for Stability	160

- 6.1 Models of pure untyped λ -calculus 162
- 6.2 Clique Models 164
- 6.3 Stable λ -models 172
- 6.4 Conclusions and comparison with related works 176

1 Introduction

Formal semantics of programming languages is the area of computer science, that aims at studying in a mathematical way the meaning of the constructs of a given programming language. There are two methods for giving semantics description of a programming language, the *operational approach* and the *denotational approach*.¹

The operational approach consists in defining an abstract machine with several components and set of primitive instructions. The semantics of a program is defined in terms of the operations it induces in this machine. Following this point of view, two programs are equivalent according to the operational semantics whenever they are interchangeable in all contexts, without affecting the observational outcome of computation. The denotational approach consists instead in giving a “semantic valuation function” which maps syntactic constructs of the program to abstract values (numbers, truth values, function, etc.) which they denote. The main feature of such valuation function is to be *compositional*: the value denoted by a construct is specified in terms of the values denoted by its syntactic sub-components. Denotational semantics focuses its attention on the values which may denote a program; this means that two programs are denotationally equivalent if they have the same denotation in the model itself. The main objective of denotational semantics is to abstract away from some operational aspects of computation, in order to capture essential properties of programs, in particular the abstract property of a mathematical object corresponding to the operational notions of deterministic and/or sequential computation.

The precision of denotational semantics is measured w.r.t. a particular operational semantics of a programming language, by comparing the two equivalences they induce in the language. Denotational semantics is *correct* w.r.t. operational semantics, when two operationally distinct programs are never interpreted on the same denotational object by the semantic valuation function. Denotational semantics is said to be *complete* w.r.t. operational semantics, when two operationally equal programs are interpreted on the same denotational object. Finally, denotational semantics is *fully abstract* when the operational equivalence coincides with the denotational one.

An important contribution in denotational semantics was given by D. Scott [Scott, 1972]. He gave a model for the untyped λ -calculus, which is fully abstract w.r.t. a well established operational semantics. His model was focused on studying *extensional* properties of programs (i.e. the relation between input and output of programs), abstracting completely from their operational aspects.

Denotational semantics is concerned with modelling datatypes as sets with structure and procedures (methods) as mathematical functions preserving this structure. In

¹There is also a third method, which is called *axiomatic*, but it is not treated in this thesis

particular, in Scott-style denotational semantics, all data managed by programs are structured in (Scott)-domains, which are particular kinds of partial orders. Data can be either finite structures (like a number or a program defined on finitely many finite inputs) called *finite elements*, or infinite ones (like a program defined on infinitely many finite inputs).

Programs are interpreted as functions between two domains, having the property of being (*Scott*)-*continuous*. A function is continuous when it is monotonic and it always observes a finite part of its input, to give a particular output. From a more domain theoretic point of view, a function between two domains is continuous when the suprema of all directed sets commute with respect to it. Continuous functions form a Scott-domain where they are *extensionally ordered* making the category of Scott-domains and Continuous Functions cartesian closed.

Unfortunately, Scott-continuity does not capture the operational notion of sequentiality [Milner, 1977], thus further refinements of Scott semantics were proposed. A remarkable refinement was the one given by Berry, [Berry, 1978], who introduced the notion of *stable function*. A function is stable when it is continuous and for every “finite” part of its output, there exists a unique minimum “finite” part of input, which causes it.

1.1 The linear big bang

In 1986, a **linear big bang** happened, when J.Y. Girard studied a denotational semantics for System F, a polymorphic extension of typed λ -calculus [Girard, 1986]. In this work he provides a further refinement of stable semantics, by introducing binary qualitative domains which became Coherence Spaces in [Girard, 1987]. These spaces provide a pleasant decomposition of stable functions via linear functions and exponential domain constructors. From this, he built up Linear Logic [Girard, 1987] where structural rules (weakening and contraction) got for the first time a full logical status, with associated exponential connectives. What differentiates linear logic from previous logical system is that it is a *resource sensitive* logic, where intuitively, formulae are seen as resources and exponential modalities give account on the way of using them: weakening of a formula corresponds to resource discarding, while contraction of a formula corresponds to resource duplication.

Linear Logic gives surely an important contribution for the research in theoretical computer science. In fact, since its inception, Linear Logic has inspired the introduction of many formal languages with different computational notions: resource-conscious evaluation, categorical language, implicit computational complexity, and so on. But also the future developments in denotational semantics - and in particular game semantics - have been influenced by this new logic, which focuses on dynamical aspects of computation, rather than on static ones. Game semantics aims in overcoming the defects of classical denotational model, trying to focus to the interactive aspects of computation. In game semantics, computation is described in term of a game between two players, Player (or Eloise), which is the program itself and Opponent (or

Abelard), which is the execution environment. A strategy, denoting a program is the set of all possible plays between the program itself and the environment. So game models are not classical functional models but *trace models*.

The formal connection between Linear Logic and Game Semantics was established in the work of A. Blass [Blass, 1992]. This work inspired many authors in studying game semantics as a model for computation, by leading to the solution of the full abstraction problem for PCF [Abramsky et al., 2000, Hyland and Ong, 2000], a well known open problem posed by Milner in [Milner, 1977].

The most important aspect, pointed out by Girard, is surely the *decomposition of the Intuitionistic implication* $A \Rightarrow B$ into two novel connectives. The first one is the *linear implication* $A \multimap B$, which claims the consumption of the hypothesis A to obtain the conclusion B , while the second one is a modal connective $!A$, which claims that A can be weakened or contracted finitely many times and that from $!A$ it is possible to derive linearly A (the so called *dereliction*). We have

$$A \Rightarrow B = !A \multimap B$$

This analysis pointed out the notion of *linearity* in computation. However, linearity has many facets and it is not so simple as it may appear on a first sight. In fact linearity in functional programming languages and λ -calculi can be considered in many respects and at least five kinds of linearity emerge in the literature: they are *typing linearity*, *syntactical linearity*, *denotational linearity*, *reduction linearity* and *operational linearity*.

The simplest notion of linearity is *syntactical linearity*. Syntactical linearity is the property enjoyed by terms of a programming language or a calculus and it claims a linear use of variables in terms, i.e. in a term each λ abstraction binds exactly one variable occurrence. All terms of linear λ -calculus are syntactically linear and it is called linear because each linear λ -term normalizes in a number of step linear in its length [Hindley, 1989].

Typing linearity is the property enjoyed by a logical system or a typed calculus whose typing rules do not admit weakening and contraction as primitive rules. The Multiplicative Additive fragment of Linear Logic is a logical system enjoying typing linearity; notice that, in this system, a limited form of contraction is allowed and it is present in the additive connectives. Some typed calculi enjoying typing linearity can be found in [Alves et al., 2008, Mackie, 1994], in which additive formulae are used to type a conditional construct.

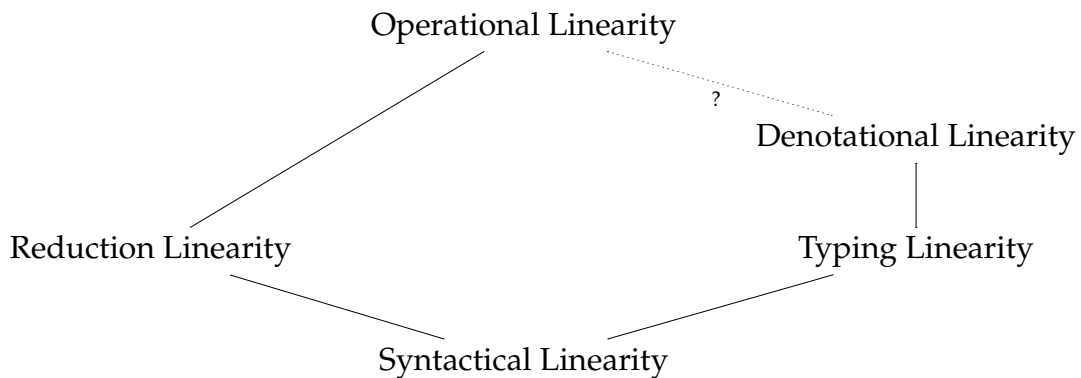
Denotational linearity is the property enjoyed by terms of a programming language whose denotation in a correct denotational model is a linear object. From a domain theoretic point of view, a linear object is a *linear function*, i.e. a continuous function in which the suprema of all bounded subset commute, while from a game semantics point of view, a linear object is a *linear strategy* i.e. a strategy in which no repetitions of the same move in a play is allowed. All terms of \mathcal{SLPCF} [Gaboardi and Paolini, 2007, Paolini and Piccolo, 2008] are denotationally linear.

Reduction linearity is the property enjoyed by terms of a calculus, whose redexes are not duplicated during reduction, for all possible reduction strategies. Simple

terms of λ -calculus [Klop, 2007] enjoy reduction linearity. Finally *operational linearity* is a relaxation of reduction linearity which claims the non-duplication of redexes, following a particular reduction strategy.

A syntactically linear term of a (λ -calculus like) language is also typing linear and denotationally linear. However a typing linear term is not necessarily syntactically linear because typing linearity allows a limited form of weakening and contraction of variables. For example if we consider the linear conditional construct of \mathcal{SLPCF} (denoted with lif), we have that the term $\lambda f. \text{lif } x \ f3 \ f5$ is typing linear (we can type it using an implicit additive contraction) but it is not syntactically linear since f appears twice. Finally, observe that the property of being syntactically linear is not preserved by operational equality (in particular by expansion) while denotational linearity is preserved by operational equality. We can use this fact to build the following example: if $I = \lambda x.x$, then the term $(\lambda x.xx)I$ is denotationally linear (since it is denotationally equal to the identity), but it is not syntactically linear.

Under the assumption of having the β rule as the only rewriting rule of the calculus, we can also observe that a syntactically linear term does not duplicate redexes during reduction, by construction. Thus a syntactically linear term is also reduction linear. However, a reduction linear term is not necessarily syntactically linear. To see this consider any term N which does not contain any redex; then $(\lambda x.xx)N$ is a reduction linear term but not syntactically linear. Finally, reduction linearity and denotational linearity are two incomparable notions. We can reuse the example above to say that there may be reduction linear terms that are not denotationally linear. To see that there are denotationally linear terms which are not reduction linear, consider the term $(\lambda x.xx)(II)$; it is denotationally linear, because it is denotationally equal to the identity, but it is not reduction linear, because we can choose to reduce the outermost redex and this duplicates the innermost one. Finally, we conjecture that denotationally linear terms are also operationally linear. The following diagram summarizes the comparisons among different notions of linearity.



1.2 Linearity

In this thesis we propose an exploration of the notion of linearity, with particular attention to its denotational aspects. Our choice is motivated from the fact that this

aspect of linearity was not so studied in the literature, except for works studying linearity from a categorical point of view.

Our analysis starts with the study of denotational linearity in a sequential setting. Our point of departure is the language $\mathcal{S}\ell\text{PCF}$ (acronym for “Semantically Linear PCF”), introduced in [Gaboardi and Paolini, 2007, Paolini and Piccolo, 2008]. $\mathcal{S}\ell\text{PCF}$ is not linear in a syntactical sense, namely some kind of variables may appear more than once. However this language is linear in a denotational sense, since it was conceived to be the syntactical counterpart of Linear Stable Functions among Coherence Spaces. In this thesis, we propose an abstract notion of model of $\mathcal{S}\ell\text{PCF}$, in which terms of $\mathcal{S}\ell\text{PCF}$ are seen as syntactical description of morphisms of suitable symmetric monoidal closed categories. This categorical vision allows us to study denotational linearity in a more wider sense. We show that there are at least two interesting concrete models of $\mathcal{S}\ell\text{PCF}$, which are instance of our abstract definition and these models are both adequate with respect to the operational semantics of $\mathcal{S}\ell\text{PCF}$. More specifically, one model is built in the Scott Domain setting while the other model is built in the Coherence Space setting. We show a full abstraction result for the model built in the Coherence Space setting. Then we discuss the significance of our results from the higher type computability point of view, by addressing the universality problem.

Our analysis continues with the issue of identifying those processes which describe the execution of $\mathcal{S}\ell\text{PCF}$ programs. For this purpose, we introduce the process language linProc , i.e. a typed process calculus based on the calculus of solos [Laneve and Victor, 2003]. Our purpose is twofold. We want to describe in a faithful way all evaluation strategies of $\mathcal{S}\ell\text{PCF}$ programs, to enable to process redexes of $\mathcal{S}\ell\text{PCF}$ in a parallel way. But also we want to give a syntactical tool to describe in a finitary way those game semantics strategies which are interpretation of $\mathcal{S}\ell\text{PCF}$, in the same as in [Hyland and Ong, 1995, Berger et al., 2001]. There are several motivations behind this work. From a programming language point of view, we provide a tool for study both parallel evaluation strategies and equivalence of programs. From a process calculus point of view, we give a faithful translation from a calculus to an other calculus, thus allowing to simulate in the process calculus all possible evaluation strategies of programs. From a game semantics point of view, we suggest a parallel description of strategies by avoiding useless causality.

The analysis of linearity comes to an end (with respect to this thesis) with the study of denotational linearity in the field of process languages and studying their connection with linear game semantics. The linear game model we choose is an extended version of Girard’s Ludics [Girard, 2001]. Ludics arises from prior works in game semantics, and it has the aim to build a foundational pre-logical framework upon which ordinary logics and type systems are to be built. We choose Ludics since its interactive nature is able to take into account the parallel and concurrent aspects of computation and it is close to the kind of vision of process calculi like π -calculus. More specifically, we study a fragment of linear π -calculus [Yoshida et al., 2004], which has Ludics as correct model. Our aim is to propose Ludics as an abstract mathematical tool to study some important properties of processes, like liveness or deadlock-freeness, which can be considered as the concurrent counterpart of solvability in λ -calculus. In

this perspective, we give an interpretation of processes of the finitary fragment of the linear π -calculus into ludics strategies, being fully abstract with respect to a suitable notion of observational equivalence on processes. Finally we show how to extend this interpretation to a more powerful fragment of linear π -calculus, which allows the presence of recursive definitions.

1.3 ... and beyond

Probably the reader may ask some questions about the meaning of the title of this thesis. In fact the word *beyond* may seem quite mysterious. So, let us explain a little bit the word *beyond*, with respect to the various notions of linearity we introduce in the previous section.

Often in literature, the adjective “linear” is not used in order to denote some strictly linear analysis, but in a wider sense. Consider, for example Linear Logic, that adds to its linear fragment (MALL) some exponential construction allowing weakening and contraction. This last part is the “beyond linearity”. Namely, “going beyond linearity” means that the linear core of a language is opportunely extended in order to obtain more expressiveness by safeguarding the properties of the core.

From a syntactical point of view “going beyond linearity” means that in a linear term, we have some special kind of variables which can be weakened or they may occur many times. From a typing point of view, going “beyond linearity” means simply adding an exponential modality on which weakening and contraction is allowed.

However, from a denotational point of view, it is more difficult to define the meaning of “going beyond linearity”, since we need to use explicitly a notion of semantic exponential. Using a categorical language “going beyond linearity” from a linear object means to build the free commutative comonoid obtained from it, which satisfies certain naturality conditions (see [Melliès, 2003] for more detailed definitions). Concretely, from a domain theoretic point of view, “going beyond linearity” for functions between domains means constructing an exponential domain in which all linear functions between this domain and an other correspond to the continuous functions between the non-exponentiated domain and the other.

In the last part of this thesis, we study how to go “beyond linearity” again from a denotational point of view. In particular, we use intersection types to mimic the kind of semantic exponential construction described above. Intersection types were introduced by Coppo and Dezani [Coppo and Dezani-Ciancaglini, 1980] to increase the typability power of Curry’s type discipline.

The approach we use is known as *logical semantics* and we apply it in the setting of Coherence Space. Informally speaking, we use types as labels to denote tokens of a suitable coherence space. The simple types (i.e. atomic types plus arrow types without intersection) describe the behavior of a linear function, because the trace of a linear function between two coherence spaces is a subset of the cartesian product of the set of their tokens (satisfying certain coherence conditions). We want to use intersection types as an exponential: the intersection of two types denoting two tokens will be a

type denoting the clique containing these two tokens. Thus, allowing intersection on the left-hand side of an arrow type means describing linear function from the finite clique space of a coherence space to another coherence space, i.e. a stable function.

After doing this, we build a type assignment system, which assigns types of this kind to λ -terms and we define the interpretation of a term to be the set of types the system is able to assign to it. This corresponds to a logical description of a λ -model built in the category of Coherence Space and Stable Function where λ -terms are interpreted into cliques. The fact that the type system assigns a type to a term M is logically equivalent to the fact that the token corresponding to that type belongs to the clique denoting M .

Thus, intersection types assignment system are a tool to reason in a finitary way on the denotational interpretation of λ -terms, in the way described above. In particular, we define two type assignment system, parametric with respect to a coherence relation on types and we prove that, when the instantiation of the parameter satisfies certain conditions, our two type systems induces models of λ -calculus. Lastly, we show that such systems give a logical characterization of two classes of models built in the category of Girard's Coherence Spaces and Stable Functions.

1.4 Structure of the thesis

Summarizing, in this thesis, an exploration of the notions of linearity is proposed. In particular, linearity is seen as a common ingredient of many denotational models, from the classical domain-based denotational semantics up to the modern game semantics. In this point of view, we propose the study of processes as a link between functional static aspects of programs and dynamic aspects of programs, related to their evaluation. Processes are both entities generated by the evaluation of a program and tools to describe in a finitary way game-semantics strategies. Moreover, we wish to go beyond strict linearity, by using intersection types as a semantic tool which breaks linearity allowing weakening and contraction.

Thus, this thesis will be divided into two parts. Part 1 is titled **Linearity in Denotational Semantics** while Part 2 is titled **Behind Linearity Using Intersection Types**

Linearity in Denotational Semantics. In the first part, we propose a study of linearity as a fundamental ingredient in denotational semantics. This part consists of three chapters. In chapter 3, titled **Semantically Linear PCF** we propose the study of a denotationally linear programming language, called $\mathcal{S}lPCF$. we will show that the model of Scott Domain and Strict Continuous Function is adequate with respect to it and a full abstraction result is shown for the model of Coherence Spaces and Linear Stable Functions. In chapter 4, titled **A Process Model for Linear Programs**, we propose a process language, called $linProc$, to give a syntactical model of $\mathcal{S}lPCF$. This language is able to describe all evaluation strategies of programs of $\mathcal{S}lPCF$. The spirit of this language is to describe in a finitary way strategies of game semantics, as in [Hyland and Ong, 1995].

In chapter 5, titled **Ludics strategies and the π -calculus**, we will propose Ludics as denotational model to study properties of processes of linear π -calculus [Yoshida et al., 2004]. Namely, we will show a fully abstract encoding from π -processes of the finitary fragment of linear π -calculus to Ludics strategies. Then we address the problem of extending such a result to more expressive calculus, equipped with recursion.

Behind Linearity using Intersection Types. In the second part, we want to study intersection types, as a tool to break with strict linearity. This part consists of one chapter, titled **Logical Semantics for Stability** where we use intersection types as a tool to extend linear calculi with exponential modalities, allowing weakening and contraction. In particular we study intersection type assignment systems as tool to reason about the denotational interpretation of terms of untyped λ -calculus..

2 Technical Preliminaries

In this chapter, we review some technical preliminaries, that we will need in the sequel. In particular, in Section 2.1 we remind some basics notion of Category Theory; in Section 2.2 we review some notions concerning λ -calculus and functional programming languages; finally in Section 2.3 we discuss some denotational models of these languages.

2.1 Category Theory

In this section we recall some basics notions of Category Theory. In particular, this section is divided into three subsections. In Subsection 2.1.1, we recall the definition of category, functors and natural transformation. In Subsection 2.1.2, we remind the definition of adjoint functors and monads. In Subsection 2.1.3, we remind the notion of Cartesian Closed Category. Finally, in Subsection 2.1.4, we recall the definition of Monoidal Category, which is the base of many models of Linear Logic [Girard, 1987]. Much of the content of this section is taken from [Asperti and Longo, 1991, Crole, 1993, MacLane, 1998, Blute and Scott, 2004].

2.1.1 Categories, Functors and Natural Transformations

Definition 2.1.1 (Category). *A category \mathbb{C} consists of*

- *a collection of objects $Obj(\mathbb{C})$;*
- *for each pair of objects A, B , a collection $\mathbb{C}(A, B)$ of morphisms. In the following we will write $f : A \rightarrow B$ or $A \xrightarrow{f} B$ to denote a morphism f in $\mathbb{C}(A, B)$;*
- *given two morphisms $f : A \rightarrow B$ and $b : B \rightarrow C$, a composition \circ mapping them into the morphism $g \circ f : A \rightarrow C$;*
- *for each object A , an identity $id_A : A \rightarrow A$;*

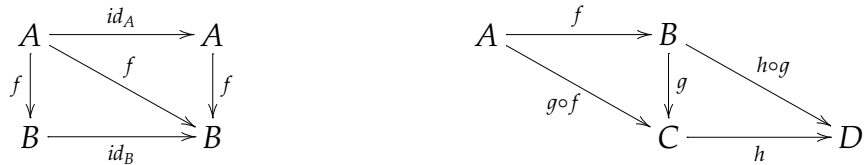
satisfying the axioms,

$$h \circ (g \circ f) = (h \circ g) \circ f \tag{2.1}$$

$$f \circ id_A = f = id_B \circ f \tag{2.2}$$

for any $f : A \rightarrow B, g : B \rightarrow C, h : C \rightarrow D$ where A, B, C, D are in $Obj(\mathbb{C})$.

A standard methodology in category theory to describe formal equations is the use of **commutative diagrams**. In a diagram, a morphism $f : A \rightarrow B$ is drawn as an arrow from A to B labelled with f . A diagram **commutes** if the composition of the morphisms along any path between two fixed objects is equal. For example, we can describe axioms (2.1) and (2.2) in terms of the following commutative diagrams. For a more detailed explanation, we refer to [Asperti and Longo, 1991, MacLane, 1998].



A category is called **large** or **small** depending upon whether its collection of objects is respectively a proper class or a set, in the sense of Gödel-Bernays set theory. A category \mathbb{C} is **locally small** if $\mathbb{C}(A, B)$ is a set for all objects A, B .

Remark 2.1.1. *From now on, when not specified otherwise, given a category \mathbb{C} , we will assume that \mathbb{C} is a locally small category.*

Many familiar classes of structures in mathematics and logic can be organised into categories. In the following, we give some examples, which will be useful in the sequel.

- The most familiar one is $\mathbb{S}et$, which is the large category having all sets as Objects and all set-theoretic functions as morphisms. The identity and the composition are defined in the usual way.
- An other simple example of category is $\mathbb{1}$, which is the small category having one object and one (identity) morphism.

For other examples of categories, we refer again to [Asperti and Longo, 1991, MacLane, 1998].

There are many ways of forming new categories out of old ones. Two basic operations are the following:

Dual Category. The dual category \mathbb{C}^{op} has the same objects of \mathbb{C} but reversed morphisms. More formally, if \mathbb{C} is a category, then its dual \mathbb{C}^{op} is defined in the following way:

- $Obj(\mathbb{C}^{op}) = Obj(\mathbb{C})$.
- $\mathbb{C}^{op}(A, B) = \mathbb{C}(B, A)$ for all $A, B \in Obj(\mathbb{C})$.
- Given $f \in \mathbb{C}^{op}(B, A)$ and $g \in \mathbb{C}^{op}(C, B)$ we define their composition $f \circ g$ in \mathbb{C}^{op} as the composition $g \circ f$ in \mathbb{C}

Product Category. If \mathbb{C} and \mathbb{D} are two categories, we define $\mathbb{C} \times \mathbb{D}$ as:

- $Obj(\mathbb{C} \times \mathbb{D}) = Obj(\mathbb{C}) \times Obj(\mathbb{D})$

- $\mathbb{C} \times \mathbb{D}((A_1, A_2), (B_1, B_2)) = \mathbb{C}(A_1, B_1) \times \mathbb{D}(A_2, B_2)$ for each $A_1, B_1 \in \text{Obj}(\mathbb{C})$ and $A_2, B_2 \in \text{Obj}(\mathbb{D})$.
- Composition and identities are defined componentwise.

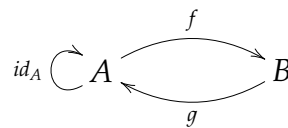
We will see other operations in the following of this section. An other useful notion in category theory is the one presented in the following definitions.

Definition 2.1.2 (Mono/Epi). *Given a morphism $f : B \rightarrow C$*

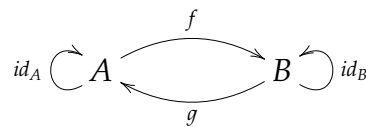
- *it is mono iff for any pair of morphisms $g, h : A \rightarrow B$ we have that $f \circ g = f \circ h$ implies $g = h$.*
- *it is epi iff for any pair of morphisms $g, h : C \rightarrow D$ we have that $g \circ f = h \circ f$ implies $g = h$.*

Definition 2.1.3. *Let $f : A \rightarrow B$ and $g : B \rightarrow A$ be two morphisms*

- *the pair (f, g) is called retraction pair when $g \circ f = id_A$ or equivalently when the following diagram commutes*



- *the pair (f, g) is called isomorphism (or simply iso) when $g \circ f = id_A$ and $f \circ g = id_B$ or equivalently when the following diagram commutes.*



Given a retraction pair (f, g) , where $f : A \rightarrow B$ and $g : B \rightarrow A$, we will say that *the object B contains A as retract* and we write $B \triangleright A$. By abusing notation, when $f : A \rightarrow B$ is a morphism of an iso pair, we will say that f is an *isomorphism* and we write $A \cong_f B$ (sometimes, we use to omit the subscript when clear from the context or uninteresting). Moreover, given an isomorphism $f : A \rightarrow B$, we denote its inverse arrow with $f^{-1} : B \rightarrow A$. Throughout this thesis we might talk of isomorphisms between objects without reference to the considered category. Generally, in this case, we are implicitly referring to Set , where isomorphisms are bijective functions, or to some other category which will be clear from the context.

Definition 2.1.4 (Functor). *Let \mathbb{C} and \mathbb{D} be categories. A (covariant) functor $F : \mathbb{C} \rightarrow \mathbb{D}$ consists of a mapping associating to each object $A \in \text{Obj}(\mathbb{C})$ an object $F(A) \in \text{Obj}(\mathbb{D})$ and to each morphisms $f \in \mathbb{C}(A, B)$ a morphism $F(f) \in \mathbb{D}(F(A), F(B))$ such that the following conditions hold*

- $F(id_A) = id_{F(A)}$ for all $A \in Obj(\mathbb{C})$.
- $F(g \circ f) = F(g) \circ F(f)$ for all $f : A \rightarrow B, g : B \rightarrow C$ with $A, B, C \in Obj(\mathbb{C})$.

Intuitively, a functor is a mapping between categories which preserves the identity and the composition between morphisms. A functor $F : \mathbb{C} \rightarrow \mathbb{D}$ is **full** (resp. **faithful**) when the induced mapping $F : \mathbb{C}(A, B) \rightarrow \mathbb{D}(F(A), F(B))$ on morphisms is surjective (resp. injective).

Given two functors $F : \mathbb{A} \rightarrow \mathbb{B}$ and $G : \mathbb{B} \rightarrow \mathbb{C}$, we can define their composition $G \circ F$ in the obvious way and it is still a functor. Furthermore, for any category \mathbb{A} , we can define the identity functor $id_{\mathbb{A}} : \mathbb{A} \rightarrow \mathbb{A}$ which is the identity in both object and morphism part. These observations allow us to define the category $\mathbb{C}at$ of small categories, where its morphisms are functors between them.

A **contravariant functor** $F : \mathbb{C} \rightarrow \mathbb{D}$ is a functor $F : \mathbb{C}^{op} \rightarrow \mathbb{D}$, i.e. a functor reversing the direction of the morphisms of \mathbb{C} . A functor whose domain is a product of category is also called a **bifunctor**. In the following, we give some example of functors, which will be useful in the following.

Inclusion Functor. A **subcategory** \mathbb{C} of a category \mathbb{D} (written $\mathbb{C} \subseteq \mathbb{D}$) is a collection of objects and morphism of \mathbb{D} such that

- for each $f : A \rightarrow B$ of \mathbb{C} we have $A, B \in Obj(\mathbb{C})$.
- for each object A of \mathbb{C} we have that the morphism id_A is in \mathbb{C} .
- for each pair of composable morphisms $f : A \rightarrow B$ and $g : B \rightarrow C$ of \mathbb{C} , we have that their composition $g \circ f : A \rightarrow C$ (made in \mathbb{D}) is in \mathbb{C} .

Clearly the definition above ensures that a subcategory is itself a category. Given a subcategory \mathbb{C} of \mathbb{D} , the mapping which sends objects and morphisms of \mathbb{C} to themselves in \mathbb{D} defines a functor called **inclusion functor**. This functor is faithful and when it is full we say that \mathbb{C} is a **full subcategory** of \mathbb{D} .

Hom Functors. If $A \in Obj(\mathbb{C})$ where \mathbb{C} is a locally small category, we have the mutually dual covariant and contravariant hom functors.

1. *Covariant hom functor:* $\mathbb{C}(A, -) : \mathbb{C} \rightarrow \mathbb{S}et$ given by:

$$\begin{aligned} B &\mapsto \mathbb{C}(A, B) \\ B \xrightarrow{f} C &\mapsto \mathbb{C}(A, f) : \mathbb{C}(A, B) \rightarrow \mathbb{C}(A, C) \\ &\text{where } \mathbb{C}(A, f)(g) = f \circ g \end{aligned}$$

2. *Contravariant hom functor:* $\mathbb{C}(-, A) : \mathbb{C}^{op} \rightarrow \mathbb{S}et$ given by:

$$\begin{aligned} B &\mapsto \mathbb{C}(B, A) \\ B \xrightarrow{f} C &\mapsto \mathbb{C}(f, A) : \mathbb{C}(C, A) \rightarrow \mathbb{C}(B, A) \\ &\text{where } \mathbb{C}(f, A)(g) = g \circ f \end{aligned}$$

3. *Hom bifunctor*: $\mathbb{C}(-, +) : \mathbb{C}^{op} \times \mathbb{C} \rightarrow \mathbb{Set}$ given by:

$$\begin{aligned} (A, B) &\mapsto \mathbb{C}(A, B) \\ (A_1, A_2) \xrightarrow{(f, g)} (B_1, B_2) &\mapsto \mathbb{C}(f, g) : \mathbb{C}(A_1, A_2) \rightarrow \mathbb{C}(B_1, B_2) \\ &\text{where } \mathbb{C}(f, g)(h) = (g \circ h) \circ f \end{aligned}$$

As hinted above, functors can be composed and in particular, for any category \mathbb{D}' and \mathbb{D}'' , $\mathbb{C}(-, +)$ can be composed with any two functors $F' : \mathbb{D}' \rightarrow \mathbb{C}^{op}$ and $F'' : \mathbb{D}'' \rightarrow \mathbb{C}$ to obtain the bifunctor $\mathbb{C}(F'(-), F''(+)) : \mathbb{D}' \times \mathbb{D}'' \rightarrow \mathbb{Set}$

Definition 2.1.5 (Natural Transformation). *Given two functors $F, G : \mathbb{C} \rightarrow \mathbb{D}$, a natural transformation $\alpha : F \rightarrow G$ is a collection of morphisms $\{\alpha_C : F(C) \rightarrow G(C) \mid C \in \text{Obj}(\mathbb{C})\}$ in \mathbb{D} , such that for any morphism $f : A \rightarrow B$ in \mathbb{C} , the following diagram commutes.*

$$\begin{array}{ccc} F(A) & \xrightarrow{\alpha_A} & G(A) \\ F(f) \downarrow & & \downarrow G(f) \\ F(B) & \xrightarrow{\alpha_B} & G(B) \end{array}$$

The diagram above is called *naturality square associated with f* .

Let $F, G, H : \mathbb{C} \rightarrow \mathbb{D}$ be functors. Given two natural transformations $\alpha : F \rightarrow G$ and $\beta : G \rightarrow H$, we can define their (vertical) **composition** $\beta \circ \alpha$ in a componentwise way $\{\beta_A \circ \alpha_A \mid A \in \text{Obj}(\mathbb{C})\}$. Moreover, we can define the **identity natural transformation** $id_F : F \rightarrow F$ to be the natural transformation with identity at all components: $id_F = \{id_{F(A)} \mid A \in \text{Obj}(\mathbb{C})\}$. Thus, we can define a category having functors between \mathbb{C} and \mathbb{D} as objects and natural transformations as morphisms. This category is called **functor category** and it is denoted with $[\mathbb{C}, \mathbb{D}]$.

Since natural transformations are morphisms in a functor category, we can specialise the notion of isomorphism to them. Thus, a natural transformation $\alpha : F \rightarrow G$ is a **natural isomorphism** when there is a natural transformation $\beta : G \rightarrow F$ such that $\beta \circ \alpha = id_F$ and $\alpha \circ \beta = id_G$.

Proposition 2.1.1. *A natural transformation is an isomorphism of functor iff each component is an isomorphism.*

Proof. See [Asperti and Longo, 1991, p. 47]. □

2.1.2 Adjonctions and (Co)-Monads

Definition 2.1.6 (Left/Right Adjoint). *Let $F : \mathbb{C} \rightarrow \mathbb{D}$ and $G : \mathbb{D} \rightarrow \mathbb{C}$ be two functors. We say that F is the left adjoint to G or G is the right adjoint to F (denoted $F \dashv G$) when there is a natural isomorphism between $\mathbb{D}(F(-), +)$ and $\mathbb{C}(-, G(+))$.*

In other words, given two functors $F : \mathbb{C} \rightarrow \mathbb{D}$ and $G : \mathbb{D} \rightarrow \mathbb{C}$, F is the left adjoint to G when there is a bijection between $\mathbb{D}(F(A), B)$ and $\mathbb{C}(A, G(B))$ which is natural in both A and B .

Indeed, the statement that $F : \mathbb{C} \rightarrow \mathbb{D}$ is a left adjoint to $G : \mathbb{D} \rightarrow \mathbb{C}$ is equivalent to the following property of the functor F : for each object $A \in \text{Obj}(\mathbb{C})$ there is an object $F(A) \in \text{Obj}(\mathbb{D})$ and a morphism $\eta_A : A \rightarrow G(F(A))$, such that for any morphism $f : A \rightarrow G(B)$ there is a unique $f^* : F(A) \rightarrow B$ making the following diagram commute.

$$\begin{array}{ccc} G(F(A)) & & \\ \eta_A \uparrow & \searrow G(f^*) & \\ A & \xrightarrow{f} & G(B) \end{array}$$

Dually, the fact that G is a right adjoint to F is equivalent to the fact that for every object $A \in \text{Obj}(\mathbb{D})$ there is an object $G(A) \in \text{Obj}(\mathbb{C})$ and a morphism $\varepsilon_A : F(G(A)) \rightarrow A$, such that for any morphism $f : F(B) \rightarrow A$ there is a unique $f^* : B \rightarrow G(A)$ making the following diagram commute.

$$\begin{array}{ccc} F(G(A)) & & \\ \varepsilon_A \downarrow & \swarrow F(f^*) & \\ A & \xleftarrow{f} & F(B) \end{array}$$

The two properties enunciated above are called *universal properties* of the adjunction. Their equivalence relationship with adjunctions is proved in [Crole, 1993, Proposition 2.10.15 p.83] or in [MacLane, 1998, Theorem 2 p.83].

Let $F : \mathbb{C} \rightarrow \mathbb{D}$ and $G : \mathbb{D} \rightarrow \mathbb{C}$ be functors such that $F \dashv G$ and let η_A and ε_A be as defined above. Observe that

- $\eta = \{\eta_A \mid A \in \text{Obj}(\mathbb{C})\} : id_{\mathbb{C}} \rightarrow G(F(-))$ and
- $\varepsilon = \{\varepsilon_A \mid A \in \text{Obj}(\mathbb{D})\} : F(G(-)) \rightarrow id_{\mathbb{D}}$

are two natural transformations and the quadruple $(F, G, \eta, \varepsilon)$ is called **adjunction from \mathbb{C} to \mathbb{D}** , while η and ε are called respectively **unit** and **co-unit** of the adjunction.

We conclude the section by giving the definition of monad and co-monad and clarifying their connection with adjunctions. Let $T : \mathbb{C} \rightarrow \mathbb{C}$ be an endo-functor. We denote $T^n = T^{n-1} \circ T$. Moreover, if $\mu : T^2 \rightarrow T$ is a natural transformation, then we denote the natural transformations $\{\mu_C \mid C \in \text{Obj}(\mathbb{C})\}$ and $\{\mu_{T(C)} \mid C \in \text{Obj}(\mathbb{C})\}$ respectively with $T\mu : T^3 \rightarrow T^2$ and $\mu T : T^3 \rightarrow T^2$.

Definition 2.1.7 (Monad/Co-Monad). A monad over a category \mathbb{C} is a triple (T, μ, η) where $T : \mathbb{C} \rightarrow \mathbb{C}$ is a functor, $\mu : T^2 \rightarrow T$ and $\eta : id_{\mathbb{C}} \rightarrow T$ are two natural transformations called respectively multiplication and unit of the monad, such that the following diagrams commute.

$$\begin{array}{ccc}
 T^3 & \xrightarrow{\mu^T} & T^2 \\
 T\mu \downarrow & & \downarrow \mu \\
 T^2 & \xrightarrow{\mu} & T
 \end{array}
 \qquad
 \begin{array}{ccc}
 T & \xrightarrow{\eta^T} & T^2 \xleftarrow{T\eta} & T \\
 \downarrow id_T & & \downarrow \mu & \swarrow id_T \\
 & & T &
 \end{array}$$

A co-monad over a category \mathbb{C} is a triple (T, δ, ε) where $T : \mathbb{C} \rightarrow \mathbb{C}$ is a functor, $\delta : T \rightarrow T^2$ and $\varepsilon : T \rightarrow id_{\mathbb{C}}$ are natural transformations called respectively co-multiplication and co-unit of the co-monad, such that the following diagrams commute.

$$\begin{array}{ccc}
 T^3 & \xleftarrow{\delta^T} & T^2 \\
 T\delta \uparrow & & \uparrow \delta \\
 T^2 & \xleftarrow{\delta} & T
 \end{array}
 \qquad
 \begin{array}{ccc}
 T & \xleftarrow{\varepsilon^T} & T^2 \xrightarrow{T\varepsilon} & T \\
 \downarrow id_T & & \downarrow \delta & \swarrow id_T \\
 & & T &
 \end{array}$$

The canonical examples of monads and co-monads arise from a pair of adjoint functors $F \dashv G$, as shown by the following proposition.

Proposition 2.1.2. *Let $(F, G, \eta, \varepsilon)$ be an adjunction from \mathbb{C} to \mathbb{D} . Then*

- $(G(F(-)), G\varepsilon F, \eta)$ is a monad on \mathbb{C}
- $(F(G(-)), G\eta F, \varepsilon)$ is a co-monad on \mathbb{D} .

Proof. See [Asperti and Longo, 1991, Proposition 5.4.1 p. 104]. □

In fact, every monad (and consequently every co-monad) arises from a pair of adjoint functors. In this thesis, we are interested on those arisen by a co-monads, hence we will introduce the Kleisli Category over a co-monad T and the Eilemberg-Moore Category of co-algebras of a co-monad T .

Definition 2.1.8 (Kleisli Category). *Let (T, δ, ε) be a co-monad over the category \mathbb{C} . We define the category $\mathbb{D} = \mathbb{C}_T$, called Kleisli Category over the co-monad T , in the following way.*

- The set of objects is given by $Obj(\mathbb{D}) = Obj(\mathbb{C})$.
- The set of morphisms between two objects A, B is $\mathbb{D}(A, B) = \mathbb{C}(T(A), B)$.
- The identity $id_A : A \rightarrow A$ in \mathbb{D} is $\varepsilon_A : T(A) \rightarrow A$ in \mathbb{C} .
- The composition $g \circ f$ of two morphisms of \mathbb{D} , $f : A \rightarrow B$ and $g : B \rightarrow C$ is defined in term of the composition of \mathbb{C} (where $f : T(A) \rightarrow B$ and $g : T(B) \rightarrow C$ in \mathbb{C}) as $(g \circ T(f)) \circ \delta_A$

The pair of functors $F : \mathbb{C}_T \rightarrow \mathbb{C}$ and $G : \mathbb{C} \rightarrow \mathbb{C}_T$ with $F \dashv G$ are given by:

- given $A \in Obj(\mathbb{C}_T) = Obj(\mathbb{C})$ we let $F(A) = T(A)$ and given $f \in \mathbb{C}_T(A, B) = \mathbb{C}(T(A), B)$, we let $F(f) = T(f) \circ \delta_A$

- given $A \in \text{Obj}(\mathbb{C})$ we let $G(A) = A \in \text{Obj}(\mathbb{C}_T)$ and given $f \in \mathbb{C}(A, B)$, we let $G(f) = f \circ \varepsilon_A$.

A **T -coalgebra** of a co-monad (T, δ, ε) is a morphism $\alpha : A \rightarrow T(A)$ which is such that the following diagrams commute

$$\begin{array}{ccc}
 A & \xrightarrow{\alpha} & T(A) \\
 & \searrow \text{id}_A & \downarrow \varepsilon_A \\
 & & A
 \end{array}
 \qquad
 \begin{array}{ccc}
 A & \xrightarrow{\alpha} & T(A) \\
 \downarrow \alpha & & \downarrow T(\alpha) \\
 T(A) & \xrightarrow{\delta_A} & T^2(A)
 \end{array}$$

Definition 2.1.9 (Eilemberg-Moore Category of Co-Algebras). *The Eilemberg-Moore Category of Co-Algebras of a co-monad, denoted with \mathbb{C}^T , is defined in the following way.*

- The set of objects $\text{Obj}(\mathbb{C}^T)$ is the set of T -coalgebra of the co-monads T
- Given two co-algebras $\alpha : A \rightarrow T(A)$ and $\beta : B \rightarrow T(B)$ a co-algebra morphism is any morphism $f : A \rightarrow B$ commuting with the co-algebra structure, i.e. such that the following diagram commutes.

$$\begin{array}{ccc}
 A & \xrightarrow{f} & B \\
 \downarrow \alpha & & \downarrow \beta \\
 T(A) & \xrightarrow{T(f)} & T(B)
 \end{array}$$

The pair of functors $F : \mathbb{C}^T \rightarrow \mathbb{C}$ and $G : \mathbb{C} \rightarrow \mathbb{C}^T$ with $F \dashv G$ are given by:

- $F(A \xrightarrow{\alpha} T(A)) = A$ and $F(h) = h$.
- $G(A) = T(A) \xrightarrow{\delta_A} T^2(A)$ and $G(f) = T(f)$

All co-algebras of the form $T(A) \xrightarrow{\delta_A} T^2(A)$ are called **free co-algebras** and it can be shown that the Kleisli category of a co-monad T is equivalent to the full subcategory of the Eilemberg-Moore category consisting of free co-algebras [MacLane, 1998, p.147].

2.1.3 Cartesian Closed Categories

Let $\mathbb{1}$ be the category having one object and one morphism. We will denote with \star the unique object of such a category.

Definition 2.1.10 (Cartesian Category). *A category \mathbb{C} is said to be cartesian when the following two functors $F : \mathbb{C} \rightarrow \mathbb{1}$ and $G : \mathbb{C} \rightarrow \mathbb{C} \times \mathbb{C}$, defined as*

$$\begin{array}{ccc}
 F : \mathbb{C} & \rightarrow & \mathbb{1} & & G : \mathbb{C} & \rightarrow & \mathbb{C} \times \mathbb{C} \\
 A & \mapsto & \star & & A & \mapsto & (A, A) \\
 A \xrightarrow{f} B & \mapsto & \star \xrightarrow{\text{id}_\star} \star & & A \xrightarrow{f} B & \mapsto & (A, A) \xrightarrow{(f, f)} (B, B)
 \end{array}$$

admit right adjoints, respectively denoted with $\top : \mathbb{1} \rightarrow \mathbb{C}$ and $\times : \mathbb{C} \times \mathbb{C} \rightarrow \mathbb{C}$.

The above definition deserves some explanation. Observe that the above defined functor $F : \mathbb{C} \rightarrow \mathbb{1}$ is the unique functor to the one-object category. Postulating that F has a right adjoint corresponds to saying that \mathbb{C} has an object $\top(\star)$ such that for any other object $A \in \text{Obj}(\mathbb{C})$ there is a unique morphism $!_A : A \rightarrow \top(\star)$, since $\mathbb{C}(A, \top(\star)) \cong \{id_\star\} \cong \mathbb{1}(F(A), \star)$. This object is called **terminal object**. Finally, we can observe that $!_A$ is the morphism corresponding to id_\star through the isomorphism given by the adjonction. Furthermore, postulating that G has a right adjoint corresponds to saying that, given any two object $A, B \in \text{Obj}(\mathbb{C})$, there is an object $A \times B$ such that for all C we have a natural isomorphism $\mathbb{C}(C, A) \times \mathbb{C}(C, B) \cong \mathbb{C}(C, A \times B)$. If we let $C = A \times B$, then the identity morphism $id_{A \times B}$ in $\mathbb{C}(A \times B, A \times B)$ corresponds through the isomorphism to the pair of projections $(\pi_1, \pi_2) \in \mathbb{C}(A \times B, A) \times \mathbb{C}(A \times B, B)$. Moreover, given two morphisms $f : C \rightarrow A$ and $g : C \rightarrow B$, we can define again through the isomorphism a pairing $\langle f, g \rangle : C \rightarrow A \times B$. It follows from the naturality law of adjonction that all these morphisms satisfies the following equations, for all $f : C \rightarrow A, g : C \rightarrow B$ and $h : C \rightarrow A \times B$.

$$\pi_1 \circ \langle f, g \rangle = f \tag{2.3}$$

$$\pi_2 \circ \langle f, g \rangle = g \tag{2.4}$$

$$\langle \pi_1 \circ h, \pi_2 \circ h \rangle = h \tag{2.5}$$

The bifunctor \times is called **categorical product**. The categorical product, when it exists, is unique up to isomorphism. In a dual way, we can define co-cartesian categories, as following.

Definition 2.1.11 (Co-cartesian Category). *A category \mathbb{C} is co-cartesian when the functors F, G defined in Definition 2.1.10 have specified left adjoints, denoted respectively with \perp and $+$.*

In a dual way, the object $\perp(\star)$ is called **initial object** and the bifunctor $+$ is called **co-product**.

Notation 2.1.1. *From now on, with an abuse of notation, we will denote respectively with \top and \perp , the terminal object and the initial object of a category, if it admits them.*

Definition 2.1.12 (Cartesian Closed Category). *Let \mathbb{C} be a cartesian category. We say that it is cartesian closed when for all objects $A \in \text{Obj}(\mathbb{C})$ the functor $- \times A : \mathbb{C} \rightarrow \mathbb{C}$, defined as*

$$\begin{aligned} - \times A : \mathbb{C} &\rightarrow \mathbb{C} \\ C &\mapsto C \times A \\ C \xrightarrow{f} D &\mapsto C \times A \xrightarrow{\langle f, id_A \rangle} D \times A \end{aligned}$$

has a **specified right adjoint** denoted $A \Rightarrow - : \mathbb{C} \rightarrow \mathbb{C}$.

Postulating that $- \times A$ has a right adjoint $A \Rightarrow -$ means that there is a natural isomorphism $\mathbb{C}(C \times A, B) \cong \mathbb{C}(C, A \Rightarrow B)$ for all $C \in \text{Obj}(\mathbb{C})$. If we let $C = A \Rightarrow B$, then the identity morphism $id_{A \Rightarrow B}$ in $\mathbb{C}(A \Rightarrow B, A \Rightarrow B)$ corresponds through the isomorphism to the evaluation morphism $eval : A \Rightarrow B \times A \rightarrow B$. Conversely, given $f : C \times A \rightarrow B$, we can define again through the isomorphism, the morphism $curry(f) : C \rightarrow A \Rightarrow B$. It follows that all these morphisms satisfies the following equations, for all $f : C \times A \rightarrow B$ and for all $g : C \rightarrow A \Rightarrow B$.

$$eval \circ (curry(f) \times id_A) = f \tag{2.6}$$

$$curry(eval \circ (g \times id_A)) = g \tag{2.7}$$

The object B^A is called **exponential object**.

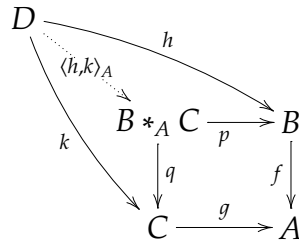
Definition 2.1.13. A Cartesian Closed Category \mathbb{C} (a category admitting a terminal object suffices) is well pointed (or it has enough points) when for any two distinct $f, g : A \rightarrow B$, there exists a morphism $p : \mathbb{T} \rightarrow A$ such that $f \circ p \neq g \circ p$.

We conclude the section by defining a categorical construction which is a generalisation of the categorical product. It is called **pull-back** and it is defined as follows

Definition 2.1.14 (Pull-back). Given two morphisms $f : B \rightarrow A$ and $g : C \rightarrow A$, the pullback of (f, g) is an object $B *_A C$ and two morphisms $p : B *_A C \rightarrow B, q : B *_A C \rightarrow C$, such that

1. $f \circ p = g \circ q : B *_A C \rightarrow A$
2. for every other triple $(D, h : D \rightarrow B, k : D \rightarrow C)$ such that $g \circ k = f \circ h$, there exists a unique arrow $\langle h, k \rangle_A : D \rightarrow B *_A C$ such that $p \circ \langle h, k \rangle_A = h$, and $q \circ \langle h, k \rangle_A = k$.

A typical pull-back diagram is the following.



The lower square is also called *pull-back square*

Proposition 2.1.3. Let \mathbb{C} be a category admitting a terminal object \mathbb{T} and pull-back for every pair of morphisms. Then it admits categorical product for every pair of object.

Proof. The proof follows just by defining $A \times B = A *_T B$ and see that it satisfies all the required axioms for the categorical product. More details can be found in [Asperti and Longo, 1991, Proposition 2.6.5 p. 29]. \square

2.1.4 Monoidal categories

Definition 2.1.15 (Monoidal category). A monoidal category consists of a category \mathbb{C} , a bifunctor $\otimes : \mathbb{C} \times \mathbb{C} \rightarrow \mathbb{C}$ called tensor product, an object $\mathbf{1} \in \text{Obj}(\mathbb{C})$ and three natural isomorphisms α, λ, ρ such that:

- $\alpha_{A,B,C} : A \otimes (B \otimes C) \cong (A \otimes B) \otimes C$ is natural for all $A, B, C \in \text{Obj}(\mathbb{C})$, and the pentagonal diagram

$$\begin{array}{ccccc}
 A \otimes (B \otimes (C \otimes D)) & \xrightarrow{\alpha_{A,B,C \otimes D}} & (A \otimes B) \otimes (C \otimes D) & \xrightarrow{\alpha_{A \otimes B,C,D}} & ((A \otimes B) \otimes C) \otimes D \\
 \downarrow \text{id}_A \otimes \alpha_{B,C,D} & & & & \uparrow \alpha_{A,B,C} \otimes \text{id}_D \\
 A \otimes ((B \otimes C) \otimes D) & \xrightarrow{\alpha_{A,B \otimes C,D}} & & & (A \otimes (B \otimes C)) \otimes D
 \end{array}$$

commutes for all $A, B, C, D \in \text{Obj}(\mathbb{C})$

- $\lambda_A : \mathbf{1} \otimes A \cong A$ and $\rho_A : A \otimes \mathbf{1} \cong A$ are natural for all $A \in \text{Obj}(\mathbb{C})$, and the triangular diagram

$$\begin{array}{ccc}
 A \otimes (\mathbf{1} \otimes C) & \xrightarrow{\alpha_{A,\mathbf{1},C}} & (A \otimes \mathbf{1}) \otimes C \\
 \searrow \text{id}_A \otimes \lambda_C & & \swarrow \rho_A \otimes \text{id}_C \\
 & A \otimes C &
 \end{array}$$

commutes for all $A, C \in \text{Obj}(\mathbb{C})$ and also

$$\lambda_{\mathbf{1}} = \rho_{\mathbf{1}} : \mathbf{1} \otimes \mathbf{1} \rightarrow \mathbf{1}$$

We sometimes call α the *associativity law* of \otimes , ρ the *right identity law* of \otimes and λ the *left identity law* of \otimes .

Observe the analogy between Monoidal Categories and monoids: it turns out that the tensor product is associative and it has a neutral element (up to isomorphism). It seems natural to extend monoidal categories to symmetric monoidal categories by asking the tensor product to be commutative too (up to isomorphism).

Definition 2.1.16 (Co-Monoid). Let \mathbb{C} be a monoidal category. A co-monoid (C, d_C, e_C) consists of an object $C \in \text{Obj}(\mathbb{C})$ together with two morphisms $d_C : C \rightarrow C \otimes C$ and $e_C : C \rightarrow \mathbf{1}$ such that the following diagrams are commutative.

$$\begin{array}{ccccc}
 C \otimes (C \otimes C) & \xleftarrow{\alpha_{C,C,C}^{-1}} & (C \otimes C) \otimes C & \xleftarrow{d_C \otimes \text{id}_C} & C \otimes C \\
 \text{id}_C \otimes d_C \uparrow & & & & \uparrow d_C \\
 C \otimes C & \xleftarrow{d_C} & & & C
 \end{array}$$

$$\begin{array}{ccccc}
 \mathbf{1} \otimes C & \xleftarrow{e_C \otimes \text{id}_C} & C \otimes C & \xrightarrow{\text{id}_C \otimes e_C} & C \otimes \mathbf{1} \\
 & \swarrow \lambda_C^{-1} & \uparrow d_C & \searrow \rho_C^{-1} & \\
 & & C & &
 \end{array}$$

Given two co-monoids (C, d_C, e_C) and (D, d_D, e_D) in \mathbb{C} , a **co-monoid morphism** is any morphism $f : C \rightarrow D$ such that

$$d_D \circ f = (f \otimes f) \circ d_C \quad e_D \circ f = e_C$$

With these morphisms, the co-monoids in \mathbb{C} constitute a category, denoted $\text{CoMon}_{\mathbb{C}}$.

Definition 2.1.17 (Symmetric Monoidal Category). *A monoidal category \mathbb{C} is said to be symmetric when it is equipped with an isomorphism*

$$\gamma_{A,B} : A \otimes B \cong B \otimes A$$

natural in $A, B \in \text{Obj}(\mathbb{C})$, such that the diagrams

$$\begin{array}{ccc} \gamma_{A,B} \circ \gamma_{B,A} = id_{A \otimes B} & \varrho_B = \lambda_B \circ \gamma_{B,1} : B \otimes \mathbf{1} \cong B & \\ \\ \begin{array}{ccccc} A \otimes (B \otimes C) & \xrightarrow{\alpha_{A,B,C}} & (A \otimes B) \otimes C & \xrightarrow{\gamma_{A \otimes B, C}} & C \otimes (A \otimes B) \\ \downarrow id_A \otimes \gamma_{B,C} & & & & \downarrow \alpha_{C,A,B} \\ A \otimes (C \otimes B) & \xrightarrow{\alpha_{A,C,B}} & (A \otimes C) \otimes B & \xrightarrow{\gamma_{A,C} \otimes id_B} & (C \otimes A) \otimes B \end{array} \end{array}$$

all commute.

Let \mathbb{C} be a symmetric monoidal category. A comonoid (C, d_C, e_C) in \mathbb{C} is said to be **commutative** when $d_C = \gamma_{C,C} \circ d_C$.

Observe that any Cartesian Category \mathbb{C} (resp. Co-cartesian Category) is a Symmetric Monoidal Category, by taking \times (resp. $+$) as tensor product and the terminal object \mathbf{T} (resp. the initial object $\mathbf{1}$) as neutral element of the tensor product. However, the converse does not hold. Roughly, the tensor product differs from the Categorical Product in that it has no projections and pairings satisfying Equations (2.3), (2.4) and (2.5).

Definition 2.1.18 (Monoidal Functor). *Let \mathbb{C} and \mathbb{D} be two monoidal categories. A monoidal functor between \mathbb{C} and \mathbb{D} consists of the following three items:*

- *an ordinary functor $F : \mathbb{C} \rightarrow \mathbb{D}$.*
- *for objects $A, B \in \text{Obj}(\mathbb{C})$, a morphism $m_{A,B} : F(A) \otimes F(B) \rightarrow F(A \otimes B)$, which is natural in both A and B .*
- *for the units $\mathbf{1}_{\mathbb{C}}$ of \mathbb{C} and $\mathbf{1}_{\mathbb{D}}$ of \mathbb{D} , a morphism $m_1 : \mathbf{1}_{\mathbb{D}} \rightarrow F(\mathbf{1}_{\mathbb{C}})$ of \mathbb{D}*

Together, these must make all the following three diagrams, involving the structural maps $\alpha^{\mathbb{C}}$, $\lambda^{\mathbb{C}}$, $\varrho^{\mathbb{C}}$ of \mathbb{C} and $\alpha^{\mathbb{D}}$, $\lambda^{\mathbb{D}}$, $\varrho^{\mathbb{D}}$ of \mathbb{D} , commute in \mathbb{D} .

$$\begin{array}{ccc} F(A) \otimes (F(B) \otimes F(C)) & \xrightarrow{\alpha_{F(A), F(B), F(C)}^{\mathbb{D}}} & (F(A) \otimes F(B)) \otimes F(C) \\ \downarrow id_{F(A)} \otimes m_{B,C} & & \downarrow m_{A,B} \otimes id_{F(C)} \\ F(A) \otimes (F(B \otimes C)) & & F(A \otimes B) \otimes F(C) \\ \downarrow m_{A, B \otimes C} & & \downarrow m_{A \otimes B, C} \\ F(A \otimes (B \otimes C)) & \xrightarrow{F(\alpha_{A,B,C}^{\mathbb{C}})} & F((A \otimes B) \otimes C) \end{array}$$

$$\begin{array}{ccc}
 F(B) \otimes \mathbf{1}_{\mathbb{D}} & \xrightarrow{\varrho_{F(B)}^{\mathbb{D}}} & F(B) \\
 \downarrow id_{F(B)} \otimes m_1 & & \uparrow F(\varrho_B^{\mathbb{C}}) \\
 F(B) \otimes F(\mathbf{1}_{\mathbb{C}}) & \xrightarrow{m_{B, \mathbf{1}_{\mathbb{C}}}} & F(B \otimes \mathbf{1}_{\mathbb{C}})
 \end{array}
 \quad
 \begin{array}{ccc}
 \mathbf{1}_{\mathbb{D}} \otimes F(B) & \xrightarrow{\lambda_{F(B)}^{\mathbb{D}}} & F(B) \\
 \downarrow m_1 id_{F(B)} & & \uparrow F(\lambda_B^{\mathbb{C}}) \\
 F(\mathbf{1}_{\mathbb{C}}) \otimes F(B) & \xrightarrow{m_{\mathbf{1}_{\mathbb{C}}, B}} & F(\mathbf{1}_{\mathbb{C}} \otimes B)
 \end{array}$$

We will denote a monoidal functor between \mathbb{C} and \mathbb{D} simply as $(F, m) : \mathbb{C} \rightarrow \mathbb{D}$. A monoidal functor (F, m) is said to be **strong** when both m_1 and $m_{A,B}$ are isomorphisms. If \mathbb{C} and \mathbb{D} are also symmetric, then the monoidal functor $(F, m) : \mathbb{C} \rightarrow \mathbb{D}$ is said to be **symmetric** when also the following diagram involving the structural maps $\gamma^{\mathbb{C}}$ of \mathbb{C} and $\gamma^{\mathbb{D}}$ of \mathbb{D} commutes in \mathbb{D} .

$$\begin{array}{ccc}
 F(A) \otimes F(B) & \xrightarrow{\gamma_{F(A), F(B)}^{\mathbb{D}}} & F(B) \otimes F(A) \\
 \downarrow m_{A,B} & & \downarrow m_{B,A} \\
 F(A \otimes B) & \xrightarrow{F(\gamma_{A,B}^{\mathbb{C}})} & F(B \otimes A)
 \end{array}$$

Definition 2.1.19 (Monoidal Natural Transformation). *A monoidal natural transformation between monoidal functors $(F, m) : \mathbb{C} \rightarrow \mathbb{D}$ and $(G, n) : \mathbb{C} \rightarrow \mathbb{D}$ is an ordinary natural transformation $\theta : F \rightarrow G$ such that all the following diagrams commute in \mathbb{D} .*

$$\begin{array}{ccc}
 F(A) \otimes F(B) & \xrightarrow{m_{A,B}} & F(A \otimes B) \\
 \downarrow \theta_A \otimes \theta_B & & \downarrow \theta_{A \otimes B} \\
 G(A) \otimes G(B) & \xrightarrow{n_{A,B}} & G(A \otimes B)
 \end{array}
 \quad
 \begin{array}{ccc}
 F(\mathbf{1}_{\mathbb{C}}) & \xrightarrow{\theta_{\mathbf{1}_{\mathbb{C}}}} & G(\mathbf{1}_{\mathbb{C}}) \\
 \swarrow m_1 & & \nearrow n_1 \\
 & \mathbf{1}_{\mathbb{D}} &
 \end{array}$$

Definition 2.1.20 (Monoidal Co-monad). *A monoidal functor (T, m) is a monoidal co-monad if $\langle T, \delta, \varepsilon \rangle$ is also a co-monad with δ and ε being monoidal natural transformations.*

By analogy with Cartesian Closed Categories, it is natural to ask that the tensor product have an appropriate adjoint, and this leads to the following definition.

Definition 2.1.21 (Symmetric Monoidal Closed Category). *A symmetric category \mathbb{C} is closed when for all $B \in \text{Obj}(\mathbb{C})$ the functor $- \otimes B : \mathbb{C} \rightarrow \mathbb{C}$ has a specified right adjoint $B \multimap - : \mathbb{C} \rightarrow \mathbb{C}$.*

Given a morphisms $f : A \otimes B \rightarrow C$, we denote with $\text{curry}(f) : A \rightarrow B \multimap C$ and $\text{eval} : A \multimap B \otimes A \rightarrow B$ the two morphisms obtained from the adjunction in the same way as for the case of Cartesian Closed Categories. It is not difficult to see that Equations (2.6) and (2.7) holds also for symmetric monoidal closed categories.

The last two types of category we introduce are special cases of Symmetric Monoidal Closed Categories and they are given by the following definitions.

Definition 2.1.22 (*-Autonomous Category). *A *-autonomous category \mathbb{C} is a symmetric monoidal closed category with a distinguished dualizing object \perp (which may be different from the terminal object), such that (letting $A^\perp = A \multimap \perp$), the canonical map $v_A : A \rightarrow A^{\perp\perp}$ is an isomorphism for all A .*

*-Autonomous Category are applied especially to provide models of Classical Linear Logic [Girard, 1987]. In particular, given a *-autonomous category \mathbb{C} , it is possible to define $\wp : \mathbb{C} \rightarrow \mathbb{C}$ in terms of the tensor product and the dualizing object, as $A \wp B = (A^\perp \otimes B^\perp)^\perp$.

In general, given a *-autonomous category, it could be possibly the case that $\wp \neq \otimes$ and there is no morphism $f : A \otimes B \rightarrow A \wp B$ (having such a morphism corresponds to validate the **Mix** rule in LL). However, it is possible to prove that in every *-autonomous category having the tensor unit as dualizing object there exists a canonical morphism $A \otimes B \rightarrow A \wp B$ [Cockett and Seely, 1997]. The next notion corresponds to a *-autonomous category in which $\otimes = \wp$.

Definition 2.1.23 (Compact Closed Category). *A compact closed category \mathbb{C} is a symmetric monoidal closed category where every object $A \in \text{Obj}(\mathbb{C})$ has a dual A^* and two canonical morphisms $\eta_A : \mathbf{1} \rightarrow A^* \otimes A$ and $\nu_A : A \otimes A^* \rightarrow \mathbf{1}$ such that the following diagrams commute.*

$$\begin{array}{ccc}
 A & \xrightarrow{\varrho_A} & A \otimes \mathbf{1} \xrightarrow{id_A \otimes \eta_A} & A \otimes (A^* \otimes A) & & A^* & \xrightarrow{\lambda_{A^*}} & \mathbf{1} \otimes A^* \xrightarrow{\eta_A \otimes id_{A^*}} & (A^* \otimes A) \otimes A^* \\
 & \searrow & & \downarrow \alpha_{A, A^*, A}^{-1} & & & \searrow & & \downarrow \alpha_{A^*, A, A^*} \\
 & & & (A \otimes A^*) \otimes A & & & & & A^* \otimes (A \otimes A^*) \\
 & & & \downarrow \nu_A \otimes id_A & & & & & \downarrow id_{A^*} \otimes \nu_A \\
 & & & \mathbf{1} \otimes A & & & & & A \otimes \mathbf{1} \\
 & & & \downarrow \lambda_A^{-1} & & & & & \downarrow \varrho_{A^*}^{-1} \\
 & & & A & & & & & A^*
 \end{array}$$

The following result is well known and it relates *-autonomous categories to compact closed ones.

Proposition 2.1.4. *Compact closed categories are *-autonomous categories with $\mathbf{1}$ as dualizing object and such that $(A \otimes B)^\perp = A^\perp \otimes B^\perp$.*

2.2 Functional programming languages and calculi

In this section we are going to introduce the programming language PCF, which is general enough to represent a core of a functional language. It has been introduced in [Plotkin, 1977] by Plotkin. PCF is based on λ -calculus and it is a re-visitation of the Scott's language LCF [Scott, 1993], which instead is based on combinatory logic. For ease of exposition, it is good to make a clear distinction between reduction systems and programming languages. A reduction system is a formal language with rewrite rules. A programming language may be regarded as a reduction system with some additional features. For our purpose, it is an essential feature of a programming language to be equipped with a specific reduction strategy. So, in this section, we will first introduce λ -calculus with its β -reduction rule, which is the rewriting system on which PCF is based. Then we will introduce the language PCF together with its operational semantics, which gives a strategy on the β -reduction.

2.2.1 λ -calculi and simply typed λ -calculi

In this section we will briefly recall the definition of λ -calculus with constants.

Definition 2.2.1. We assume a enumerable set of constants Const , ranging over c, c_1, c_2, \dots and a enumerable set of variables Var ranging over x, y, z, w, \dots .

1. Terms of λ -calculus are defined by the following syntax:

$$M ::= x \mid c \mid \lambda x.M \mid MM$$

We use M, N, \dots to range over terms of λ -calculus.

2. We remind that $\lambda x.M$ binds all the free occurrences of the variable x in M . Thus we denote with $FV(M)$ the set of free variables of M , while $M[N/x]$ denotes the capture free substitution of all occurrences of x in M by N .
3. The reduction system of the calculus consists of the contextual closure of the following rewriting rules

$$(\lambda x.M)N \rightarrow_{\beta} M[N/x] \quad (2.8)$$

$$\lambda x.Mx \rightarrow_{\eta} M \text{ if } x \notin FV(M) \quad (2.9)$$

plus additional rules (denoted with \rightarrow_{δ}) dealing with constants. We denote with $\rightarrow_{\beta\eta} = \rightarrow_{\beta} \cup \rightarrow_{\eta}$ and with $\rightarrow = \rightarrow_{\beta} \cup \rightarrow_{\eta} \cup \rightarrow_{\delta}$. \rightarrow_{β}^* , \rightarrow_{η}^* , $\rightarrow_{\beta\eta}^*$, \rightarrow^* denote respectively the transitive closure of \rightarrow_{β} , \rightarrow_{η} , $\rightarrow_{\beta\eta}$ and \rightarrow . $=_{\beta}$, $=_{\eta}$, $=_{\beta\eta}$ and $=_{\beta\eta\delta}$ are the minimal equivalences induced respectively by \rightarrow_{β} , \rightarrow_{η} , $\rightarrow_{\beta\eta}$ and \rightarrow .

A sub-term of a term M is called *redex* when it is under the form of the left hand side of a reduction rule. We will use notations from [Barendregt, 1984]. \equiv denotes the syntactical identity between terms, which is defined up to α -equivalence, i.e. bound variables renaming avoiding variable clashes. Observe that, the choice of the set Const of constant symbols is part of the definition of the language. If the set of constants Const is empty, then we will speak of **pure λ -calculus**. For this calculus, some important syntactical properties can be proved, like *confluence*, (called also *Church Rosser property*), *standardisation* and many others. For an introduction see [Ronchi Della Rocca and Paolini, 2004]. However, there may be many possible reasonable choices for Const which may give rise to many possible λ -calculi, as we will see in the following.

Types can be assigned to terms of the calculus to provide in a syntactical way semantic information on the behaviour of the terms. In fact using types, it is possible to characterise terms of the calculus enjoying good properties, like termination.

Definition 2.2.2 (Types). Let us assume a non-empty finite set Atoms of atomic types ranging over a, a_1, a_2, \dots . Types are generated by the following grammar

$$\sigma ::= a \mid \sigma \Rightarrow \sigma$$

$\frac{}{\Gamma, \mathbf{x}^\tau \vdash \mathbf{x}^\tau : \tau} \text{ (var)}$	$\frac{}{\Gamma \vdash \mathbf{c} : \sigma_{\mathbf{c}}} \text{ (const)}$	$\frac{\Gamma, \mathbf{y}^{\sigma_2}, \mathbf{x}^{\sigma_1}, \Delta \vdash \mathbf{M} : \tau}{\Gamma, \mathbf{x}^{\sigma_1}, \mathbf{y}^{\sigma_2}, \Delta \vdash \mathbf{M} : \tau} \text{ (e)}$
$\frac{\Gamma, \mathbf{x}^\sigma \vdash \mathbf{M} : \tau}{\Gamma \vdash \lambda \mathbf{x}^\sigma. \mathbf{M} : \sigma \Rightarrow \tau} \text{ (}\Rightarrow\text{I)}$		$\frac{\Gamma \vdash \mathbf{M} : \sigma \Rightarrow \tau \quad \Gamma \vdash \mathbf{N} : \sigma}{\Gamma \vdash \mathbf{MN} : \tau} \text{ (}\Rightarrow\text{E)}$

 Table 2.1: Generic Type Assignment System of a λ -calculus.

A typing system is a logical system which allow us to formalise when typing judgements are valid. For our purpose, in the following, we follow the typed approach (*à la Church*) instead of the type-assignment approach (*à la Curry*).

We begin by partitioning the set of variables Var , into subsets Var^σ which are non-empty, countable for every type σ and pairwise disjoint for any two distinct types σ, σ' . We will use the notation $\mathbf{x}^\sigma, \mathbf{y}^\sigma, \dots$ to range over variables in Var^σ . A **basis** $\Gamma = \mathbf{x}_1^{\sigma_1}, \dots, \mathbf{x}_n^{\sigma_n}$ is a finite list of variables in Var , where each variable appears at most once. We use Greek capital letters to range over basis. As usual, we use the comma to denote concatenation of basis as well as single variables. A **typing judgement** is under the form $\Gamma \vdash \mathbf{M} : \sigma$ where Γ is a basis, \mathbf{M} is a term and σ is a type. In the following definition, we assume having already fixed the sets Const and Atoms .

Definition 2.2.3 (Typing system). *Let us fix a type $\sigma_{\mathbf{c}}$, for all constants \mathbf{c} of the language. A typing judgement is said to be valid when it is the conclusion of a derivation consisting of the rules given in Table 2.1.*

The most basic property a typing system should have is **subject reduction**. It asks that the set of typed terms of the same type is closed under reduction. More formally a type system enjoys subject reduction when for all terms \mathbf{M} such that $\Gamma \vdash \mathbf{M} : \sigma$, if $\mathbf{M} \rightarrow \mathbf{N}$ then $\Gamma \vdash \mathbf{N} : \sigma$. A typing system which does not enjoy subject reduction is almost useless, if we want to study what kind of properties are preserved under reduction.

Definition 2.2.4 (σ -contexts). *Let $[\cdot]^\sigma$ be a special constant of type σ . The set of σ -contexts Ctx_σ is generated by the following grammar*

$$C ::= [\cdot]^\sigma \mid \mathbf{x} \mid \mathbf{c} \mid \lambda \mathbf{x}^\tau. C \mid CC$$

We denote with \mathbf{M}^σ a term having type σ . Let \mathbf{N}^σ be a term and let $C \in \text{Ctx}_\sigma$. We denote with $C[\mathbf{N}]$ the term obtained by replacing all the occurrences of $[\cdot]^\sigma$ with \mathbf{N} and allowing the capture of its free variables. Clearly \mathbf{N} is a term such that $\Gamma \vdash \mathbf{N} : \sigma$ and $C \in \text{Ctx}_\sigma$ do not imply that $C[\mathbf{N}]$ is a term of simply typed λ -calculus. For example, \mathbf{y}^σ is a term of simply typed λ -calculus, but if we consider the context $C = (\lambda \mathbf{x}^\sigma. \mathbf{xx})[\cdot]^\sigma$, we do not have that $C[\mathbf{y}^\sigma]$ is a term of simply typed λ -calculus, because the sub-term $\lambda \mathbf{x}^\sigma. \mathbf{xx}$ is not typable.

We give in the following, some examples of typed λ -calculi obtained from the generic type assignment system.

Pure simply typed λ -calculus. Assume we have fixed a set of atomic types. If we take the pure λ -calculus as base language, we obtain the *pure simply typed λ -calculus* or simply *λ^{\Rightarrow} -calculus*. It enjoys subject reduction and it is sound with respect to *strong normalisation*; it means that if a term M is such that $\Gamma \vdash M : \sigma$, for some Γ and σ , then there is no infinite reduction sequences starting from M .

λ -calculus with natural numbers, conditional and fix point operator. Let us take as atomic type the type ι called *type of natural numbers* and let us take \Rightarrow as the only type constructor. Let us consider the set of constants Const obtained by adding the following operators:

- the constant $\underline{0}$, having type ι ;
- the operator succ , having type $\iota \Rightarrow \iota$;
- the operator pred , having type $\iota \Rightarrow \iota$;
- the operator if , having type $\iota \Rightarrow \iota \Rightarrow \iota \Rightarrow \iota$;
- for every type σ , the operator \mathbf{Y}^σ , having type $(\sigma \Rightarrow \sigma) \Rightarrow \sigma$.

We will write \underline{n} for $\text{succ}(\dots(\text{succ}(\underline{0}))\dots)$ where succ is applied n -times to $\underline{0}$.

The rewriting rules dealing with the constant we added are the following.

$$\text{pred}(\text{succ}(\underline{n})) \rightarrow_\delta \underline{n} \quad (2.10)$$

$$\text{if } \underline{0} \ M \ N \rightarrow_\delta \ M \quad (2.11)$$

$$\text{if } \underline{n+1} \ M \ N \rightarrow_\delta \ N \quad (2.12)$$

$$\mathbf{Y}^\sigma(M) \rightarrow_\delta M(\mathbf{Y}^\sigma(M)) \quad (2.13)$$

The so obtained calculus is called *λ -calculus with natural numbers, conditional and fix point operators* or simply *$\lambda^{\Rightarrow, \iota, \text{if}, \mathbf{Y}}$ -calculus*. The typing system enjoys subject reduction, but it is not sound with respect to strong normalisation, due to the presence of \mathbf{Y} . It is possible to prove that this typed calculus is Turing-complete, i.e. it is possible to represent in it all computable functions between natural numbers.

$\lambda^{\Rightarrow, \iota, \text{if}, \mathbf{Y}}$ -calculus is the typed calculus on which many functional programming languages are based; an example will be PCF which will be introduced in the following section.

2.2.2 The programming language PCF

PCF - acronym for Programming language for Computable Functions - was introduced by Plotkin in [Plotkin, 1977] and it is based on the Logic of Computable Function (LCF for short) introduced by Scott in 1969 in the until recently unpublished paper [Scott, 1993]. As already anticipated, the formal system of PCF is obtained from $\lambda^{\Rightarrow, \iota, \text{if}, \mathbf{Y}}$ -calculus by imposing a specific evaluation strategy, which is given by the leftmost strategy (the redex in the leftmost position is reduced first).

Let $\mathcal{P} = \{M \mid \emptyset \vdash M : \iota\}$ be the set of *programs* and let $\mathcal{N} = \{\underline{0}, \dots, \underline{n}, \dots\}$ be the set of *numerals*. The notion of leftmost reduction strategy is formalised in the following definition.

Definition 2.2.5. We define the predicate $M \Downarrow \underline{n}$ (where $M \in \mathcal{P}$ and $\underline{n} \in \mathcal{N}$) to be true when it is the conclusion of a derivation consisting of the following rules.

$$\frac{}{\underline{0} \Downarrow \underline{0}} \quad \frac{M \Downarrow \underline{n}}{\text{succ } M \Downarrow \underline{n+1}} \quad \frac{M \Downarrow \underline{n+1}}{\text{pred } M \Downarrow \underline{n}} \quad \frac{M \Downarrow \underline{0}}{\text{pred } M \Downarrow \underline{0}} \quad \frac{M \Downarrow \underline{0} \quad P \Downarrow \underline{n}}{\text{if } M P Q \Downarrow \underline{n}}$$

$$\frac{M \Downarrow \underline{m+1} \quad Q \Downarrow \underline{n}}{\text{if } M P Q \Downarrow \underline{n}} \quad \frac{M[N/x] P_1 \dots P_i \Downarrow \underline{n}}{(\lambda x^\sigma.M) N P_1 \dots P_i \Downarrow \underline{n}} \quad \frac{(M(Y^\sigma M)) P_1 \dots P_i \Downarrow \underline{n}}{(Y^\sigma M) P_1 \dots P_i \Downarrow \underline{n}}$$

The following proposition relates the evaluation predicate with the rewriting rules.

Proposition 2.2.1. Let $M, N \in \mathcal{P}$ such that $M =_{\beta\eta\delta} N$. Then for any numeral \underline{n} we have

$$M \Downarrow \underline{n} \iff N \Downarrow \underline{n}$$

Proof. It follows from the application of Church Rosser Theorem and Standardisation Theorem. \square

Corollary 2.2.1. Let $M \in \mathcal{P}$ and \underline{n} be a numeral. Then $M =_{\beta\eta\delta} \underline{n} \iff M \Downarrow \underline{n}$.

The definition of observational pre-order can be obtained as in the previous section.

Definition 2.2.6 (Operational Equivalence). Let M^σ, N^σ be terms.

- $M \lesssim_\sigma N$ whenever, for all $C \in \text{Ctx}_\sigma$ such that $C[M], C[N] \in \mathcal{P}$, if $C[M] \Downarrow \underline{n}$ then $C[N] \Downarrow \underline{n}$
- $M \approx_\sigma N$ whenever $M \lesssim_\sigma N$ and $N \lesssim_\sigma M$.

Given two closed terms M^σ, N^σ , we call C a *separating context* when $C[M] \Downarrow \underline{n}$ and it is not the case that $C[N] \Downarrow \underline{n}$. It follows easily by definition that two term M^σ, N^σ are operationally distinct if and only if there exists a separating context.

There are several variants of PCF (with booleans, with a test for zero ...). Here we mention a variant presented in [Gunter, 1992], where the fix-point operator Y is replaced with a μ -abstraction $\mu x^\sigma.M$, having the following typing and evaluation rules.

$$\frac{\Gamma, x : \sigma \vdash M : \sigma}{\Gamma \vdash \mu x^\sigma.M : \sigma} \quad \frac{(M[\mu x^\sigma.M/x]) P_1 \dots P_i \Downarrow \underline{n}}{(\mu x^\sigma.M) P_1 \dots P_i \Downarrow \underline{n}}$$

Using μ -abstraction we can define the term $\mu x^{(\sigma \Rightarrow \sigma) \Rightarrow \sigma}. \lambda y^{\sigma \Rightarrow \sigma}. y(xy)$ which behaves like the fix-point operator Y^σ . Conversely, using the fix-point operator we can define the term $Y^\sigma(\lambda x^\sigma.M)$ which behaves like $\mu x^\sigma.M$.

2.3 Denotational Models

2.3.1 Categorical Model

The fundamental idea of a categorical treatment of models of typed calculi is that types should be interpreted as objects of an appropriate category and typed terms should be interpreted as morphisms on that category. The following definition has been taken from [de Paiva, 2006].

Definition 2.3.1 (Categorical Model). *A category \mathbb{C} is said to be a categorical model of a typed language \mathcal{L} when there is a function $\llbracket \cdot \rrbracket^{\mathbb{C}}$ mapping types into objects and type derivations into morphisms, such that*

- *For all type derivation $\Gamma \vdash M : \sigma$, there is a morphism $\llbracket \Gamma \vdash M : \sigma \rrbracket^{\mathbb{C}} : \llbracket \Gamma \rrbracket^{\mathbb{C}} \rightarrow \llbracket \sigma \rrbracket^{\mathbb{C}}$*
- *For all equalities between terms $M =_{\mathcal{L}} N$ where $\Gamma \vdash M : \sigma$ and $\Gamma \vdash N : \sigma$, we have $\llbracket \Gamma \vdash M : \sigma \rrbracket^{\mathbb{C}} = \llbracket \Gamma \vdash N : \sigma \rrbracket^{\mathbb{C}}$.*

A categorical model is said to be complete when there exists a category \mathbb{C} such that $\llbracket \Gamma \vdash M : \sigma \rrbracket^{\mathbb{C}}$ and $\llbracket \Gamma \vdash N : \sigma \rrbracket^{\mathbb{C}}$ are interpreted in the same morphism if and only if $M =_{\mathcal{L}} N$ is derivable.

Categorical models of typed λ -calculi

We now recall the definition of categorical model of a simply typed λ -calculus. For more details and proofs, see [Asperti and Longo, 1991, Crole, 1993, Lambek and Scott, 1986].

In the following definition, we assume that the set Const of constant symbols has already been fixed. Let \mathbb{C} be a cartesian category; we remind that $!_A : A \rightarrow \mathbb{T}$ is the unique morphism from the object $A \in \text{Obj}(\mathbb{C})$ to the terminal object \mathbb{T} ; moreover, we denote with $\gamma_{A,B} : A \times B \rightarrow B \times A$, the natural commutativity law of the categorical product: we remind that the categorical product is also a tensor product (see Section 2.1.4).

Definition 2.3.2. *Let \mathbb{C} be a cartesian closed category. A categorical model of simply typed λ -calculus consists of a structure $\mathcal{M}^{\mathbb{C}}$ specified by:*

- *for every atomic type a , an object $\llbracket a \rrbracket^{\mathbb{C}}$ of \mathbb{C} ;*
- *for every type σ , an object $\llbracket \sigma \rrbracket^{\mathbb{C}}$ of \mathbb{C} , such that $\llbracket \sigma \Rightarrow \tau \rrbracket^{\mathbb{C}} = \llbracket \sigma \rrbracket^{\mathbb{C}} \Rightarrow \llbracket \tau \rrbracket^{\mathbb{C}}$;*
- *for every constant symbol c of type σ_c a morphism $\llbracket c \rrbracket^{\mathbb{C}} : \mathbb{T} \rightarrow \llbracket \sigma_c \rrbracket^{\mathbb{C}}$.*
- *Given a context $\Gamma = \mathbf{x}_1^{\sigma_1}, \dots, \mathbf{x}_n^{\sigma_n}$, we set $\llbracket \Gamma \rrbracket^{\mathbb{C}} = \llbracket \sigma_1 \rrbracket^{\mathbb{C}} \times \dots \times \llbracket \sigma_n \rrbracket^{\mathbb{C}}$. Thus, for every term M such that $\Gamma \vdash M : \sigma$, a morphism*

$$\llbracket \Gamma \vdash M : \sigma \rrbracket^{\mathbb{C}} : \llbracket \Gamma \rrbracket^{\mathbb{C}} \rightarrow \llbracket \sigma \rrbracket^{\mathbb{C}}$$

defined by induction on the derivation of $\Gamma \vdash M : \sigma$, whose typing rules are given in Table 2.1 as follows;

1. $\llbracket \mathbf{x}_1^{\sigma_1}, \dots, \mathbf{x}_n^{\sigma_n} \vdash \mathbf{x}_i : \sigma_i \rrbracket^{\mathbb{C}} = \pi_i : \llbracket \sigma_1 \rrbracket^{\mathbb{C}} \times \dots \times \llbracket \sigma_n \rrbracket^{\mathbb{C}} \rightarrow \llbracket \sigma_i \rrbracket^{\mathbb{C}}$
2. $\llbracket \Gamma \vdash \mathbf{c} : \sigma_{\mathbf{c}} \rrbracket^{\mathbb{C}} = \llbracket \mathbf{c} \rrbracket^{\mathbb{C}} \circ !_{\llbracket \Gamma \rrbracket^{\mathbb{C}}} : \llbracket \Gamma \rrbracket^{\mathbb{C}} \rightarrow \llbracket \sigma_{\mathbf{c}} \rrbracket^{\mathbb{C}}$
3. $\llbracket \Gamma, \mathbf{x}^{\sigma_1}, \mathbf{y}^{\sigma_2}, \Delta \vdash \mathbf{M} : \tau \rrbracket^{\mathbb{C}} = f \circ (id_{\llbracket \Gamma \rrbracket^{\mathbb{C}}} \times \gamma_{\llbracket \sigma_1 \rrbracket^{\mathbb{C}}, \llbracket \sigma_2 \rrbracket^{\mathbb{C}}} \times id_{\llbracket \Delta \rrbracket^{\mathbb{C}}})$ where
 $f = \llbracket \Gamma, \mathbf{y}^{\sigma_2}, \mathbf{x}^{\sigma_1}, \Delta \vdash \mathbf{M} : \tau \rrbracket^{\mathbb{C}} : \llbracket \Gamma \rrbracket^{\mathbb{C}} \times \llbracket \sigma_2 \rrbracket^{\mathbb{C}} \times \llbracket \sigma_1 \rrbracket^{\mathbb{C}} \times \llbracket \Delta \rrbracket^{\mathbb{C}} \rightarrow \llbracket \tau \rrbracket^{\mathbb{C}};$
4. $\llbracket \Gamma \vdash \lambda \mathbf{x}^{\sigma}. \mathbf{M} : \sigma \Rightarrow \tau \rrbracket^{\mathbb{C}} = \text{curry}(f)$ where
 $f = \llbracket \Gamma, \mathbf{x}^{\sigma} \vdash \mathbf{M} : \tau \rrbracket^{\mathbb{C}} : \llbracket \Gamma \rrbracket^{\mathbb{C}} \times \llbracket \sigma \rrbracket^{\mathbb{C}} \rightarrow \llbracket \tau \rrbracket^{\mathbb{C}}.$
5. $\llbracket \Gamma \vdash \mathbf{M}\mathbf{N} : \tau \rrbracket^{\mathbb{C}} = \text{eval} \circ \langle f, g \rangle$ where
 $f = \llbracket \Gamma \vdash \mathbf{M} : \sigma \Rightarrow \tau \rrbracket^{\mathbb{C}} : \llbracket \Gamma \rrbracket^{\mathbb{C}} \rightarrow \llbracket \sigma \Rightarrow \tau \rrbracket^{\mathbb{C}}$
 $g = \llbracket \Gamma \vdash \mathbf{N} : \sigma \rrbracket^{\mathbb{C}} : \llbracket \Gamma \rrbracket^{\mathbb{C}} \rightarrow \llbracket \sigma \rrbracket^{\mathbb{C}}.$

Lemma 2.3.1 (Substitution Lemma). *Let \mathbf{N} be a term such that $\Gamma = \mathbf{x}_1^{\sigma_1}, \dots, \mathbf{x}_n^{\sigma_n} \vdash \mathbf{N} : \tau$ and let \mathbf{M}_i such that $\Delta \vdash \mathbf{M}_i : \sigma_i$ for $i = 1$ to n . Then $\Delta \vdash \mathbf{N}[\tilde{\mathbf{M}}/\tilde{\mathbf{x}}] : \tau$ and*

$$\llbracket \Delta \vdash \mathbf{N}[\tilde{\mathbf{M}}/\tilde{\mathbf{x}}] : \tau \rrbracket^{\mathbb{C}} = \llbracket \Gamma \vdash \mathbf{N} : \tau \rrbracket^{\mathbb{C}} \circ \langle \llbracket \Delta \vdash \mathbf{M}_1 : \sigma_1 \rrbracket^{\mathbb{C}}, \dots, \llbracket \Delta \vdash \mathbf{M}_n : \sigma_n \rrbracket^{\mathbb{C}} \rangle$$

Proof. The proof is a straightforward induction on the derivation of $\Gamma \vdash \mathbf{N} : \tau$. See [Crole, 1993, Lemma 4.5.2 page 169]. \square

Proposition 2.3.1 (Soundness). *Let $\mathcal{M}^{\mathbb{C}}$ be a categorical model of a typed λ -calculus and let \mathbf{M}, \mathbf{N} such that $\Gamma \vdash \mathbf{M}, \mathbf{N} : \sigma$. If $\mathbf{M} =_{\beta\eta} \mathbf{N}$ then $\llbracket \Gamma \vdash \mathbf{M} : \sigma \rrbracket^{\mathbb{C}} = \llbracket \Gamma \vdash \mathbf{N} : \sigma \rrbracket^{\mathbb{C}}$.*

Proof. The proof is by induction on the derivation of $\mathbf{M} =_{\beta} \mathbf{N}$. We develop the cases of β -reduction and η -reduction.. In the following, we let $f = \llbracket \Gamma, \mathbf{x}^{\sigma} \vdash \mathbf{M} : \tau \rrbracket^{\mathbb{C}}$.

$$\begin{aligned} \llbracket \Gamma \vdash (\lambda \mathbf{x}^{\sigma}. \mathbf{M})\mathbf{N} : \tau \rrbracket^{\mathbb{C}} &= \text{eval} \circ \langle \llbracket \Gamma \vdash \lambda \mathbf{x}^{\sigma}. \mathbf{M} : \sigma \Rightarrow \tau \rrbracket^{\mathbb{C}}, \llbracket \Gamma \vdash \mathbf{N} : \sigma \rrbracket^{\mathbb{C}} \rangle \\ &= \text{eval} \circ \langle \text{curry}(f), \llbracket \Gamma \vdash \mathbf{N} : \sigma \rrbracket^{\mathbb{C}} \rangle \\ &= \text{eval} \circ (\text{curry}(f) \times id_{\llbracket \sigma \rrbracket^{\mathbb{C}}}) \circ \langle id_{\llbracket \Gamma \rrbracket^{\mathbb{C}}}, \llbracket \Gamma \vdash \mathbf{N} : \sigma \rrbracket^{\mathbb{C}} \rangle \\ &= f \circ \langle id_{\llbracket \Gamma \rrbracket^{\mathbb{C}}}, \llbracket \Gamma \vdash \mathbf{N} : \sigma \rrbracket^{\mathbb{C}} \rangle \\ &= \llbracket \mathbf{M}[\mathbf{N}/\mathbf{x}] \rrbracket^{\mathbb{C}} \end{aligned}$$

where the last two steps are obtained since the category is cartesian closed and by Lemma 2.3.1. For the η -reduction, we have

$$\begin{aligned} \llbracket \Gamma \vdash \lambda \mathbf{x}^{\sigma}. \mathbf{M}\mathbf{x} : \sigma \Rightarrow \tau \rrbracket^{\mathbb{C}} &= \text{curry}(\llbracket \Gamma, \mathbf{x}^{\sigma} \vdash \mathbf{M}\mathbf{x} : \tau \rrbracket^{\mathbb{C}}) \\ &= \text{curry}(\text{eval} \circ \langle \llbracket \Gamma, \mathbf{x}^{\sigma} \vdash \mathbf{M} : \sigma \Rightarrow \tau \rrbracket^{\mathbb{C}}, \llbracket \Gamma, \mathbf{x}^{\sigma} \vdash \mathbf{x}^{\sigma} \rrbracket^{\mathbb{C}} \rangle) \\ &= \text{curry}(\text{eval} \circ (\llbracket \Gamma \vdash \mathbf{M} : \sigma \Rightarrow \tau \rrbracket^{\mathbb{C}} \times id_{\llbracket \sigma \rrbracket^{\mathbb{C}}})) \\ &= \llbracket \Gamma \vdash \mathbf{M} : \sigma \Rightarrow \tau \rrbracket^{\mathbb{C}} \end{aligned}$$

\square

Proposition 2.3.2 (Completeness). *Let \mathbf{M}, \mathbf{N} such that $\Gamma \vdash \mathbf{M}, \mathbf{N} : \sigma$. Then there is a Cartesian Closed Category \mathbb{C} such that $\llbracket \Gamma \vdash \mathbf{M} : \sigma \rrbracket^{\mathbb{C}} = \llbracket \Gamma \vdash \mathbf{N} : \sigma \rrbracket^{\mathbb{C}}$ if and only if $\mathbf{M} =_{\beta\eta} \mathbf{N}$.*

Proof. Let us consider the language obtained from simply typed λ -calculus by augmenting types with products the unit type and by augmenting terms with a pairing constructs, projections and a constant for the unit type (together with the reduction rules obeying the corresponding categorical equations). It suffices to prove that the category obtained by taking types as objects and equivalence classes of terms of this language, according to $=_{\beta\eta}$ plus the equations above mentioned, as morphism is Cartesian Closed. This construction is usually called *term model*. A proof of this can be found in [Lambek and Scott, 1986]. \square

Remark 2.3.1 (Lawvere-Lambek Isomorphism). *From the above results, Lambek was able to formulate a more general correspondence between typed λ -calculi (with product types) and Cartesian Closed Categories. Let **CCC** be the category having Cartesian Closed Categories as objects and functors which preserve the cartesian closed structure as morphisms. Let **Typed λ** be the category whose objects are typed λ -calculi (not necessarily freely generated) and whose morphisms are translations, i.e. homomorphic interpretations. There is a natural equivalence of categories, i.e. there are functors F and G*

$$\begin{array}{ccc} & F & \\ \text{Typed}\lambda & \xrightarrow{\quad} & \text{CCC} \\ & G & \end{array}$$

such that $F \circ G \cong id_{\text{CCC}}$ and $G \circ F \cong id_{\text{Typed}\lambda}$. Given a Cartesian Closed Category \mathbb{C} , we can associate the language $G(\mathbb{C})$, called *internal language* of \mathbb{C} . Roughly speaking, the types of $G(\mathbb{C})$ are the objects of \mathbb{C} and terms are obtained using the arrows of \mathbb{C} (where $\text{curry}(-)$ corresponds to λ -abstraction and eval to application); reduction rules are obtained from equalities in \mathbb{C} . Conversely given a language \mathcal{L} , $F(\mathcal{L})$ is cartesian closed category generated by the term model of \mathcal{L} (in an analogous way as in the proof of the proposition above). This correspondence is known as *Lawvere-Lambek Isomorphism* and it is the categorical counterpart of *Curry-Howard Isomorphism* (for more details see [Lambek and Scott, 1986]).

The following definition has been given in [Hyland and Ong, 2000], in order to establish a very general categorical context for recursion theory. There they adopt Platek's original conceptions [Platek, 1966], to make higher types, the conditional (or the definition by cases) at all types and fix points at all types, the basis for their discussion.

In the following definition, we denote with $\Delta : A \rightarrow A \times A$ the morphism $\langle id_A, id_A \rangle : A \rightarrow A \times A$ (which is usually called *diagonal morphism*).

Definition 2.3.3 (C-fix Category). *A c-fix category is a cartesian closed category \mathbb{C} equipped with the following additional structure.*

Numerals. *Let N be an object of \mathbb{C} equipped with maps $0 : \mathbb{T} \rightarrow N$, $\text{succ} : N \rightarrow N$. A numeral $n : \mathbb{T} \rightarrow N$ is defined by induction to be $0 : \mathbb{T} \rightarrow N$ for the base case and $n + 1$ is the composite of $\text{succ} \circ n : \mathbb{T} \rightarrow N$. We say that N is a simple object of numerals*

when there is a morphism $\text{pred} : N \rightarrow N$ such that the following diagram

$$\begin{array}{ccc} \mathbf{T} & \xrightarrow{n+1} & N \\ & \searrow n & \downarrow \text{pred} \\ & & N \end{array}$$

commutes.

Conditional. A map $\text{if} : A \times A \times A \rightarrow N$, called conditional morphism such that the following diagram

$$\begin{array}{ccccc} A \times A \cong \mathbf{T} \times A \times A & \xrightarrow{0 \times \text{id}_A \times \text{id}_A} & N \times A \times A & \xleftarrow{n+1 \times \text{id}_A \times \text{id}_A} & A \times A \cong \mathbf{T} \times A \times A \\ & \searrow \pi_1 & \downarrow \text{if} & \swarrow \pi_2 & \\ & & N & & \end{array}$$

commutes

Fix point. For every object $A \in \text{Obj}(\mathbb{C})$ a map $Y_A : A \Rightarrow A \rightarrow A$, called fix point operator such that the following diagram

$$\begin{array}{ccc} A \Rightarrow A & \xrightarrow{\Delta} & (A \Rightarrow A) \times (A \Rightarrow A) \\ \downarrow Y_A & & \downarrow \text{id}_A \times Y_A \\ A & \xleftarrow{\text{eval}} & (A \Rightarrow A) \times A \end{array}$$

commutes.

Observe that the choice of the simple object of numerals N , of the conditional morphism and the fix point operator are part of the definition of a c-fix category. Let \mathbb{C} be a c-fix category (a category admitting a simple object of numerals suffices); observe that the simple object of numerals N gives a categorical representation of numbers as morphisms between \mathbf{T} and N . Observe that, given two different natural number, it is not required that the corresponding morphisms are different.

Definition 2.3.4 (Categorical model of $\lambda^{\Rightarrow, \iota, \text{if}, Y}$ -calculus). A categorical model of $\lambda^{\Rightarrow, \iota, \text{if}, Y}$ -calculus consists of a c-fix category \mathbb{C} and a mapping $\llbracket \cdot \rrbracket^{\mathbb{C}}$ such that it forms a model of the underlying simply typed λ -calculus and satisfying the following conditions.

1. $\llbracket \vdash \underline{0} : \iota \rrbracket^{\mathbb{C}} = 0$, $\llbracket \vdash \text{succ} : \iota \Rightarrow \iota \rrbracket^{\mathbb{C}} = \text{curry}(\text{succ})$, $\llbracket \vdash \text{pred} : \iota \Rightarrow \iota \rrbracket^{\mathbb{C}} = \text{curry}(\text{pred})$
2. $\llbracket \vdash \text{if} : \iota \Rightarrow \iota \Rightarrow \iota \Rightarrow \iota \rrbracket^{\mathbb{C}} = \text{curry}(\text{curry}(\text{curry}(\text{if})))$
3. $\llbracket \vdash Y^\sigma : (\sigma \Rightarrow \sigma) \Rightarrow \sigma \rrbracket^{\mathbb{C}} = \text{curry}(Y^{\llbracket \sigma \rrbracket^{\mathbb{C}}})$

Proposition 2.3.3 (Soundness). Let $\mathcal{M}^{\mathbb{C}}$ be a categorical model of $\lambda^{\Rightarrow, \iota, \text{if}, Y}$ -calculus and let M, N such that $\Gamma \vdash M, N : \sigma$. If $M =_{\beta\eta\delta} N$ then $\llbracket \Gamma \vdash M : \sigma \rrbracket^{\mathbb{C}} = \llbracket \Gamma \vdash N : \sigma \rrbracket^{\mathbb{C}}$.

Proof. See [Hyland and Ong, 2000]. \square

Proposition 2.3.4 (Completeness). *Let \mathbb{M}, \mathbb{N} such that $\Gamma \vdash \mathbb{M}, \mathbb{N} : \sigma$. $\mathbb{M} =_{\beta\eta\delta} \mathbb{N}$ if and only if for all c-fix categories \mathbb{C} we have $\llbracket \Gamma \vdash \mathbb{M} : \sigma \rrbracket^{\mathbb{C}} = \llbracket \Gamma \vdash \mathbb{N} : \sigma \rrbracket^{\mathbb{C}}$.*

Proof. It suffices to prove that the category obtained by taking types as objects and equivalence classes of terms (according to $=_{\beta\eta\delta}$) as morphism is a c-fix category. See [Hyland and Ong, 2000]. \square

Models of PCF

In this section we will extend the notion of model of a typed calculus to model of PCF and we introduce some basic properties of a model of PCF. The definition appearing here has been taken from [Curien, 2007].

In this section, we assume that the considered category has enough points. Moreover in the interpretation function, we will omit the apex denoting the considered category and given a term \mathbb{M}^σ , we will write simply $\llbracket \mathbb{M} \rrbracket$ instead of $\llbracket \Gamma \vdash \mathbb{M} : \sigma \rrbracket$ to lighten the notation.

A *categorical model* of PCF is a categorical model of the underlying $\lambda^{\Rightarrow, \iota, \text{if}, Y}$ -calculus.

Remark 2.3.2. *In [Hyland and Ong, 2000], an equivalent (and in our opinion, nicer) definition of model of PCF is given, which makes use of the Isomorphism of Lawvere-Lambek (see Remark 2.3.1). Let \mathbf{PCF} be the c-fix category obtained from the term model of PCF (where terms are quotiented with respect to the observational equivalence). A categorical model of PCF is a functor from the category \mathbf{PCF} to any c-fix category \mathbb{C} which preserves the structure of c-fix category.*

Since a programming language is not just a term rewriting system, but also a language with an operational semantics, we would like that models of PCF enjoy additional properties, in order to study in a purely mathematical way the operational equivalence of PCF. A first requirement for a model of PCF is to be correct.

Definition 2.3.5 (Correctness). *A model of PCF is said to be correct when for all $\mathbb{M}^\sigma, \mathbb{N}^\sigma$ we have $\llbracket \mathbb{M}^\sigma \rrbracket = \llbracket \mathbb{N}^\sigma \rrbracket$ implies $\mathbb{M} \approx_\sigma \mathbb{N}$.*

We can reduce the issue of proving the correctness of the model to the issue of proving the adequacy of the model, namely we ask that the evaluation of programs is correctly interpreted in the model itself.

Definition 2.3.6 (Adequacy). *A model of PCF is said to be adequate when for all $\mathbb{M} \in \mathcal{P}$, $\llbracket \mathbb{M} \rrbracket = \llbracket \underline{n} \rrbracket$ is equivalent to $\mathbb{M} \Downarrow \underline{n}$*

Proposition 2.3.5. *If a model of PCF is adequate then it is also correct.*

Proof. Suppose $\llbracket \mathbb{M} \rrbracket = \llbracket \underline{n} \rrbracket$ and let $C[\mathbb{M}] \Downarrow \underline{n}$. Then $\llbracket C[\mathbb{M}] \rrbracket = \llbracket \underline{n} \rrbracket$ by adequacy. But $\llbracket C[\mathbb{N}] \rrbracket = \underline{n}$ by hypothesis and by definition of model. Thus, we can conclude $C[\mathbb{N}] \Downarrow \underline{n}$ again by adequacy. The case $C[\mathbb{N}] \Downarrow \underline{n}$ can be dealt in the same way. \square

A much stronger property is that the equational theory induced by the model is exactly the operational equivalence. This property is called full abstraction and it is formalised in the following definition.

Definition 2.3.7 (Full Abstraction). *A model of PCF is said to be fully abstract when for all $\mathbb{M}^\sigma, \mathbb{N}^\sigma$ we have $\llbracket \mathbb{M}^\sigma \rrbracket = \llbracket \mathbb{N}^\sigma \rrbracket$ if and only if $\mathbb{M} \approx_\sigma \mathbb{N}$.*

The following property is closely related to full abstraction.

Definition 2.3.8 (Definable Separability). *A model of PCF enjoys definable separability when for any two distinct points $f, g : \mathbb{T} \rightarrow \llbracket \sigma \rrbracket$ there exists an open term $\mathbf{x}^\sigma \vdash \mathbb{M} : \iota$ such that $\llbracket \mathbb{M} \rrbracket \circ f \neq \llbracket \mathbb{M} \rrbracket \circ g$.*

Proposition 2.3.6. *If a model of PCF is adequate and enjoys definable separability, then it is also fully abstract.*

Proof. One direction follows by Proposition 2.3.5. For the other direction let us consider two closed terms \mathbb{M}, \mathbb{N} such that $\vdash \mathbb{M}, \mathbb{N} : \sigma$ and $\llbracket \mathbb{M} \rrbracket \neq \llbracket \mathbb{N} \rrbracket$. By definable separability, there is an open term $\mathbf{x}^\sigma \vdash \mathbb{P} : \iota$ such that $\llbracket \mathbb{P} \rrbracket \circ \llbracket \mathbb{M} \rrbracket \neq \llbracket \mathbb{P} \rrbracket \circ \llbracket \mathbb{N} \rrbracket$. Then we can build the context $C = (\lambda \mathbf{x}^\sigma. \mathbb{P}).[\cdot]$ and prove that it is separating by adequacy. \square

We conclude the section with the notion of universality. A model of PCF is said to be universal when every point of an object is definable by a term of the language.

Definition 2.3.9 (Universality). *A model of PCF is universal when for all $f : \mathbb{T} \rightarrow \llbracket \sigma \rrbracket$ there is a closed term \mathbb{M}^σ such that $\llbracket \vdash \mathbb{M} : \sigma \rrbracket = f$.*

Universality implies full abstraction, as shown by the following proposition.

Proposition 2.3.7. *If a model of PCF is adequate and universal, then it is also fully abstract.*

Proof. One direction follows by Proposition 2.3.5. For the other direction let us consider two closed terms \mathbb{M}, \mathbb{N} such that $\vdash \mathbb{M}, \mathbb{N} : \tau_1 \Rightarrow \dots \tau_k \Rightarrow \iota$ and $\llbracket \mathbb{M} \rrbracket \neq \llbracket \mathbb{N} \rrbracket$. Since the category has enough points and by closure of the category, there are $g_1 : \mathbb{T} \rightarrow \llbracket \tau_1 \rrbracket, \dots, g_k : \mathbb{T} \rightarrow \llbracket \tau_k \rrbracket$ such that $\text{eval} \circ (\llbracket \mathbb{M} \rrbracket \times \langle g_1, \dots, g_k \rangle) \neq \text{eval} \circ (\llbracket \mathbb{N} \rrbracket \times \langle g_1, \dots, g_k \rangle)$. Since the model is universal then there are closed \mathbb{P}_i such that $\llbracket \mathbb{P}_i \rrbracket = g_i$. Thus we can build the following context $C[\cdot] = [\cdot]_{\mathbb{P}_1} \mathbb{P}_k$ which is separating by adequacy. So we can conclude. \square

However, as we will see in the following sections, to establish full abstraction we may require something weaker than universality of the model.

2.3.2 Scott Domains

The following definitions has been taken from [Gunter, 1992].

A *partial order* or *poset* is a pair $\langle D, \sqsubseteq \rangle$ where D is a set and \sqsubseteq is an order relation, often noted simply as D . A subset $X \subseteq D$ is *bounded* if and only if it has an upper bound; sometimes $x, y \in D$ are said *consistent* when there is $z \in D$ such that $x, y \sqsubseteq z$.

An element of D is *bottom* and denoted \perp if and only if $\perp \sqsubseteq d$ for each $d \in D$. A *least upper bound* or *supremum* or *join* of $X \subseteq D$ (denoted with $\bigsqcup X$) is the minimum element d of D , if it exists, such that $x \sqsubseteq d$, for each $x \in X$. A partial order D is *flat* when, for all $x, y \in D$, if $x \sqsubseteq z$ then $x = \perp$ or $x = y$. A nonempty subset X of D is *directed* if $\forall x, x' \in X \exists x'' \in X$ such that $x \sqsubseteq x''$ and $x' \sqsubseteq x''$, namely for each pair of elements of X there is an upper bound in X . A *directed complete partial order* (often abbreviated with *cpo*) is a poset D such that if $X \subseteq D$ is directed then there is $\bigsqcup X \in D$. A cpo is said to be *bounded complete* is a poset D such that if $X \subseteq D$ is bounded then there is $\bigsqcup X \in D$. Observe that a bounded complete cpo always admits a bottom element, since \emptyset is bounded (and so $\bigsqcup \emptyset = \perp$). An element $d \in D$ is said to be *compact* when for all directed $X \subseteq D$, if $d \sqsubseteq \bigsqcup X$ then there is $x \in X$ such that $d \sqsubseteq x$. A cpo D is said to be *ω -algebraic* when, for every $x \in D$, the set $X = \{a \sqsubseteq x \mid a \text{ compact}\}$ is directed and $\bigsqcup X = x$.

Definition 2.3.10 (Scott Domain). *A Scott Domain D is an ω -algebraic bounded complete cpo.*

Let A, B be Scott Domains. A function $f : A \rightarrow B$ is *monotonic* if and only if $\forall x, x' \in A$ if $x \sqsubseteq_A x'$ then $f(x) \sqsubseteq_B f(x')$. A monotonic function $f : A \rightarrow B$ is *continuous* when for every directed set $X \subseteq A$ we have $f(\bigsqcup X) = \bigsqcup f(X)$. A continuous function $f : A \rightarrow B$ is *strict* when $f(\perp) = \perp$. Given two function $f, g : A \rightarrow B$ we write $f \sqsubseteq g$ if for all $x \in A$ we have $f(x) \sqsubseteq g(x)$. We call this order *pointwise order* or *extensional order*.

Proposition 2.3.8. *Let $f : A \rightarrow B$ be a monotonic function. Then f is continuous iff for all compact y such that $y \sqsubseteq f(x)$ we have that there exists a compact $x_0 \sqsubseteq x$ such that $f(x_0) = y$.*

The category **Bcdom** having Scott Domains as objects and continuous functions as morphisms is Cartesian Closed.

Definition 2.3.11 (Cartesian Product). *Let A, B two Scott Domains. We define $A \times B = \{\langle a, b \rangle \mid a \in A, b \in B\}$ ordered with the componentwise order, i.e. $\langle a_1, b_1 \rangle \sqsubseteq \langle a_2, b_2 \rangle$ if $a_1 \sqsubseteq a_2$ and $b_1 \sqsubseteq b_2$.*

Definition 2.3.12 (Exponential Object). *Let A, B two Scott Domains. We define $A \Rightarrow B$ to be the Scott Domain of continuous functions between A and B extensionally ordered.*

Moreover the category **Bcdom** admits a simple object for numerals N , which is the flat domain of natural numbers defined as $N = \mathbb{N} \cup \{\perp\}$ and $d_1 \sqsubseteq d_2$ if either $d_1 = \perp$ or $d_1 = d_2$. It admits also a fix-point operator, defined as

$$\text{fix}(f) = \bigsqcup_n f^n(\perp_A)$$

where $f : A \rightarrow A$ is a continuous function and f^n is the n -times iteration of f , namely $f^0 = \text{id}_A$ and $f^{n+1} = f \circ f^n$. By Knaster-Tarsky Fix Point Theorem, it is also the least fix point of the function f .

Proposition 2.3.9. *Let $f : A \rightarrow A$ be a continuous function and let $g : A \rightarrow N$ and let $n \in \mathbb{N}$. Then*

$$g(\text{fix}(f)) = n \Rightarrow \exists k. g(f^k(\perp_A)) = n$$

Corollary 2.3.1. ***Bcdom** is a c-fix category.*

It follows from previous corollary that **Bcdom** is also a model of PCF. In [Plotkin, 1977] Gordon Plotkin proved that this model is adequate for the operational semantics of PCF.

Proposition 2.3.10 ([Plotkin, 1977]). ***Bcdom** is a correct model of PCF.*

We call this model, the *Scott model* of PCF. What about full abstraction of the Scott model of PCF? First of all, we say that the Scott model of PCF enjoys **compact definability** if for every compact element $c \in \llbracket \sigma \rrbracket$ of the domain there exists a closed term M^σ such that $\llbracket M^\sigma \rrbracket = c$. More generically, we can extend the notion of compact definability to all models admitting a notion of compact element.

Proposition 2.3.11 ([Milner, 1977, Plotkin, 1977]). *The Scott model of PCF is fully abstract if and only if it enjoys compact definability.*

For the leftward direction, let M^σ, N^σ such that $\llbracket M \rrbracket \neq \llbracket N \rrbracket$, where $\sigma = \sigma_1 \Rightarrow \dots \Rightarrow \iota$. By ω -algebraicity and by Proposition 2.3.8, there are compact $x_i \in \llbracket \sigma_i \rrbracket$ such that $\llbracket M \rrbracket(x_1, \dots, x_n) \neq \llbracket N \rrbracket(x_1, \dots, x_n)$. Since the model enjoys compact definability we can build the context $C = [\]P_1 \dots P_n$ where P_i is the term defining the compact x_i , which allows to separate the two terms M, N by adequacy. For the rightward direction, let us consider the following continuous function $por : N \times N \rightarrow N$, called *parallel or*¹.

$$por(0, \perp) = 0 \quad por(\perp, 0) = 0 \quad por(n+1, m+1) = 1$$

This function is not definable by a term of the language. The proof of this uses alternative adequate models of PCF, like the stable model which we will introduce in the following section. For a formal proof of this, see [Plotkin, 1977]. Hence consider the following two closed terms M_1, M_2 . Let Ω be a divergent term of PCF (it is not difficult to see that we can build one of this kind for every type of PCF)

$$M_1 = \lambda f^{\iota \Rightarrow \iota \Rightarrow \iota}. \text{if } (f \underline{0} \Omega) \text{ (if } (f \Omega \underline{0}) \text{ (if } (f \underline{1} \underline{1}) \underline{5} \Omega) \Omega) \Omega \quad M_2 = \Omega$$

These two terms are operationally equivalent, since we need a term behaving like *por* to build a separating context. But for the same reason, they are interpreted into different morphisms.

Corollary 2.3.2. *The Scott Model is not fully abstract for PCF.*

¹Note that we code booleans with numbers, where 0 stands for the value true while a non-zero number stands for false

In [Plotkin, 1977] PCF was extended with an additional operator called `pif` (a parallel conditional operator), whose evaluation rules are the following.

$$\frac{M \Downarrow \underline{0} \quad L \Downarrow \underline{n}}{\text{pif } M \text{ L R } \Downarrow \underline{n}} \quad \frac{M \Downarrow \underline{m+1} \quad R \Downarrow \underline{n}}{\text{pif } M \text{ L R } \Downarrow \underline{n}} \quad \frac{L \Downarrow \underline{n} \quad R \Downarrow \underline{n}}{\text{pif } M \text{ L R } \Downarrow \underline{n}}$$

This operator allows to define all compact elements of the domains which are interpretation of types, thus **Bcdom** is fully abstract for the so obtained language.

To achieve universality, it is necessary to consider the subcategory of **Bcdom** consisting of only effectively computable functions; we call this category **EffBcdom**. Unfortunately this model is not universal, even for PCF + `pif`. Gordon Plotkin in [Plotkin, 1977] extended PCF + `pif` with an additional operator, called \exists and he proved that **EffBcdom** is universal for PCF + `por` + \exists .

2.3.3 Coherence Spaces

Coherence spaces are a simple framework for Berry's stable functions [Berry, 1978], developed by Girard [Girard, 1987]. Proof details can be found in [Girard et al., 1989].

Definition 2.3.13. A coherence space X is a pair $\langle |X|, \supseteq_X \rangle$ where $|X|$ is a set of tokens called the web of X and \supseteq_X is a reflexive and symmetric relation between tokens of $|X|$ called the coherence relation² on X . A clique x of X is a subset of $|X|$ containing pairwise coherent tokens. The set of cliques of X is denoted $Cl(X)$. $Cl_{fin}(X)$ denotes the set of finite cliques of $Cl(X)$.

If X is a coherence space then $Cl(X)$ form a Scott Domain with respect to the set-theoretical inclusion. In particular,

- $\emptyset \in Cl(X)$ is the bottom element and $\{a\} \in Cl(X)$, for each $a \in |X|$,
- if $y \subseteq x$ and $x \in Cl(X)$ then $y \in Cl(X)$,
- if $D \subseteq Cl(X)$ is directed then $\bigcup D \in Cl(X)$,
- the set of compact elements is $Cl_{fin}(X)$.

Definition 2.3.14. Let X and Y be coherence spaces.

- If $f : Cl(X) \rightarrow Cl(Y)$ is a continuous function then f is stable if $\forall x, x' \in Cl(X)$, $x \cup x' \in Cl(X)$ implies $f(x \cap x') = f(x) \cap f(x')$.
- If $f : Cl(X) \rightarrow Cl(Y)$ is a stable function then f is linear if $f(x) = \bigcup_{a \in x} f(\{a\})$ and $f(\emptyset) = \emptyset$.

Some well-known characterisations hold for these functions.

²The strict incoherence \simeq_X is the complementary relation of \supseteq_X ; the incoherence \asymp_X is the union of relations \simeq_X and $=$; the strict coherence \frown_X is the complementary relation of \asymp_X .

Proposition 2.3.12. *Let X and Y be coherence spaces and $f : Cl(X) \rightarrow Cl(Y)$ be a monotone function.*

- *f is stable iff $\forall x \in Cl(X) \forall b \in f(x) \exists x_0 \subseteq_{fin} x$ such that $b \in f(x_0)$ and $\forall x' \subseteq x$, if $b \in f(x')$ then $x_0 \subseteq x'$.*
- *f is linear iff $\forall x \in Cl(X), \forall b \in f(x) \exists ! a \in x$ s.t. $b \in f(\{a\})$.*

We call **LinCoh** and **StabCoh** respectively the category of Coherence Spaces and Linear Function and the category of Coherence Spaces and Stable Functions.

The category LinCoh.

We start our analysis on the former, by showing that it is a Symmetric Monoidal Closed category, which is also Cartesian.

LinCoh is a symmetric monoidal category, and the tensor product is defined as follows.

Definition 2.3.15 (Tensor Product). *Let X and Y be coherence spaces. $X \otimes Y$ is the coherence space having $|X \otimes Y| = |X| \times |Y|$ as web, while $(a, b) \circ_{X \otimes Y} (a', b')$ if $a \circ_X a'$ and $b \circ_Y b'$. We also define $\mathbf{1} = \langle \{*\}, \circ_1 \rangle$ with $* \circ_1 *$.*

Linear functions can be represented by cliques, thus making the category monoidal closed.

Definition 2.3.16 (Linear Function Space). *Let X and Y be coherence spaces. $X \multimap Y$ is the coherence space having $|X \multimap Y| = |X| \times |Y|$ as web, while $(a, b) \circ_{X \multimap Y} (a', b')$ is defined as,*

$$a =_X a' \text{ implies } b =_Y b' \text{ and } a \frown_X a' \text{ implies } b \frown_Y b'$$

Let X and Y be coherence spaces. The *trace* of a linear function $f : Cl(X) \rightarrow Cl(Y)$ is the set defined as follows:

$$\text{tr}(f) = \{(a, b) \in |X| \times |Y| \mid b \in f(\{a\})\} \quad (2.14)$$

Let X and Y be coherence spaces and let $t \in Cl(X \multimap Y)$. Let us define the map $\text{eval}(t, -) : Cl(X) \rightarrow Cl(Y)$ to be the function such that

$$\text{eval}(t, x) = \{b \in |Y| \mid \exists a \in x, (a, b) \in t\} \quad (2.15)$$

Lemma 2.3.2. *If $f : Cl(X) \rightarrow Cl(Y)$ is a linear function then $\text{tr}(f) \in Cl(X \multimap Y)$. If $t \in Cl(X \multimap Y)$ then $\text{eval}(t, -) : Cl(X) \rightarrow Cl(Y)$ is a linear function.*

Finally, **LinCoh** is also Cartesian, with the cartesian product defined as follows. Instead of using the usual notation for it (namely $X \times Y$), we use the most suggestive $X \& Y$ to denote it.

Definition 2.3.17 (Cartesian Product). *Let X and Y be two coherence spaces. $X \& Y$ is the coherence space having $|X \& Y| = |X| + |Y|$ as web, while $\circ_{A\&B}$ is the smallest symmetric relation such that*

- $a \circ_X a'$ implies $in_1(a) \circ_{X\&Y} in_1(a')$.
- $b \circ_Y b'$ implies $in_2(b) \circ_{X\&Y} in_2(b')$.
- $in_1(a) \circ in_2(b)$ always.

The category **StabCoh**.

Let us now switch our attention into the category **StabCoh** of Coherence Space and stable functions. It is Cartesian Closed and the stable function space is defined as follows.

Definition 2.3.18. *Let X and Y be coherence spaces.*

$X \Rightarrow Y$ is the coherence space having $|X \Rightarrow Y| = Cl_{fin}(X) \times |Y|$ as web, while if $(x_0, b_0), (x_1, b_1) \in |X \Rightarrow Y|$, then $(x_0, b_0) \circ_{X \Rightarrow Y} (x_1, b_1)$ under the following conditions:

1. $x_0 \cup x_1 \in Cl(X)$ implies $b_0 \circ_Y b_1$;
2. $x_0 \cup x_1 \in Cl(X)$ and $b_0 = b_1$ imply $x_0 = x_1$.

The previous definition can be reformulated in term of strict coherence as follows.

Proposition 2.3.13. $(x_0, b_0) \frown_{X \Rightarrow Y} (x_1, b_1)$ iff $x_0 \cup x_1 \in Cl(X)$ implies $b_0 \frown_Y b_1$.

Stable function can be represented as cliques, as shown by the following definition.

Definition 2.3.19. *Let X and Y be coherence spaces.*

The trace $tr(f)$ of the stable function $f : Cl(X) \rightarrow Cl(Y)$ is the set of pairs $(x_0, b) \in Cl_{fin}(X) \times |Y|$ such that $b \in f(x_0)$ and $\forall x \subseteq x_0, b \in f(x)$ implies $x = x_0$.

The bridge between stable functions and cliques follows.

Lemma 2.3.3. *If $f : Cl(X) \rightarrow Cl(Y)$ is a stable function then $tr(f) \in Cl(X \Rightarrow Y)$.*

Let X, Y be coherence spaces and $t \in Cl(X \Rightarrow Y)$ and $x \in Cl(X)$. Let us define $\mathcal{F}(t) : Cl(X) \rightarrow Cl(Y)$ be the function such that

$$\mathcal{F}(t)(x) = \{b \in |Y| \mid \exists x_0 \in Cl(X) \ (x_0, b) \in t \wedge x_0 \subseteq x\}.$$

Lemma 2.3.4. *If $t \in Cl(X \Rightarrow Y)$ then $\mathcal{F}(t) : Cl(X) \rightarrow Cl(Y)$ is a stable function.*

The category **StabCoh** is a full subcategory of the categories of qualitative domains and dI-domains endowed with stable functions. It is not difficult to see that **StabCoh** is a c-fix category, thus it is also a model of PCF. This model is again correct but not fully abstract.

To this purpose, Luca Paolini, in [Paolini, 2006], introduced the language StPCF. This language is an extension of PCF with two additional operators, called `gor` and `strict?`, whose evaluation rules are the following.

$$\frac{M_1^l \Downarrow \underline{0} \quad M_2^l \Downarrow \underline{1}}{\text{gor } M_1^l M_2^l M_3^l \Downarrow \underline{0}} \quad \frac{M_2^l \Downarrow \underline{0} \quad M_3^l \Downarrow \underline{1}}{\text{gor } M_1^l M_2^l M_3^l \Downarrow \underline{0}} \quad \frac{M_3^l \Downarrow \underline{0} \quad M_1^l \Downarrow \underline{1}}{\text{gor } M_1^l M_2^l M_3^l \Downarrow \underline{0}}$$

$$\frac{M^{l \rightarrow l'} \underline{0} \Downarrow \quad M^{l \rightarrow l'}(\Omega^{l'}) \Uparrow}{\text{strict? } M^{l \rightarrow l'} \Downarrow \underline{0}} \quad \frac{M^{l \rightarrow l'} \underline{0} \Downarrow \quad M^{l \rightarrow l'}(\Omega^{l'}) \Downarrow}{\text{strict? } M^{l \rightarrow l'} \Downarrow \underline{0}}$$

Observe that the evaluation rule we give for `strict?` is not effective. However, an effective description of the evaluation has been given in [Paolini, 2006]. With this extension, he was able to establish a full abstraction result for **StabCoh** [Paolini, 2006].

Part I

Linearity in Denotational Semantics

The notion of linearity in denotational semantics became known after the works by Girard on Linear Logic [Girard, 1987]. Girard discovered a pleasant decomposition of intuitionistic implication into two new connectives: a binary connective, *linear implication* and an unary connective called *exponential*. From this work, several notions of linearity emerged in the literature and the most important ones have been already introduced in the previous paragraph.

In this part of the thesis, I study mainly the relationship between typing linearity, syntactical linearity and denotational linearity. I begin my analysis from the latter, by studying two mathematical frameworks which are intrinsically linear: the symmetric monoidal closed category of Coherence Spaces and linear functions [Girard, 1987] and the monistic framework of Ludics. For both models, a language being fully abstract for them is proposed; it is shown that \mathcal{SLPCF} enjoys full abstraction with respect to the least full sub-category of Coherence Spaces and linear function, consisting of the infinite flat domain of natural numbers and closed under linear arrow; in the same way, it is shown that a fragment of π -calculus enjoys full abstraction with respect to Girard's Ludics. About the two languages, it is possible to observe that both enjoy typing linearity, but \mathcal{SLPCF} is not syntactically linear, while the considered fragment of π -calculus is. Moreover, a further analysis on \mathcal{SLPCF} is made, by describing processes an \mathcal{SLPCF} -program generates by using linProc , a linearly typed process calculus based on the calculus of solos [Laneve and Victor, 2003].

This part will consist of three chapters, which are respectively titled **Semantically Linear Programming Languages**, **A Process Model for Linear Programs** and **Ludics Strategies and the π -calculus**

3 Semantically Linear PCF

ABSTRACT.

We propose a paradigmatic programming language (called *SℓPCF*) which is linear in a semantic sense. *SℓPCF* is not syntactically linear, namely its programs can contain more than one occurrences of the same variable. We study two denotational models of *SℓPCF*, respectively built in the setting of Scott Domains and Coherence Spaces. We prove that the Scott Model is adequate for the operational semantics of *SℓPCF*, while for the Coherence Space Model a full abstraction result holds. Furthermore, we discuss the independence of new syntactical operators and we address the universality problem.

Introduction

Coherence Spaces [Girard, 1987] are the result of the deep analysis of stable semantics [Berry, 1978] done by Jean-Yves Girard. These spaces provide a pleasant decomposition of stable functions via linear functions and exponential domain constructors. Such a decomposition is patently reflected in linear logic syntax. Since its inception, linear logic has inspired the introduction of many formal languages with sometimes different goals: resource-conscious evaluation, categorical language, implicit computational complexity, and so on.

This decomposition have been then generalised in the categorical study of models of Linear Logic (for an introduction to categorical models of Linear Logic, see [Melliès, 2003, de Paiva, 2006] among others). In particular, in this chapter, we are interested in a particular model of LL, which is given by Benton, Bierman, Hyland and de Paiva's Linear Categories [Benton et al., 1992]. All these categories are symmetric monoidal closed and they are equipped with a symmetric monoidal co-monad $!$ used to interpret the exponential modality and satisfying certain properties (see Section 3.1.2). The idea is to impose enough conditions on the co-monad in order to make its induced Kleisli category a Cartesian Closed Category with exponential object $A \Rightarrow B = !A \multimap B$. Observe that the original construction does not require this, but it would actually be the case, if the monoidal closed category is also cartesian.

In this chapter, we introduce a paradigmatic programming language (called *SℓPCF*)

which is linear in a semantic sense. More specifically, terms of $\mathcal{S}\ell\text{PCF}$ are syntactical descriptions of morphisms of suitable Linear Categories. For this purpose, we introduce the notion of $\mathcal{S}\ell\lambda$ Category which extends in the natural way the definition of Linear Category, and we show that $\mathcal{S}\ell\text{PCF}$ -terms can be soundly interpret into morphisms of such a category. By analogy with the notion of c -fix category (see Definition 2.3.3) we ask that this category admits a morphism acting like a “conditional” and a morphism acting like a “fix-point operator”. Furthermore, to interpret ground values, we require the existence of a distinguished object N with the usual zero and successor and predecessor morphisms satisfying the expected equation. However, since variables of ground type can be freely duplicated and erased, we need to ask that all numeral morphisms behaves properly with respect to the co-monad $!$. For this purpose we ask that N is itself a co-monoid and that there exists a $!$ -coalgebra $p : N \rightarrow !N$ which is also co-monoidal; furthermore we ask that all numeral morphisms are both co-algebraic and co-monoidal.

This notion of $\mathcal{S}\ell\lambda$ Category is also general enough to contain many interesting concrete models. In particular in this chapter we study two interesting models built respectively in the Scott Domain setting and the Coherence Space setting. We show that the model consisting of Scott Domains and Strict Continuous Functions is adequate with respect to the operational semantics of $\mathcal{S}\ell\text{PCF}$. We also show a full abstraction result for the category **LinCoh** of Coherence Spaces and Linear Functions. Finally, we show the relevance of our study from an higher-type computability point of view. We introduce new programming features for $\mathcal{S}\ell\text{PCF}$ and we show that they are independent from the remaining constructors of the language.

This analysis teaches us that linearity can be considered in many respects. Three main kinds of linearity emerge: syntactical, operational and denotational. The syntactical one claims a linear use of variables in terms. This kind of linearity is sometimes considered the computational counterpart of the linear logic linearity. $\mathcal{S}\ell\text{PCF}$ is not linear in this syntactical sense. However, $\mathcal{S}\ell\text{PCF}$ is linear in a denotational sense, since only purely linear functions can be defined in the language, without any kind of exponentiation. Finally, if redexes are not duplicated during the evaluation, then a notion of operational linearity arises. $\mathcal{S}\ell\text{PCF}$ is not endowed with a linear operational evaluation. Operational linearity is related to the notion of *simple term* [Klop, 2007] in λ -calculus, suggesting a more general linearity than the operational one, unrelated to a specific strategy.

A very general motivation behind our results is that linearity which is embedded in our language is more generous than the strict syntactical one, so it can be used in order to improve resource-conscious results related to Linear Logic. $\mathcal{S}\ell\text{PCF}$ is Turing-Complete. However, type-respecting recursive functions in StPCF are strictly less than that of the languages StPCF [Paolini, 2006] and $\text{PCF} + \text{H}$ [Longley, 2002]. In other words, linear functions between two coherence spaces are less than stable functions between the same domains. Since the syntactical constraints of $\mathcal{S}\ell\text{PCF}$ forbid useless duplications of redexes, we are utterly convinced that $\mathcal{S}\ell\text{PCF}$ can be fruitfully exploited in research fields like optimal evaluation (see for instance [Asperti and Guerrini, 1998, Petersen et al., 2003, Turner and Wadler, 1999]),

implicit computational complexity (see for instance [Asperti and Roversi, 2002, Bellantoni et al., 2000, Dal Lago et al., 2004]), linear computation (see for instance [Alves et al., 2006, Dal Lago, 2005]). In [Paolini and Piccolo, 2009] and in the following chapter, we study processes behind our programming language by translating programs in calculus of Solos [Laneve and Victor, 2003]. Such a translation is related to the description of game-semantics done in [Berger et al., 2001, Hyland and Ong, 1995, Yoshida et al., 2004], by using the π -calculus [Sangiorgi and Walker, 2001]. We plan also to study correspondences between Ludics [Girard, 2001] and the linear model, considered above.

The content of this section is an extended version of the article [Gaboardi and Piccolo, 2009] written with Marco Gaboardi and the article [Paolini and Piccolo, 2008] written with Luca Paolini.

3.1 Semantically Linear λ -calculus

In this section, we introduce a term rewriting system whose terms are a syntactical description of morphisms of a symmetric monoidal category. More specifically, this calculus was introduced in [Paolini and Piccolo, 2008], as the underlying calculus of \mathcal{SLPCF} . We call this calculus $\mathcal{S}\ell\lambda$ -calculus, acronym of *Semantically Linear λ -calculus*. $\mathcal{S}\ell\lambda$ -calculus is an opportune syntactical adaptation of *simply typed λ -calculus with conditional and fix point operator*, thus we will use the same syntactical conventions we introduced there (see Section 2.2.1).

3.1.1 Term rewriting system

Truth-values of $\mathcal{S}\ell\lambda$ -calculus are encoded as integers (zero encodes “true” while any other numeral stands for “false”).

Definition 3.1.1 ($\mathcal{S}\ell\lambda$ -types). *The set of $\mathcal{S}\ell\lambda$ -types is defined as follows, $\sigma, \tau ::= \iota \mid (\sigma \multimap \tau)$ where ι is the only atomic type (i.e. the type of numerals), \multimap is the only type constructor and σ, τ, \dots are meta-variables ranging over types.*

As customary \multimap associates to right. Hence $\sigma_1 \multimap \sigma_2 \multimap \sigma_3$ is an abbreviation for $\sigma_1 \multimap (\sigma_2 \multimap \sigma_3)$. It is easy to see that all types τ have the shape $\tau_1 \multimap \dots \multimap \tau_n \multimap \iota$, for some type τ_1, \dots, τ_n where $n \geq 0$. Let $\text{Var}^\sigma, \text{SVar}^\sigma$ be enumerable disjoint sets of variables of type σ . The set of *ground* variables is Var^ι , the set of *higher-order* variables is $\text{HVar} = \bigcup_{\sigma, \tau} \text{Var}^{\sigma \multimap \tau}$, and the whole set of variables is $\text{Var} = \text{Var}^\iota \cup \text{HVar} \cup \text{SVar}$. Letters \mathbf{x}^σ range over variables in Var^σ , letters $\mathbf{y}^\iota, \mathbf{z}^\iota, \dots$ range over variables in Var^ι , letters $\mathbf{f}^{\sigma \multimap \tau}, \mathbf{g}^{\sigma \multimap \tau}, \dots$ range over variables in HVar , while $F^\sigma, F_1^\sigma, F_2^\sigma, \dots$ range over *stable variables*, namely variables in SVar^σ . The reason of the name “stable variable” will be explained in Section 3.2. Last, \varkappa will denote any kind of variables. Latin letters $\mathbb{M}, \mathbb{N}, \mathbb{L}, \dots$ range over terms.

$\frac{}{\vdash \underline{0} : \iota}$ (z)	$\frac{}{\vdash \text{succ} : \iota \multimap \iota}$ (s)	$\frac{}{\vdash \text{pred} : \iota \multimap \iota}$ (p)	$\frac{\Gamma, \mathcal{X}_2^{\sigma_2}, \mathcal{X}_1^{\sigma_1}, \Delta \vdash \mathbf{M} : \tau}{\Gamma, \mathcal{X}_1^{\sigma_1}, \mathcal{X}_2^{\sigma_2}, \Delta \vdash \mathbf{M} : \tau}$ (ex)
$\frac{}{\mathbf{x}^l \vdash \mathbf{x} : \iota}$ (gv)	$\frac{}{\mathbf{f}^{\sigma \multimap \tau} \vdash \mathbf{f} : \sigma \multimap \tau}$ (hv)	$\frac{}{\mathbf{F}^\sigma \vdash \mathbf{F} : \sigma}$ (sv)	$\frac{\Gamma, \mathbf{x}^\sigma \vdash \mathbf{M} : \tau}{\Gamma \vdash \lambda \mathbf{x}^\sigma. \mathbf{M} : \sigma \multimap \tau}$ (λ)
$\frac{\Gamma \cap \Delta = \emptyset \quad \Gamma \vdash \mathbf{M} : \sigma \multimap \tau \quad \Delta \vdash \mathbf{N} : \sigma}{\Gamma, \Delta \vdash \mathbf{M}\mathbf{N} : \tau}$ (ap)		$\frac{\Gamma, \mathbf{x}_1^l, \mathbf{x}_2^l \vdash \mathbf{M} : \tau}{\Gamma, \mathbf{x}^l \vdash \mathbf{M}[\mathbf{x}/\mathbf{x}_1, \mathbf{x}_2] : \tau}$ (gc)	
$\frac{\Gamma \vdash \mathbf{M} : \tau}{\Gamma, \mathbf{x}^l \vdash \mathbf{M} : \tau}$ (gw)	$\frac{\Gamma \cap \Delta = \emptyset \quad \Gamma \vdash \mathbf{M} : \iota \quad \Delta \vdash \mathbf{L} : \iota \quad \Delta \vdash \mathbf{R} : \iota}{\Gamma, \Delta \vdash \text{elif } \mathbf{M} \mathbf{L} \mathbf{R} : \iota}$ (lif)		
$\frac{\Gamma, \mathbf{F}_1^\sigma, \mathbf{F}_2^\sigma \vdash \mathbf{M} : \tau}{\Gamma, \mathbf{F}^\sigma \vdash \mathbf{M}[\mathbf{F}/\mathbf{F}_1, \mathbf{F}_2] : \tau}$ (sc)	$\frac{\Gamma \vdash \mathbf{M} : \tau}{\Gamma, \mathbf{F}^\sigma \vdash \mathbf{M} : \tau}$ (sw)	$\frac{\Gamma^l, \Delta^*, \mathbf{F}^\sigma \vdash \mathbf{M} : \sigma}{\Gamma^l, \Delta^* \vdash \mu \mathbf{F}. \mathbf{M} : \sigma}$ (μ)	

Table 3.1: Type assignment system for $\mathcal{S}\ell\lambda$ -calculus

A **basis** Γ is a finite list of variables in Var , where each variable appears at most once. We denote with Γ^* (resp Γ^l) a basis Γ containing variables in SVar (resp. in GVar). We will denote with Γ, Δ the concatenation of two basis and with $\Gamma \cap \Delta$ the intersection of two basis, defined in the expected way.

Definition 3.1.2. *Typed terms of $\mathcal{S}\ell\lambda$ -calculus are defined by using a type assignment proving judgement of the shape $\Gamma \vdash \mathbf{M} : \sigma$, in Table 3.1.*

Typing rules deserve some explanation. As expected $\underline{0}$ has type ι (rule **(z)**) and succ , pred (corresponding to the successor and predecessor functions) has type $\iota \multimap \iota$ (rules **(s)** and **(p)**). Using successor and zero, we can define the numeral \underline{n} to be $\text{succ}(\dots(\text{succ}(\underline{0}))\dots)$ where succ is applied n -times to $\underline{0}$. The exchange rule **(ex)** allows to permute variables of the basis. Rules **(gv)**, **(hv)** and **(sv)** are the rules introducing the three different kinds of variables of the language which are respectively ground variables, higher-order variables and stable variables. Rules **(λ)** and **(ap)** are the standard \multimap -introduction and \multimap -elimination rules. Rule **(lif)** introduces the conditional construct. Observe that in both rule **(ap)** and **(lif)**, we require in the premise that the two basis Γ and Δ do not have any common higher-order variable, since variables of this kind are treated linearly. Thus weakening and explicit contraction on them is not allowed: there is only an implicit additive contraction on these variables made by the rule **(lif)**. Instead, weakening and explicit contraction is allowed on ground variables and stable variables by the rules **(gc)**, **(gw)**, **(sc)** and **(sw)**. This justifies the shape of rule **(μ)** which asks that only terms without any free occurrence of an higher-order variable can be used for recursion.

Sometimes types will be omitted when they are clear from the context or uninteresting. Note that given types of all variables of a term \mathbf{M} , there is a unique σ such that \mathbf{M} has type σ (sometimes denoted with \mathbf{M}^σ). However, the typing derivation to obtain it is not necessary unique, due to the presence of structural rules. Sometimes,

parentheses are omitted, always by respecting the following conventions: application associates to the left and application binds more tightly than abstraction, i.e. $\lambda x.MNL = (\lambda x.((MN)L))$. Free variables of terms are defined as expected. A term M is *closed* if and only if $FV(M) = \emptyset$, otherwise M is *open*. Terms are considered up to α -equivalence, namely a bound variable can be renamed provided no free variable is captured. Moreover, $M[\underline{n}/y]$, $M[N/f]$ and $M[N/F]$ denote the expected capture-free substitutions. Let $\mathcal{P} = \{M' \mid FV(M') = \emptyset\}$ be the set of *programs* and let $\mathcal{N} = \{\underline{0}, \dots, \underline{n}, \dots\}$ be the set of *numerals*.

Definition 3.1.3. We denote \rightsquigarrow the firing (without any context-closure) of one of the following rules:

$$\begin{array}{lll} (\lambda f^{\sigma \rightarrow \tau}.M)N \rightsquigarrow_{\beta} M[N/f] & (\lambda z'.M)\underline{n} \rightsquigarrow_{\iota} M[\underline{n}/z] & \mu F.M \rightsquigarrow_{\Upsilon} M[\mu F.M/F] \\ \text{pred}(\text{succ } \underline{n}) \rightsquigarrow_{\delta} \underline{n} & \text{lif } \underline{0} L R \rightsquigarrow_{\delta} L & \text{lif } \underline{n+1} L R \rightsquigarrow_{\delta} R \end{array}$$

We call *redex* each term or sub-term having the shape of a left-hand side of rules defined above. We denote $\rightarrow_{\mathcal{S}}$ the contextual closure of \rightsquigarrow . Moreover, we denote $\rightarrow_{\mathcal{S}}^*$ and $=_{\mathcal{S}}$ respectively, the reflexive and transitive closure of $\rightarrow_{\mathcal{S}}$ and the reflexive, symmetric and transitive closure of $\rightarrow_{\mathcal{S}}$.

In Definition 3.1.3, we implicitly assume that terms on the left-hand side of the rules are well typed. We remark that \rightsquigarrow_{β} formalises a call-by-name parameter passing in case of an higher-order argument. On the other hand, \rightsquigarrow_{ι} formalises a call-by-value parameter passing, namely the reduction can fire only when the argument is a numeral. As done in [Berry et al., 1985], it is easy to prove properties as subject-reduction, post-position of δ -rules in a sequence of reductions, the confluence and a standardisation theorem. Let us observe that terms of $\mathcal{S}\ell\lambda$ -calculus are not reduction linear. Let us consider the term $(\lambda f^{\iota \rightarrow \iota}. \text{lif } x \ f2 \ f3)(\lambda x'.((\lambda z'.z)\underline{5}))$: the contraction of the outermost redex will duplicate the innermost one.

Definition 3.1.4. Let $[\sigma]$ be a special constant of type σ . The set of σ -context Ctx_{σ} is generated by the following grammar:

$$\begin{array}{l} \text{C}[\sigma] ::= [\sigma] \mid \lambda^{\tau} \mid F^{\tau} \mid \underline{0} \mid \text{succ} \mid \text{pred} \\ \mid \text{lif } \text{C}[\sigma] \ \text{C}[\sigma] \ \text{C}[\sigma] \mid (\lambda \lambda. \text{C}[\sigma]) \mid (\text{C}[\sigma] \text{C}[\sigma]) \mid \mu F. \text{C}[\sigma] \end{array}$$

$\text{C}[\mathbb{N}^{\sigma}]$ denotes the result obtained by replacing all the occurrences of $[\sigma]$ in the context $\text{C}[\sigma]$ by the term \mathbb{N}^{σ} and by allowing the capture of its free variables.

Clearly, if \mathbb{N}^{σ} is a $\mathcal{S}\ell\lambda$ -term and $\text{C}[\sigma] \in \text{Ctx}_{\sigma}$, then it might be not the case that that $\text{C}[\mathbb{N}^{\sigma}]$ is a $\mathcal{S}\ell\lambda$ -term. For example consider a term M^{σ} having some free occurrences of higher order variables and consider a context $\text{C}[\sigma]$ having more than one occurrence of $[\sigma]$. Clearly $\text{C}[M] \notin \mathcal{S}\ell\text{PCF}$.

3.1.2 Categorical model of $S\ell\lambda$ -calculus

In this section we define a categorical model of $S\ell\lambda$ -calculus and we prove its soundness with respect to $=_{\mathcal{S}}$. We will assume some familiarity with Category Theory, in particular the notions of monoidal categories, co-monoids, co-monads, adjunctions and monoidal functors. For an introduction, see either Chapter 2 Section 2.1 of this thesis or [MacLane, 1998]. We begin by recalling the definition of Linear Category, given by Benton, Bierman, Hyland and de Paiva [Benton et al., 1992], which proposes a categorical notion of a model of Intuitionistic Linear Logic.

Definition 3.1.5 (Linear Category [Benton et al., 1992]). A Linear Category $\langle \mathbb{L}, !, \delta, \varepsilon, q, d, e \rangle$ consists of

- a symmetric monoidal closed category $\langle \mathbb{L}, \otimes, \multimap, \mathbf{1} \rangle$;
- a symmetric monoidal co-monad $\langle !, \delta, \varepsilon, q_{A,B}, q_{\mathbf{1}} \rangle : \mathbb{L} \rightarrow \mathbb{L}$, such that
 1. for every free $!$ -coalgebra $\langle !A, \delta_A \rangle$ there are two distinguished monoidal natural transformations with components $d_A : !A \rightarrow !A \otimes A$ and $e_A : !A \rightarrow \mathbf{1}$ which form a commutative comonoid and are coalgebra morphisms;
 2. whenever $f : \langle !A, \delta_A \rangle \rightarrow \langle !B, \delta_B \rangle$ is a coalgebra morphism between free coalgebras, then it is also a co-monoid morphism.

We will call a monoidal co-monad $!$ satisfying the above conditions, an exponential co-monad.

A Linear Category provides a sound categorical model of Intuitionistic Linear Logic [Benton et al., 1992]. A $S\ell\lambda$ -category will be a Linear Category, thus every $S\ell\lambda$ -Category is a model of Intuitionistic Linear Logic. However, it is necessary to augment it with other opportune features, in order to generate a sound categorical model of $S\ell\lambda$ -calculus.

Numerals

First of all, we need a canonical object to interpret ground type ι and opportune morphisms to interpret successor and predecessor. The following definition is an adaptation to monoidal categories of the definition of “simple object of numerals” given in [Hyland and Ong, 2000] (see Definition 2.3.3).

Definition 3.1.6 (Monoidal Object of Numerals). Let \mathbb{C} be a symmetric monoidal category. Let N be an object equipped with two morphisms $0 : \mathbf{1} \rightarrow N$ and $\text{succ} : N \rightarrow N$. A numeral $n : \mathbf{1} \rightarrow N$ is defined inductively as the map $0 : \mathbf{1} \rightarrow N$ for the base case, while the map $n + 1 : \mathbf{1} \rightarrow N$ is equal to $\text{succ} \circ n$. N is said to be a monoidal object of numerals when it is also equipped with a morphism $\text{pred} : N \rightarrow N$ such that the following diagram commutes

$$\begin{array}{ccc}
 \mathbf{1} & \xrightarrow{n+1} & N \\
 & \searrow n & \downarrow \text{pred} \\
 & & N
 \end{array}$$

The definition above is very weak. It is in fact not required that given two numerals $m : \mathbf{1} \rightarrow N$ and $n : \mathbf{1} \rightarrow N$ with $n \neq m$ (viewed as numbers), they are distinct morphisms in \mathbb{C} . Moreover the definition given above does not allow to represent neither recursive nor primitive recursive functions in \mathbb{C} . An analogous situation is also present in the definition of simple object of numerals given in [Hyland and Ong, 2000].

For sake of completeness, we now compare the above definition with the definition given in [Paré and Román, 1988]. It extends the notion of natural number object, which was specifically defined for cartesian categories in [Román, 1989], to any monoidal category.

Definition 3.1.7 ([Paré and Román, 1988, Mackie et al., 1993]). *Let \mathbb{C} be a symmetric monoidal closed category. By a natural number object in \mathbb{C} we mean an object N and two morphisms $0 : \mathbf{1} \rightarrow N$ and $\text{succ} : N \rightarrow N$ such that, given any pair of morphisms $c : \mathbf{1} \rightarrow A$ and $f : A \rightarrow A$ there is a unique $h : N \rightarrow A$ making the following diagrams commute.*

$$\begin{array}{ccc} \mathbf{1} & \xrightarrow{0} & N \\ & \searrow c & \downarrow h \\ & & A \end{array} \qquad \begin{array}{ccc} N & \xrightarrow{\text{succ}} & N \\ \downarrow h & & \downarrow h \\ A & \xrightarrow{f} & A \end{array}$$

In [Paré and Román, 1988], Paré and Román show that in any symmetric monoidal category \mathbb{C} with a natural number object, the theory of primitive recursive functions can be developed. This is done by considering the category of commutative co-monoids $\mathbb{C}\text{CoMon}_{\mathbb{C}}$ in \mathbb{C} , which is cartesian [Paré and Román, 1988] and where the theory of natural number objects is well developed. In detail, if $\langle C, d_C, e_C \rangle$ and $\langle D, d_D, e_D \rangle$ are two commutative co-monoids, then its cartesian product is given by $\langle C \otimes D, d_C \otimes d_D, e_C \otimes e_D \rangle$, while the pairing and the projections are defined as

$$\begin{aligned} \pi_1 & \text{ is the composite of } C \otimes D \xrightarrow{id_C \otimes e_D} C \otimes \mathbf{1} \xrightarrow{\rho} C \\ \pi_2 & \text{ is the composite of } C \otimes D \xrightarrow{e_C \otimes id_D} \mathbf{1} \otimes D \xrightarrow{\lambda} D \\ \langle f, g \rangle & \text{ is the composite of } E \xrightarrow{d_E} E \otimes E \xrightarrow{f \otimes g} C \otimes D \end{aligned}$$

for $f : E \rightarrow C$ and $g : E \rightarrow D$. The terminal object is $\mathbf{1}$.

More specifically in [Paré and Román, 1988] it is shown that if N is a natural number object, then it is a commutative co-monoid, by taking the morphisms $w_N : N \rightarrow \mathbf{1}$ and $c_N : N \rightarrow N \otimes N$ to be the unique morphisms making the following diagrams commute

$$\begin{array}{ccc} \mathbf{1} & \xrightarrow{0} & N \\ \downarrow id_{\mathbf{1}} & & \downarrow w_N \\ & & \mathbf{1} \end{array} \qquad \begin{array}{ccc} N & \xrightarrow{\text{succ}} & N \\ \downarrow w_N & & \downarrow w_N \\ \mathbf{1} & \xrightarrow{id_{\mathbf{1}}} & \mathbf{1} \end{array} \qquad \begin{array}{ccc} \mathbf{1} \cong \mathbf{1} \otimes \mathbf{1} & \xrightarrow{0} & N \\ \downarrow 0 \otimes 0 & & \downarrow c_N \\ & & N \otimes N \end{array} \qquad \begin{array}{ccc} N & \xrightarrow{\text{succ}} & N \\ \downarrow c_N & & \downarrow c_N \\ N \otimes N & \xrightarrow{\text{succ} \otimes \text{succ}} & N \otimes N \end{array}$$

Furthermore $0 : \mathbf{1} \rightarrow N$ and $\text{succ} : N \rightarrow N$ are both comonoid morphisms. Thus, all numerals are co-monoid morphisms, and all primitive recursive functions can

be represented, in the same way as they were represented in a Cartesian Category [Román, 1989]. Observe again that the above definition of *natural number object* does not require that given two numerals $n : \mathbf{1} \rightarrow N$ and $m : \mathbf{1} \rightarrow N$ with $n \neq m$ (viewed as numbers) are distinct morphisms in \mathbb{C} . But in [Paré and Román, 1988], it has been shown that if this holds and if \mathbb{C} is monoidal closed, then \mathbb{C} is equivalent to the one-object one-morphism category $\mathbb{1}$ (an analogous fact holds also for Cartesian Closed Categories [Hyland and Ong, 2000]).

The following proposition is a corollary of the above statement.

Proposition 3.1.1. *Let \mathbb{C} be a symmetric monoidal closed category with a natural number object N . Then N is a monoidal object of numerals.*

Proof. Let $h : N \rightarrow N \otimes N$ be the unique morphism making the following diagrams commute (the pairing and projections in the category $\mathbb{C}_{\text{OMONC}}$ are defined above).

$$\begin{array}{ccc} \mathbf{1} & \xrightarrow{0} & N \\ & \searrow \langle 0,0 \rangle & \downarrow h \\ & & N \otimes N \end{array} \quad \begin{array}{ccc} N & \xrightarrow{\text{succ}} & N \\ \downarrow h & & \downarrow h \\ N \otimes N & \xrightarrow{\langle \text{succ} \circ \pi_1, \pi_1 \rangle} & N \otimes N \end{array}$$

Thus, a choice for $\text{pred} : N \rightarrow N$ could be the following

$$\text{pred is the composite of } N \xrightarrow{h} N \otimes N \xrightarrow{\pi_2} N$$

We prove by induction on n that $n = \text{pred} \circ n + 1$. We remind that $\mathbb{C}_{\text{OMONC}}$ is a cartesian category. For the base case, we have

$$\begin{aligned} \text{pred} \circ 1 &= \pi_2 \circ h \circ \text{succ} \circ 0 \\ &= \pi_2 \circ \langle \text{succ} \circ \pi_1, \pi_1 \rangle \circ h \circ 0 \\ &= \pi_2 \circ \langle \text{succ} \circ \pi_1, \pi_1 \rangle \circ \langle 0, 0 \rangle \\ &= \pi_2 \circ \langle 1, 0 \rangle \\ &= 0 \end{aligned}$$

For the inductive case, we have

$$\begin{aligned} \text{pred} \circ n + 2 &= \pi_2 \circ h \circ \text{succ} \circ n + 1 \\ &= \pi_2 \circ \langle \text{succ} \circ \pi_1, \pi_1 \rangle \circ h \circ \text{succ} \circ n \\ &= \pi_2 \circ \langle \text{succ} \circ \pi_1, \pi_1 \rangle \circ \langle \text{succ} \circ \pi_1, \pi_1 \rangle \circ h \circ n \\ &= \pi_2 \circ \langle \text{succ} \circ \text{succ} \circ \pi_1, \text{succ} \circ \pi_1 \rangle \circ h \circ n \\ &= \text{succ} \circ \pi_1 \circ h \circ n \\ &= \text{succ} \circ \pi_2 \circ \langle \text{succ} \circ \pi_1, \pi_1 \rangle \circ h \circ n \\ &= \text{succ} \circ \text{pred} \circ n + 1 \\ &= n + 1 \end{aligned}$$

where in the last line we use inductive hypothesis. □

We now give a notion of natural number object in Linear Categories. We observe that a monoidal object of numerals is too weak, in order to be a sound interpretation of the type ι of $\mathcal{S}\ell\text{PCF}$. The structure of monoidal object of numerals should be enriched to obtain an *exponential object of numerals*; it will be a monoidal object of numerals with additional morphisms allowing to duplicate and weaken occurrences of them and whose other morphisms respects the comonoidal structure induced by the exponential co-monad.

Definition 3.1.8 (Exponential object of numerals). *Let $\langle \mathbb{L}, !, \delta, \varepsilon, q, d, e \rangle$ be a Linear Category. An exponential object of numerals is a $!$ -coalgebra $\langle N, p \rangle$ such that*

1. N is a monoidal object of numerals.
2. There exists two morphisms $w_N : N \rightarrow \mathbf{1}$ and $c_N : N \rightarrow N \otimes N$ which form a commutative co-monoid and are such that
 - a) $0 : \mathbf{1} \rightarrow N$ and $\text{succ} : N \rightarrow N$ are both co-algebras and co-monoid morphisms.
 - b) $p : N \rightarrow !N$ is a co-monoid morphism.

Corollary 3.1.1. *Let $\langle \mathbb{L}, !, \delta, \varepsilon, q, d, e \rangle$ be a Linear Category and let N be a natural number object such that $\langle N, p \rangle$ is a $!$ -coalgebra satisfying*

1. $p : N \rightarrow !N$ is a co-monoid morphism.
2. $0 : \mathbf{1} \rightarrow N$ and $\text{succ} : N \rightarrow N$ are coalgebra morphisms

Then $\langle N, p \rangle$ is an exponential object of numerals.

$\mathcal{S}\ell\lambda$ -category

We are now ready to give the definition of $\mathcal{S}\ell\lambda$ -category. An $\mathcal{S}\ell\lambda$ -category is a Linear Category admitting an exponential object of numerals, together with a “conditional-like” morphism and a fix-point morphism for every object B in the Kleisli category over the co-monad $!$, which is cartesian closed. This leads to the following definition.

Definition 3.1.9 ($\mathcal{S}\ell\lambda$ Category). *A $\mathcal{S}\ell\lambda$ Category is a linear category $\mathcal{L} = \langle \mathbb{L}, !, \delta, \varepsilon, q, d, e \rangle$ such that*

Numerals. \mathbb{L} admits an exponential object of numerals $\langle N, p \rangle$.

Conditional Operator. \mathbb{L} is cartesian and there exists a morphism $\ell\text{if} : N \otimes (N \times N) \rightarrow N$ such that, for all $f, g : \mathbf{1} \rightarrow N$, the following diagram commutes

$$\begin{array}{ccccc}
 \mathbf{1} \cong \mathbf{1} \otimes \mathbf{1} & \xrightarrow{0 \otimes \langle f, g \rangle} & N \otimes (N \times N) & \xleftarrow{n+1 \otimes \langle f, g \rangle} & \mathbf{1} \cong \mathbf{1} \otimes \mathbf{1} \\
 & \searrow f & \downarrow \ell\text{if} & \swarrow g & \\
 & & N & &
 \end{array}$$

Fix-Point Operator. The Kleisli category $\mathbb{L}_!$ (which is Cartesian Closed) admits a fix-point operator $fix_B : !(B \multimap B) \rightarrow B$ for any object B . We remind that, by the Kleisli construction, we have that the following diagram commutes.

$$\begin{array}{ccc}
 !(B \multimap B) & \xrightarrow{d_{!B \multimap B}} & !(B \multimap B) \otimes !(B \multimap B) \\
 \downarrow fix_B & & \downarrow \varepsilon_{!B \multimap B} \otimes (!fix_B \circ \delta_{!B \multimap B}) \\
 B & \xleftarrow{eval} & (B \multimap B) \otimes B
 \end{array}$$

Lemma 3.1.1. Every numeral morphism $n : N \rightarrow N$ is both a co-algebra and a co-monoid morphism.

Proof. By induction on the definition of n . If $n = 0$ then it is obvious by definition. If $n = succ \circ m$, then it follows by inductive hypothesis and from the fact that $succ$ is a coalgebra and co-monoid morphism and that these properties are closed under composition. \square

Given the above monoidal functor $(!, q) : \mathbb{L} \rightarrow \mathbb{L}$ we define $q_{A_1, \dots, A_n} : !A_1 \otimes \dots \otimes !A_n \rightarrow !(A_1 \otimes \dots \otimes A_n)$ (by induction on $n > 2$) to be the morphism $q_{A_1, A_2 \otimes \dots \otimes A_n} \circ (id_{A_1} \otimes q_{A_2, \dots, A_n})$. Moreover, we remind that $\varrho_A : A \otimes \mathbf{1} \rightarrow A$ is the right identity law of \otimes , while $\gamma_{A,B} : A \otimes B \rightarrow B \otimes A$ is the symmetric law of \otimes . Sometimes subscripts will be omitted when clear from the context.

Definition 3.1.10 (Categorical $\mathcal{S}\ell\lambda$ -model). A categorical $\mathcal{S}\ell\lambda$ -model consists of

- A $\mathcal{S}\ell\lambda$ Category $\langle \mathcal{L}, N, p, c_N, w_N, lif, fix \rangle$, where $\mathcal{L} = \langle \mathbb{L}, !, \delta, \varepsilon, q, e, d \rangle$.
- A mapping associating to every $\mathcal{S}\ell\lambda$ -type σ , an object $\llbracket \sigma \rrbracket$ of \mathbb{L} such that $\llbracket \iota \rrbracket = N$ and $\llbracket \sigma \multimap \tau \rrbracket = \llbracket \sigma \rrbracket \multimap \llbracket \tau \rrbracket$.
- Given a basis Γ we define $\llbracket \Gamma \rrbracket$ by induction as $\llbracket \emptyset \rrbracket = \mathbf{1}$, $\llbracket \mathbf{x}^\sigma, \Delta \rrbracket = \llbracket \sigma \rrbracket \otimes \llbracket \Delta \rrbracket$ and $\llbracket F^\sigma, \Delta \rrbracket = \llbracket \sigma \rrbracket \otimes \llbracket \Delta \rrbracket$. Moreover, given a basis Γ such that $\Gamma^\iota = \mathbf{x}_1^\iota, \dots, \mathbf{x}_n^\iota$ (resp. $\Gamma^* = F_1^{\sigma_1}, \dots, F_n^{\sigma_n}$) we denote with $p_\Gamma = \underbrace{p \otimes \dots \otimes p}_n$ (resp. $\delta_\Gamma = \delta_{\llbracket \sigma_1 \rrbracket} \otimes \dots \otimes \delta_{\llbracket \sigma_n \rrbracket}$).

Given a term M such that $\Gamma \vdash M : \sigma$ we associate a morphism $\llbracket \Gamma \vdash M : \sigma \rrbracket : \llbracket \Gamma \rrbracket \rightarrow \llbracket \sigma \rrbracket$, such that

1. $\llbracket \vdash \emptyset : \iota \rrbracket = 0$
2. $\llbracket \vdash succ : \iota \multimap \iota \rrbracket = \text{curry}(succ)$
3. $\llbracket \vdash pred : \iota \multimap \iota \rrbracket = \text{curry}(pred)$
4. $\llbracket \mathbf{x}^\iota \vdash \mathbf{x} : \iota \rrbracket = id_N$
5. $\llbracket f^{\sigma \multimap \tau} \vdash f : \sigma \multimap \tau \rrbracket = id_{\llbracket \sigma \multimap \tau \rrbracket}$
6. $\llbracket F^\sigma \vdash F : \sigma \rrbracket = \varepsilon_{\llbracket \sigma \rrbracket}$
7. $\llbracket \Gamma^\iota, \Delta^* \vdash \mu F.M : \sigma \rrbracket = fix_{\llbracket \sigma \rrbracket} \circ !\text{curry}(\llbracket \Gamma^\iota, \Delta^*, F^\sigma \vdash M : \sigma \rrbracket) \circ q \circ (p_\Gamma \otimes \delta_\Delta)$
8. $\llbracket \Gamma \vdash \lambda \mathbf{x}^\sigma.M : \sigma \multimap \tau \rrbracket = \text{curry}(\llbracket \Gamma, \mathbf{x}^\sigma \vdash M : \tau \rrbracket)$

9. $\llbracket \Gamma, \kappa_1^{\sigma_1}, \kappa_2^{\sigma_2}, \Delta \vdash M : \tau \rrbracket = \llbracket \Gamma, \kappa_2^{\sigma_2}, \kappa_1^{\sigma_1}, \Delta \vdash M : \tau \rrbracket \circ (id_{\llbracket \Gamma \rrbracket} \otimes \gamma_{\llbracket \sigma_1 \rrbracket, \llbracket \sigma_2 \rrbracket}} \otimes id_{\llbracket \Delta \rrbracket})$
10. $\llbracket \Gamma, \Delta \vdash MN : \tau \rrbracket = eval \circ (\llbracket \Gamma \vdash M : \sigma \multimap \tau \rrbracket \otimes \llbracket \Delta \vdash N : \sigma \rrbracket)$.
11. $\llbracket \Gamma, \Delta \vdash \ell if M L R : \iota \rrbracket = \ell if \circ (\llbracket \Gamma \vdash M : \iota \rrbracket \otimes \langle \llbracket \Delta \vdash L : \iota \rrbracket, \llbracket \Delta \vdash R : \iota \rrbracket \rangle)$.
12. $\llbracket \Gamma, x^t \vdash M[x/x_1, x_2] : \tau \rrbracket = \llbracket \Gamma, x_1^t, x_2^t \vdash M : \tau \rrbracket \circ id_{\llbracket \Gamma \rrbracket} \otimes c_N$
13. $\llbracket \Gamma, F^\sigma \vdash M[F/F_1, F_2] : \tau \rrbracket = \llbracket \Gamma, F_1^\sigma, F_2^\sigma \vdash M : \tau \rrbracket \circ id_{\llbracket \Gamma \rrbracket} \otimes d_{\llbracket \sigma \rrbracket}$
14. $\llbracket \Gamma, x^t \vdash M : \tau \rrbracket = \llbracket \Gamma \vdash M : \tau \rrbracket \circ \rho \circ id_{\llbracket \Gamma \rrbracket} \otimes w_N$
15. $\llbracket \Gamma, F^\sigma \vdash M : \tau \rrbracket = \llbracket \Gamma \vdash M : \tau \rrbracket \circ \rho \circ id_{\llbracket \Gamma \rrbracket} \otimes e_{\llbracket \sigma \rrbracket}$

The following lemmas are the standard semantic Substitution Lemma. The key point to prove these lemmas is to show that the transformation induced by the typing rules is natural on the unchanged components of the sequent. Let us observe that the substitution of a ground or high-order variable respectively with a numeral or a term is modelled directly with the composition in \mathbb{L} (see Lemma 3.1.2 and Lemma 3.1.3), while the substitution of a stable variable with a term is modelled with the composition in the category of !-coalgebras (see Lemma 3.1.4).

For sake of simplicity, in the proofs we will relax a bit the definition of types and basis. First of all, in the syntax of types, we allow types prefixed with a !. Thus, given a basis $\Gamma = \kappa_1^{\sigma_1}, \dots, \kappa_n^{\sigma_n}$, we denote with $!\Gamma = \kappa_1^{!\sigma_1}, \dots, \kappa_n^{!\sigma_n}$. Observe that, for us ! is just a syntactical annotation which will be interpreted with the corresponding exponential co-monad; for this reason, we will adapt in the canonical way the interpretation function on the so obtained types and basis. For example, if $\langle \mathbb{L}, !, \delta, \varepsilon, q, e, d, N, p, c_N, w_N, \ell if, fix \rangle$ is an $\mathcal{S}\ell\lambda$ Category and given a basis Γ^* (resp. Γ'), we have $\delta_\Gamma : \llbracket \Gamma \rrbracket \rightarrow \llbracket !\Gamma \rrbracket$ (resp. $p_\Gamma : \llbracket \Gamma \rrbracket \rightarrow \llbracket !\Gamma \rrbracket$).

Lemma 3.1.2. *Let M be such that $\Gamma, x^t, \Delta \vdash M : \sigma$. Then $\llbracket \Gamma, \Delta \vdash M[\underline{n}/x] : \sigma \rrbracket = \llbracket \Gamma, x^t, \Delta \vdash M : \sigma \rrbracket \circ (id_{\llbracket \Gamma \rrbracket} \otimes n \otimes id_{\llbracket \Delta \rrbracket})$.*

Proof. The proof is by induction on the derivation of $\Gamma, x^t \vdash M : \sigma$ and by cases on the last applied rule.

- case **(ex)**. Straightforward.
- case **(gv)**. Obvious, since $\llbracket x[\underline{n}/x] \rrbracket = id_N \circ n = n$ as expected.
- case **(λ)**. Then $\Gamma, x^t \vdash \lambda f.M : \sigma \multimap \tau$ is direct consequence of $\Gamma, f^\sigma \vdash M : \tau$. Thus, we have

$$\begin{aligned}
\llbracket \Gamma, x^t \vdash \lambda f.M[\underline{n}/x] : \sigma \multimap \tau \rrbracket &= \text{curry}(\llbracket \Gamma, f^\sigma \vdash M[\underline{n}/x] : \tau \rrbracket) \\
&= \text{curry}(\llbracket \Gamma, x^t, f^\sigma \vdash M : \tau \rrbracket \circ (id_{\llbracket \Gamma \rrbracket} \otimes n \otimes id_\sigma)) \\
&= \text{curry}(\llbracket \Gamma, x^t, f^\sigma \vdash M : \tau \rrbracket) \circ (id_{\llbracket \Gamma \rrbracket} \otimes n) \\
&= \llbracket \Gamma, x^t \vdash \lambda f.M : \sigma \multimap \tau \rrbracket \circ (id_{\llbracket \Gamma \rrbracket} \otimes n)
\end{aligned}$$

where the first row follows by interpretation, the second row follows by induction, the third row follows by naturality of $\text{curry}(-)$ and the fourth row follows again by interpretation.

- case **(ap)**. This case follows by induction and by functoriality of \otimes .
- case **(gc)**. The only interesting case is $\Gamma, \mathbf{x}^t \vdash \mathbb{M}[\mathbf{x}/\mathbf{x}_1, \mathbf{x}_2] : \sigma$ consequence of $\Gamma, \mathbf{x}_1^t, \mathbf{x}_2^t \vdash \mathbb{M} : \sigma$. Thus we have

$$\begin{aligned}
\llbracket \Gamma \vdash \mathbb{M}[\mathbf{x}/\mathbf{x}_1, \mathbf{x}_2][\underline{n}/\mathbf{x}] : \sigma \rrbracket &= \llbracket \Gamma \vdash \mathbb{M}[\underline{n}/\mathbf{x}_1, \underline{n}/\mathbf{x}_2] : \sigma \rrbracket \\
&= \llbracket \Gamma, \mathbf{x}_1^t, \mathbf{x}_2^t \vdash \mathbb{M} : \sigma \rrbracket \circ id_{\llbracket \Gamma \rrbracket} \otimes n \otimes n \\
&= \llbracket \Gamma, \mathbf{x}_1^t, \mathbf{x}_2^t \vdash \mathbb{M} : \sigma \rrbracket \circ (id_{\llbracket \Gamma \rrbracket} \otimes c_N) \circ (id_{\llbracket \Gamma \rrbracket} \otimes n) \\
&= \llbracket \Gamma, \mathbf{x}^t \vdash \mathbb{M} : \sigma \rrbracket \circ id_{\llbracket \Gamma \rrbracket} \otimes n
\end{aligned}$$

where the first row follows by definition of substitution, the second row follows by induction, the third row follows by Lemma 3.1.1 and the fourth row follows by interpretation.

- case **(gw)**. The only interesting case is $\Gamma, \mathbf{x}^t \vdash \mathbb{M} : \sigma$ consequence of $\Gamma \vdash \mathbb{M} : \sigma$. Thus we have

$$\begin{aligned}
\llbracket \Gamma \vdash \mathbb{M}[\underline{n}/\mathbf{x}] : \sigma \rrbracket &= \llbracket \Gamma \vdash \mathbb{M} : \sigma \rrbracket \\
&= \llbracket \Gamma \vdash \mathbb{M} : \sigma \rrbracket \circ \varrho \circ id_{\llbracket \Gamma \rrbracket} \otimes id_1 \circ \varrho^{-1} \\
&= \llbracket \Gamma \vdash \mathbb{M} : \sigma \rrbracket \circ \varrho \circ (id_{\llbracket \Gamma \rrbracket} \otimes w_N) \circ (id_{\llbracket \Gamma \rrbracket} \otimes n) \circ \varrho^{-1} \\
&= \llbracket \Gamma, \mathbf{x}^t \vdash \mathbb{M} : \sigma \rrbracket \circ id_{\llbracket \Gamma \rrbracket} \otimes n \circ \varrho^{-1}
\end{aligned}$$

where the first row follows by definition of substitution, the second row follows since ϱ is a natural isomorphism, the third row follows by Lemma 3.1.1 and by observing that $\langle \mathbf{1}, \varrho^{-1}, id_1 \rangle$ is trivially a commutative co-monoid and the fourth row follows by interpretation.

- case **(lif)**. The only interesting case is $\Gamma, \Delta, \mathbf{x}^t \vdash \text{lif } \mathbb{M} \mathbb{L} \mathbb{R}$ direct consequence of $\Gamma \vdash \mathbb{M} : \iota$ and $\Delta, \mathbf{x}^t \vdash \mathbb{L} : \iota$ and $\Delta, \mathbf{x}^t \vdash \mathbb{R} : \iota$. Thus we have

$$\begin{aligned}
\llbracket \Gamma, \Delta \vdash \text{lif } \mathbb{M} \mathbb{L} \mathbb{R}[\underline{n}/\mathbf{x}] : \iota \rrbracket &= \llbracket \Gamma, \Delta \vdash \text{lif } \mathbb{M} \mathbb{L}[\underline{n}/\mathbf{x}] \mathbb{R}[\underline{n}/\mathbf{x}] : \iota \rrbracket \\
&= \text{lif} \circ \llbracket \Gamma \vdash \mathbb{M} : \iota \rrbracket \otimes \langle \llbracket \Delta, \mathbf{x}^t \vdash \mathbb{L} : \iota \rrbracket \circ id_{\llbracket \Delta \rrbracket} \otimes n, \llbracket \Delta, \mathbf{x}^t \vdash \mathbb{R} : \iota \rrbracket \circ id_{\llbracket \Delta \rrbracket} \otimes n \rangle \\
&= \text{lif} \circ \llbracket \Gamma \vdash \mathbb{M} : \iota \rrbracket \otimes \langle \llbracket \Delta, \mathbf{x}^t \vdash \mathbb{L} : \iota \rrbracket, \llbracket \Delta, \mathbf{x}^t \vdash \mathbb{R} : \iota \rrbracket \rangle \circ id_{\llbracket \Delta \rrbracket} \otimes n \\
&= \llbracket \Gamma, \Delta, \mathbf{x}^t \vdash \text{lif } \mathbb{M} \mathbb{L} \mathbb{R} : \iota \rrbracket \circ id_{\llbracket \Gamma \rrbracket} \otimes id_{\llbracket \Delta \rrbracket} \otimes n
\end{aligned}$$

where the first row follows by definition of substitution, the second row follows by induction, the third row follows by naturality of pairing and the fourth row follows by interpretation.

- case **(sc)** and **(sw)** are straightforward.
- case **(μ)**. Then $\Gamma^t, \mathbf{x}^t, !\Delta \vdash \mu F. \mathbb{M} : \sigma$ is direct consequence of $\Gamma^t, \mathbf{x}^t, !\Delta, F^\sigma \vdash \mathbb{M} : \sigma$. Let $f = \llbracket \Gamma^t, \mathbf{x}^t, !\Delta, F^\sigma \vdash \mathbb{M} : \sigma \rrbracket$. Then this case follows from the commutativity of the following diagram.

$$\begin{array}{ccccc}
\llbracket \Gamma \rrbracket \otimes \mathbf{1} \otimes \llbracket \Delta \rrbracket & \xrightarrow{p_\Gamma \otimes q_1 \otimes \delta_\Delta} & \llbracket !\Gamma \rrbracket \otimes !\mathbf{1} \otimes \llbracket !\Delta \rrbracket & \xrightarrow{q} & !(\llbracket \Gamma \rrbracket \otimes \mathbf{1} \otimes \llbracket \Delta \rrbracket) \\
id_{\llbracket \Gamma \rrbracket} \otimes n \otimes id_{\llbracket \Delta \rrbracket} \downarrow & & id_{\llbracket !\Gamma \rrbracket} \otimes !n \otimes id_{\llbracket !\Delta \rrbracket} \downarrow & & !(id_{\llbracket \Gamma \rrbracket} \otimes n \otimes id_{\llbracket \Delta \rrbracket}) \downarrow \\
\llbracket \Gamma \rrbracket \otimes N \otimes \llbracket \Delta \rrbracket & \xrightarrow{p_\Gamma \otimes p \otimes \delta_\Delta} & \llbracket !\Gamma \rrbracket \otimes !N \otimes \llbracket !\Delta \rrbracket & \xrightarrow{q} & !(\llbracket \Gamma \rrbracket \otimes N \otimes \llbracket \Delta \rrbracket) \xrightarrow{! \text{curry}(f)} !(\llbracket \sigma \rrbracket \multimap \llbracket \sigma \rrbracket) \\
& & & & \swarrow \text{!curry}(f \circ id_{\llbracket \Gamma \rrbracket} \otimes n \otimes id_{\llbracket \Delta \rrbracket})
\end{array}$$

where the square on the left commutes since Lemma 3.1.1, the central square commutes since $\langle !, q \rangle$ is a monoidal functor and the triangle on the right commutes since the category is monoidal closed and by functoriality. Thus, we have

$$\begin{aligned}
\llbracket \Gamma', \Delta^* \vdash \mu F.M[\underline{n}/\underline{x}] : \sigma \rrbracket &= \text{fix}_{\llbracket \sigma \rrbracket} \circ !\text{curry}(\llbracket \Gamma', \Delta^*, F^\sigma \vdash M[\underline{n}/\underline{x}] : \sigma \rrbracket) \circ q \circ (p_\Gamma \otimes \delta_\Delta) \\
&= \text{fix}_{\llbracket \sigma \rrbracket} \circ !\text{curry}(f \circ \text{id}_{\llbracket \Gamma \rrbracket} \otimes n \otimes \text{id}_{\llbracket \Delta \rrbracket}) \circ q \circ (p_\Gamma \otimes \delta_\Delta) \\
&= \text{fix}_{\llbracket \sigma \rrbracket} \circ !\text{curry}(f) \circ q \circ (p_\Gamma \otimes p \otimes \delta_\Delta) \circ (\text{id}_{\llbracket \Gamma \rrbracket} \otimes n \otimes \text{id}_{\llbracket \Delta \rrbracket}) \\
&= \llbracket \Gamma', \mathbf{x}', \Delta^* \vdash \mu F.M : \sigma \rrbracket \circ (\text{id}_{\llbracket \Gamma \rrbracket} \otimes n \otimes \text{id}_{\llbracket \Delta \rrbracket})
\end{aligned}$$

where the first row follows by interpretation, the second row follows by induction, the third row follows by commutativity of the above diagram and the fourth row follows again by interpretation. \square

Lemma 3.1.3. *Let M, N be such that $\Gamma, F^\sigma \vdash M : \tau$ and $\Delta \vdash N : \sigma$, with $\Gamma \cap \Delta = \emptyset$. Then $\llbracket \Gamma, \Delta \vdash M[N/f] : \tau \rrbracket = \llbracket \Gamma, F^\sigma \vdash M : \tau \rrbracket \circ \text{id}_{\llbracket \Gamma \rrbracket} \otimes \llbracket \Delta \vdash N : \sigma \rrbracket$*

Proof. The proof is by induction on the derivation of $\Gamma, F^\sigma \vdash M : \tau$. All cases are straightforward, just by observing that the operation on morphisms induced by the interpretation are natural in the interpretation of the unchanged components of the sequent. \square

Lemma 3.1.4. *Let M, N be such that $\Gamma, F^\sigma \vdash M : \tau$ and $\Delta_1^l, \Delta_2^* \vdash N : \sigma$, with $\Gamma \cap \Delta_1 \cap \Delta_2 = \emptyset$. Then $\llbracket \Gamma, \Delta_1^l, \Delta_2^* \vdash M[N/F] : \tau \rrbracket = \llbracket \Gamma, F^\sigma \vdash M : \tau \rrbracket \circ (\text{id}_{\llbracket \Gamma \rrbracket} \otimes (!\llbracket \Delta_1^l, \Delta_2^* \vdash N : \sigma \rrbracket \circ q \circ (p_{\Delta_1} \otimes \delta_{\Delta_2})))$*

Proof. The proof is by induction on the derivation of $\Gamma, F^\sigma \vdash M : \tau$. All cases are straightforward except the following.

- case (sc). The only interesting case is $\Gamma, F^\sigma \vdash M[F/F_1, F_2] : \tau$ direct consequence of $\Gamma, F_1^\sigma, F_2^\sigma \vdash M : \tau$. If we let $f = \llbracket \Delta_1^l, \Delta_2^* \vdash N : \sigma \rrbracket$, then this case follows by the commutativity of the following diagram (to light the notation, we omitted some subscripts).

$$\begin{array}{ccc}
\llbracket \Delta_1, \Delta_2 \rrbracket & \xrightarrow{\overbrace{(c_N \otimes \dots \otimes c_N) \otimes (d \otimes \dots \otimes d)}^{n \quad m}} & \llbracket \Delta_1, \Delta_1, \Delta_2, \Delta_2 \rrbracket \cong \llbracket \Delta_1, \Delta_2, \Delta_1, \Delta_2 \rrbracket \\
\downarrow p_{\Delta_1} \otimes \delta_{\Delta_2} & & \downarrow p_{\Delta_1} \otimes p_{\Delta_1} \otimes \delta_{\Delta_2} \otimes \delta_{\Delta_2} \\
\llbracket !\Delta_1, !\Delta_2 \rrbracket & \xrightarrow{\overbrace{d \otimes \dots \otimes d}^{n+m}} & \llbracket !\Delta_1, !\Delta_1, !\Delta_2, !\Delta_2 \rrbracket \cong \llbracket !\Delta_1, !\Delta_2, !\Delta_1, !\Delta_2 \rrbracket \\
\downarrow q & & \downarrow q \otimes q \\
!\llbracket \Delta_1, \Delta_2 \rrbracket & \xrightarrow{d_{\llbracket \Delta_1, \Delta_2 \rrbracket}} & !\llbracket \Delta_1, \Delta_2 \rrbracket \otimes !\llbracket \Delta_1, \Delta_2 \rrbracket \\
\downarrow !f & & \downarrow !f \otimes !f \\
!\llbracket \sigma \rrbracket & \xrightarrow{d_{\llbracket \sigma \rrbracket}} & !\llbracket \sigma \rrbracket \otimes !\llbracket \sigma \rrbracket
\end{array}$$

where the first square on the top commutes since p and δ are co-monoid morphisms and since the involved co-monoid are commutative, the central square commutes since d is a monoidal natural transformation and the square on the bottom commutes since being $!f$ a co-algebra morphism between free co-algebra, it is also a co-monoid morphism. Thus we have

$$\begin{aligned}
& \llbracket \Gamma, \Delta_1, \Delta_2^* \vdash \mathbb{M}[F/F_1, F_2][N/F] : \tau \rrbracket \\
&= \llbracket \Gamma, \Delta_1, \Delta_2, \Delta_1, \Delta_2 \vdash \mathbb{M}[N/F_1][N/F_2] : \tau \rrbracket \circ \gamma \circ (id_{\llbracket \Gamma \rrbracket} \otimes (c_N \otimes \cdots \otimes c_N) \otimes (d \otimes \cdots \otimes d)) \\
&= \llbracket \Gamma, F_1^\sigma, F_2^\sigma \vdash \mathbb{M} : \tau \rrbracket \circ (id_{\llbracket \Gamma \rrbracket} \otimes (!f \otimes !f) \circ (q \otimes q) \circ (p_{\Delta_1} \otimes \delta_{\Delta_2} \otimes p_{\Delta_1} \otimes \delta_{\Delta_2})) \circ \gamma \circ \\
&\quad ((c_N \otimes \cdots \otimes c_N) \otimes (d \otimes \cdots \otimes d)) \\
&= \llbracket \Gamma, F_1^\sigma, F_2^\sigma \vdash \mathbb{M} : \tau \rrbracket \circ id_{\llbracket \Gamma \rrbracket} \otimes (d \circ !f \circ q \circ p_{\Delta_1} \otimes \delta_{\Delta_2}) \\
&= \llbracket \Gamma, F^\sigma \vdash \mathbb{M}[F/F_1, F_2] \rrbracket \circ id_{\llbracket \Gamma \rrbracket} \otimes (!f \circ q \circ p_{\Delta_1} \otimes \delta_{\Delta_2})
\end{aligned}$$

where the first row follows by interpretation, the second row follows by induction, the third row follows by commutativity of the above square and the fourth row follows by interpretation.

- case (sw). The only interesting case is $\Gamma, F^\sigma \vdash \mathbb{M} : \tau$ direct consequence of $\Gamma \vdash \mathbb{M} : \sigma$. If we let $f = \llbracket \Delta_1^t, \Delta_2^* \vdash \mathbb{N} : \sigma \rrbracket$ then this case follows by commutativity of the following diagram.

$$\begin{array}{ccccccc}
\llbracket \Gamma, \Delta_1, \Delta_2 \rrbracket & \xrightarrow{id_{\llbracket \Gamma \rrbracket} \otimes p_{\Delta_1} \otimes \delta_{\Delta_2}} & \llbracket \Gamma, !\Delta_1, !\Delta_2 \rrbracket & \xrightarrow{id_{\llbracket \Gamma \rrbracket} \otimes q} & \llbracket \Gamma \rrbracket \otimes !\llbracket \Delta_1, \Delta_2 \rrbracket & \xrightarrow{id_{\llbracket \Gamma \rrbracket} \otimes !f} & \llbracket \Gamma \rrbracket \otimes !\llbracket \sigma \rrbracket \\
\downarrow id_{\llbracket \Gamma \rrbracket} \otimes (w_N \otimes \cdots \otimes w_N) \otimes (e \otimes \cdots \otimes e) & & \downarrow id_{\llbracket \Gamma \rrbracket} \otimes (e \otimes \cdots \otimes e) & & \downarrow id_{\llbracket \Gamma \rrbracket} \otimes e_{\llbracket \Delta_1, \Delta_2 \rrbracket} & & \downarrow id_{\llbracket \Gamma \rrbracket} \otimes e_{\llbracket \sigma \rrbracket} \\
\llbracket \Gamma \rrbracket \otimes \mathbf{1} \cong \llbracket \Gamma \rrbracket & \cong & \llbracket \Gamma \rrbracket \otimes \mathbf{1} \cong \llbracket \Gamma \rrbracket & \cong & \llbracket \Gamma \rrbracket \otimes \mathbf{1} \cong \llbracket \Gamma \rrbracket & \cong & \llbracket \Gamma \rrbracket \otimes \mathbf{1} \cong \llbracket \Gamma \rrbracket
\end{array}$$

where the first square on the left commutes since p and δ are co-monoid morphisms, the central square commutes since e is a monoidal natural transformation and the last square on the right commutes since, being $!f$ a co-algebra morphism between free co-algebras, it is also a co-monoid morphism. Thus we have

$$\begin{aligned}
\llbracket \Gamma, \Delta_1^t, \Delta_2^* \vdash \mathbb{M}[N/F] : \tau \rrbracket &= \llbracket \Gamma, \Delta_1^t, \Delta_2^* \vdash \mathbb{M} : \tau \rrbracket \\
&= \llbracket \Gamma \vdash \mathbb{M} : \tau \rrbracket \circ \rho \circ ((w_N \otimes \cdots \otimes w_N) \otimes (e \otimes \cdots \otimes e) \otimes id_{\llbracket \Gamma \rrbracket}) \\
&= \llbracket \Gamma \vdash \mathbb{M} : \tau \rrbracket \circ \rho \circ (id_{\llbracket \Gamma \rrbracket} \otimes (e_{\llbracket \sigma \rrbracket} \circ !f \circ q \circ (p_{\Delta_1} \otimes \delta_{\Delta_2}))) \\
&= \llbracket \Gamma, F^\sigma \vdash \mathbb{M} : \tau \rrbracket \circ (id_{\llbracket \Gamma \rrbracket} \otimes (!f \circ q \circ (p_{\Delta_1} \otimes \delta_{\Delta_2})))
\end{aligned}$$

where the first row follows by definition of substitution, the second row follows by interpretation, the third row follows by the commutativity of the above square and the fourth row follows by interpretation.

- case (μ). Then $\Gamma^t, \Delta^*, F^\sigma \vdash \mu F_1. \mathbb{M} : \tau$ is direct consequence of $\Gamma^t, \Delta^*, F^\sigma, F_1^\tau \vdash \mathbb{M} : \tau$. If we let $f = \llbracket \Delta_1^t, \Delta_2^* \vdash \mathbb{N} : \sigma \rrbracket$ and $g = \llbracket \Gamma^t, \Delta^*, F^\sigma, F_1^\tau \vdash \mathbb{M} : \tau \rrbracket$, then this case follows by commutativity of the diagram in Figure 3.1.2. For its commutativity, we use the fact that p and δ are co-algebras, δ and q are monoidal natural transformation and $!f$ is a morphism between free co-algebras.

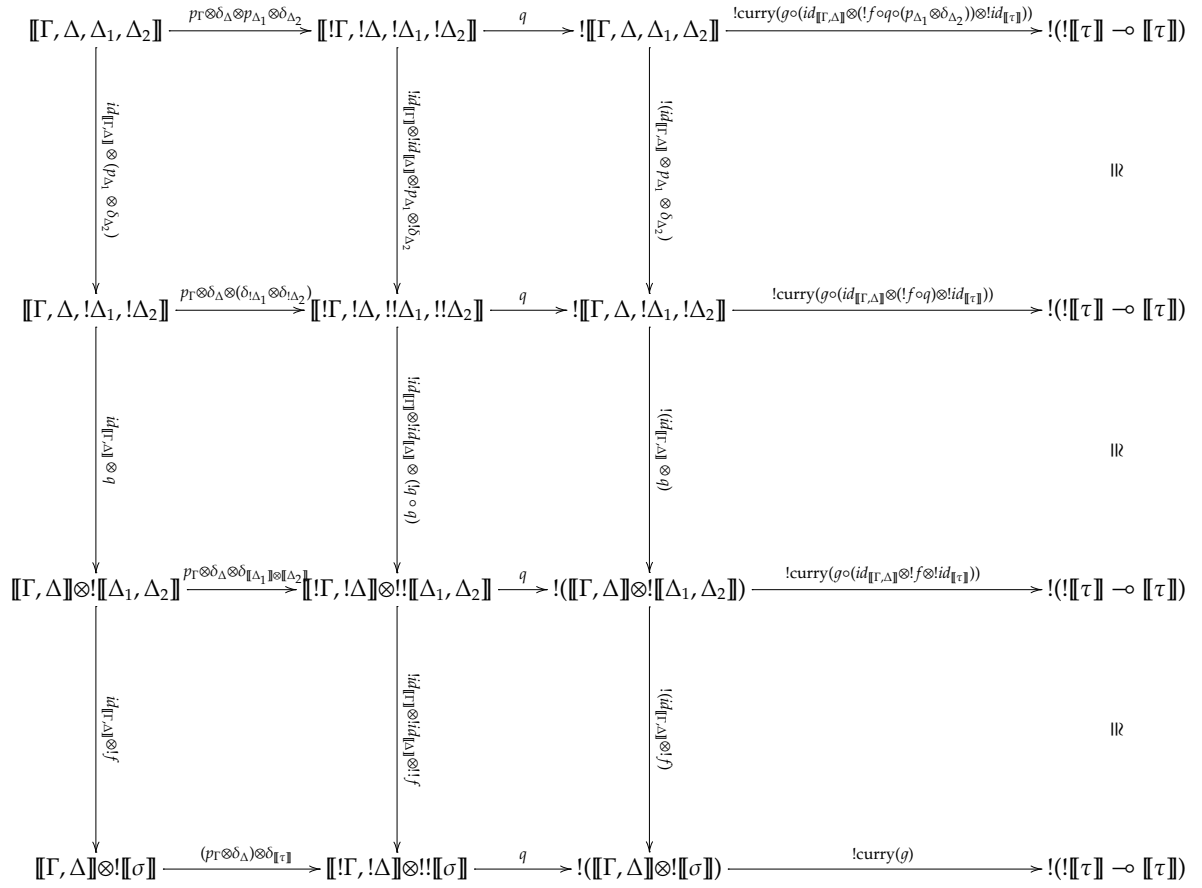


Figure 3.1: Diagram for the proof of Lemma 3.1.4

Hence, we can conclude the following

$$\begin{aligned}
\llbracket \Gamma^l, \Delta^*, \Delta_1^l, \Delta_2^* \vdash \mu F_1.M[N/F] : \tau \rrbracket &= \text{fix}_{\llbracket \tau \rrbracket} \circ !\text{curry}(\llbracket \Gamma, \Delta, \Delta_1, \Delta_2, F_1^\tau \vdash M[N/F] : \tau \rrbracket) \circ q \circ \\
&\quad (p_\Gamma \otimes \delta_\Delta \otimes p_{\Delta_1} \otimes \delta_{\Delta_2}) \\
&= \text{fix}_{\llbracket \tau \rrbracket} \circ !\text{curry}(g \circ (id_{\llbracket \Gamma, \Delta \rrbracket} \otimes (!f \circ q \circ (p_{\Delta_1} \otimes \delta_{\Delta_2}))) \otimes !id_{\llbracket \tau \rrbracket})) \circ q \circ (p_\Gamma \otimes \delta_\Delta \otimes p_{\Delta_1} \otimes \delta_{\Delta_2}) \\
&= \text{fix}_{\llbracket \tau \rrbracket} \circ !\text{curry}(g) \circ q \circ (p_\Gamma \otimes \delta_\Delta \otimes \delta_{\llbracket \tau \rrbracket}) \circ (id_{\llbracket \Gamma, \Delta \rrbracket} \otimes (!f \circ q \circ (p_{\Delta_1} \otimes \delta_{\Delta_2}))) \\
&= \llbracket \Gamma, \Delta, F^\sigma \vdash \mu F_1.M : \tau \rrbracket \circ (id_{\llbracket \Gamma, \Delta \rrbracket} \otimes (!f \circ q \circ (p_{\Delta_1} \otimes \delta_{\Delta_2})))
\end{aligned}$$

where the first row follows by interpretation, the second row follows by induction, the third row follows by the commutativity of the diagram in Figure 3.1.2 and the fourth row follows again by interpretation. \square

Theorem 3.1.1 (Soundness). *Let M, N such that $\Gamma \vdash M, N : \sigma$.*

If $M =_{\mathcal{S}} N$ then $\llbracket \Gamma \vdash M : \sigma \rrbracket = \llbracket \Gamma \vdash N : \sigma \rrbracket$

Proof. The proof is by induction on the derivation of $M =_{\mathcal{S}} N$. We develop a few cases.

- case $M = (\lambda f^\tau.M_1)M_2$ and $N = M_1[M_2/f]$. Then we have

$$\begin{aligned}
\llbracket \Gamma, \Delta \vdash M : \sigma \rrbracket &= \text{eval} \circ (\text{curry}(\llbracket \Gamma, f^\tau \vdash M_1 : \sigma \rrbracket) \otimes \llbracket \Delta \vdash M_2 : \tau \rrbracket) \\
&= \llbracket \Gamma, f^\tau \vdash M_1 : \sigma \rrbracket \circ (id_{\llbracket \Gamma \rrbracket} \otimes \llbracket \Delta \vdash M_2 : \tau \rrbracket) \\
&= \llbracket \Gamma, \Delta \vdash M_1[M_2/f] : \sigma \rrbracket \\
&= \llbracket \Gamma, \Delta \vdash N : \sigma \rrbracket
\end{aligned}$$

where in the third line we used Lemma 3.1.3

- case $M = (\lambda x^t.M_1)\underline{n}$ and $N = M_1[\underline{n}/x^t]$. Then we have

$$\begin{aligned}
\llbracket \Gamma \vdash M : \sigma \rrbracket &= \text{eval} \circ (\text{curry}(\llbracket \Gamma, x^t \vdash M_1 : \sigma \rrbracket) \otimes n) \\
&= \llbracket \Gamma, x^t \vdash M_1 : \sigma \rrbracket \circ (id_{\llbracket \Gamma \rrbracket} \otimes n) \\
&= \llbracket \Gamma \vdash M_1[\underline{n}/x^t] : \sigma \rrbracket \\
&= \llbracket \Gamma \vdash N : \sigma \rrbracket
\end{aligned}$$

where in the third line we used Lemma 3.1.2.

- case $M = \mu F.M_1$ and $N = M_1[\mu F.M_1/F]$. First of all, if we let $f = \llbracket \Gamma^l, \Delta^*, F^\sigma \vdash M_1 : \sigma \rrbracket$, let us observe that the following diagram commutes

$$\begin{array}{ccccc}
\llbracket \Gamma, \Delta \rrbracket & \xrightarrow{(c_N \otimes \dots \otimes c_N) \otimes (d \otimes \dots \otimes d)} & \llbracket \Gamma, \Gamma, \Delta, \Delta \rrbracket \cong \llbracket \Gamma, \Delta, \Gamma, \Delta \rrbracket & \xrightarrow{id_{\llbracket \Gamma, \Delta \rrbracket} \otimes p_\Gamma \otimes \delta_\Delta} & \llbracket \Gamma, \Delta \rrbracket \otimes \llbracket !\Gamma, !\Delta \rrbracket \\
\downarrow p_\Gamma \otimes \delta_\Delta & & \downarrow p_\Gamma \otimes p_\Gamma \otimes \delta_\Delta \otimes \delta_\Delta & & \downarrow id_{\llbracket \Gamma, \Delta \rrbracket} \otimes p_\Gamma \otimes \delta_\Delta \\
\llbracket !\Gamma, !\Delta \rrbracket & \xrightarrow{d \otimes \dots \otimes d} & \llbracket !\Gamma, !\Gamma, !\Delta, !\Delta \rrbracket \cong \llbracket !\Gamma, !\Delta, !\Gamma, !\Delta \rrbracket & \xrightarrow{\varepsilon_{\Gamma, \Delta} \otimes \delta_{\Gamma, !\Delta}} & \llbracket \Gamma, \Delta \rrbracket \otimes \llbracket !!\Gamma, !!\Delta \rrbracket \\
\downarrow q & & \downarrow q \otimes q & & \downarrow id_{\llbracket \Gamma, \Delta \rrbracket} \otimes (!q \circ q) \\
!\llbracket \Gamma, \Delta \rrbracket & \xrightarrow{d} & !\llbracket \Gamma, \Delta \rrbracket \otimes !\llbracket \Gamma, \Delta \rrbracket & \xrightarrow{\varepsilon_{\llbracket \Gamma, \Delta \rrbracket} \otimes \delta_{\llbracket \Gamma, \Delta \rrbracket}} & \llbracket \Gamma, \Delta \rrbracket \otimes !\llbracket \Gamma, \Delta \rrbracket \\
\downarrow !\text{curry}(f) & & \downarrow !\text{curry}(f) \otimes !\text{curry}(f) & & \downarrow \text{curry}(f) \otimes !\text{curry}(f) \\
!(\llbracket \sigma \rrbracket \multimap \llbracket \sigma \rrbracket) & \xrightarrow{d_{\llbracket \sigma \rrbracket \multimap \llbracket \sigma \rrbracket}} & !(\llbracket \sigma \rrbracket \multimap \llbracket \sigma \rrbracket) \otimes !(\llbracket \sigma \rrbracket \multimap \llbracket \sigma \rrbracket) & \xrightarrow{\varepsilon_{\llbracket \sigma \rrbracket \multimap \llbracket \sigma \rrbracket} \otimes \delta_{\llbracket \sigma \rrbracket \multimap \llbracket \sigma \rrbracket}}} & (\llbracket \sigma \rrbracket \multimap \llbracket \sigma \rrbracket) \otimes !(\llbracket \sigma \rrbracket \multimap \llbracket \sigma \rrbracket)
\end{array}$$

where the left square on the top commutes since p and δ are co-monoid morphisms, the right square on the top commutes since p and δ are co-algebras

(observe that we used both commutative diagrams of the definition of co-algebra) and by bi-functoriality of \otimes , the left square on the middle commutes since d is a monoidal natural transformation, the right square on the middle commutes since δ and ε are monoidal natural transformations, and finally the two squares on the bottom commutes respectively because being $!curry(f)$ a co-algebra morphism between free co-algebra, it is also a co-monoid morphism, by naturality of ε and δ and by bi-functoriality of \otimes . Thus, we have,

$$\begin{aligned}
\llbracket \Gamma^l, \Delta^* \vdash \mathbf{M} : \sigma \rrbracket &= fix_{\llbracket \sigma \rrbracket} \circ !curry(f) \circ q \circ (p_\Gamma \otimes \delta_\Delta) \\
&= eval \circ (\varepsilon_{\llbracket \sigma \rrbracket \rightarrow \llbracket \sigma \rrbracket} \otimes (!fix_{\llbracket \sigma \rrbracket} \circ \delta_{\llbracket \sigma \rrbracket \rightarrow \llbracket \sigma \rrbracket})) \circ d_{\llbracket \sigma \rrbracket \rightarrow \llbracket \sigma \rrbracket} \circ !curry(f) \circ q \circ (p_\Gamma \otimes \delta_\Delta) \\
&= eval \circ (curry(f) \otimes (!fix_{\llbracket \sigma \rrbracket} \circ !curry(f) \circ (!q \circ q)) \circ (p_{!_\Gamma} \otimes \delta_{!_\Delta})) \circ (p_\Gamma \otimes \delta_\Delta) \\
&\quad \circ ((c_N \otimes \cdots \otimes c_N) \otimes (d \otimes \cdots \otimes d)) \\
&= f \circ id_{\llbracket \Gamma, \Delta \rrbracket} \otimes (!\llbracket \Gamma^l, \Delta^* \vdash \mu F.M : \sigma \rrbracket \circ q \circ (p_\Gamma \otimes \delta_\Delta)) \circ ((c_N \otimes \cdots \otimes c_N) \otimes (d \otimes \cdots \otimes d)) \\
&= \llbracket \Gamma^l, \Delta^*, \Gamma^l, \Delta^* \vdash \mathbf{N} : \sigma \rrbracket \circ ((c_N \otimes \cdots \otimes c_N) \otimes (d \otimes \cdots \otimes d))
\end{aligned}$$

where in the second line we used the fix-point law, in the third line we use the commutativity of the above diagram, in the fourth line we use the definition of interpretation and the naturality of q and since the category is monoidal closed and finally in the fifth line we use Lemma 3.1.4. Then we can conclude by interpretation. \square

About completeness, we conjecture that it holds too, but we leave this issue for future developments.

3.2 Semantically Linear PCF

In this section we will introduce the language $\mathcal{S}lPCF$, which is a syntactical restriction of the language PCF. The programming language $\mathcal{S}lPCF$ is obtained from $\mathcal{S}l\lambda$ -calculus, by imposing a particular evaluation strategy, on closed terms. Let us observe that rules in Definition 3.1.3 are very close to a reduction strategy, since they impose a call by name evaluation on higher-order parameters and a call by value evaluation on ground parameter.

However, the strategy we impose on $\mathcal{S}lPCF$ is the lazy reduction strategy, as evaluation strategy on terms; the lazy reduction strategy reduces at every step the leftmost redex which is not under a λ -abstraction. For instance, the term $(\lambda x^t.((\lambda z^t.z)\underline{5}))(\underline{pred}\underline{3})$ reduces to $(\lambda x^t.((\lambda z^t.z)\underline{5}))(\underline{2})$ according to this strategy.

Let us observe that this strategy is not linear in an operational sense, since it duplicates redexes. Consider again the term $(\lambda f^{t \rightarrow t}. \ell i f \ x \ f\underline{2} \ f\underline{3})(\lambda x^t.((\lambda z^t.z)\underline{5}))$: if we reduce the leftmost redex, we will duplicate the innermost one.

The following definition formalises better this evaluation strategy, in a big-step operational semantics style.

$\frac{}{\underline{0} \Downarrow \underline{0}} \text{ (0)}$	$\frac{M \Downarrow \underline{n}}{\text{succ } M \Downarrow \text{succ } \underline{n}} \text{ (s)}$	$\frac{M \Downarrow \underline{0} \quad L \Downarrow \underline{m}}{\text{lif } M \ L \ R \Downarrow \underline{m}} \text{ (ifl)}$	$\frac{M \Downarrow \text{succ}(\underline{n}) \quad R \Downarrow \underline{m}}{\text{lif } M \ L \ R \Downarrow \underline{m}} \text{ (ifr)}$	$\frac{M \Downarrow \text{succ } \underline{n}}{\text{pred } M \Downarrow \underline{n}} \text{ (p)}$
$\frac{M[N/x]P_1 \cdots P_i \Downarrow \underline{n}}{(\lambda x^{\sigma \rightarrow \tau}.M)NP_1 \cdots P_i \Downarrow \underline{n}} \text{ (\lambda}^{\rightarrow}\text{)}$	$\frac{M[\mu F.M/F]P_1 \cdots P_i \Downarrow \underline{n}}{(\mu F.M)P_1 \cdots P_i \Downarrow \underline{n}} \text{ (\mu)}$	$\frac{N \Downarrow \underline{m} \quad M[m/x]P_1 \cdots P_i \Downarrow \underline{n}}{(\lambda x^t.M)NP_1 \cdots P_i \Downarrow \underline{n}} \text{ (\lambda}^t\text{)}$		

Table 3.2: Operational Semantics of $\mathcal{S}\text{PCF}$.

Definition 3.2.1. The evaluation relation $\Downarrow \subseteq \mathcal{P} \times \mathcal{N}$ is the effective relation inductively defined by the rules of Table 3.2. If there exists a numeral \underline{n} such that $M \Downarrow \underline{n}$ then we say that M converges, and we write $M \Downarrow$, otherwise we say that it diverges, and we write $M \Uparrow$.

The language $\mathcal{S}\text{PCF}$ endowed with an operational semantics, has been introduced in [Gaborardi and Paolini, 2007]. This language was then extended with an additional operator, called *which?* in [Paolini and Piccolo, 2008] and a full abstraction result for the model Coherence Spaces and Linear Functions was proved. This operator will be discussed in Section 3.5.2

The relation \Downarrow implements a *call-by-value* parameter passing policy in the ground case and *call-by-name* parameter passing policy in the high-order case.

The operational semantics of the language is soundly given with respect to the underlying calculus, as shown by the following theorem; it formalise a lazy leftmost reduction strategy.

Lemma 3.2.1. Let M, N be such that $\Gamma \vdash M : \sigma$ and $\Gamma \vdash N : \sigma$, with $M \rightsquigarrow N$. Suppose that $C[M], C[N] \in \mathcal{P}$ for some C . Then if $C[N] \Downarrow \underline{n}$ then $C[M] \Downarrow \underline{n}$.

Proof. The proof is by induction on the shape of the context, by considered in each case the last applied rule of derivation of $C[N] \Downarrow \underline{n}$. \square

Theorem 3.2.1. $M \Downarrow \underline{n}$ if and only if $M \rightarrow_{\mathcal{S}}^* \underline{n}$, for all term M .

Proof. $M \Downarrow \underline{n}$ implies $M \rightarrow_{\mathcal{S}}^* \underline{n}$ straightforwardly. The other direction follows by induction on the number of reduction steps of $M \rightarrow_{\mathcal{S}}^* \underline{n}$. The case of zero steps is immediate. The inductive case follows by Lemma 3.2.1. \square

Syntax deserves some discussion. We remark that pred is a partial operator, namely $\text{pred } \underline{0}$ diverges. The evaluation of $\text{lif } M^l \ L^l \ R^l$ asks to evaluate exactly one sub-term between L^l and R^l . Since first-order strict stable functions are linear, we need to add a fix-point operator to our language. Unfortunately, the least fix-point of a linear function is always the bottom of the considered domain, because strictness. In order to overcome this problem we recall the fix-point theorem [Gunter, 1992].

Theorem 3.2.2 (Knaster-Tarsky). Let D be a cpo. If $f : D \rightarrow D$ is monotonic then it has a least fixed point $\text{fix}(f) \in D$.

We now justify the introduction of stable variables. Those variables are used in order to program continuous (w.r.t. stable order) non-strict functions from a linear

coherence space L to itself, so their fix-points will be elements of L . Syntactically, a stable variable will be used without linear constraints. We do not permit to λ -abstract those variables, since they will be used only in order to obtain fix-points. Turing-Completeness was proved for $\mathcal{S}\ell\text{PCF}$ in [Gaboardi and Paolini, 2007].

Lemma 3.2.2. *Let $\mathbb{M}^\tau, \mathbb{N}^\sigma \in \mathcal{S}\ell\text{PCF}$.*

1. *If $\text{HVar} \cap \text{FV}(\mathbb{M}^\tau) \cap \text{FV}(\mathbb{N}^\sigma) = \emptyset$ and $\mathbf{x}^\sigma \in \text{HFV}(\mathbb{M}^\tau)$ then $\mathbb{M}^\tau[\mathbb{N}^\sigma/\mathbf{x}^\sigma] \in \mathcal{S}\ell\text{PCF}$.*
2. *If $\text{FV}(\mathbb{N}^\sigma) \subseteq \text{SVar} \cup \text{Var}^t$ then $\mathbb{M}^\tau[\mathbb{N}^\sigma/F^\sigma] \in \mathcal{S}\ell\text{PCF}$.*
3. *If \mathbb{N} does not contain high-order free variables and $\mathcal{X}_1, \dots, \mathcal{X}_n \in \text{Var}$ then $\mathbb{M}[\mathbb{N}/\mathcal{X}_1, \dots, \mathbb{N}/\mathcal{X}_n] \in \mathcal{S}\ell\text{PCF}$.*

Proof. By structural induction on terms. Proof details can be found in [Gaboardi, 2007, Lemma 91,p.149]. \square

Lemma 3.2.2 implies that evaluation is well-defined.

Example. Let us see how to write a $\mathcal{S}\ell\text{PCF}$ -term $\mathbb{M}^{l \rightarrow o \rightarrow l}$ which calculates the sum of two natural number. The term \mathbb{M} is the following

$$\mu F^{l \rightarrow o \rightarrow l}. \lambda x^l. \lambda y^l. \ell\text{if } (x) (y) (\text{succ } F (\text{pred } x) y)$$

Let us prove that for all $\underline{n}, \underline{m}$, we have $\mathbb{M}\underline{n}\underline{m} \Downarrow \underline{n} + \underline{m}$, by induction on \underline{n} . For the case $\underline{n} = \underline{0}$, we can build the following typing derivation.

$$\frac{\frac{\frac{\underline{0} \Downarrow \underline{0}}{\underline{m} \Downarrow \underline{m}} \quad \ell\text{if } (\underline{0}) (\underline{m}) (\text{succ } \mathbb{M} (\text{pred } \underline{0}) \underline{m}) \Downarrow \underline{m}}{\underline{0} \Downarrow \underline{0}} \quad (\lambda y^l. \ell\text{if } (\underline{0}) (y) (\text{succ } \mathbb{M} (\text{pred } \underline{0}) y)) \underline{m} \Downarrow \underline{m}}{\underline{0} \Downarrow \underline{0}} \quad (\lambda x^l. \lambda y^l. \ell\text{if } (x) (y) (\text{succ } \mathbb{M} (\text{pred } x) y)) \underline{0} \underline{m} \Downarrow \underline{m}}{\underline{0} \underline{m} \Downarrow \underline{m}}$$

For the case $\underline{n} = \underline{k} + \underline{1}$, we can build the following typing derivation.

$$\frac{\frac{\frac{\frac{\frac{\vdots}{\mathbb{M} (\text{pred } \underline{k} + \underline{1}) \underline{m} \Downarrow \underline{k} + \underline{m}}{\underline{k} + \underline{1} \Downarrow \underline{k} + \underline{1}} \quad \text{succ } \mathbb{M} (\text{pred } \underline{k} + \underline{1}) \underline{m} \Downarrow \underline{k} + \underline{m} + \underline{1}}{\underline{m} \Downarrow \underline{m}} \quad \ell\text{if } (\underline{k} + \underline{1}) (\underline{m}) (\text{succ } \mathbb{M} (\text{pred } \underline{k} + \underline{1}) \underline{m}) \Downarrow \underline{k} + \underline{m} + \underline{1}}{\underline{k} + \underline{1} \Downarrow \underline{k} + \underline{1}} \quad (\lambda y^l. \ell\text{if } (\underline{k} + \underline{1}) (y) (\text{succ } \mathbb{M} (\text{pred } \underline{k} + \underline{1}) y)) \underline{m} \Downarrow \underline{k} + \underline{m} + \underline{1}}{\underline{k} + \underline{1} \Downarrow \underline{k} + \underline{1}} \quad (\lambda x^l. \lambda y^l. \ell\text{if } (x) (y) (\text{succ } \mathbb{M} (\text{pred } x) y)) \underline{k} + \underline{1} \underline{m} \Downarrow \underline{k} + \underline{m} + \underline{1}}{\underline{k} + \underline{1} \underline{m} \Downarrow \underline{k} + \underline{m} + \underline{1}}$$

where $\mathbb{M} (\text{pred } \underline{k} + \underline{1}) \underline{m} \Downarrow \underline{k} + \underline{m}$ follows easily by induction hypothesis.

Definition 3.2.2 (Operational Equivalence). Let $M^\sigma, N^\sigma \in \mathcal{SLPCF}$, such that $\text{SFV}(M), \text{SFV}(N) \subseteq \{F_1^{\sigma_1}, \dots, F_n^{\sigma_n}\}$

1. $M \lesssim_\sigma N$ whenever, for all closed term $P_1^{\sigma_1}, \dots, P_n^{\sigma_n}$, for all $C[\sigma]$ s.t. $C[M[P_1/F_1, \dots, P_n/F_n]], C[N[P_1/F_1, \dots, P_n/F_n]] \in \mathcal{P}$, if $C[M[P_1/F_1, \dots, P_n/F_n]] \Downarrow \underline{n}$ then $C[N[P_1/F_1, \dots, P_n/F_n]] \Downarrow \underline{n}$.
2. $M \approx_\sigma N$ if and only if $M \lesssim_\sigma N$ and $N \lesssim_\sigma M$.

Observe that the operational equivalence between two terms M, N is defined in terms of all substitutions of closed terms to all free occurrences of stable variables in M and N . This is done since stable variables cannot be λ -abstracted, and we did not find a way different from substitution to assign a whatever term to a stable variable (we will leave this issue for a future development).

It is easy to verify that \lesssim_σ is a preorder while \approx_σ is a congruence. For the sequel, it is convenient to define some abbreviations. We put

$$\Omega^\iota = \mu F^\iota . F^\iota$$

Moreover, if $\sigma_\emptyset = \mu_1 \multimap \dots \multimap \mu_m \multimap \iota$ for some $m \in \mathbb{N}$, then

$$\Omega^{\sigma_\emptyset \multimap \dots \multimap \sigma_n \multimap \iota} = \lambda x_\emptyset^{\sigma_\emptyset} \dots x_n^{\sigma_n} . \text{lift}(\Omega^{\sigma_1 \multimap \dots \multimap \sigma_n \multimap \iota} x_1^{\sigma_1} \dots x_n^{\sigma_n})(x_\emptyset \Omega^{\mu_1} \dots \Omega^{\mu_m})(x_\emptyset \Omega^{\mu_1} \dots \Omega^{\mu_m}).$$

By using Ω^σ it is possible to define approximants of a term having shape $\mu F . M^\sigma$ as follows,

$$\mu^0 F . M^\sigma = \Omega^\sigma, \quad \mu^{n+1} F . M^\sigma = M[\mu^n F . M/F].$$

The following lemma describes the operational behaviour of the above terms.

Lemma 3.2.3. Let $M_0^{\sigma_0}, \dots, M_m^{\sigma_m}$ be a sequence of closed terms ($m \geq 0$).

1. $\Omega^{\sigma_\emptyset \multimap \dots \multimap \sigma_m \multimap \iota} M_0 \dots M_m$ is a program and $\Omega^{\sigma_\emptyset \multimap \dots \multimap \sigma_m \multimap \iota} M_0 \dots M_m \Uparrow$.
2. Let $(\mu F . P^\sigma) M_0 \dots M_m$ be a program. $(\mu F . P^\sigma) M_0 \dots M_m \Downarrow \underline{n}$ if and only if $(\mu^{k+1} F . P^\sigma) M_0 \dots M_m \Downarrow \underline{n}$, for some $k \in \mathbb{N}$.

Proof. 1. The proof can be done by induction on m . The base case is easy, while all the other cases follows directly from inductive hypothesis.

2. Both implications can be proved by induction on derivations proving the hypothesis. The proof is quite involved, but standard. See for example [Gunter, 1992, Section 4.4].

□

3.3 A Scott-Domain semantics of $\mathcal{S}\ell\text{PCF}$

In this section we give an interpretation of $\mathcal{S}\ell\text{PCF}$ into a category built in the Scott Domain setting. This category is obtained by taking Scott Domains as objects and strict continuous functions as morphisms and it is called **StrictBcdom**. This setting was studied in [Dezani-Ciancaglini et al., 1986, Egidi et al., 1992], to define classes of models for the call by value λ -calculus.

This section is divided into three subsections: in the first, we recall the definition of **StrictBcdom** and we show that it is a Linear Category, by taking the lifting construct as the exponential co-monad; in the second, we show that **StrictBcdom** is also an $\mathcal{S}\ell\lambda$ -category, by choosing an opportune natural number object; in the third, we show that the considered model is adequate for the operational semantics of $\mathcal{S}\ell\text{PCF}$.

3.3.1 The category **StrictBcdom**

Let **StrictBcdom** be the category such that:

- the objects are *Scott Domains*;
- the morphisms are the *strict continuous functions*, namely those continuous functions that always take the bottom element of the source object to the bottom element of the target object.

We denote with \perp_D the least element of a Scott Domain D . The following result is taken from [Pravato et al., 1999].

Proposition 3.3.1 ([Pravato et al., 1999]). **StrictBcdom** is a Linear Category.

Proof. • \otimes is the *smash product*

$$D_1 \otimes D_2 = \{\langle d_1, d_2 \rangle \mid d_1 \in D_1, d_2 \in D_2, d_1 \neq \perp_{D_1}, d_2 \neq \perp_{D_2}\} \cup \{\perp_{D_1 \otimes D_2}\}$$

with unit $\mathbf{1} = \{\perp, \top\}$, being \perp smaller than \top , and for any $f : D_1 \rightarrow D_2$ and $g : D_3 \rightarrow D_4$, we define $f \otimes g : D_1 \otimes D_3 \rightarrow D_2 \otimes D_4$ as:

$$f \otimes g(d) = \begin{cases} \langle f(d_1), g(d_2) \rangle & \text{if } d = \langle d_1, d_2 \rangle \text{ and } f(d_1) \neq \perp_{D_2} \text{ and } g(d_2) \neq \perp_{D_4} \\ \perp_{D_2 \otimes D_4} & \text{otherwise} \end{cases}$$

- $D_1 \multimap D_2$ is the cpo of strict continuous functions from D_1 to D_2 ordered point-wise (which is a Scott Domain).
- $\langle \mathbf{StrictBcdom}, \otimes, \multimap, \mathbf{1} \rangle$ is a symmetric monoidal closed category, by setting
 - $\text{curry}(-) : \mathbf{StrictBcdom}(D_1 \otimes D_2, D_3) \rightarrow \mathbf{StrictBcdom}(D_1, D_2 \multimap D_3)$ such that $\text{curry}(f)(\perp_{D_1}) = \perp_{D_2 \multimap D_3}$ and $\text{curry}(f)(d_1)(d_2) = f(\langle d_1, d_2 \rangle)$ if $d_2 \neq \perp_{D_2}$ and $\text{curry}(f)(d_1)(\perp_{D_2}) = \perp_{D_3}$.

- $\text{eval} : (D_1 \multimap D_2) \otimes D_1 \rightarrow D_2$ is $\text{eval}(\langle f, d_1 \rangle) = f(d_1)$ and $\text{eval}(\perp_{(D_1 \multimap D_2) \otimes D_1}) = \perp_{D_2}$.
- The *exponential co-monad* is the lifting functor $(-)_\perp$, namely:
 - D_\perp is the Scott Domain obtained from D by adding a new bottom element \perp .
 - for any $f : D_1 \rightarrow D_2$, the morphism $f_\perp : D_{1\perp} \rightarrow D_{2\perp}$ is $f_\perp(d) = f(d)$ if $d \in D_1$ while $f_\perp(\perp) = \perp$
- $\varepsilon_D : D_\perp \rightarrow D$ is $\varepsilon_D(d) = d$ if $d \in D$, while $\varepsilon_D(\perp) = \perp_D$.
- $\delta_D : D_\perp \rightarrow D_{\perp\perp}$ is $\delta_D(d) = d$ if $d \in D$, while $\delta_D(\perp) = \perp'$.
- $q_{D_1, D_2} : D_{1\perp} \otimes D_{2\perp} \rightarrow (D_1 \otimes D_2)_\perp$ and $q_1 : \mathbf{1} \rightarrow \mathbf{1}_\perp$ are:

$$q_{D_1, D_2}(d) = \begin{cases} \langle d_1, d_2 \rangle & \text{if } d = \langle d_1, d_2 \rangle \text{ and } d_1 \neq \perp_{D_1} \text{ and } d_2 \neq \perp_{D_2} \\ \perp & \text{otherwise} \end{cases}$$

$$q_1(\top) = \top \qquad q_1(\perp_1) = \perp$$

- $e_D : D_\perp \rightarrow \mathbf{1}$ is $e_D(d) = \top$ if $d \in D$, while $e_D(\perp) = \perp_1$
- $d_D : D_\perp \rightarrow D_\perp \otimes D_\perp$ is $d_D(d) = \langle d, d \rangle$ if $d \in D$ while $d_D(\perp) = \perp_{D_\perp \otimes D_\perp}$.

□

In order to make **StrictBcdom** a $\mathcal{S}\ell\lambda$ Category, thus a model of $\mathcal{S}\ell\lambda$ -calculus it is necessary to choose an opportune exponential object for numerals.

Proposition 3.3.2. *StrictBcdom is an $\mathcal{S}\ell\lambda$ Category.*

Proof. 1. Let N be the *flat domain of natural number*, i.e. $N = \mathbb{N} \cup \{\perp_N\}$ with $d_1 \sqsubseteq d_2 \in N$ if either $d_1 = \perp_N$ or $d_1 = d_2$. Then it is an exponential object of numeral, by setting

- $\text{succ} : N \rightarrow N$ be such that $\text{succ}(d) = d + 1$ if $n \in \mathbb{N}$ and $\text{succ}(\perp_N) = \perp_N$, while
- $\text{pred} : N \rightarrow N$ be such that $\text{pred}(d + 1) = d$ if $d \in \mathbb{N}$, $\text{pred}(0) = \text{pred}(\perp_N) = \perp_N$.
- $p : N \rightarrow N_\perp$ be such that $p(d) = d$ if $d \in \mathbb{N}$ and $p(\perp_N) = \perp$.
- $w_N : N \rightarrow \mathbf{1}$ be such that $w_N(d) = \top$ if $d \in \mathbb{N}$ and $w_N(\perp_N) = \perp_1$.
- $c_N : N \rightarrow N \otimes N$ be such that $c_N(d) = \langle d, d \rangle$ if $d \in \mathbb{N}$ and $c_N(\perp_N) = \perp_{N \otimes N}$.

It is routine to check that all the required diagrams for having an exponential object for numerals commute.

2. **StrictBcdom** is cartesian by setting $D_1 \times D_2 = \{\langle d_1, d_2 \rangle \mid d_1 \in D_1, d_2 \in D_2\}$ ordered componentwise. It is easy to see that projections $\pi_1 : D_1 \times D_2 \rightarrow D_1$ and $\pi_2 :$

$D_1 \times D_2 \rightarrow D_2$ are strict continuous functions. Thus we define the *conditional function* as follows

$$\text{lif}(d) = \begin{cases} d_1 & \text{if } d = \langle 0, \langle d_1, d_2 \rangle \rangle \\ d_2 & \text{if } d = \langle n + 1, \langle d_1, d_2 \rangle \rangle \text{ with } n \in \mathbb{N} \\ \perp_N & \text{otherwise} \end{cases}$$

It is routine to check that lif is strict continuous and that the required diagram commutes.

3. The Kleisli category over the co-monad $(-)_\perp$ is equivalent to the category **Bcdom**, which admits a fix point operator for every object. □

Remark 3.3.1. Observe that, in spite of the fact that \otimes is the cartesian product in the category of co-monoids of **StrictBcdom**, it is not the case that \otimes is the cartesian product in **StrictBcdom**. In particular in the category of co-monoids of **StrictBcdom**, $N \otimes N$ is the cartesian product of the pair of comonoids (N, N) (the commutativity of the usual diagram can be proved just by observing that \perp_N is not a co-monoid morphism, since $\text{id}_1 \neq w_N(\perp_N) = \perp_1$). However $N \otimes N$ is not the cartesian product of the pair of objects (N, N) in **StrictBcdom**.

We denote with $C[\![-]\!]$ the interpretation function obtained from Definition 3.1.10; it follows from Theorem 3.1.1 that this interpretation function induces an equational theory on $\mathcal{S}\ell\lambda$ -terms which contains $=_{\mathcal{S}\ell}$ and it is contextually closed. Hence, this model is sound for $\mathcal{S}\ell\text{PCF}$ or more formally we have the following theorem.

Theorem 3.3.1 (Soundness). $\mathbb{M} \Downarrow \underline{n}$ implies $C[\![\mathbb{M}]\!] = C[\![\underline{n}]\!]$

Proof. $\mathbb{M} \Downarrow \underline{n}$ implies $\mathbb{M} =_{\mathcal{S}\ell} \underline{n}$ by Theorem 3.2.1. Thus the result follows by Proposition 3.3.2 and Theorem 3.1.1. □

3.3.2 Adequacy and correctness

The denotational semantics $C[\![-]\!]$ is said to be *adequate* when $C[\![\mathbb{M}]\!] = [\![\underline{n}]\!]$ and $\mathbb{M} \Downarrow \underline{n}$ are logically equivalent for any program \mathbb{M} , numeral \underline{n} . The following adequacy proof is an adaptation of a proof of Plotkin given in [Plotkin, 1977].

Definition 3.3.1. The “computability predicate” is defined by the following cases.

- Case $FV(\mathbb{M}^\sigma) = \emptyset$.
 - Subcase $\sigma = \iota$. $\text{Comp}(\mathbb{M}^\iota)$ if and only if $C[\![\mathbb{M}]\!] = C[\![\underline{n}]\!]$ implies $\mathbb{M} \Downarrow \underline{n}$.
 - Subcase $\sigma = \mu \multimap \tau$. $\text{Comp}(\mathbb{M}^{\mu \multimap \tau})$ if and only if $\text{Comp}(\mathbb{M}^{\mu \multimap \tau} \mathbb{N}^\mu)$ for each closed \mathbb{N}^μ such that $\text{Comp}(\mathbb{N}^\mu)$.
- Case $FV(\mathbb{M}^\sigma) = \{\chi_1^{\tau_1}, \dots, \chi_n^{\tau_n}\}$, for some $n \geq 1$.
 $\text{Comp}(\mathbb{M}^\sigma)$ if and only if $\text{Comp}(\mathbb{M}[\mathbb{N}_1/\chi_1, \dots, \mathbb{N}_n/\chi_n])$ for each closed $\mathbb{N}_i^{\tau_i}$ such that $\text{Comp}(\mathbb{N}_i^{\tau_i})$.

Lemma 3.3.1 states an equivalent formulation of computability predicate.

Lemma 3.3.1. *Let $M^{\tau_1 \multimap \dots \multimap \tau_m \multimap \iota} \in \mathcal{S}\ell\text{PCF}$ and $FV(M) = \{\mathcal{X}_1^{\mu_1}, \dots, \mathcal{X}_n^{\mu_n}\}$ ($n, m \in \mathbb{N}$). $\text{Comp}(M)$ if and only if $C[\llbracket M[N_1/\mathcal{X}_1, \dots, N_n/\mathcal{X}_n]P_1 \dots P_m \rrbracket] = C[\llbracket \underline{n} \rrbracket]$ implies $M[N_1/\mathcal{X}_1, \dots, N_n/\mathcal{X}_n]P_1 \dots P_m \Downarrow \underline{n}$ for each closed $N_i^{\mu_i}$ and $P_j^{\tau_j}$ such that $\text{Comp}(N_i)$ and $\text{Comp}(P_j)$ where $i \leq n, j \leq m$.*

Adequacy follows immediately by next lemma.

Lemma 3.3.2. *If $M^\sigma \in \mathcal{S}\ell\text{PCF}$ then $\text{Comp}(M^\sigma)$.*

Proof. The proof is done by induction on the “untyped syntax shape” of terms. We consider only some cases, the other are easier (see [Paolini, 2006, Plotkin, 1977]).

- $M = \mathcal{X}$. Let $\sigma = \tau_1 \multimap \dots \multimap \tau_m \multimap \iota$, where $m \in \mathbb{N}$. Let P^σ and $N_i^{\tau_i}$ for $1 \leq i \leq m$ be closed terms such that $\text{Comp}(P^\sigma)$ and $\text{Comp}(N_i^{\tau_i})$. By definition, $\text{Comp}(P^\sigma)$ imply that, if $C[\llbracket P[N_1 \dots N_m] \rrbracket] = C[\llbracket \underline{n} \rrbracket]$ then $P[N_1 \dots N_m] \Downarrow \underline{n}$, by definition of the computability predicate.
- $M = NP$. Assume $N^{\tau \multimap \sigma}$ and P^τ for types σ and τ . By induction hypothesis $\text{Comp}(N^{\tau \multimap \sigma})$ and $\text{Comp}(P^\tau)$ and the proof follows by definition of the computability predicate.
- $M = \lambda x.Q$. Assume x^μ and Q^τ for types μ and τ . Let $FV(M) = \{\mathcal{X}_1^{\mu_1}, \dots, \mathcal{X}_k^{\mu_k}\}$ for $k \geq 0$ and $\tau = \tau_1 \multimap \dots \multimap \tau_h \multimap \iota$, where $h \geq 0$. Let $N_1^{\mu_1}, \dots, N_k^{\mu_k}, P_0^\mu, P_1^{\tau_1}, \dots, P_h^{\tau_h}$ be closed terms such that $\text{Comp}(N_i)$ and $\text{Comp}(P_j)$ for $1 \leq i \leq k$ and $0 \leq j \leq h$ respectively. Thus $\text{Comp}(Q^\tau[P_0/x][N_1/\mathcal{X}_1, \dots, N_k/\mathcal{X}_k]P_1 \dots P_h)$, since $\text{Comp}(Q^\tau)$ holds by induction hypothesis. Consider the case $\mu \neq \iota$ and suppose

$$C[\llbracket (\lambda x^\mu.Q^\tau)[N_1/\mathcal{X}_1, \dots, N_k/\mathcal{X}_k]P_0 \dots P_h \rrbracket] = C[\llbracket \underline{n} \rrbracket]$$

Therefore $C[\llbracket Q^\tau[P_0/x][N_1/\mathcal{X}_1, \dots, N_k/\mathcal{X}_k]P_1 \dots P_h \rrbracket] = C[\llbracket \underline{n} \rrbracket]$ by Theorem 3.1.1 and Proposition 3.3.2. Thus $Q^\tau[P_0/x][N_1/\mathcal{X}_1, \dots, N_k/\mathcal{X}_k]P_1 \dots P_h \Downarrow \underline{n}$ by induction hypothesis. So, $(\lambda x^\mu.Q^\tau)[N_1/\mathcal{X}_1, \dots, N_k/\mathcal{X}_k]P_0 \dots P_h \Downarrow \underline{n}$ by the evaluation rule (λ^-) . Now, suppose $\mu = \iota$ and

$$C[\llbracket (\lambda x^\mu.Q^\tau)[N_1/\mathcal{X}_1, \dots, N_k/\mathcal{X}_k]P_0 \dots P_h \rrbracket] = C[\llbracket \underline{n} \rrbracket],$$

Since eval is strict, we must have $C[\llbracket P_\emptyset \rrbracket] = C[\llbracket \underline{m} \rrbracket]$ for some \underline{m} . But $\text{Comp}(P_\emptyset^\mu)$ by induction and $C[\llbracket P_\emptyset \rrbracket] = C[\llbracket \underline{m} \rrbracket]$ imply $P_\emptyset \Downarrow \underline{m}$. Hence

$$C[\llbracket Q^\tau[\underline{m}/x][N_1/\mathcal{X}_1, \dots, N_k/\mathcal{X}_k]P_1 \dots P_h \rrbracket] = C[\llbracket \underline{n} \rrbracket]$$

by Theorem 3.1.1 and Proposition 3.3.2. Therefore

$$Q^\tau[\underline{m}/x][N_1/\mathcal{X}_1, \dots, N_k/\mathcal{X}_k]P_1 \dots P_h \Downarrow \underline{n}$$

by induction hypothesis. The proof follows by applying the evaluation rule (λ') .

- $M = \ell\text{if } M_1 M_2 M_3$. Assume $\text{FV}(M) = \{\chi_1^{\sigma_1}, \dots, \chi_n^{\sigma_n}\}$ and let $P_1^{\sigma_1}, \dots, P_n^{\sigma_n}$ be closed terms such that $\text{Comp}(P_i)$ for all i . Suppose that

$$\llbracket M[P_1/\chi_1, \dots, P_n/\chi_n] \rrbracket = \llbracket \underline{m} \rrbracket$$

Then, by interpretation (namely by definition of the function ℓif), either

$$\llbracket M_1[P_1/\chi_1, \dots, P_n/\chi_n] \rrbracket = \llbracket \underline{0} \rrbracket \text{ and } \llbracket M_2[P_1/\chi_1, \dots, P_n/\chi_n] \rrbracket = \llbracket \underline{m} \rrbracket$$

or

$$\llbracket M_1[P_1/\chi_1, \dots, P_n/\chi_n] \rrbracket = \llbracket \underline{k+1} \rrbracket \text{ and } \llbracket M_3[P_1/\chi_1, \dots, P_n/\chi_n] \rrbracket = \llbracket \underline{m} \rrbracket$$

In the first case we have $M_1[P_1/\chi_1, \dots, P_n/\chi_n] \Downarrow \underline{0}$ and $M_2[P_1/\chi_1, \dots, P_n/\chi_n] \Downarrow \underline{m}$ by inductive hypothesis; thus we conclude by the evaluation rule (ifl). For the other case, we conclude again by using inductive hypothesis and the evaluation rule (ifr).

- $M = \mu F.N$. Assume M^σ, N^σ for some type σ . Let $\text{FV}(M) = \{\chi_1^{\mu_1}, \dots, \chi_k^{\mu_k}\}$ for $k \geq 0$ and $\sigma = \tau_1 \multimap \dots \multimap \tau_h \multimap \iota$, where $h \geq 0$. By induction on h , likewise to the corresponding proof of [Plotkin, 1977]. The case $h = 0$ is trivial, so assume $h \geq 1$. Assume $N_1^{\mu_1}, \dots, N_k^{\mu_k}$ and $P_1^{\tau_1}, \dots, P_h^{\tau_h}$ be closed terms such that $\text{Comp}(N_i)$ and $\text{Comp}(P_j)$ for $1 \leq i \leq k$ and $1 \leq j \leq h$ respectively. Let $C[\llbracket (\mu F.N^\sigma[N_1/\chi_1, \dots, N_k/\chi_k])P_1 \dots P_h \rrbracket] = C[\llbracket \underline{n} \rrbracket]$. By Proposition 2.3.9 and by unfolding of definitions, we have

$$\begin{aligned} C[\llbracket (\mu F.N^\sigma[N_1/\chi_1, \dots, N_k/\chi_k])P_1 \dots P_h \rrbracket] = \\ C[\llbracket (\mu^k F.N^\sigma[N_1/\chi_1, \dots, N_k/\chi_k])P_1 \dots P_h \rrbracket] \end{aligned}$$

for some $k \in \mathbb{N}$. Thus, $\mu^k F.N^\sigma[Q'/v_1, \dots, Q'/v_m]P_1 \dots P_h \Downarrow \underline{n}$, by the previous points of this lemma. The proof follows by Lemma 3.2.3. □

Theorem 3.3.2 (Adequacy). $M \Downarrow \underline{n}$ if and only if $C[\llbracket M \rrbracket] = C[\llbracket \underline{n} \rrbracket]$.

Proof. It follows from Lemma 3.3.2 and Theorem 3.3.1. □

The considered model is however not fully abstract for $\mathcal{S}\ell\text{PCF}$, since we are not able to define all compact elements of the model with a term of the language. The full abstraction problem for $\mathcal{S}\ell\text{PCF}$ will be discussed in detail in Section 3.5.

3.4 A coherence space semantics for $\mathcal{S}\ell\text{PCF}$

In this section we will move on a particular model of $\mathcal{S}\ell\text{PCF}$, which is the category **LinCoh** of coherence space and linear function. We will see that such a model will be finer than the model based on Scott Domains presented in previous Section, since a full abstraction result can be established.

It is well known that **LinCoh** is a Linear Category. In this Section we show that it is also an $\mathcal{S}\ell\lambda$ Category, by choosing a suitable exponential object for numeral. Moreover we show that the induced model is adequate and that a full abstraction result for closed $\mathcal{S}\ell\text{PCF}$ -terms holds.

3.4.1 More on Coherence Spaces

In this section we recall a well-known result about the category of Coherence Spaces and Linear Functions **LinCoh**, i.e. that **LinCoh** is a Linear Category.

We begin by defining a suitable exponential co-monad $!$ in **LinCoh**.

Definition 3.4.1. *Let X be a coherence space. We define $!X$ to be the coherence space having as web $!|X| = Cl_{fin}(X)$ and as coherence relation $x \supset_{!X} y$ if $x \cup y \in Cl(X)$. Given a linear function $f : Cl(X) \rightarrow Cl(Y)$, we define $!f : Cl(!X) \rightarrow Cl(!Y)$ to be such that*

$$\text{tr}(!f) = \left\{ (x, y) \in !|X| \multimap !|Y| \mid \begin{array}{l} \forall a \in x. \exists b \in y. (a, b) \in \text{tr}(f) \\ \forall b \in y. \exists a \in x. (a, b) \in \text{tr}(f) \end{array} \right\}$$

Proposition 3.4.1. *$!$ is an exponential co-monad.*

Proof. We sketch the proof. More details can be found in [Pravato et al., 1999, Melliès, 2003]. $!$ is a co-monad by taking $\delta_X : Cl(!X) \rightarrow Cl(!X)$ and $\varepsilon_X : Cl(!X) \rightarrow Cl(X)$ be such that

- $\text{tr}(\delta_X) = \{(x, y) \in !|X| \multimap !|X| \mid y = \{x_1, \dots, x_n\}, x = \bigcup x_i\}$
- $\text{tr}(\varepsilon_X) = \{(\{a\}, a) \mid a \in |X|\}$

$\langle !X, d_X, e_X \rangle$ is a commutative co-monoid by taking

- $d_X : Cl(!X) \rightarrow Cl(!X \otimes !X)$ be such that $\text{tr}(d_X) = \{(x, y) \in !|X| \multimap !|X \otimes !X| \mid y = (y_1, y_2), x = y_1 \cup y_2\}$
- $e_X : Cl(!X) \rightarrow Cl(\mathbf{1})$ be such that $\text{tr}(e_X) = \{(\emptyset, *)\}$.

$!$ is a symmetric monoidal co-monad by taking q such that

$$\text{tr}(q_{X,Y}) = \{((x, y), z) \in !(|X \otimes !Y|) \multimap !(|X \otimes Y|) \mid z = \{(a_1, b_1), \dots, (a_n, b_n)\}, a_i \in x, b_i \in y\}$$

$$\text{tr}(q_{\mathbf{1}}) = \{(*, \emptyset), (*, \{*\})\}$$

where $*$ is the unique token of the coherence space $\mathbf{1}$. It is routine to check the commutativity of all required diagrams. \square

Let us recall that the Kleisli category over the co-monad $!$ denoted **LinCoh**_! is equivalent to the category **StabCoh** of Coherence Spaces and Stable Functions, which is Cartesian Closed [Melliès, 2003].

3.4.2 LinCoh is a model for $\mathcal{S}\ell\text{PCF}$

In this section we show that **LinCoh** is an $\mathcal{S}\ell\lambda$ Category.

Conditional Operator

Conditional operator can be realised in **LinCoh** by the linear function $\ellif : Cl(\mathbf{N} \otimes (\mathbf{N} \& \mathbf{N})) \rightarrow Cl(\mathbf{N})$ which is such that

$$\text{tr}(\ellif) = \{((0, \text{inl}(n)), n) \mid n \in \mathbb{N}\} \cup \{((m+1, \text{inr}(n)), n) \mid n, m \in \mathbb{N}\}$$

We can easily prove the following.

Proposition 3.4.2. $\ellif(0, \langle x, y \rangle) = x$ and $\ellif(n+1, \langle x, y \rangle) = y$ for all $x, y \in Cl(\mathbf{N})$.

Proof. Straightforward. □

Fix-Point Operator

In order to define a fix-point operator for every coherence space in the Kleisli category **LinCoh**_! over the co-monad $\langle !, \delta, \varepsilon \rangle$, we make use of the Knaster-Tarsky Fix Point Theorem.

Given a coherence space $X = \langle |X|, \subset_X \rangle$, the required map $\text{fix}_X : Cl(X \Rightarrow X) \rightarrow Cl(X)$ in **LinCoh**_! satisfies the following equation.

$$\text{fix}_X = x \in Cl(X \Rightarrow X) \mapsto \text{eval}(x, \text{fix}_X(x))$$

Thus, we can define $\text{fix}_X : Cl(X \Rightarrow X) \rightarrow Cl(X)$ to be the stable function whose trace is the least fix point of the following stable functional $F : Cl((X \Rightarrow X) \Rightarrow X) \rightarrow Cl((X \Rightarrow X) \Rightarrow X)$, defined as

$$F = x \in Cl((X \Rightarrow X) \Rightarrow X) \mapsto \text{tr}(f \in Cl(X \Rightarrow X) \mapsto \text{eval}(f, \text{eval}(x, f)))$$

In a more detail, given $n \in \mathbb{N}$ we can define $F^n : (X \Rightarrow X) \Rightarrow X$ by induction on n as follows.

$$F^0 = \emptyset \in Cl((X \Rightarrow X) \Rightarrow X) \quad F^{n+1} = \text{tr}(f \in Cl(X \Rightarrow X) \mapsto \text{eval}(f, F^n(f)))$$

Since F is stable, then it is also continuous and by Knaster-Tarsky we conclude that $\text{tr}(\text{fix}_X) = \bigcup_{n \in \mathbb{N}} F^n$ is well defined. Finally, it is easy to see that the above equation holds.

Proposition 3.4.3 (Rationality). *Let X be a coherence space and let $f : Cl(X) \rightarrow Cl(X)$, $g : Cl(X) \rightarrow Cl(\mathbf{N})$ be two stable functions and let p be a numeral. Then*

$$g(\text{fix}_X(\text{tr}(f))) = \{p\} \Rightarrow \exists k \text{ such that } g(f^k(\emptyset)) = \{p\}$$

where f^k is the k -time iteration of f .

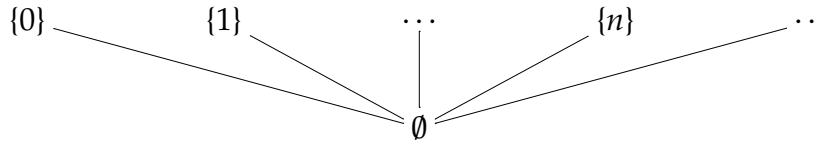
Proof. Since function application is continuous, we have

$$\begin{aligned} g(\text{fix}_X(\text{tr}(f))) &= g(\bigcup_n F^n(\text{tr}(f))) \\ &= g(\bigcup_n f^n(\emptyset)) \\ &= \bigcup_n g(f^n(\emptyset)) \\ &= \{p\} \end{aligned}$$

Thus we can conclude that there is a k such that $g(f^k(\emptyset)) = \{p\}$. \square

Numerals

We need to exhibit an exponential object for numerals in \mathbf{LinCoh} . The natural choice is to start from the usual infinite flat domain of natural number which is denoted by the coherence space $\mathbf{N} = \langle \mathbb{N}, \circlearrowright_{\mathbf{N}} \rangle$ where \mathbb{N} is the set of natural numbers and $\circlearrowright_{\mathbf{N}}$ is the smallest reflexive relation on \mathbb{N} . The space of its cliques can be depicted as follows.



There are also natural choices for the successor function $\text{succ} : Cl(\mathbf{N}) \rightarrow Cl(\mathbf{N})$ and the predecessor function $\text{pred} : Cl(\mathbf{N}) \rightarrow Cl(\mathbf{N})$, whose definition is similar the one given for Scott Domains. They are linear functions between the clique space of natural numbers.

Proposition 3.4.4. \mathbf{N} is an object for numerals in the category \mathbf{LinCoh} .

Proof. By unfolding the definitions, we can easily prove that $\text{pred}(\text{succ}(\{n\})) = \{n\}$ for all $n \in \mathbb{N}$. \square

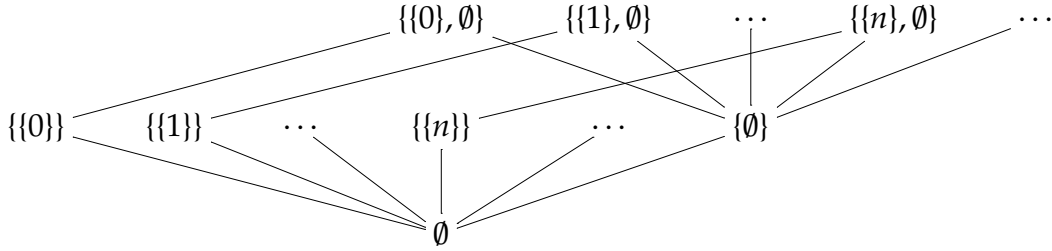
The linear functions $c_{\mathbf{N}} : Cl(\mathbf{N}) \rightarrow Cl(\mathbf{N} \otimes \mathbf{N})$ and $w_{\mathbf{N}} : Cl(\mathbf{N}) \rightarrow Cl(\mathbf{1})$ realising weakening and contraction on ground data can be chosen in the canonical way as follows.

- $c_{\mathbf{N}} : Cl(\mathbf{N}) \rightarrow Cl(\mathbf{N} \otimes \mathbf{N})$ is the function such that $\text{tr}(c_{\mathbf{N}}) = \{(n, (n, n)) \mid n \in \mathbb{N}\}$
- $w_{\mathbf{N}} : Cl(\mathbf{N}) \rightarrow Cl(\mathbf{1})$ is the function such that $\text{tr}(w_{\mathbf{N}}) = \{(n, *) \mid n \in \mathbb{N}\}$

Proposition 3.4.5. $c_{\mathbf{N}}$ and $w_{\mathbf{N}}$ are both linear functions and they are such that $\langle \mathbf{N}, c_{\mathbf{N}}, w_{\mathbf{N}} \rangle$ is a commutative co-monoid.

Proof. Straightforward, by unfolding definitions. \square

We need now to give a linear function realising the promotion from \mathbf{N} to $!\mathbf{N}$. The clique space of the latter is depicted below.



We can observe that there are many ways to embed \mathbf{N} into $!\mathbf{N}$, but there is only one possible choice for the $!$ -coalgebra $p : Cl(\mathbf{N}) \rightarrow Cl(!\mathbf{N})$, which is given by the following proposition.

Proposition 3.4.6. *Let $p : Cl(\mathbf{N}) \rightarrow Cl(!\mathbf{N})$ be a linear function such that*

$$\text{tr}(p) = \{(n, \{n\}) \mid n \in \mathbb{N}\} \cup \{(n, \emptyset) \mid n \in \mathbb{N}\}$$

Then

1. $p : Cl(\mathbf{N}) \rightarrow Cl(!\mathbf{N})$ is a co-algebra
2. $0 : \mathbf{1}$ and $\text{succ} : \mathbf{N} \rightarrow \mathbf{N}$ are both co-algebra and co-monoid morphisms.
3. $p : Cl(\mathbf{N}) \rightarrow Cl(!\mathbf{N})$ is a co-monoid morphism.

Proof. (1.) It is easy to see that $\varepsilon_{\mathbf{N}} \circ p = \text{id}_{\mathbf{N}}$. To prove that $!p \circ p = \delta_{\mathbf{N}} \circ p$, where

- $\text{tr}(!p) = \{(\{n\}, \{\{n\}\}) \mid n \in \mathbb{N}\} \cup \{(\{n\}, \{\emptyset\}) \mid n \in \mathbb{N}\} \cup \{(\{n\}, \{\{n\}, \emptyset\}) \mid n \in \mathbb{N}\} \cup \{(\emptyset, \emptyset)\}$
- $\text{tr}(\delta_{\mathbf{N}}) = \{(\{n\}, \{\{n\}\}) \mid n \in \mathbb{N}\} \cup \{(\{n\}, \{\{n\}, \emptyset\}) \mid n \in \mathbb{N}\} \cup \{(\emptyset, \emptyset), (\emptyset, \{\emptyset\})\}$

we have $(!p \circ p)(\emptyset) = (\delta_{\mathbf{N}} \circ p)(\emptyset) = \emptyset$ by linearity and $(!p \circ p)(\{n\}) = !p(\{\emptyset, \{n\}\}) = \{\emptyset, \{\emptyset\}, \{\{n\}\}, \{\{n\}, \emptyset\}\} = \delta_{\mathbf{N}}(\{\emptyset, \{n\}\}) = (\delta_{\mathbf{N}} \circ p)(\{n\})$ by unfolding definitions. (2.) and (3.) are easy. \square

Soundness Theorem

From the results proved in the previous paragraphs, it follows that \mathbf{LinCoh} is actually a model of $\mathcal{S}\ell\lambda$ -calculus. More specifically, we can prove the following.

Proposition 3.4.7. $\langle \mathbf{LinCoh}, \mathbf{N}, p, c_{\mathbf{N}}, w_{\mathbf{N}}, \text{lif} \rangle$ is an $\mathcal{S}\ell\lambda$ Category.

Let $\mathcal{L}[\![\cdot]\!]$ be the interpretation function induced by the above $\mathcal{S}\ell\lambda$ Category, by using Definition 3.1.10. Sometimes typing judgement of terms will be omitted inside the interpretation functions when clear from the context or uninteresting. It follows from Theorem 3.1.1 that this interpretation function induces an equational theory on $\mathcal{S}\ell\lambda$ -terms which contains $=_{\mathcal{S}}$ and it is contextually closed. Hence, this model is sound for $\mathcal{S}\ell\text{PCF}$ or more formally we have the following theorem.

Theorem 3.4.1 (Soundness). $M \Downarrow \underline{n}$ implies $\mathcal{L}[\![M]\!] = \mathcal{L}[\![\underline{n}]\!]$

Proof. $M \Downarrow \underline{n}$ implies $M =_{\mathcal{S}} \underline{n}$ by Theorem 3.2.1. Thus the result follows by Proposition 3.4.7 and Theorem 3.1.1. \square

3.4.3 Adequacy and Correctness

The denotational semantics is said to be *adequate* when $\mathcal{L}[\llbracket \mathbb{M} \rrbracket] = \mathcal{L}[\llbracket \underline{n} \rrbracket]$ and $\mathbb{M} \Downarrow \underline{n}$ are logically equivalent for any program \mathbb{M} , numeral \underline{n} . We straightforwardly adapt a proof of Plotkin [Plotkin, 1977] for Scott-continuous domains, based on a computability argument in Tait style.

Definition 3.4.2. The “computability predicate” is defined by the following cases.

- Case $FV(\mathbb{M}^\sigma) = \emptyset$.
 - Subcase $\sigma = \iota$. $\text{Comp}(\mathbb{M}^\iota)$ if and only if $\mathcal{L}[\llbracket \mathbb{M} \rrbracket] = \mathcal{L}[\llbracket \underline{n} \rrbracket]$ implies $\mathbb{M} \Downarrow \underline{n}$.
 - Subcase $\sigma = \mu \multimap \tau$. $\text{Comp}(\mathbb{M}^{\mu \multimap \tau})$ if and only if $\text{Comp}(\mathbb{M}^{\mu \multimap \tau} \mathbb{N}^\mu)$ for each closed \mathbb{N}^μ such that $\text{Comp}(\mathbb{N}^\mu)$.
- Case $FV(\mathbb{M}^\sigma) = \{\mathcal{X}_1^{\tau_1}, \dots, \mathcal{X}_n^{\tau_n}\}$, for some $n \geq 1$.
 $\text{Comp}(\mathbb{M}^\sigma)$ if and only if $\text{Comp}(\mathbb{M}[\mathbb{N}_1/\mathcal{X}_1, \dots, \mathbb{N}_n/\mathcal{X}_n])$ for each closed $\mathbb{N}_i^{\tau_i}$ such that $\text{Comp}(\mathbb{N}_i^{\tau_i})$.

Lemma 3.4.1 states an equivalent formulation of computability predicate.

Lemma 3.4.1. Let $\mathbb{M}^{\tau_1 \multimap \dots \multimap \tau_m \multimap \iota} \in \mathcal{S}\ell\text{PCF}$ and $FV(\mathbb{M}) = \{\mathcal{X}_1^{\mu_1}, \dots, \mathcal{X}_n^{\mu_n}\}$ ($n, m \in \mathbb{N}$). $\text{Comp}(\mathbb{M})$ if and only if $\mathcal{L}[\llbracket \mathbb{M}[\mathbb{N}_1/\mathcal{X}_1, \dots, \mathbb{N}_n/\mathcal{X}_n] \mathbb{P}_1 \dots \mathbb{P}_m \rrbracket] = \mathcal{L}[\llbracket \underline{n} \rrbracket]$ implies $\mathbb{M}[\mathbb{N}_1/\mathcal{X}_1, \dots, \mathbb{N}_n/\mathcal{X}_n] \mathbb{P}_1 \dots \mathbb{P}_m \Downarrow \underline{n}$ for each closed $\mathbb{N}_i^{\mu_i}$ and $\mathbb{P}_j^{\tau_j}$ such that $\text{Comp}(\mathbb{N}_i)$ and $\text{Comp}(\mathbb{P}_j)$ where $i \leq n, j \leq m$.

Adequacy follows immediately by next lemma.

Lemma 3.4.2. If $\mathbb{M}^\sigma \in \mathcal{S}\ell\text{PCF}$ then $\text{Comp}(\mathbb{M}^\sigma)$.

Proof. The proof is done by induction on the “untyped syntax shape” of terms. Observe that, linear functions between Coherence Space are strict, so almost all the cases are similar to the ones of Section 3.3.2. The only different one is the following.

Case $\mathbb{M} = \mu F.N$. Assume $\mathbb{M}^\sigma, \mathbb{N}^\sigma$ for some type σ . Let $FV(\mathbb{M}) = \{\mathcal{X}_1^{\mu_1}, \dots, \mathcal{X}_k^{\mu_k}\}$ for $k \geq 0$ and $\sigma = \tau_1 \multimap \dots \multimap \tau_h \multimap \iota$, where $h \geq 0$. By induction on h , likewise to the corresponding proof of [Plotkin, 1977]. The case $h = 0$ is trivial, so assume $h \geq 1$. Assume $\mathbb{N}_1^{\mu_1}, \dots, \mathbb{N}_k^{\mu_k}$ and $\mathbb{P}_1^{\tau_1}, \dots, \mathbb{P}_h^{\tau_h}$ be closed terms such that $\text{Comp}(\mathbb{N}_i)$ and $\text{Comp}(\mathbb{P}_j)$ for $1 \leq i \leq k$ and $1 \leq j \leq h$ respectively. Let $\mathcal{L}[\llbracket (\mu F.N^\sigma[\mathbb{N}_1/\mathcal{X}_1, \dots, \mathbb{N}_k/\mathcal{X}_k]) \mathbb{P}_1 \dots \mathbb{P}_h \rrbracket] = \mathcal{L}[\llbracket \underline{n} \rrbracket]$. By Proposition 3.4.3 and by unfolding of definitions, we have

$$\begin{aligned} \mathcal{L}[\llbracket (\mu F.N^\sigma[\mathbb{N}_1/\mathcal{X}_1, \dots, \mathbb{N}_k/\mathcal{X}_k]) \mathbb{P}_1 \dots \mathbb{P}_h \rrbracket] = \\ \mathcal{L}[\llbracket (\mu^k F.N^\sigma[\mathbb{N}_1/\mathcal{X}_1, \dots, \mathbb{N}_k/\mathcal{X}_k]) \mathbb{P}_1 \dots \mathbb{P}_h \rrbracket] \end{aligned}$$

for some $k \in \mathbb{N}$. Thus, $\mu^k F.N^\sigma[Q'/v_1, \dots, Q'/v_m] \mathbb{P}_1 \dots \mathbb{P}_h \Downarrow \underline{n}$, by the previous points of this lemma. The proof follows by Lemma 3.2.3. \square

Corollary 3.4.1 (Adequacy). The linear interpretation is adequate for $\mathcal{S}\ell\text{PCF}$.

As usual the adequacy implies the correctness. Note that correctness implies that our terms are strict in all arguments, for all orders.

Theorem 3.4.2 (Correctness). *The linear interpretation is correct for \mathcal{SLPCF} .*

Proof. Let M^σ and N^σ such that $\mathcal{L}[\llbracket M \rrbracket] = \mathcal{L}[\llbracket N \rrbracket]$. For simplicity, we assume that $\text{SFV}(M) = \text{SFV}(N) = \emptyset$; the extension is straightforward. Let $C[\sigma]$ such that $C[M], C[N] \in \mathcal{P}$. If $C[M] \Downarrow \underline{n}$ for some value \underline{n} , then $\mathcal{L}[\llbracket C[M] \rrbracket] = \mathcal{L}[\llbracket \underline{n} \rrbracket]$ by Theorem 3.4.1. Since $\mathcal{L}[\llbracket C[N] \rrbracket] = \mathcal{L}[\llbracket C[M] \rrbracket] = \mathcal{L}[\llbracket \underline{n} \rrbracket]$ by Theorem 3.1.1 and Proposition 3.4.7, $C[N] \Downarrow \underline{n}$ by adequacy. By definition of operational equivalence the proof is done. \square

3.4.4 Token Definability

If $\sigma \in \mathbb{T}$ then, as usual, its order $\text{order}(\sigma)$ is defined by $\text{order}(\iota) = 0$ and $\text{order}(\sigma_1 \multimap \dots \multimap \sigma_k \multimap \iota) = 1 + \max\{\text{order}(\sigma_i) \mid i \in [1, k]\}$.

We show that, all tokens of coherence spaces which are interpretation of types, are definable. We remind that, if M and N are closed ground terms then $M \doteq N$ is an abbreviation for the application of

$$\mu F^{\iota \multimap \iota \multimap \iota}. \lambda x^t y^t. \ell \text{if } x \ (\ell \text{if } y \ \underline{0} \ \underline{1}) \ (\ell \text{if } y \ \underline{1} \ (F(\text{pred } x)(\text{pred } y)))$$

to M and N . It is easy to check that

$$\mathcal{L}[\llbracket M \doteq N \rrbracket] = \begin{cases} 0 & \llbracket M \rrbracket = m = \llbracket N \rrbracket, \\ 1 & \llbracket M \rrbracket = m \neq n = \llbracket N \rrbracket, \\ \emptyset & \text{otherwise.} \end{cases}$$

Let N_0 and N_1 be an abbreviation for the term

$$\ell \text{if } N_0 \ (\ell \text{if } N_1 \ \underline{0} \ \underline{1}) \ (\ell \text{if } N_1 \ \underline{1} \ \underline{1})$$

Lemma 3.4.3 (Token Definability). *Let $\sigma \in \mathbb{T}$. If $a \in |\mathcal{L}[\llbracket \sigma \rrbracket]|$ then there is a closed term M^σ such that $\mathcal{L}[\llbracket M^\sigma \rrbracket] = \{a\}$.*

Proof. The proof is done by induction on the order of σ . The case $\text{order}(\sigma) = 0$ is trivial. Assume $\text{order}(\sigma) = 1$, thus $\sigma = \iota_1 \multimap \dots \multimap \iota_k \multimap \iota$. Hence, given $a \in |\mathcal{L}[\llbracket \sigma \rrbracket]|$, a has the shape (n_1, \dots, n_k, n) where $n_1, \dots, n_k, n \in \mathbb{N}$. If

$$\begin{aligned} M^\sigma &= \lambda x_1^t \dots x_k^t. \\ &\ell \text{if } ((x_1 \doteq \underline{n}_1) \text{and } (x_2 \doteq \underline{n}_2) \text{and } \dots \text{and } (x_k \doteq \underline{n}_k)) \ \underline{n} \ \Omega^t \end{aligned}$$

then $\mathcal{L}[\llbracket M^\sigma \rrbracket]_\rho = \{a\}$, for all ρ .

Finally, assume $\sigma = \sigma_1 \multimap \dots \multimap \sigma_k \multimap \iota$ where $\text{order}(\sigma_i) > 1$ and $\sigma_i = \tau_1^i \multimap \dots \multimap \tau_{h_i}^i \multimap \iota$ for $1 \leq i \leq k$. If $a \in |\mathcal{L}[\llbracket \sigma \rrbracket]|$ then a has shape $((a_1^1, \dots, a_{h_1}^1, n_1), \dots, (a_1^k, \dots, a_{h_k}^k, n_k), n')$ where $n_1, \dots, n_k, n' \in \mathbb{N}$ and $a_j^i \in |\mathcal{L}[\llbracket \tau_j^i \rrbracket]|$, for each $i \in [1, k]$ and $j \in [1, h_i]$. By inductive hypothesis, for all $i \in [1, k]$ and $j \in [1, h_i]$, there exists a closed term $M^{\tau_j^i}$ such that $\mathcal{L}[\llbracket M^{\tau_j^i} \rrbracket] = \{a_j^i\}$. Thus,

$$\lambda x_1^{\sigma_1} \dots \lambda x_k^{\sigma_k}. \ell \text{if } \left(\begin{array}{l} (x_1 M^{\tau_1^1} \dots M^{\tau_{h_1}^1} \doteq \underline{n}_1) \\ \text{and } \dots \text{ and} \\ (x_k M^{\tau_1^k} \dots M^{\tau_{h_k}^k} \doteq \underline{n}_k) \end{array} \right) \ \underline{n}' \ \Omega^t$$

is the the closed term defining the considered token. \square

Lemma 3.4.4 (Definable Separability). *Let $\sigma \in \mathbb{T}$. For all distinct $f, g \in Cl(\mathcal{L}[\![\sigma]\!])$ there exists a closed term $M^{\sigma \rightarrow \iota} \in \mathcal{S}PCF$ such that $eval(\mathcal{L}[\![M^{\sigma \rightarrow \iota}]\!], f) \neq eval(\mathcal{L}[\![M^{\sigma \rightarrow \iota}]\!], g)$.*

Proof. Let $\sigma = \sigma_1 \rightarrow \dots \rightarrow \sigma_k \rightarrow \iota$. Since $f \neq g$ there exists a such that $a \in f$ but $a \notin g$. Assume $a = (a_1, \dots, a_k, n)$ (where $a_1 \in |\mathcal{L}[\![\sigma_1]\!]|, \dots, a_k \in |\mathcal{L}[\![\sigma_k]\!]|, n \in \mathbb{N}$). By lemma 3.4.3, for each $i \in [1, k]$ there is $N^{\sigma_i} \in \mathcal{S}PCF$ such that $\mathcal{L}[\![N^{\sigma_i}]\!] = a_i$. By choosing $M^{\sigma \rightarrow \iota}$ as $\lambda x^\sigma. \text{if} ((x N^{\sigma_1} \dots N^{\sigma_k}) \doteq \underline{n}) \underline{0} \underline{1}$, the proof follows. \square

Theorem 3.4.3 (Closed Completeness). *Let N_1, N_2 such that $SFV(N_1) = SFV(N_2) = \emptyset$. If $N_1 \approx_\sigma N_2$ then $\mathcal{L}[\![N_1]\!] = \mathcal{L}[\![N_2]\!]$.*

Proof. Let us prove the contraposition. Let us assume $\mathcal{L}[\![N_1]\!] \neq \mathcal{L}[\![N_2]\!]$, with $\mathbf{x}_1^{\sigma_1}, \dots, \mathbf{x}_k^{\sigma_k} \vdash N_1, N_2 : \sigma$. Thus there exists $a_1 \in |\mathcal{L}[\![\sigma_1]\!]|, \dots, a_k \in |\mathcal{L}[\![\sigma_k]\!]|$ such that $\mathcal{L}[\![N_1]\!](\{a_1\}, \dots, \{a_k\}) \neq \mathcal{L}[\![N_2]\!](\{a_1\}, \dots, \{a_k\})$, since $\mathcal{L}[\![N_1]\!], \mathcal{L}[\![N_2]\!]$ are linear functions. By definable separability, there exists $M^{\sigma \rightarrow \iota}$ such that

$$eval(\mathcal{L}[\![M^{\sigma \rightarrow \iota}]\!])(\mathcal{L}[\![N_1]\!](\{a_1\}, \dots, \{a_k\})) = n_1 \neq eval(\mathcal{L}[\![M^{\sigma \rightarrow \iota}]\!])(\mathcal{L}[\![N_2]\!](\{a_1\}, \dots, \{a_k\}))$$

By Lemma 3.4.3, there exists closed M_1, \dots, M_k such that $\mathcal{L}[\![M_i]\!] = \{a_i\}$. Observe that we use here the hypothesis of $SFV(N_1) = SFV(N_2) = \emptyset$; if the terms N_1, N_2 have a free occurrence of a stable variable, then Lemma 3.4.3 is not sufficient to guarantee the existence of M_i . Thus we can build the following context $C = M((\lambda \mathbf{x}_1^{\sigma_1} \dots \lambda \mathbf{x}_k^{\sigma_k} [\cdot]) M_1 \dots M_k)$. By adequacy, $C[N_1] \Downarrow \underline{n}_1$ but it is not the case that $C[N_2] \Downarrow \underline{n}_2$. So $N_1 \not\approx_\sigma N_2$. \square

Corollary 3.4.2 (Closed Full abstraction). *Let N_1, N_2 such that $SFV(N_1) = SFV(N_2) = \emptyset$. Then*

$$N_1 \approx_\sigma N_2 \iff \mathcal{L}[\![N_1]\!] = \mathcal{L}[\![N_2]\!]$$

The above result is “morally” a full abstraction result for $\mathcal{S}PCF$, since when studying equivalences among programs, we are interested only in terms having no free occurrences of stable variables, since only these terms correspond to “purely” linear function (without any kind of exponentiation). Stable variables are used only to program fix-points of stable endo-transformations on a linear coherence space, thus they cannot be λ -abstracted.

A more general full abstraction result (for terms having free occurrences of stable variables) does not hold. This issue will be discussed in detail in the next section.

3.5 Linear Extensions of $\mathcal{S}PCF$

In this section we give two possible linear extensions of $\mathcal{S}PCF$, namely we introduce two different programming features for $\mathcal{S}PCF$. We prove that they are not syntactic sugar for $\mathcal{S}PCF$, i.e. they are not definable in terms of other language constructs. These two operators are inspired from the coherence space semantics of $\mathcal{S}PCF$, we give in the previous section. More specifically, we introduce two different operators,

called $G\text{or}^2$ and which?. The first is a parallel operator which is close to a gustave function [Berry, 1979]. The second is very close to a catch construct and it allows us to deal with exceptions in a linear setting. Thus, the model we defined in the previous section is not universal for $\mathcal{S}\ell\text{PCF}$.

3.5.1 Second Order Gustave Or

In spite of the separability result we give in the previous section, Corollary 3.4.2 cannot be extended to terms having free occurrences of stable variables. This means that the linear model is not fully abstract for $\mathcal{S}\ell\text{PCF}$. In this section we provide a counter-example for full abstraction. The technique we use is the same used to show that the Scott-Continuous model is not fully abstract for PCF [Plotkin, 1977]. We show that $\mathcal{S}\ell\text{PCF}$ is not able to define all finite elements of the model, by giving a linear function and showing that it is not definable by a $\mathcal{S}\ell\text{PCF}$ -term, since it is not strongly stable, in the sense of Bucciarelli-Ehrhard [Bucciarelli and Ehrhard, 1991]. Based on this function, we give two terms (one having free occurrences of stable variables) which are operationally equivalent, but interpreted into two different morphisms of **LinCoh**.

Let us consider the following finite clique of the coherence space $((\mathbf{N} \multimap \mathbf{N}) \multimap (\mathbf{N} \multimap \mathbf{N}) \multimap (\mathbf{N} \multimap \mathbf{N})) \multimap \mathbf{N}$.

$$\left\{ \begin{array}{l} ((0,0), (0,1), (1,0), 1) \\ ((1,0), (0,0), (0,1), 2) \\ ((0,1), (1,0), (0,0), 3) \\ ((1,1), (1,1), (1,1), 4) \end{array} \right\}$$

It corresponds to the linear function $G\text{or}^2$ defined as follows.

$$G\text{or}^2(f_1, f_2, f_3) = \begin{cases} \{1\} & \text{if } f_1(\{0\}) = \{0\}, f_2(\{0\}) = \{1\}, f_3(\{1\}) = \{0\} \\ \{2\} & \text{if } f_1(\{1\}) = \{0\}, f_2(\{0\}) = \{0\}, f_3(\{0\}) = \{1\} \\ \{3\} & \text{if } f_1(\{0\}) = \{1\}, f_2(\{1\}) = \{0\}, f_3(\{0\}) = \{0\} \\ \{4\} & \text{if } f_1(\{1\}) = \{1\}, f_2(\{1\}) = \{1\}, f_3(\{1\}) = \{1\} \\ \emptyset & \text{otherwise} \end{cases}$$

We can add $G\text{or}^2 : (\iota \multimap \iota) \multimap (\iota \multimap \iota) \multimap (\iota \multimap \iota) \multimap \iota$ to $\mathcal{S}\ell\text{PCF}$ with the operational semantics described in Table 3.3. It is easy to check that the interpretation of $G\text{or}^2$ is the trace given above and that the closed full abstraction hold again for this extension of $\mathcal{S}\ell\text{PCF}$.

The next two Subsections will be devoted to show that the $G\text{or}^2$ is not strongly stable in the sense of Antonio Bucciarelli and Thomas Ehrhard [Bucciarelli and Ehrhard, 1991, Bucciarelli and Ehrhard, 1994, Ehrhard, 1995]. For sake of simplicity, we will prove simply that a boolean version of $G\text{or}^2$ is not a strongly stable function.

$\frac{P_0 \underline{0} \Downarrow \underline{0} \quad P_1 \underline{0} \Downarrow \underline{1} \quad P_2 \underline{1} \Downarrow \underline{0}}{Gor \ P_0 P_1 P_2 \Downarrow \underline{1}} \quad (Gor_0^2)$	$\frac{P_0 \underline{1} \Downarrow \underline{0} \quad P_1 \underline{0} \Downarrow \underline{0} \quad P_2 \underline{0} \Downarrow \underline{1}}{Gor \ P_0 P_1 P_2 \Downarrow \underline{2}} \quad (Gor_1^2)$
$\frac{P_0 \underline{0} \Downarrow \underline{1} \quad P_1 \underline{1} \Downarrow \underline{0} \quad P_2 \underline{0} \Downarrow \underline{0}}{Gor \ P_0 P_1 P_2 \Downarrow \underline{3}} \quad (Gor_2^2)$	$\frac{P_0 \underline{1} \Downarrow \underline{1} \quad P_1 \underline{1} \Downarrow \underline{1} \quad P_2 \underline{1} \Downarrow \underline{1}}{Gor \ P_0 P_1 P_2 \Downarrow \underline{4}} \quad (Gor_3^2)$

Table 3.3: A Second Order Gustave OR.

Since Gor is not strongly stable, it is not definable in $\mathcal{S}PCF$. On this fact, we can build the desired counter-example of $\mathcal{S}PCF$. First of all we introduce some notation: we define \underline{n}^m the term $\lambda x^t. \ellif \ x \doteq \underline{m} \ \underline{n} \ \Omega^t$; it is easy to see that $\text{tr}(\mathcal{L}[\underline{n}^m]) = \{(m, n)\}$. Consider the following two terms M^t, N^t , which are such that $\Gamma \vdash M, N : \iota$, with $\Gamma = F^{(\iota \multimap \iota) \multimap (\iota \multimap \iota) \multimap (\iota \multimap \iota) \multimap \iota}$. Observe that the subterm $\ellif \ M_1 \begin{matrix} M_2 \\ M_3 \end{matrix}$ of N should be read as $\ellif \ M_1 \ M_2 \ M_3$

$$M = \Omega^t$$

$$N = \ellif \ F(\underline{0}^0)(\underline{0}^1)(\underline{1}^0) \ \ellif \ F(\underline{1}^0)(\underline{0}^0)(\underline{0}^1) \ \ellif \ F(\underline{0}^1)(\underline{1}^0)(\underline{0}^0) \ \ellif \ F(\underline{1}^1)(\underline{1}^1)(\underline{1}^1) \ \Omega^t$$

It is easy to see that $\mathcal{L}[\Gamma \vdash M : \iota] \neq \mathcal{L}[\Gamma \vdash N : \iota]$. Unfortunately $M \approx_t N$, since it is necessary a term behaving like Gor to separate the two terms. But such a term cannot exist, since Gor is not strongly stable.

Remark 3.5.1. Let N be the Scott Domain corresponding to the infinite flat domain of natural number defined in Section 3.3. Then the function $Gor: (N \multimap N) \multimap (N \multimap N) \multimap (N \multimap N) \multimap N$ defined in the analogous way as above for the Scott Domain setting is a strict continuous function. Thus, the same counterexample as above can be used to show that **StrictBcdom** is not fully abstract for $\mathcal{S}PCF$.

Strongly stable model

We recall basic definitions about strong stability, in the sense of Antonio Bucciarelli and Thomas Ehrhard, see [Bucciarelli and Ehrhard, 1994, Ehrhard, 1995].

Definition 3.5.1. A qualitative domain [Girard, 1986] is a pair $\langle |Q|, Q \rangle$ where $|Q|$ is a set (called the web) and Q is a subset of $\wp(|Q|)$ satisfying the following conditions:

- $\emptyset \in Q$ and, if $a \in |Q|$ then $\{a\} \in Q$
- if $x \in Q$ and if $y \subseteq x$ then $y \in Q$
- if $D \subseteq Q$ is directed with respect to inclusion, the $\bigcup D \in Q$

The elements of Q are called states of the qualitative domain, and the qualitative domain itself will also be denoted Q .

Property 3.5.1. *A qualitative domain Q is a coherence space when for all $u \subseteq |Q|$, if for all $a, b \in u, \{a, b\} \in Q$ then $u \in Q$.*

If $\langle D, \leq \rangle$ is a poset and $A, B \subseteq D$ then we say that A is Egli-Milner smaller than B (written $A \sqsubseteq_{EM} B$) if

$$\forall x \in A. \exists y \in B. x \leq y \quad \text{and} \quad \forall y \in B. \exists x \in A. x \leq y$$

Definition 3.5.2 (Qualitative domain with coherence). *A qualitative domain with coherence is a pair $\langle Q, C(Q) \rangle$ where Q is a qualitative domain while $C(Q)$ is a subset of $\wp_{fin}(Q)$ such that*

- if $x \in Q$ then $\{x\} \in C(Q)$
- if $A \in C(Q)$ and $B \sqsubseteq_{EM} A$ then $B \in C(Q)$
- if D_1, \dots, D_n is a family of directed subset of Q such that for any $d_1 \in D_1, \dots, d_n \in D_n$ the set $\{d_1, \dots, d_n\} \in C(Q)$ then $\{\sqcup D_1, \dots, \sqcup D_n\} \in C(Q)$

Intuitively, the notion of coherence corresponds to the notion of linear coherence in a sequential structure, see [Bucciarelli and Ehrhard, 1994, Ehrhard, 1995].

Example 3.5.1. *Let us consider the qualitative domain with coherence $\langle \mathbf{B}, C(\mathbf{B}) \rangle$ of booleans,*

- $|\mathbf{B}| = \{\mathbf{tt}, \mathbf{ff}\}$
- $\mathbf{B} = \{\emptyset, \{\mathbf{tt}\}, \{\mathbf{ff}\}\}$
- $C(\mathbf{B}) = \left\{ \begin{array}{l} \{\emptyset\}, \{\{\mathbf{tt}\}\}, \{\{\mathbf{ff}\}\}, \{\emptyset, \{\mathbf{tt}\}\}, \{\emptyset, \{\mathbf{ff}\}\}, \\ \{\emptyset, \{\mathbf{tt}\}, \{\mathbf{ff}\}\} \end{array} \right\}$

Morphisms between qualitative domains adapts the definition for coherence space. Let X and Y be qualitative domains.

- If $f : X \rightarrow Y$ is a monotone function then f is continuous if and only if $f(\sqcup D) = \sqcup \{f(x) \mid x \in D\}$, for each directed $D \subseteq X$.
- If $f : X \rightarrow Y$ is a continuous function then f is stable if and only if $\forall x, x' \in X, x \sqcup x' \in X$ implies $f(x \sqcap x') = f(x) \sqcap f(x')$.
- If $f : X \rightarrow Y$ is a stable function then f is linear if and only if $f(\perp) = \perp$ and $\forall x, x' \in X$ bounded $f(x \sqcup x') = f(x) \sqcup f(x')$.

Definition 3.5.3 (Strongly stable function). *A function $f : Q \rightarrow R$ between two qualitative domain with coherence is strongly stable if*

- for all $X \in C(Q)$ then $f(X) \in C(R)$

- $f(\prod X) = \prod_{a \in X} f(a)$.

The strongly stable functions are similar to stable functions, but they have to preserve coherence as well as intersections of coherent sets of states and not just of bounded ones. The category obtained taking as objects qualitative domains with coherence and as morphisms the strongly stable functions is cartesian closed.

Definition 3.5.4 (Product). *Let Q and R two qualitative domain with coherence. Their product $Q \times R$ is $\langle Q \& R, C(Q \times R) \rangle$ where $Q \& R$ is the product of beneath qualitative domains and, $X \in C(Q \times R)$ iff $\pi_1(X) \in C(Q)$ and $\pi_2(X) \in C(R)$.*

A set $X \subseteq A \times B$ is a *pairing* if $\pi_1(X) = A$ and $\pi_2(X) = B$. Recall that the set of strongly stable function between from Q to R is a qualitative domain [Ehrhard, 1995].

Definition 3.5.5 (Exponential object). *Let Q and R two qualitative domains with coherence. Let R^Q be the qualitative domain of strongly stable function between from Q to R . The exponential object $Q \rightarrow R$ is $\langle R^Q, C(Q \rightarrow R) \rangle$ where, for all $F \subseteq R^Q$, for all $X \in C(Q)$, for all pairing $P \subseteq X \times F$, F belongs to $C(Q \rightarrow R)$ iff by assuming $ev(P) = \{f(x) \mid \langle x, f \rangle \in P\}$, the following two conditions hold*

- $ev(P) \in C(R)$
- $\prod ev(P) = (\prod F)(\prod X)$

Example 3.5.2. *Let us consider the qualitative domain $\mathbf{B} \rightarrow \mathbf{B}$ of strongly stable function between booleans. Let $a, b \in \{\mathbf{tt}, \mathbf{ff}\}$ we define some functions in $\mathbf{B} \rightarrow \mathbf{B}$ as follows*

$$b^a = \begin{cases} b & \text{if } x = a \\ \perp & \text{otherwise} \end{cases}$$

We noted \perp the bottom of qualitative domains, corresponding to the empty clique in case of coherence spaces.

We sketch the proof that $F = \{\mathbf{tt}^{\mathbf{tt}}, \mathbf{ff}^{\mathbf{tt}}, \mathbf{tt}^{\mathbf{ff}}\}$ belongs to $C(\mathbf{B} \rightarrow \mathbf{B})$. For all $X \in C(\mathbf{B})$ there are two cases.¹

- If $\emptyset \in X$ then for every pairing $P \subseteq X \times F$, so we have $\emptyset \in ev(P)$, since the considered functions are strict. Thus $ev(P) \in C(\mathbf{B})$ (see Example 3.5.1).
- If $\emptyset \notin X$ then X is singleton (see Example 3.5.1) and easily we can check that for every pairing $P \subseteq X \times F$, so we have $ev(P) \in C(\mathbf{B})$.

The second condition of Definition 3.5.5, about the commutation of greatest lower bound, can be easily checked to be true.

¹For sake of conciseness, we use the exponential notation in order to denote pairs of booleans: exponent (base) corresponds respectively to first (second) projection.

$$\text{bGor}_{k_1, k_2, k_3, k_4}^2(f_1, f_2, f_3) = \begin{cases} k_1 & \text{if } f_1(\mathbf{tt}) = \mathbf{tt}, f_2(\mathbf{ff}) = \mathbf{tt}, f_3(\mathbf{tt}) = \mathbf{ff} \\ k_2 & \text{if } f_1(\mathbf{tt}) = \mathbf{ff}, f_2(\mathbf{tt}) = \mathbf{tt}, f_3(\mathbf{ff}) = \mathbf{tt} \\ k_3 & \text{if } f_1(\mathbf{ff}) = \mathbf{tt}, f_2(\mathbf{tt}) = \mathbf{ff}, f_3(\mathbf{tt}) = \mathbf{tt} \\ k_4 & \text{if } f_1(\mathbf{ff}) = \mathbf{ff}, f_2(\mathbf{ff}) = \mathbf{ff}, f_3(\mathbf{ff}) = \mathbf{ff} \\ \perp & \text{otherwise} \end{cases}$$

where $k_1, k_2, k_3, k_4 \in \{\mathbf{tt}, \mathbf{ff}\}$

Table 3.4: A boolean version of a Second Order Gustave-OR.

Second Order Gustave Or is not strongly stable

For sake of simplicity, we prove simply that a boolean version of Gor^2 is not a strongly stable function and we replace everywhere $\underline{0}$ by \mathbf{tt} and $\text{succ } \underline{n}$ by \mathbf{ff} , respecting our notations.

Let us consider the second-order Gustave-Or defined in Table 3.4. It is straightforward to check that the definition of bGor^2 is well given and it is a boolean generalisation of a Second Order Gustave Or.

We prove that $\text{bGor}_{k_1, k_2, k_3, k_4}^2$ is not a strongly stable function (see Definition 3.5.3) for all $k_1, k_2, k_3, k_4 \in \{\mathbf{tt}, \mathbf{ff}\}$. Let us consider the functions in Example 3.5.2 above. We take 3 elements of

$$(\mathbf{B} \multimap \mathbf{B}) \times (\mathbf{B} \multimap \mathbf{B}) \times (\mathbf{B} \multimap \mathbf{B}),$$

namely $t_1 = \langle \mathbf{tt}^{\mathbf{tt}}, \mathbf{tt}^{\mathbf{ff}}, \mathbf{ff}^{\mathbf{tt}} \rangle,$
 $t_2 = \langle \mathbf{ff}^{\mathbf{tt}}, \mathbf{tt}^{\mathbf{tt}}, \mathbf{tt}^{\mathbf{ff}} \rangle,$
 $t_3 = \langle \mathbf{tt}^{\mathbf{ff}}, \mathbf{ff}^{\mathbf{tt}}, \mathbf{tt}^{\mathbf{tt}} \rangle.$

The set $I = \{t_1, t_2, t_3\}$ is in the coherence set of the domain corresponding to $(\mathbf{B} \multimap \mathbf{B}) \times (\mathbf{B} \multimap \mathbf{B}) \times (\mathbf{B} \multimap \mathbf{B})$, since the projections are in the coherence set of $\mathbf{B} \multimap \mathbf{B}$. We have two cases.

- If $k_1 = k_2 = k_3$ then $\prod \text{bGor}_{k_1, k_2, k_3, k_4}^2(I) = k_1$, but $\text{bGor}_{k_1, k_2, k_3, k_4}^2(\prod I) = \text{bGor}_{k_1, k_2, k_3, k_4}^2(\emptyset) = \emptyset$.
- Otherwise, there exists $i, j \in \{1, 2, 3\}$ such that $k_i \neq k_j$, thus

$$\text{bGor}_{k_1, k_2, k_3, k_4}^2(I) \notin C(\mathbf{B})$$

since

$$\{k_i, k_j\} \subseteq \text{bGor}_{k_1, k_2, k_3, k_4}^2(I)$$

and $\emptyset \notin \text{bGor}_{k_1, k_2, k_3, k_4}^2(I)$.

Therefore $\text{bGor}_{k_1, k_2, k_3, k_4}^2$ is not strongly stable, for all k_1, k_2, k_3, k_4 .

$$\boxed{\frac{\mathbb{M}(\lambda x'. \text{if}(\overbrace{\text{pred} \dots \text{pred}}^{\underline{k}} x) \underline{k} \Omega') \Downarrow \underline{n} \quad [\underline{n}, \underline{k}] \Downarrow \underline{r}}{\text{which? } \mathbb{M}^{(l \rightarrow o) \rightarrow o} \Downarrow \underline{r}}} \quad (w^{\underline{k}})$$

Table 3.5: The which? operator

3.5.2 The which? -operator

In this section, we introduce a further operator to the language and we show its independence. Such an operator is called *which?*. In order to define its operational semantics, we need pairing and projections operators on natural numbers. If $m, n \in \mathbb{N}$ then $2^m(2n + 1) - 1$ is our pairing function. Projections from $z \in \mathbb{N}$ can be defined by functions

$$\begin{aligned} \min_{x \leq z}[(\exists y)_{\leq z}(z = \langle x, y \rangle)] \\ \min_{y \leq z}[(\exists x)_{\leq z}(z = \langle x, y \rangle)]. \end{aligned}$$

Such functions induce a bijection, details can be found either, in p.41,p.73 of [Cutland, 1980] or in p.47 of [Davis and Weyuker, 1983].

Notation 3.5.1. We write $[\underline{n}, \underline{m}]$ as an abbreviation, for the application of a term coding the pairing function to \underline{n} and \underline{m} .

You can easily verify that

$$\text{Sum} \equiv \mu F. \lambda x' y'. \text{if } x \ y \ (F(\text{pred } x)(\text{succ } y))$$

defines the addition,

$$\text{Prod} \equiv \mu F. \lambda x' y'. \text{if } x \ \underline{0} \ (\text{Sum } y \ (F(\text{pred } x) \ y))$$

defines the multiplication and

$$\text{Exp2} \equiv \mu F. \lambda x'. \text{if } x \ \underline{1} \ (\text{Prod } \underline{2} \ (F(\text{pred } x)))$$

defines the exponentiation of base 2. Hence,

$$\lambda x' y'. \text{pred}(\text{Prod}(\text{Exp2 } x)(\text{succ}(\text{Prod } \underline{2} \ y)))$$

defines the pairing function as syntactic sugar.

The operational semantics of *which?* is given in Table 3.5. The evaluation of *which?* \mathbb{M} is a pattern for infinite rules, similarly to rules for the operator \exists of Plotkin [Plotkin, 1977]. Note that *which?* $\mathbb{M} \Downarrow$ if and only if $\mathbb{M}(\lambda x'. x) \Downarrow$; moreover, linearity assures that \mathbb{M} applies its argument $\lambda x'. x$ exactly to a unique numeral \underline{k} , in Table 3.2. A deterministic evaluation can be formalised likewise that of the operator *strict?* presented in [Paolini, 2006].

The following three Subsections will discuss in a more detail such novel operator. In the first one, we give some programming examples of the use of *which?*. In the second one, we discuss how to add *which?* to a PCF-like language and we compare it to other operators introduced in the languages StPCF and SPCF . In the third one, we show that *which?* is not syntactic sugar for $\mathcal{S}\ell\text{PCF}$.

Programming with `which?`

The operator `which?` of $\mathcal{S}\ell\text{PCF}$ plays a role similar to `strict?` of StPCF [Paolini, 2006]. Likewise, we can use `which?` to define some useful primitives in $\mathcal{S}\ell\text{PCF}$ in order to deal with a simple kind of exceptions in a linear setting.

First of all, let us introduce the following operator called `@wh?` : $((\iota \multimap \iota) \multimap \iota) \multimap (\iota \multimap \iota) \multimap \iota$ with the following operational semantics,

$$\frac{\mathbb{M}(\lambda x'. \text{if}(\overbrace{\text{pred} \dots \text{pred } x}^k) (\mathbb{N} \underline{k}) \Omega') \Downarrow \underline{n} \quad [\underline{n}, \underline{k}] \Downarrow \underline{r}}{\text{@wh? } \mathbb{M}^{(\iota \multimap \iota) \multimap \iota} \mathbb{N}^{\iota \multimap \iota} \Downarrow \underline{r}} \quad (\text{@wh? }^k)$$

Note that `@wh? M N` \Downarrow if and only if $\mathbb{M} \mathbb{N} \Downarrow$. This control operator gives back the result of the application of the functional \mathbb{M} to the function \mathbb{N} , together with the unique numeral passed by \mathbb{M} as argument to \mathbb{N} .

Example. Let us write a term `Tryn` of type $((\iota \multimap \iota) \multimap \iota) \multimap (\iota \multimap \iota) \multimap \iota$ which takes in input a functional $F^{(\iota \multimap \iota) \multimap \iota}$ and a function $f^{\iota \multimap \iota}$. It tries to evaluate the term $F f$ and it raise an exception (giving back $\underline{0}$) if during the evaluation, F observes f on a particular number \underline{n} . Otherwise it gives back the successor of the result of the evaluation. In the following we denote with $\pi_1^{\iota \multimap \iota}$ and $\pi_2^{\iota \multimap \iota}$ the two terms coding the projections of the pairing function we use. The following term

$$\lambda F^{(\iota \multimap \iota) \multimap \iota}. \lambda f^{\iota \multimap \iota}. (\lambda x'. \text{if}(\pi_2^{\iota \multimap \iota} x) \doteq \underline{n} \ (\underline{0}) \ (\text{succ } \pi_1 x)) (\text{@wh? } F f)$$

codes `Tryn`.

If $\mathbb{M}^{(\iota \multimap \iota) \multimap \iota}$ is a term then `@wh? M (λx'.x)` behaves in the same way as `which? M`. So `@wh?` may appear more powerful than `which?`. However the term

$$\lambda f^{\iota \multimap \iota}. \lambda g^{\iota \multimap \iota}. \text{which?} (\lambda h^{\iota \multimap \iota}. f(\lambda x'. g(h x)))$$

behaves in the same way as `@wh?`. Therefore, by replacing `which?` with `@wh?` in $\mathcal{S}\ell\text{PCF}$, we program the same linear functions.

Adding `which?` to PCF-like languages

If we add `which?` together with the operational rule of Table 3.5 to PCF, then `which?` may return a non-deterministic result when applied to non-linear functions. For instance, if $R = \lambda f^{\iota \multimap \iota}. \underline{5}$ then both `which? R` $\Downarrow [\underline{0}, \underline{5}]$ and `which? R` $\Downarrow [\underline{1}, \underline{5}]$. However, we can add the operator `dwh?` to PCF together the same operational rule of `which?` in Table 3.5 with the following additional hypothesis,

$$\forall h \leq k \quad \mathbb{M}(\lambda x'. \text{if}(\overbrace{\text{pred} \dots \text{pred } x}^h) \underline{k} \ \Omega') \Uparrow$$

This hypothesis asks simply, for the minimum k satisfying the rule of `which?`, so the determinism is recovered. Such an hypothesis is inane for `which?` in $\mathcal{S}\ell\text{PCF}$, in the sense that, if we add `dwh?` to $\mathcal{S}\ell\text{PCF}$ we obtain exactly `which?`. Unfortunately, the given evaluation rule for `dwh?` is not effective. However, an effective evaluation of `dwh?` can be formalised likewise that of the operator `strict?` [Paolini, 2006].

It is easy to verify that `dwh?` can be programmed in $\text{PCF} + \text{strict?}$ and so in StPCF [Paolini, 2006]. Recall that `strict?` ^{$(l \rightarrow l) \rightarrow l$} $\mathbb{M}^{l \rightarrow l}$ checks whether the function $\mathbb{M}^{l \rightarrow l}$ is strict. If $\mathbb{M}^{l \rightarrow l}$ is strict then the evaluation of `strict? M` returns $\underline{0}$, while if $\mathbb{M}^{l \rightarrow l}$ is not strict but $\mathbb{M}^{l \rightarrow l} \underline{0}$ converges then `strict? M` returns $\underline{1}$. Recall that `dwh? M` \Downarrow if and only if $\mathbb{M}(\lambda x^l. x)$ \Downarrow . Let \doteq be as in Section 3.4.4, it is easy to check that

$$T = \lambda f^{(l \rightarrow l) \rightarrow l} x^l. \text{strict?}^{l \rightarrow l} (\lambda y^l. f(\lambda z^l. \text{if } (x \doteq z) z (\text{if } y z z)))$$

tests whether f passes x^l as argument of $\lambda z^l. z$. Thus `dwh?` can be translated in the following term of $\text{PCF} + \text{strict?}$.

$$\lambda f^{(l \rightarrow l) \rightarrow l}. (\mu F^{l \rightarrow l} \lambda x^l. \text{if } (T f x) (\Gamma x, f(\lambda z^l. \text{if } (z \doteq x) z \Omega^l) \Uparrow) (F (\text{succ}(x)))) \underline{0}$$

Note that `dwh?` is stable and also strongly stable, since $\text{PCF} + \text{strict?}$ is (see [Paolini, 2006]).

We can also compare `dwh?` with the catch operator, which is present in the language SPCF [Cartwright et al., 1994]. `catch xl` in \mathbb{M}^l binds the occurrences of x in \mathbb{M} . Informally, it asks the evaluation of \mathbb{M} , if the computation of \mathbb{M}^l asks the evaluation of the variable x^l then the computation of `catch xl` in \mathbb{M}^l terminates giving $\underline{0}$. Otherwise, if the computation of \mathbb{M}^l terminates on a numeral \underline{n} without using x , then `catch xl` in \mathbb{M}^l returns $\text{succ}(\underline{n})$. We can write a term of SPCF having the same behaviour of `dwh?` using `catch`, in the following way. Consider the following term

$$T' = \text{catch } y \text{ in } f(\lambda z^l. \text{if } (x \doteq z) z y)$$

T' evaluates to $\underline{0}$ (i.e. causes an error) if f does not pass x as argument of $\lambda z^l. z$. Otherwise it evaluates to the successor of the result of the application of f to $\lambda z^l. z$. So the term

$$\lambda f^{(l \rightarrow l) \rightarrow l}. (\mu F^{l \rightarrow l} \lambda x^l. \text{if } T' (F \text{succ}(x)) (\Gamma x, f(\lambda z^l. \text{if } (z \doteq x) z \Omega^l) \Uparrow)) \underline{0}$$

behaves like `dwh?`.

We can conclude that `which?` can be used in a fruitful way, in order to manage some kind of “linear” exceptions in $\mathcal{S}\ell\text{PCF}$. Note that `which?` is able to perform observations that are subtly linear: in fact `which? M` gives back at once the parameter passed by \mathbb{M} during its evaluation with the identity function together with the result of the evaluation, in a unique evaluation of the such an application. Thus, the evaluation is done by respecting operational linearity.

Semantics of `which?`

We will show that `which?` can be interpreted in a linear function, so as a corollary, we will obtain that `which?` is also a Scott-continuous and stable function (see [Asperti and Longo, 1991, pp. 29]). We will interpret `which?` as the linear function $\text{which} : ((\mathbb{N} \multimap \mathbb{N}) \multimap \mathbb{N}) \multimap \mathbb{N}$ having trace

$$\text{tr}(\text{which}) = \{((n, n), m), \lceil n, m \rceil \mid n, m \in \mathbb{N}\}$$

With an abuse of notation, above we denoted $\lceil n, k \rceil$ the number coding the pair (n, k) according to the considered pairing function. In a functional way, we define $\text{which} : Cl((\mathbb{N} \multimap \mathbb{N}) \multimap \mathbb{N}) \rightarrow Cl(\mathbb{N})$ to be the function

$$\text{which}(f) = \begin{cases} \lceil n, k \rceil & \text{if } \exists n. f(\{(n, n)\}) = \{k\} \\ \emptyset & \text{otherwise} \end{cases}$$

Proposition 3.5.1. *Let $f : Cl(\mathbb{N} \multimap \mathbb{N}) \rightarrow Cl(\mathbb{N})$ be a linear function. If $f(\text{tr}(id_{\mathbb{N}})) \neq \emptyset$ if and only if there exists a unique $n \in \mathbb{N}$ such that $f(\{(n, n)\}) = f(\text{tr}(id_{\mathbb{N}})) \neq \emptyset$.*

Proof. The \Leftarrow direction follows by monotonicity. To prove the \Rightarrow direction, we first prove the existence of such an n . Suppose by contradiction that for all n we have $f(\{(n, n)\}) = \emptyset$. Since $f(\text{tr}(id_{\mathbb{N}})) \neq \emptyset$, by continuity, there must exist a finite clique $x \subseteq_{fin} \text{tr}(id_{\mathbb{N}})$ such that $f(x) = f(\text{tr}(id_{\mathbb{N}})) \neq \emptyset$. x is not empty since $f(\emptyset) = \emptyset$ by linearity, thus $x = \{(n_1, n_1), \dots, (n_k, n_k)\}$. Again by linearity we have $f(x) = f(\{(n_1, n_1)\}) \cup \dots \cup f(\{(n_k, n_k)\}) = \emptyset$. Contradiction. To prove the unicity, suppose there are n_1, n_2 such that $f(\{(n_1, n_1)\}) \neq \emptyset$ and $f(\{(n_2, n_2)\}) \neq \emptyset$. Observe that by monotonicity, it is never the case that $f(\{(n_1, n_1)\}) \neq f(\{(n_2, n_2)\})$. So $f(\{(n_1, n_1)\}) = f(\{(n_2, n_2)\}) = \{k\}$ for some k . By stability we have $f(\{(n_1, n_1)\} \cap \{(n_2, n_2)\}) = \{k\}$. If $n_1 \neq n_2$ we would have $f(\emptyset) = \{k\}$ which leads to the desired contradiction. So we conclude. \square

Observe that, by Proposition 3.5.1, `which` is a well defined linear function.

It is easy to see that, when setting $\mathcal{L}[\llbracket \text{which?} \rrbracket] = \text{curry}(\text{which})$, the denotational semantics is again sound and adequate with respect to the operational semantics of $\mathcal{S}\ell\text{PCF} + \text{which?}$. In the same way, the result of closed full abstraction holds too for such an extension.

Now we ask ourselves if there is a program of $\mathcal{S}\ell\text{PCF}$ having the same semantics of `which?`. The answer is negative!

Proposition 3.5.2. *`which?` is not syntactic sugar for $\mathcal{S}\ell\text{PCF}$.*

Proof. If M and N are programs then $M \doteq N$ be defined as in Section 3.4.4. Let

$$\begin{aligned} F_{\mathbf{k}}^\sigma &= \lambda f^{t \rightarrow t}. \text{if } ((f\mathbf{k}) \doteq \mathbf{k}) \ \underline{\mathbf{0}} \ \Omega^t, & M_1 &= \Omega^t, \\ M_2 &= \text{if } \left(((F^{\sigma \rightarrow \sigma} F_{\underline{\mathbf{0}}}^\sigma) \doteq \lceil \underline{\mathbf{0}}, \underline{\mathbf{0}} \rceil) \text{ and } ((F^{\sigma \rightarrow \sigma} F_{\underline{\mathbf{1}}}^\sigma) \doteq \lceil \underline{\mathbf{0}}, \underline{\mathbf{1}} \rceil) \right) \ \underline{\mathbf{9}} \ \Omega^t \end{aligned}$$

be $\mathcal{S}\ell\text{PCF}$ terms. Clearly the evaluation of $M_1[\text{which?}/F]$ should diverge, while $M_2[\text{which?}/F]$ should return $\underline{\mathbf{9}}$, making the two terms operationally different in $\mathcal{S}\ell\text{PCF} + \text{which?}$.

We prove that M_1, M_2 are operationally equivalent in $\mathcal{S}PCF$, using the correctness of the Scott Semantics given in Section 3.3. In the following, we denote with N to be the Scott Domain corresponding to the infinite flat domain of natural numbers and we use the notation introduced in Section 3.3.

Assume there is a strict continuous function $f : \mathbf{1} \rightarrow ((N \multimap N) \multimap N) \multimap N$ such that $C\llbracket M_1 \rrbracket \circ f_{\perp} \circ q_1 \neq C\llbracket M_2 \rrbracket \circ f_{\perp} \circ q_1$. Then $f(\top)(C\llbracket F_0 \rrbracket) = 0$ and $f(\top)(C\llbracket F_1 \rrbracket) = 1$. By monotonicity this implies that $f(\top)(C\llbracket F_0 \rrbracket \sqcup C\llbracket F_1 \rrbracket) = d \neq \perp_N$ and such that $0, 1 \sqsubseteq d$. The existence of such a d would imply $0 = 1$ which leads to the desired contradiction. This concludes the proof. \square

Using a similar argument, we can show that dwh? is not syntactic sugar neither for PCF nor for PCF + por. More details can be found in [Paolini and Piccolo, 2008].

Proposition 3.5.3. *There exists no program in PCF + por (and so in PCF) having the same semantics of dwh? .*

3.5.3 Towards finite definability

In this section we try to address the problem of finite definability, by introducing a novel operator, called GLin and showing that all finite cliques belonging to a coherence space which is the interpretation of a type σ such that $\text{order}(\sigma) \leq 2$ and having at most three arguments. The typing rule for GLin is the following.

$$\frac{\forall i \in \{1, 2, 3\} \Gamma_i^*, \Delta_i^l \vdash M_i : (\iota \multimap \iota) \multimap (\iota \multimap \iota) \multimap (\iota \multimap \iota) \multimap \iota}{\Gamma_1^*, \Gamma_2^*, \Gamma_3^*, \Delta_1^l, \Delta_2^l, \Delta_3^l \vdash \text{GLin } M_1 M_2 M_3 : (\iota \multimap \iota) \multimap (\iota \multimap \iota) \multimap (\iota \multimap \iota) \multimap \iota}$$

under the hypothesis that all involved basis are pairwise disjoint.

Given a closed term $F^{\iota \multimap \iota}$, we write $F \downarrow_{\underline{n}} = \lambda x^{\iota}. \text{li f } (x \doteq \underline{n}) F \underline{n} \Omega^{\iota}$. The evaluation rules of GLin is the following.

$$\frac{M_1(F_1 \downarrow_{\underline{n}_1})(F_2 \downarrow_{\underline{n}_2})(F_3 \downarrow_{\underline{n}_3}) \Downarrow \underline{0} \quad M_2(F_1 \downarrow_{\underline{n}_1})(F_2 \downarrow_{\underline{n}_2})(F_3 \downarrow_{\underline{n}_3}) \Downarrow \underline{m+1}}{\text{GLin } M_1 M_2 M_3 F_1 F_2 F_3 \Downarrow \underline{m}}$$

$$\frac{M_2(F_1 \downarrow_{\underline{n}_1})(F_2 \downarrow_{\underline{n}_2})(F_3 \downarrow_{\underline{n}_3}) \Downarrow \underline{0} \quad M_3(F_1 \downarrow_{\underline{n}_1})(F_2 \downarrow_{\underline{n}_2})(F_3 \downarrow_{\underline{n}_3}) \Downarrow \underline{m+1}}{\text{GLin } M_1 M_2 M_3 F_1 F_2 F_3 \Downarrow \underline{m}}$$

$$\frac{M_3(F_1 \downarrow_{\underline{n}_1})(F_2 \downarrow_{\underline{n}_2})(F_3 \downarrow_{\underline{n}_3}) \Downarrow \underline{0} \quad M_1(F_1 \downarrow_{\underline{n}_1})(F_2 \downarrow_{\underline{n}_2})(F_3 \downarrow_{\underline{n}_3}) \Downarrow \underline{m+1}}{\text{GLin } M_1 M_2 M_3 F_1 F_2 F_3 \Downarrow \underline{m}}$$

We denote with $\alpha = (\iota \multimap \iota) \multimap (\iota \multimap \iota) \multimap (\iota \multimap \iota) \multimap \iota$. We know that given four closed terms $M^{\alpha}, F_1^{\iota \multimap \iota}, F_2^{\iota \multimap \iota}, F_3^{\iota \multimap \iota}$, if $M F_1 F_2 F_3$ converges then there exists unique $\underline{n}_1, \underline{n}_2, \underline{n}_3$ such that $M(F_1 \downarrow_{\underline{n}_1})(F_2 \downarrow_{\underline{n}_2})(F_3 \downarrow_{\underline{n}_3})$ converges. Thus the operator GLin checks in a circular

way the evaluation of three terms $M_1^\alpha, M_2^\alpha, M_3^\alpha$ applied to the same restriction of their arguments.

To define the semantics of this operator, in the category **LinCoh**, we will define $GLin : Cl(!\mathcal{L}[\alpha] \otimes !\mathcal{L}[\alpha] \otimes !\mathcal{L}[\alpha]) \otimes (\mathbf{N} \multimap \mathbf{N}) \otimes (\mathbf{N} \multimap \mathbf{N}) \otimes (\mathbf{N} \multimap \mathbf{N}) \rightarrow Cl(\mathbf{N})$ to be a function having the following trace.

$$\left\{ \left(\left(\begin{array}{l} \{(n_1, m_1), (n_2, m_2), (n_3, m_3), 0\}, \\ \{(n_1, m_1), (n_2, m_2), (n_3, m_3), m+1\}, \\ \emptyset, \end{array} \right), (n_1, m_1), (n_2, m_2), (n_3, m_3), m \mid n_1, n_2, n_3, m_1, m_2, m_3, m \in \mathbb{N} \right) \cup \right. \\ \left. \left(\left(\begin{array}{l} \emptyset \\ \{(n_1, m_1), (n_2, m_2), (n_3, m_3), 0\}, \\ \{(n_1, m_1), (n_2, m_2), (n_3, m_3), m+1\}, \end{array} \right), (n_1, m_1), (n_2, m_2), (n_3, m_3), m \mid n_1, n_2, n_3, m_1, m_2, m_3, m \in \mathbb{N} \right) \cup \right. \\ \left. \left(\left(\begin{array}{l} \{(n_1, m_1), (n_2, m_2), (n_3, m_3), m+1\}, \\ \emptyset, \\ \{(n_1, m_1), (n_2, m_2), (n_3, m_3), 0\} \end{array} \right), (n_1, m_1), (n_2, m_2), (n_3, m_3), m \mid n_1, n_2, n_3, m_1, m_2, m_3, m \in \mathbb{N} \right) \right\}$$

Lemma 3.5.1. $\text{tr}(GLin) \in Cl(!\mathcal{L}[\alpha] \otimes !\mathcal{L}[\alpha] \otimes !\mathcal{L}[\alpha]) \otimes (\mathbf{N} \multimap \mathbf{N}) \otimes (\mathbf{N} \multimap \mathbf{N}) \otimes (\mathbf{N} \multimap \mathbf{N}) \multimap \mathbf{N}$

Proof. The proof follows by a straightforward case analysis. \square

We set

$$\mathcal{L}[\Gamma_1^*, \Gamma_2^*, \Gamma_3^*, \Delta_1^l, \Delta_2^l, \Delta_3^l \vdash \text{GLin } M_1 M_2 M_3 : \alpha] = GLin \circ \left(\begin{array}{l} (!\mathcal{L}[\Gamma_1^*, \Delta_1^l \vdash M_1 : \alpha] \circ q \circ (\delta_{\Gamma_1} \otimes p_{\Delta_1})) \otimes \\ (!\mathcal{L}[\Gamma_2^*, \Delta_2^l \vdash M_2 : \alpha] \circ q \circ (\delta_{\Gamma_2} \otimes p_{\Delta_2})) \otimes \\ (!\mathcal{L}[\Gamma_3^*, \Delta_3^l \vdash M_3 : \alpha] \circ q \circ (\delta_{\Gamma_3} \otimes p_{\Delta_3})) \end{array} \right)$$

Example. Let us give the term defining the finite clique of the function $\overset{2}{Gor}$, defined in the previous section. We will use a case construct

$$\text{case } \begin{array}{l} [x = \underline{0}] M \\ [x = \underline{1}] N \end{array}$$

defined as $\ell \text{if } (x \doteq \underline{0}) M (\ell \text{if } x \doteq \underline{1} N \Omega^t)$.

Then $\overset{2}{Gor} = \text{GLin } M_1 M_2 M_3$ where M_1, M_2, M_3 are such that

- $\text{tr}(\mathcal{L}[\![M_1]\!]) = \{((0, 0), (0, 1), (1, 0), 0), ((1, 0), (0, 0), (0, 1), 3)\}$
- $\text{tr}(\mathcal{L}[\![M_2]\!]) = \{((0, 0), (0, 1), (1, 0), 2), ((0, 1), (1, 0), (0, 0), 0), ((1, 1), (1, 1), (1, 1), 0)\}$
- $\text{tr}(\mathcal{L}[\![M_3]\!]) = \{((1, 0), (0, 0), (0, 1), 0), ((0, 1), (1, 0), (0, 0), 4), ((1, 1), (1, 1), (1, 1), 5)\}$

The term M_1 can be easily defined as follows

$$M_1 = \lambda f_1^{l-\circ l} f_2^{l-\circ l} f_3^{l-\circ l} \cdot \left(\lambda w^t \cdot \text{case } \begin{array}{l} [w = \underline{0}] \ell \text{if } (f_1 \underline{1} \doteq \underline{0} \text{ and } f_3 \underline{0} \doteq \underline{1}) \exists \Omega^t \\ [w = \underline{1}] \ell \text{if } (f_1 \underline{0} \doteq \underline{0} \text{ and } f_3 \underline{1} \doteq \underline{0}) \underline{0} \Omega^t \end{array} \right) (f_2 \underline{0})$$

The term M_2 is defined as $\text{GLin } N_1 N_2 N_3$ where

- $N_1 = \lambda f_1^{l \rightarrow l} f_2^{l \rightarrow l} f_3^{l \rightarrow l} \cdot \left(\lambda w^l . \text{case} \begin{array}{l} [w = 0](\text{lif } (f_2 \underline{0} \doteq \underline{1} \text{ and } f_3 \underline{1} \doteq \underline{0}) \underline{0} \Omega^l) \\ [w = 1](\text{lif } (f_2 \underline{1} \doteq \underline{0} \text{ and } f_3 \underline{0} \doteq \underline{0}) \underline{1} \Omega^l) \end{array} \right) (f_1 \underline{0})$
- $N_2 = \lambda f_1^{l \rightarrow l} f_2^{l \rightarrow l} f_3^{l \rightarrow l} \cdot \left(\lambda w^l . \text{case} \begin{array}{l} [w = 0](\text{lif } (f_1 \underline{0} \doteq \underline{0} \text{ and } f_2 \underline{0} \doteq \underline{1}) \underline{3} \Omega^l) \\ [w = 1](\text{lif } (f_1 \underline{1} \doteq \underline{1} \text{ and } f_2 \underline{1} \doteq \underline{1}) \underline{0} \Omega^l) \end{array} \right) (f_3 \underline{1})$
- $N_3 = \lambda f_1^{l \rightarrow l} f_2^{l \rightarrow l} f_3^{l \rightarrow l} \cdot \left(\lambda w^l . \text{case} \begin{array}{l} [w = 0](\text{lif } (f_1 \underline{0} \doteq \underline{1} \text{ and } f_3 \underline{0} \doteq \underline{0}) \underline{0} \Omega^l) \\ [w = 1](\text{lif } (f_1 \underline{1} \doteq \underline{1} \text{ and } f_3 \underline{1} \doteq \underline{1}) \underline{1} \Omega^l) \end{array} \right) (f_2 \underline{1})$

The term M_3 can be defined as $\text{GLin } P_1 P_2 P_3$, where

- $P_1 = \lambda f_1^{l \rightarrow l} f_2^{l \rightarrow l} f_3^{l \rightarrow l} \cdot \left(\lambda w^l . \text{case} \begin{array}{l} [w = 0](\text{lif } (f_1 \underline{0} \doteq \underline{1} \text{ and } f_2 \underline{1} \doteq \underline{0}) \underline{5} \Omega^l) \\ [w = 1](\text{lif } (f_1 \underline{1} \doteq \underline{0} \text{ and } f_2 \underline{0} \doteq \underline{0}) \underline{0} \Omega^l) \end{array} \right) (f_3 \underline{0})$
- $P_2 = \lambda f_1^{l \rightarrow l} f_2^{l \rightarrow l} f_3^{l \rightarrow l} \cdot \left(\lambda w^l . \text{case} \begin{array}{l} [w = 0](\text{lif } (f_2 \underline{0} \doteq \underline{0} \text{ and } f_3 \underline{0} \doteq \underline{1}) \underline{1} \Omega^l) \\ [w = 1](\text{lif } (f_2 \underline{1} \doteq \underline{1} \text{ and } f_3 \underline{1} \doteq \underline{1}) \underline{0} \Omega^l) \end{array} \right) (f_1 \underline{1})$
- $P_3 = \lambda f_1^{l \rightarrow l} f_2^{l \rightarrow l} f_3^{l \rightarrow l} \cdot \left(\lambda w^l . \text{case} \begin{array}{l} [w = 0](\text{lif } (f_1 \underline{0} \doteq \underline{1} \text{ and } f_3 \underline{0} \doteq \underline{0}) \underline{0} \Omega^l) \\ [w = 1](\text{lif } (f_1 \underline{1} \doteq \underline{1} \text{ and } f_3 \underline{1} \doteq \underline{1}) \underline{6} \Omega^l) \end{array} \right) (f_2 \underline{1})$

Using GLin we can define all finite cliques belonging to the coherence space $\mathcal{L}[\alpha]$.

Theorem 3.5.1. *Let $x \in \text{Cl}_{fin}(\mathcal{L}[\alpha])$. Then there is a closed term M^α such that $\mathcal{L}[M] = x$.*

Proof. The proof is by induction on $|x|$. The case $|x| = 0, 1$ are easy. If $|x| = 2$ then $x = \{((a_1^1, b_1^1), (a_2^2, b_2^2), (a_3^3, b_3^3), c_1), ((a_1^1, b_1^1), (a_2^2, b_2^2), (a_3^3, b_3^3), c_2)\}$. Hence there is $h \in \{1, 2, 3\}$ such that $(a_1^h, b_1^h) \smile (a_2^h, b_2^h)$ which means that $a_1^h = a_2^h$ and $b_1^h \neq b_2^h$. Thus we have that the term

$$\lambda f_1, f_2, f_3 \cdot \left(\lambda w . \text{case} \begin{array}{l} [w = \underline{b_1^h}](\text{lif } (f_{h+1} \underline{a_1^{h+1}} \doteq \underline{b_1^{h+1}} \text{ and } f_{h+2} \underline{a_1^{h+2}} \doteq \underline{b_1^{h+2}}) \underline{c_1} \Omega^l) \\ [w = \underline{b_2^h}](\text{lif } (f_{h+1} \underline{a_2^{h+1}} \doteq \underline{b_2^{h+1}} \text{ and } f_{h+2} \underline{a_2^{h+2}} \doteq \underline{b_2^{h+2}}) \underline{c_2} \Omega^l) \end{array} \right) (f_h \underline{a_1^h})$$

defines the clique x . Observe that the operation of sum we used above is the sum defined in the finite field $\mathbb{Z}_3 = \{1, 2, 3\}$.

We develop now the case $|x| > 2$. Let $x = \{d_1, \dots, d_n\}$ with $d_i = ((a_i^1, b_i^1), (a_i^2, b_i^2), (a_i^3, b_i^3), c_i)$. We denote with $d_i[c]$ the token $((a_i^1, b_i^1), (a_i^2, b_i^2), (a_i^3, b_i^3), c)$. Let us chose d_i, d_j such that $d_i \smile d_j$: then by construction, there is $h \in \{1, 2, 3\}$ such that $(a_i^h, b_i^h) \smile (a_j^h, b_j^h)$. Let us consider the following cliques:

- $x_1 = \{d_i[0], d_j[c_j + 1]\}$
- $x_2 = \{d_i[c_i + 1]\} \cup \{d_k[0] \mid d_k \in x, d_k \neq d_i, d_j\}$
- $x_3 = \{d_j[0]\} \cup \{d_k[c_k + 1] \mid d_k \in x, d_k \neq d_i, d_j\}$

By induction we have that there are terms M_1, M_2, M_3 defining the above cliques. Thus the term $\text{GLin } M_1 M_2 M_3$ defines the clique x . \square

We conclude the section with some conjectures. We think that, by using a family of operators like GLin , we can define all finite cliques belonging to a coherence space $\llbracket \sigma \rrbracket$, where $\text{order}(\sigma) = 2$. Moreover, we conjecture that $\mathcal{S}\ell\text{PCF} + \text{which?}$ plus the family of parallel operators above mentioned is the language being fully abstract for the category LinCoh .

3.6 Conclusions

Denotational semantics is usually proposed as a tool to study the equivalence of programs. However, a further use of denotational models is as an abstract tool, assisting us in the comparison and analysis of whole programming languages endowed with different type-respecting computational power (such approach, already known to Kleene, was recently rediscovered in [Longley, 2002] and pursued in [Paolini, 2006]).

Definition 3.6.1. *A model is universal for a language when every effective element ² (of domains the interpretation of types) is definable by a closed term of the language [Meyer, 1988].*

Linear functions between Coherence Spaces are also Scott-continuous with respect to extensional order, see [Asperti and Longo, 1991, pp. 29]. Before discussing the possible linear extensions of $\mathcal{S}\ell\text{PCF}$, we considered in previous section, we start by considering some well-known extensions of PCF.

It has been proved in [Plotkin, 1977] that Scott-continuous domains are fully abstract and universal for PCF^{++} , namely PCF extended with a parallel conditional pif and an existential operator \exists .

It has been shown that stable domains (dI-domains, qualitative domains, coherence spaces) give a fully abstract model [Paolini, 2006] for StPCF , a syntactical extension of PCF. The language StPCF is obtained by extending PCF with two operators: gor and strict? . gor corresponds to a Gustave-like or function, while strict? corresponds to a non extensional-monotone function.

However strict? does not respect the Scott-continuity, while pif and \exists do not respect the stability. Whence, they cannot belong to our language. On the other hand, gor respect both Scott-continuity and stability, but it's not strict, and therefore it's not linear.

In [Longley, 2002] PCF has been extended with the operator H . $\text{PCF} + \text{H}$ is universal for the strongly stable model [Bucciarelli and Ehrhard, 1991]. Note that strict? is strongly stable [Paolini, 2006], thus it can be defined in $\text{PCF} + \text{H}$. Thus H does not respect the extensionality and, then, the linearity of our model.

In [Longley, 2002, Section 11, page 77], John Longley noted that there are seemingly natural incomparable notions of higher-type computability. In contrast with the Church's thesis, there is no a maximum "higher-type computational formalisms" such that there does not exist a more generous "higher-type computational formal

²an element corresponding to a function being effectively computable

system" that subsumes both of them. The results of this chapter give us some further interesting pieces of this jigsaw puzzle.

A *partial type structure* (PST) \mathcal{T} consists of:

- a set \mathcal{T}^σ for each type σ and in particular \mathcal{T}^1 is the flat poset of natural number;
- for each σ, τ , a total "application function" $\bullet : \mathcal{T}^{\sigma \rightarrow \tau} \times \mathcal{T}^\sigma \rightarrow \mathcal{T}^\tau$.

A partial type structure \mathcal{T} is *extensional* (EPTS) if, for all type σ, τ and for all $f, g \in \mathcal{T}^{\sigma \rightarrow \tau}$, we have that

$$\forall x \in \mathcal{T}^\sigma. f \bullet x = g \bullet x \Rightarrow f = g$$

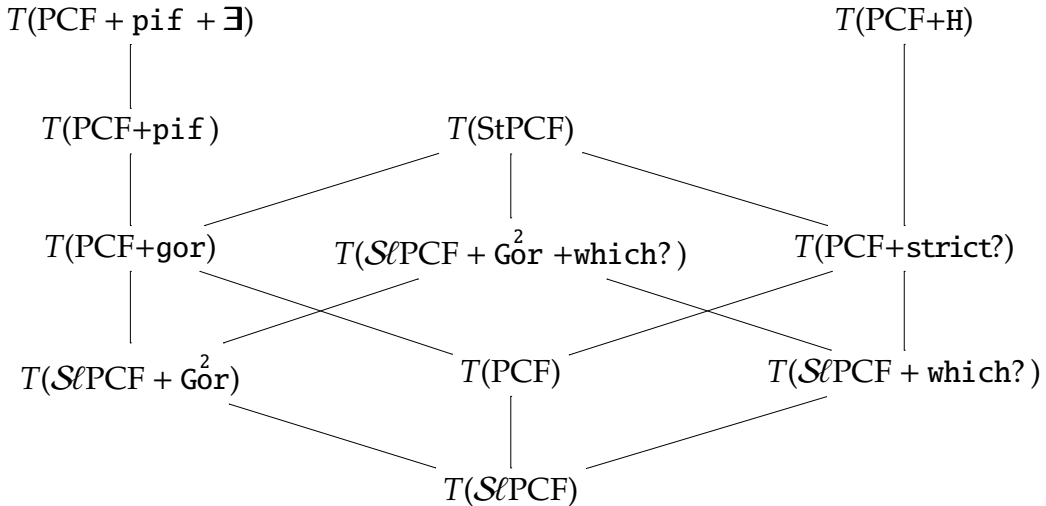
Let \mathcal{T}, \mathcal{U} be EPTSs. A *simulation* $s : \mathcal{T} \rightarrow \mathcal{U}$ consists of a total relation $s^\sigma \subseteq \mathcal{T}^\sigma \rightarrow \mathcal{U}^\sigma$, such that s^1 is the identity relation on the flat poset of natural number, while for any $f \in \mathcal{T}^{\sigma \rightarrow \tau}, g \in \mathcal{U}^{\sigma \rightarrow \tau}, x \in \mathcal{T}^\sigma, y \in \mathcal{U}^\tau$ we have

$$s^{\sigma \rightarrow \tau}(f, g) \text{ and } s^\sigma(x, y) \text{ imply } s^\tau(f \bullet x, g \bullet y)$$

If there is a simulation $s : \mathcal{T} \rightarrow \mathcal{U}$ we write $\mathcal{T} \leq \mathcal{U}$.

It is clear that EPTSs and simulations form a category. It is also easy to see that the only simulation $\mathcal{T} \rightarrow \mathcal{T}$ is the identity. So the relation \leq is a partial order on EPTSs. If \mathcal{L} is a programming language then $T(\mathcal{L})$ denotes the type structure corresponding to the term-model of \mathcal{L} built on its operational equivalence. A notion of *effective extensional partial type structure* has been also formalised in [Longley, 2002, Definition 11.2].

We complete the graphical representations of the relationship among type structures generated by PCF-like languages appearing in [Paolini, 2006] with the the language $\mathcal{S}\ell\text{PCF}$ together with its extensions with Gor^2 and which? .



Longley showed that $T(\text{PCF} + \text{pif} + \exists)$ is a maximal effective type structure and also that $T(\text{PCF} + \text{H})$ is a maximal effective type structure. Observe that, in particular $T(\text{PCF} + \text{pif})$ and $T(\text{PCF} + \text{strict?})$ are incomparable, in the sense that $T(\text{PCF} + \text{pif} +$

strict?) is not an effective type structure (see [Paolini, 2006] for a detailed proof of this). The question about the maximality of StPCF remains still open.

Further questions about $\mathcal{S}\ell\text{PCF}$ remain open. We ask if there exists a linear extension of $\mathcal{S}\ell\text{PCF}$ generating an other maximal effective type structures. In particular, we conjecture that $\mathcal{S}\ell\text{PCF} + \text{whi ch?}$ enjoys universality with respect to the linear strongly stable model. We are also working on the language describing linear continuous functions between Scott Domains³.

Some exploration can be done also on greatest lower bounds of effective EPTSs; this question arises for a well-known open problem on what is the language generating the effective type structure being the greatest lower bound of $T(\text{PCF}^{++})$ and $T(\text{PCF} + \text{H})$. We know that this language contains PCF, but Longley shows in [Longley, 2002] that it strictly contains PCF; in fact he noted that Curien's Third Counterexample [Curien, 1986] is definable in both PCF^{++} and $\text{PCF} + \text{H}$ but not in PCF.

³Let D_1, D_2 two Scott Domains. A function $f : D_1 \rightarrow D_2$ is linear when $f(\bigsqcup X) = \bigsqcup f(X)$ for all $X \subseteq D_1$ which are bounded.

4 A Process Model for Linear Programs

ABSTRACT.

We use *linProc* (i.e. a typed process calculus based on the calculus of solos) in order to express computational processes generated by *SLPCF*. We define a faithful translation of *SLPCF* on *linProc* which enables us to process redexes of *SLPCF* in a parallel way. Afterwards, we prove that a suitable observational equivalence between processes is correct w.r.t the operational semantics of *SLPCF*, via our interpretation.

Introduction

Harold Abelson, Gerald Jay and Julie Sussman, in their famous book “Structure and Interpretation of Computer Programs” [Abelson et al., 1985, Ch.1] state:

“We are about to study the idea of a computational process. Computational processes are abstract beings that inhabit computers. As they evolve, processes manipulate other abstract things called data. The evolution of a process is directed by a pattern of rules called a program. People create programs to direct processes. In effect, we conjure the spirits of the computer with our spells.”

In the first chapter, they introduce a programming language and a simple way to describe the *dynamical becoming* of the evaluation of applications of programs to inputs. The dynamic of process evolution is represented by sequences of programs related by means of rewriting rules. Fingerprints of λ -calculus pervade the book and, indeed, we are interested in a model of processes conjured by a typed λ -calculus. Unfortunately, λ -calculus lacks a satisfactory description of *interaction* between processes cooperating, competing and synchronising between themselves. To overcome such limitations, many calculi have been proposed which focus on dynamical aspects of computation with particular regards for interaction, see for instance [Hoare, 1985, Milner, 1989, Sangiorgi and Walker, 2001]. Such calculi are more intensional than λ -calculus, which instead focuses on functional aspects of computation. Although a main motivation for the comparison between lambda and process calculi

has been to study the expressiveness of process calculi, another worthy motivation is to study theories induced by equivalences in the world of processes conjured by programs, see [Milner, 1990, Milner, 1992, Sangiorgi, 1994]. The main results presented in this chapter are set in the latter research-line.

The interaction is the key idea behind the introduction of game semantics for programming language [Abramsky et al., 2000, Hyland and Ong, 2000], more precisely interaction between a program and the environment (where the program itself is intended to be executed). A seminal work exploring correspondences between process-calculi and game semantics has been presented in [Hyland and Ong, 1995], where Hyland-Ong strategies are represented by processes of an appropriately sorted polyadic π -calculus. More recently, such result has been improved by introducing an elaborate type discipline for the π -calculus in [Berger et al., 2001], where the use of linear modalities [Kobayashi et al., 1996, Kobayashi et al., 1999] is crucial. A game-independent process-based language for game strategies has been formalised, where strategies are normalised processes. Programs of PCF can be directly interpreted on such processes in a fully abstract way.

Our purpose is to deepen and advance such explorations by proposing a process-model, namely a syntactical model, built on a suitable process-calculus, inducing a corresponding semantics for $\mathcal{S}\ell$ PCF.

We are convinced that a key aspect of such explorations is linearity in many respects, moreover linearity makes analysis simpler and clearer. Accordingly, we tackle the construction of a process-model for $\mathcal{S}\ell$ PCF, already introduced in the previous chapter. The least full sub-category of coherence spaces, including the infinite flat domain (representing natural numbers) and the coherence spaces representing linear functions between domains in the model itself (by avoiding the use of exponential domain constructors) forms a fully abstract model for such programming language (see previous chapter). In order to build a process model for $\mathcal{S}\ell$ PCF, we introduce ℓ inProc, namely a process calculus based on a typed calculus of Solos [Laneve and Victor, 2003]. The calculus of Solos is a modification of the asynchronous π -calculus where explicit causal dependency is forbidden by avoiding prefixes and binding guards. We give an encoding of $\mathcal{S}\ell$ PCF in ℓ inProc which respects the operational equivalence between programs, i.e. this equivalence does not equate operationally different programs.

We note that the semantic nature of strategies in [Berger et al., 2001, Hyland and Ong, 1995] is explicitly reflected by the use of an infinitary syntax for the π -processes, on which no parallel reductions can be performed. Since we want to study processes induced by programs, our interpretation is actually given on finite processes. Moreover, we introduce some simplifications in the linear typing discipline. Hence, we break with the classical game semantic approaches, for another classical computer science approach: translation of programs (finite terms) of a language in finite terms of another language.

A translation of a calculus into another calculus is *faithful* whenever a reduction in the source calculus can be mimicked by some reductions in the target calculus. The encodings given in [Berger et al., 2001, Hyland and Ong, 1995] impose deterministic reduction strategies, so they are not faithful. This means that, there are programs

M, N translated, respectively, by processes P, Q such that $M \rightarrow_{\beta} N$, but P cannot be process-reduced to Q (although P and Q are observationally equivalent). *linProc* enables us to simulate the reduction of all redexes of \mathcal{SLPCF} , giving us a faithful encoding. Thus, no evaluation strategy is determined in advance and reduction can be actually done in an asynchronous way¹. Questions on which λ -calculus reduction strategies can be encoded in process calculi has been posed in [Milner, 1992, Sect.8]. Moreover, a minimal requirement of λ -models is to induce a congruence equivalence which contains the β -equivalence, see [Ronchi Della Rocca and Paolini, 2004]. This latter requirement is trivially induced by faithful embedding of λ -calculus. Also, we note that in [Sangiorgi and Walker, 2001, (p.467)] and [Yoshida et al., 2004] in order to faithfully mimic the β -reduction, the usual reduction-rules for replication and prefixes of process calculi are extended.

There are several motivations behind this work. From a programming language point of view we provide a tool for study both parallel evaluation strategies and equivalence of programs. From a process calculus point of view we give a fine representation of a sequential language where the redexes can actually be reduced in parallel. From a game-semantics point of view, we suggest a parallel description of strategies by avoiding useless causality.

The content of this section is an extended version of the article [Paolini and Piccolo, 2009], written with Luca Paolini.

4.1 A Linear Process Calculus

linProc is a typed process calculus, based on Solos calculus [Laneve and Victor, 2003] and it is extended to treat explicit constants. In this calculus we give the possibility to communicate names as well as ground values (i.e. integers). Since, *linProc* is conceived in order to model \mathcal{SLPCF} , some notations of the previous Chapter are overloaded.

We use two sets of names, \mathcal{Q} for *question-names* and \mathcal{A} for *answer-names*. Letters p, q, v, w, f, g, \dots range over question-names and a, b, c, \dots range over answer-names. For sake of simplicity, when useful we use u to denote all kinds of names. We write \tilde{u} for a (possible empty) finite sequence u_1, \dots, u_n of names and $|\tilde{u}|$ for its length. In order to treat ground information we use a set of variables Var ranged over by x, y, z, \dots . Last, we use a set \mathcal{H} of *process variables* ranged over by F, F_1, F_2, \dots . *linProc* is based on three syntactical categories.

¹ Gordon Plotkin in [Plotkin, 1975] remarked that the call-by-value parameter passing is hardly in accord with a strategy on (call-by-name) λ -calculus, thus he introduced $\lambda\beta_v$ -calculus

Expressions:	$E ::= \mathbf{0} \mid x \mid \text{succ}(E) \mid \text{pred}(E)$
Solos	$s ::= q\langle \tilde{p}, a \rangle \mid \bar{q}\langle \tilde{p}, a \rangle$
Processes	$P ::= s \mid a(x).(\bar{q}\langle \tilde{p}, b \rangle; P) \mid \bar{a}\langle E \rangle \mid \mathbf{0} \mid P \parallel Q \mid (\nu u^S)P \mid \text{if } E \text{ then } P \text{ else } Q$ $F^q \mid \text{rec}F.P^\ddagger$

\ddagger We assume that P contains exactly one free question-name, as a constraint on $\text{rec}F.P$ construction.

The Solos calculus has been introduced in [Laneve and Victor, 2003] as a simplification of the asynchronous π -calculus replacing prefixes by solos (not binding actions) and maintaining the restriction as unique binder.

Remark 4.1.1. *In process calculi, the purpose of a prefixing $\alpha.P$ is to freeze the continuation agent P until the action α has been consumed in a reduction. In other words, reductions involving P causally depends on the reduction involving the prefix α . Apart from this explicit causal dependency, mobile calculi possess another, implicit form of causal dependency, which relies on the scope (or restriction) operator (causal dependencies in the π -calculus have been studied in several works, see [Sangiorgi, 1995, Boreale and Sangiorgi, 1998, Degano and Priami, 1995]). Consider, for instance, the following agent: $(\nu v)(\bar{u}\langle v \rangle \parallel v\langle y \rangle)$. In this agent, the sub-process $v\langle y \rangle$ can in no way react before the solo $\bar{u}\langle v \rangle$ because the subject name v is bound. When $\bar{u}\langle v \rangle$ reacts, the scope of v is extended, possibly enabling a reaction with $v\langle y \rangle$.*

The name q is the *subject* and \tilde{p}, a are the *objects* of both solos $q\langle \tilde{p}, a \rangle$ and $\bar{q}\langle \tilde{p}, a \rangle$. Parallel composition of processes $P \parallel Q$ is as usual, commutative, associative and it has the termination $\mathbf{0}$ as neutral element. The restriction $(\nu u^S)P$ limits the scope of the name u to P . Here S is a syntactical type annotation, that will be formalised ahead in this section. We avoid the type annotations when they are clear from the context or uninteresting. We denote with $\text{FN}(P)$ the set of free names of P , defined in the standard way. *linProc* extends the Solos Calculus, in order to manipulate also ground entities and recursion. *Expressions* are build on numerals, ground variables, successor and predecessor. We define the numeral n to be $\text{succ}(\dots(\text{succ}(0)))$ where succ is applied n -times to $\mathbf{0}$. The *ground-output* process $\bar{a}\langle E \rangle$ will be used to send numerals. The *ground-input* process $a(x).(s; P)$ consists of a prefix $a(x)$ that both binds the occurrences of the ground variable x in its body P , and freezes the solo s (by forbidding its interaction). The solo will be unfrozen only after a reduction involving the corresponding prefix, while reductions in the body P are allowed, even if a value on a has not been received yet. The set of ground free variables $\text{GFV}(P)$ of a process P is defined as expected. We remark that the ground-prefix is the only binder acting on variables. Such a prefix allows the control of causal dependencies needed for modelling a call-by-value parameter passing policy on ground values. We emphasise that, ground-input makes us able to express an explicit constant-driven causality: a ground value reception enables a (potential) solo communication. The ground-driven *sum* $\text{if } E \text{ then } P \text{ else } Q$ is used to model the conditional. It acts like P , if E evaluates to $\mathbf{0}$, while it acts like Q , if E is evaluated to a numeral different from $\mathbf{0}$. We denote F^q a hold-place for a process, namely a process variable bringing the free question name q with it. Last, the *recursion* $\text{rec}F.P$ binds all the free occurrences of the process variable F in P . We remark that as a syntactical constraint, it is assumed that P contains

exactly one free question-name. The set of free process variables $FPV(P)$ is defined as expected. The substitution of a process P to a process hold-place F^q will be done by an higher-order process-substitution.

4.1.1 Typing System

In this section we define a type assignment system for $\ell inProc$. This system will type only process which respect syntactical linearity. In a linear process every name appears “once” in input and “once” in output. For example the process $(q, a_1)(q\langle a_1 \rangle || \bar{q}\langle a_1 \rangle)$ is linear, while the process $(q, q_1, q_2, q_3, a_1)(q\langle q_1, q_1, a_1 \rangle || \bar{q}\langle q_2, q_3, a_1 \rangle)$ is not linear.

Linearity for process calculi has already been studied in [Berger et al., 2001, Kobayashi et al., 1996, Kobayashi et al., 1999, Yoshida et al., 2004], mainly to ensure properties like determinism and strong normalisation. A bounded name occurs exactly once in “input” and exactly once in “output”. The main idea is to use a very elementary form of typing for names in processes, namely **action modalities**, denoted by ϵ : $+$ is the *output modality*, $-$ is the *input modality* and \updownarrow is the *neutral modality* (meaning the use of a name in both input and output mode). Each name can possess a unique modality in a process. A *duality* operation (denoted by an over-line) is defined on modalities as follows $\overline{+} = -$ and $\overline{-} = +$. Remark that $\overline{\updownarrow}$ is undefined. A partial match operator \odot on modalities is defined as

$$+ \odot - = - \odot + = \updownarrow .$$

Sorting [Milner, 1993] together with an action modality are ingredients to indicate possible usages of names in $\ell inProc$. We denote by i the *atomic sorting* for name delivering ground data, i.e. the sort related to answer-names. Since we want model a programming language, following [Abramsky et al., 2000, Hyland and Ong, 2000], our sortings will ask questions and will receive an unique answer. Thus, *composite sortings* are defined by $[\vec{\phi}, i]$, where $\vec{\phi}$ is a list (possibly empty) of composite sortings. Composite sortings are associated to question-names. The sorting shape respects straightforwardly $\mathcal{S}PCF$ type. **Sorting** (denoted with S) and **channel types** (denoted with α) are generated by the following grammars:

<i>Sorting.</i> $S ::= i \mid [\vec{\phi}, i]$	<i>Channel Type.</i> $\alpha ::= S^\epsilon$
--	--

Sorting and (non-neutral) modalities are straightforwardly related to the game-semantic notions of arenas and player/opponent of game semantics as formalised in [Berger et al., 2001, Hyland and Ong, 1995]. If ϕ is a composite sorting then ϕ^ϵ is a *question type* while i^ϵ is an *answer type*. A *dual* of a type $\alpha = S^\epsilon$, is defined by $\bar{\alpha} = \overline{S^\epsilon}$. Differently to systems presented in [Berger et al., 2001, Kobayashi et al., 1996, Kobayashi et al., 1999, Yoshida et al., 2004] our modalities do not occur inside sorting, for sake of simplicity.

In order to define our typing system we need two kinds of environments, respectively bringing type-information of free names and process variables in a

$\frac{}{\Gamma \vdash \mathbf{0} \triangleright _} \text{ (z)}$	$\frac{\Gamma \vdash P \triangleright A \quad \Gamma \vdash Q \triangleright B \quad \clubsuit(A, B)}{\Gamma \vdash P \parallel Q \triangleright A \odot B} \text{ (par)}$	$\frac{\Gamma \vdash P \triangleright A \quad \Gamma \vdash Q \triangleright A}{\Gamma \vdash \text{if } E \text{ then } P \text{ else } Q \triangleright A} \text{ (sum)}$
$\frac{\tilde{p}, a \text{ distinct}}{\Gamma \vdash q(\tilde{p}, a) \triangleright q : [\tilde{\phi}, l]^{-}, \tilde{p} : \tilde{\phi}^{-}, a : i^{-}} \text{ (in)}$	$\frac{\tilde{p}, a \text{ distinct}}{\Gamma \vdash \bar{q}(\tilde{p}, a) \triangleright q : [\tilde{\phi}, l]^{+}, \tilde{p} : \tilde{\phi}^{+}, a : i^{+}} \text{ (out)}$	
$\frac{\Gamma \vdash \bar{q}(\tilde{p}, b) \triangleright B \quad \Gamma \vdash P \triangleright A \quad \clubsuit(a : i^{-}, A, B)}{\Gamma \vdash a(x).(\bar{q}(\tilde{p}, b); P) \triangleright (a : i^{-}) \odot A \odot B} \text{ (gi)}$	$\frac{}{\Gamma \vdash \bar{a}(E) \triangleright a : i^{+}} \text{ (go)}$	$\frac{\Gamma \vdash P \triangleright A}{\Gamma \vdash P \triangleright A, u : S^{\dagger}} \text{ (w)}$
$\frac{}{\Gamma, F : \phi \vdash F^q \triangleright q : \phi^{-}} \text{ (pv)}$	$\frac{\Gamma, F : \phi \vdash P \triangleright q : \phi^{-}}{\Gamma \vdash \text{rec } F.P \triangleright q : \phi^{-}} \text{ (rec)}$	$\frac{\Gamma \vdash P \triangleright A, u : S^{\dagger}}{\Gamma \vdash (\nu u^S)P \triangleright A} \text{ (res)}$

Table 4.1: Typing rules for ℓinProc

process. An **action type** is a finite set of pairs name plus channel type, in which each name appears at most once. Observe that an action type could be possibly empty. We remark that question names are paired with question types while answer names are paired with answer types. A, B, \dots range over action types and, as usual, $A, v : \alpha$ denotes the action type $A \cup \{(v : \alpha)\}$, with $(v : \alpha) \notin A$. It can be seen as partial function from names to types. We define $\text{FN}(A)$ as the set of names appearing in A and if $A = v : \alpha, A'$ we define $A(v) = \alpha$. We define an extension of the match operator \odot to action types. Intuitively, the operation performs the union of action types and it manages the match of the names that appear in both action types. More formally, let A, B such that for all $v \in \text{FN}(A) \cap \text{FN}(B)$, $A(v) = \overline{B(v)}$. Under this hypothesis, we define $A \odot B = C$ where $\text{FN}(C) = \text{FN}(A) \cup \text{FN}(B)$ and given $v \in \text{FN}(C)$

$$C(v) = \begin{cases} A(v) & \text{if } v \in \text{FN}(A) \setminus \text{FN}(B) \\ B(v) & \text{if } v \in \text{FN}(B) \setminus \text{FN}(A) \\ S^{\dagger} & \text{if } v \in \text{FN}(A) \cap \text{FN}(B) \wedge A(v) = S^{\epsilon} \end{cases}$$

It is easy to show that \odot is a partial commutative associative operator. We write $\clubsuit(A, B)$ when $A \odot B$ is defined. A **process environments** is a set of pairs process variables plus composite sorting, in which each variable appears at most once. It is denoted by Γ and we apply to it similar conventions as those introduced for action types. Valid typing judgements have the shape $\Gamma \vdash P \triangleright A$ where Γ is a process environment, P is a process and A is an action type.

Definition 4.1.1. *A typing judgement is valid when it is conclusion of a derivation respecting the typing rules in Table 4.1.*

We allow weakening and contraction on process environments, while we treat linearly action types. The rules **(z)**, **(in)**, **(out)** and **(go)** impose correct modalities on our typing. The key rules are **(par)** and **(gi)** that check composability of sub-processes, ensuring the linearity policy. In **(gi)** we choose that the solo frozen by the ground prefix has to be an output. The only further rule composing different sub-processes is **(sum)** which is managed in an additive way. Rules **(res)** and **(w)** manage neutral names. Process environments bring the type information of process variables, then

(**pv**) and (**rec**) impose the use of F by respecting the chosen type. In (**rec**) we allow **rec**-abstraction of a process variable F , on a process having q as unique free name.

Given a process P , a *slice* of P is the process obtained from P by replacing to each summation construct one of its branches. Intuitively a slice should be thought as a possible evolution of the process. Linearity guarantees that for each slices, names are used exactly once in input or in output way.

Proposition 4.1.1 (Linearity). *If $\Gamma \vdash P \triangleright A, u : S^\epsilon$ then*

1. $\epsilon \in \{+, -\}$ implies that u occurs exactly once in all slices of P ,
2. $\epsilon = \uparrow$ implies that u occurs either zero times or twice in all slices of P ,
3. $\epsilon = \uparrow$ if and only if $\Gamma \vdash (\nu u^S)P \triangleright A$,
4. $F \notin \Gamma$ implies that $\Gamma, F : \phi \vdash P \triangleright A$,
5. $\Gamma = \Gamma', F : \phi$ and $F \notin \text{FPV}(P)$ imply that $\Gamma' \vdash P \triangleright A$.

Proof. (1.) and (2.) can be easily proved by induction on the derivation of $\Gamma \vdash P \triangleright A$. The crucial case for both is parallel composition, but it follows by a straightforward case analysis just by observing that, by construction, only mutually dual channel types in an action type can be composed and they give as a result a neutral channel type. To prove (3.), one direction is easy, while the other follows by induction on the derivation of $\Gamma \vdash (\nu u^S)P$ just by observing that the only applicable rules here are (**res**) and (**w**). (4.) and (5.) can be proved by induction on the derivation of the typing judgement. The proof is quite involved, but standard. See [Sangiorgi and Walker, 2001]. \square

4.1.2 *linProc* Reductions and Congruences

We need three different substitutions: ground substitution, name substitution and process substitution. We denote $P[\tilde{n}/\tilde{x}]$ the expected substitution of integers to variables in a process P . Recall that the ground-prefix is the only variable binder of *linProc* and note that no free variable can be captured in such substitution. We denote $P[\tilde{u}/\tilde{u}']$ the expected substitution of names to names in a process, in a capture-free way. Last, process substitution is a straightforward adaptation of higher-order π -calculus substitution [Sangiorgi and Walker, 2001]. The process substitution of a process P in another process to all occurrences of F is defined as follows

1. $s\{P/F\} = s$
2. $(Q_0 \parallel Q_1)\{P/F\} = Q_0\{P/F\} \parallel Q_1\{P/F\}$,
3. $((\nu u)Q)\{P/F\} = (\nu u)Q\{P/F\}$
4. $(\text{if } E \text{ then } Q_0 \text{ else } Q_1)\{P/F\} = \text{if } E \text{ then } (Q_0)\{P/F\} \text{ else } (Q_1)\{P/F\}$,
5. $(a(x)Q)\{P/F\} = a(x)Q\{P/F\}$,

$\frac{\forall k \leq n \quad \{v_k, v'_k\} = \{p_k, p'_k\} \text{ and } \{b, b'\} = \{a, a'\}}{(\nu v_1, \dots, v_n, b)(q\langle p_1, \dots, p_n, a \rangle \parallel \bar{q}\langle p'_1, \dots, p'_n, a' \rangle \parallel R) \rightarrow R[v'_1/v_1, \dots, v'_n/v_n, b'/b]}$			
$\frac{P \rightarrow P'}{a(x).(s;P) \rightarrow a(x).(s;P')}$		$\frac{}{a(x).(s;P) \parallel \bar{a}\langle n \rangle \longrightarrow s \parallel P[n/x]}$	
$\frac{P \rightarrow P'}{(\nu u)P \rightarrow (\nu u)P'}$	$\frac{P \equiv P' \rightarrow Q' \equiv Q}{P \rightarrow Q}$	$\frac{P \rightarrow P'}{P \parallel Q \rightarrow P' \parallel Q}$	$\frac{P \rightarrow P'}{\text{rec}F.P \rightarrow \text{rec}F.P'}$
$\frac{}{\text{if } \mathbf{0} \text{ then } P \text{ else } Q \rightarrow P}$		$\frac{P \rightarrow P'}{\text{if } E \text{ then } P \text{ else } Q \rightarrow \text{if } E \text{ then } P' \text{ else } Q}$	
$\frac{n \neq \mathbf{0}}{\text{if } n \text{ then } P \text{ else } Q \rightarrow Q}$		$\frac{Q \rightarrow Q'}{\text{if } E \text{ then } P \text{ else } Q \rightarrow \text{if } E \text{ then } P \text{ else } Q'}$	

Table 4.2: Operational Semantics of *linProc*.

6. $(\bar{a}\langle E \rangle)\{P/F\} = \bar{a}\langle E \rangle$,
7. if $F_0 \neq F$ then $F_0^p\{P/F\} = F_0^p$,
8. if $\text{FN}(P) = \{q'\}$ then $F^q\{P/F\} = P\{q/q'\}$ and

$$(\text{rec}F'.P')\{P/F\} = \begin{cases} \text{rec}F'.P' & \text{if } F' = F, \\ \text{rec}F'.(P'\{P/F\}) & \text{otherwise.} \end{cases}$$

Definition 4.1.2. The *structural congruence* \equiv between expressions and processes is the least congruence containing α -equivalences on variables, names and process-variable, and satisfying the following laws

1. $\text{pred}(\text{succ } n) \equiv n$
2. $P \parallel \mathbf{0} \equiv P$, $P \parallel Q \equiv Q \parallel P$, $(P \parallel Q) \parallel R \equiv P \parallel (Q \parallel R)$, $(\nu u)\mathbf{0} \equiv \mathbf{0}$, $\text{rec}F.P \equiv P\{\text{rec}F.P/F\}$, $(\nu u_1)(\nu u_2)P \equiv (\nu u_2)(\nu u_1)P$, $(\nu u)(P \parallel Q) \equiv P \parallel (\nu u)Q$ if $u \notin \text{FN}(P)$.
3. $\text{if } E \text{ then } \mathbf{0} \text{ else } \mathbf{0} \equiv \mathbf{0}$, $P \parallel \text{if } E \text{ then } Q \text{ else } R \equiv \text{if } E \text{ then } P \parallel Q \text{ else } P \parallel R$, $(\nu u^s)\text{if } E \text{ then } P \text{ else } Q \equiv \text{if } E \text{ then } (\nu u^s)P \text{ else } (\nu u^s)Q$, $a(x).(s; \text{if } E \text{ then } P \text{ else } Q) \equiv \text{if } E \text{ then } a(x).(s;P) \text{ else } a(x).(s;Q)$ if $x \notin \text{GFV}(E)$.
4. $(\nu u)(a(x).(s;P)) \equiv a(x).(s;(\nu u)P)$ if $u \notin \{a\} \cup \text{FN}(s)$
 $Q \parallel a(x).(s;P) \equiv a(x).(s;Q \parallel P)$ if $x \notin \text{GFV}(Q)$
 $a(x).(s; b(y).(s';P)) \equiv b(y).(s'; a(x).(s;P))$ if $x \neq y$

The structural congruence axioms presented above deserve some explanation. Rules in (1) are the usual axioms dealing with the evaluation of ground expressions. Rules in (2) are the usual structural rules of π -calculus, plus the rule dealing with recursion. Rules in (3) are the structural rules dealing with the sum. They

impose the distributive property of the sum with respect to parallel composition, restriction and ground input prefix. These laws allow us to derive congruences like $P \equiv \text{if } E \text{ then } P \text{ else } P$. Similar laws were introduced in [Beffara, 2008]. Rules in (4) are the structural rules dealing with our ground prefix, which is not as the usual input-prefix. Such rules leave untouched the solo s of the prefix $a(x).(s;P)$ and they formalise that the body P is in parallel with the prefix itself, taking care only of potential occurrences of x . The implicit causality coming from the underlying calculus of solos, together with rules in (4) will give us the possibility to mimic all reduction strategies of \mathcal{SLPCF} inside our linProc .

Definition 4.1.3. We endow linProc of a reduction relation \rightarrow , namely the least relation \rightarrow satisfying the rules of Table 4.2. As usual, we define \rightarrow^* to be the reflexive transitive closure of \rightarrow .

Remark 4.1.2. The rules describing the interaction between solos is a simplification (valid only under our linear constraints) of a more general rule presented in [Laneve and Victor, 2003] which exploit the unification in a very clever way. It is defined as follows; we write θ for a total endo-function on $\mathcal{Q} \cup \mathcal{A}$ such that $u \neq \theta(u)$ for finitely many names u . We use $\text{Dom}(\theta) = \{u \mid \theta(u) \neq u\}$ and $\text{Ran}(\theta) = \{\theta(u) \mid u \neq \theta(u)\}$. Let $\{\vec{v} = \vec{w}\}$ be the smallest equivalence relation on $\mathcal{Q} \cup \mathcal{A}$ relating each v_i with w_i and let us assume a name substitution θ agrees with the equivalence φ if for every v, w , $v\varphi w$ iff $\theta(v) = \theta(w)$. The general rule presented in [Laneve and Victor, 2003] is

$$\frac{|\vec{u}_1| = |\vec{u}_2| \quad \theta \text{ agrees with } \{u_1 = u_2\} \quad \text{Ran}(\theta) \cap \vec{u}^* = \emptyset \quad \text{Dom}(\theta) = \vec{u}^*}{(\nu \vec{u}^*)(u\langle \vec{u}_1 \rangle \parallel \bar{u}\langle \vec{u}_2 \rangle \parallel R) \rightarrow R\theta}$$

Some example can help the reader,

$$\begin{aligned} (\nu p_0, p_1, a)(q\langle q_0, q_1, a \rangle \parallel \bar{q}\langle p_0, p_1, b \rangle \parallel R) &\rightarrow R[q_0/p_0, q_1/p_1, b/a], \\ (\nu p_0, q_1, b)(q\langle q_0, q_1, a \rangle \parallel \bar{q}\langle p_0, p_1, b \rangle \parallel R) &\rightarrow R[q_0/p_0, p_1/q_1, a/b], \end{aligned}$$

but $(\nu p)(q\langle p, p, a \rangle \parallel \bar{q}\langle p_0, p_1, b \rangle \parallel R)$ cannot be reduced. We observe a difference between the managing of name passing and the managing of ground-values passing. The communication of names uses the unification mechanism, which is usual both in solos calculus and in the calculus of fusion [Parrow and Victor, 1998]; there is a perfect symmetry between name-emission and name-reception. The communication of ground values instead is asymmetric; when two answer names synchronise, all the occurrences of the ground variable bounded by the prefix-construct, are substituted with the corresponding value.

Lemma 4.1.1 (Substitution lemma). *If $\Gamma, F : \phi \vdash P \triangleright A$ and $\Gamma \vdash Q \triangleright q : \phi^-$ then $\Gamma \vdash P\{Q/F\} \triangleright A$.*

Proof. By induction on the derivation of $\Gamma, F : \phi \vdash P \triangleright A$. □

Lemma 4.1.2 (Subject congruence). *Let $\Gamma \vdash P \triangleright A$. If $P \equiv Q$ then $\Gamma \vdash Q \triangleright A$.*

Proof. The proof is as usual by induction on the derivation of $P \equiv Q$. We develop some cases.

- $P = (P_1 \parallel P_2)$ and $\Gamma \vdash P \triangleright A$ and $Q = (P_2 \parallel P_1)$. We conclude $\Gamma \vdash Q \triangleright A$ by commutativity of \odot .
- $P = (P_1 \parallel P_2) \parallel P_3$ and $\Gamma \vdash P \triangleright A$ and $Q = P_1 \parallel (P_2 \parallel P_3)$. We conclude $\Gamma \vdash Q \triangleright A$ by associativity of \odot .
- $P = (\nu u)(P_1 \parallel P_2)$ and $\Gamma \vdash P \triangleright A$ and $Q = P_1 \parallel (\nu u)P_2$. It is evident that $\Gamma \vdash Q \triangleright A$.
- $P = \text{rec}F.P_1$, $\Gamma \vdash P \triangleright A$ and $Q = P_1 \{\text{rec}F.P_1/F\}$. By Lemma 4.1.1 we have $\Gamma \vdash Q \triangleright A$.

□

Lemma 4.1.3. *Let $\Gamma \vdash (\nu u_1, \dots, u_n, b)(q\langle p_1, \dots, p_n, a \rangle \parallel \bar{q}\langle p'_1, \dots, p'_n, a' \rangle \parallel R) \triangleright A$, $q : S^\uparrow$. If $\{u_k, u'_k\} = \{p_k, p'_k\}$ for all $k \leq n$ and $\{b, b'\} = \{a, a'\}$ then $\Gamma \vdash R[u'_1/u_1, \dots, u'_n/u_n, b'/b] \triangleright A$.*

Proof. We let $B = q : [\check{\phi}, i]^-$, $\tilde{p} : \check{\phi}^-$, $a : i^-$ and $C = q : [\check{\phi}, i]^+$, $\tilde{p}' : \check{\phi}^+$, $a' : i^+$. Typing judgement $\Gamma \vdash (\nu \tilde{u})(u\langle \tilde{p}, a \rangle \parallel \bar{u}\langle \tilde{q}, b \rangle \parallel R) \triangleright A$ have the following shape

$$\frac{\frac{\tilde{p}, a \text{ distinct}}{\Gamma \vdash q\langle \tilde{p}, a \rangle \triangleright B} \text{ (in)} \quad \frac{\tilde{p}', a' \text{ distinct}}{\Gamma \vdash \bar{q}\langle \tilde{p}', a' \rangle \triangleright C} \text{ (out)} \quad \frac{\Pi}{\Gamma \vdash R \triangleright D} \text{ (par)}}{\Gamma \vdash q\langle \tilde{p}, a \rangle \parallel \bar{q}\langle \tilde{p}', a' \rangle \parallel R \triangleright q : [\check{\phi}, i]^\uparrow, \tilde{u} : \check{S}^\uparrow, b : i^\uparrow, A} \text{ (Res)}}{\Gamma \vdash (\nu \tilde{u}, b)(q\langle \tilde{p}, a \rangle \parallel \bar{q}\langle \tilde{p}', a' \rangle \parallel R) \triangleright q : [\check{\phi}, i]^\uparrow, A} \text{ (Res)}$$

where

$$B \odot C \odot D = q : [\check{\phi}, i]^\uparrow, \tilde{u} : \check{S}^\uparrow, b : i^\uparrow, A \quad (4.1)$$

and Π is a derivation of the typing judgement for $\Gamma \vdash R \triangleright D$. By Equation (4.1), we have $D = \tilde{u} : \check{S}^{\epsilon_i}, b : i^{\epsilon'}$, D' for opportune $\epsilon_i, \epsilon' \in \{+, -\}$ and D' . Consequently $A = \tilde{u}' : \check{S}^{\tilde{\epsilon}}, b' : i^{\tilde{\epsilon}'}$, D' . So $\Pi[\tilde{u}'/\tilde{u}, b'/b]$ is derivation of the typing judgement for $\Gamma \vdash R[\tilde{u}'/\tilde{u}, b'/b] \triangleright A$ (where $\Pi[\tilde{u}'/\tilde{u}, b'/b]$ is obtained from Π applying to it the corresponding name substitution). □

Theorem 4.1.1 (Subject Reduction). *Let $\Gamma \vdash P \triangleright A$. If $P \rightarrow Q$ then $\Gamma \vdash Q \triangleright A$.*

Proof. The proof is by induction on the derivation of the judgement $P \rightarrow Q$. Let us develop a few cases.

- Case $P = (\nu u_1, \dots, u_n, b)(q\langle p_1, \dots, p_n, a \rangle \parallel \bar{q}\langle p'_1, \dots, p'_n, a' \rangle \parallel R) \rightarrow R[u'_1/u_1, \dots, u'_n/u_n, b'/b] = Q$ consequence of $\{u_k, u'_k\} = \{p_k, p'_k\}$ for all $k \leq n$ and $\{b, b'\} = \{a, a'\}$. Since $\Gamma \vdash P \triangleright A$, by construction we must have $A = A'$, $q : S^\uparrow$. So by applying Lemma 4.1.3, we get $\Gamma \vdash R[u'_1/u_1, \dots, u'_n/u_n, b'/b] \triangleright A'$. We get $\Gamma \vdash R[u'_1/u_1, \dots, u'_n/u_n, b'/b] \triangleright A$ just applying a **(w)** rule.
- Case $P \rightarrow Q$ direct consequence of the fact that $P' \rightarrow Q'$ and $P \equiv P'$ and $Q \equiv Q'$. Suppose $\Gamma \vdash P \triangleright A$. By Lemma 4.1.2, we have $\Gamma \vdash P' \triangleright A$. By inductive hypothesis, we have $\Gamma \vdash Q' \triangleright A$ and we get $\Gamma \vdash Q \triangleright A$ again by Lemma 4.1.2.

□

Linearity implies that the reduction in our calculus is confluent.

Lemma 4.1.4 (Confluence). *Let $\Gamma \vdash P \triangleright A$.*

If $P \rightarrow Q_i$ for all $i \in \{0, 1\}$, then either $Q_0 \equiv Q_1$ or there is Q such that $Q_i \rightarrow Q$.

Proof. The proof is by induction on the derivation proving $P \rightarrow Q_0$. The proof is quite involved, but standard. The key observation is that by Proposition 4.1.1, two distinct communications (either made both in input or both in output) are never on the same channel, thus two different reductions are never mutually exclusive. This means that a reduction may enable other potential communications but cannot inhibit other communication which are ready in that moment. □

Since we are interested in the extensional behaviour of terms, we define a Morris like contextual equivalence as basic equality over processes.

Definition 4.1.4 (Observability). *Let $\Gamma \vdash P \triangleright a : i^+$. We use $\Gamma \vdash P \Downarrow_{\bar{a}(n)}$ to denote that $P \rightarrow^* P' \equiv (v\tilde{u})(\bar{a}(E)|P')$ where $E \equiv n$ and $a \notin \tilde{u}$.*

A relation \approx is a *typed congruence* when $\equiv \subseteq \approx$ and it is a typed equality closed under typed contexts. Moreover, $\Gamma \vdash P \approx Q \triangleright A$ is an abbreviation for $\Gamma \vdash P \triangleright A$, $\Gamma \vdash Q \triangleright A$ and $P \approx Q$. We use ρ to denote a total function from the set of ground variables to the set of numerals. Given a process P , we denote $P\rho$ to be the process obtained applying the substitution ρ to ground free variables of P .

Definition 4.1.5. \approx_E is the greatest typed congruence on processes such that, $\Gamma \vdash P \approx_E Q \triangleright A$ and $\Gamma \vdash P\rho \Downarrow_{\bar{a}(n)}$ imply $\Gamma \vdash Q\rho \Downarrow_{\bar{a}(n)}$, for all ρ .

Following [Honda and Yoshida, 1995, Yoshida et al., 2004], we can prove that \approx_E is consistent (i.e. does not equate all process), it is reduction closed, it is maximally consistent (i.e. the only typed congruence which strictly includes \approx_E is not consistent) and it equates all insensitive processes (i.e. processes that does not produce any observation) with the same type.

4.2 Processing Programs

The encoding of $\mathcal{S}PCF$ into $\mathit{linProc}$ is an adaptation of the encoding presented by Hyland and Ong in [Hyland and Ong, 1995]. First of all, types can be translated in sortings as follows, $\llbracket \tau_1 \multimap \dots \multimap \tau_k \multimap \iota \rrbracket = \llbracket \tau_1 \rrbracket, \dots, \llbracket \tau_k \rrbracket, \iota \rrbracket$ ($k \geq 0$). Before formalising the translation of programs, we give some hints.

Let M be a $\mathcal{S}PCF$ term such that $\Gamma \vdash M : \sigma_1 \multimap \dots \multimap \sigma_n \multimap \iota$ where $n \geq 0$. Encoding exploits overloads of symbols for variables of $\mathcal{S}PCF$ and symbols for names and variables of $\mathit{linProc}$. The interpretation of M is given on a process P such that $\mathit{GFV}(P) = \mathit{FV}(M) \cap \mathit{Var}^t$, $\mathit{SFV}(P) = \mathit{FV}(M) \cap \mathit{SVar}$ and $\mathit{FN}(P) = (\mathit{FV}(M) \cap \mathit{HVar}) \cup \{q\}$ where q is a fresh name, called *access-name*. Channel type of free names of P , but

To lighten the notation, from now on, we do not annotate type explicitly on processes and we denote $q(q_1, \dots, q_n, a)P = (\nu q_1, \dots, q_n, a)(q(q_1, \dots, q_n, a) P)$.	
$\llbracket \text{pred}^{l \rightarrow l} \rrbracket^{q_\epsilon} = q_\epsilon(q_1, a_\epsilon)(\bar{q}_1(a_1)(\nu q) a_1(x).(\bar{q}(a_\epsilon); q(a) \bar{a}(\text{pred}(x))))$	$\llbracket \underline{n} \rrbracket^{q_\epsilon} = q_\epsilon(a_\epsilon) \bar{a}_\epsilon(\underline{n})$
$\llbracket \text{succ}^{l \rightarrow l} \rrbracket^{q_\epsilon} = q_\epsilon(q_1, a_\epsilon)(\bar{q}_1(a_1)(\nu q) a_1(x).(\bar{q}(a_\epsilon); q(a) \bar{a}(\text{succ}(x))))$	$\llbracket \underline{x} \rrbracket^{q_\epsilon} = q_\epsilon(a_\epsilon) \bar{a}_\epsilon(\underline{x})$
$\llbracket \lambda \mathbf{x}^l. M^{\sigma_1 \rightarrow \dots \rightarrow \sigma_n \rightarrow l} \rrbracket^{q_\epsilon} = q_\epsilon(q_\emptyset, q_1, \dots, q_n, a_\epsilon) \bar{q}_\emptyset(a_\emptyset) (\nu q) a_\emptyset(x).(\bar{q}(q_1, \dots, q_n, a_\epsilon); \llbracket M \rrbracket^q)$	
$\llbracket \mathbf{f}^{\sigma_1 \rightarrow \dots \rightarrow \sigma_n \rightarrow l} \rrbracket^{q_\epsilon} = q_\epsilon(q_1, \dots, q_n, a_\epsilon) \bar{\mathbf{f}}(q_1, \dots, q_n, a_\epsilon)$	
$\llbracket M^{\sigma_1 \rightarrow \dots \rightarrow \sigma_n \rightarrow l} N^{\sigma_1} \rrbracket^{q_\epsilon} = q_\epsilon(q_2, \dots, q_n, a) (\nu p, q_1)(\bar{p}(q_1, \dots, q_n, a) \llbracket M \rrbracket^p \llbracket N \rrbracket^{q_1})$	
$\llbracket \text{if } M_1^i M_2^l M_3^q \rrbracket^{q_\epsilon} = q_\epsilon(a_\epsilon)(\nu q_1)(\llbracket M_1^i \rrbracket^{q_1} \llbracket \bar{q}_1(a_1) (\nu q_2) a_1(x).(\bar{q}_2(a_\epsilon); \text{if } x \text{ then } \llbracket M_2 \rrbracket^{q_2} \text{ else } \llbracket M_3 \rrbracket^{q_2}) \rrbracket^{q_\epsilon})$	
$\llbracket \lambda \mathbf{f}^l. M^{\sigma_1 \rightarrow \dots \rightarrow \sigma_n \rightarrow l} \rrbracket^{q_\epsilon} = q_\epsilon(\mathbf{f}, q_1, \dots, q_n, a) (\nu p)(\bar{p}(q_1, \dots, q_n, a) \llbracket M \rrbracket^p)$	
$\llbracket F^{\sigma_1 \rightarrow \dots \rightarrow \sigma_n \rightarrow l} \rrbracket^{q_\epsilon} = q_\epsilon(q_1, \dots, q_n, a) (\nu p)(\bar{p}(q_1, \dots, q_n, a) F^p)$	$\llbracket \mu F^\sigma. M^\sigma \rrbracket^{q_\epsilon} = \text{rec} F. \llbracket M \rrbracket^{q_\epsilon}$

Table 4.3: Translation of $\mathcal{S}\ell\text{PCF}$ on ℓinProc .

the access-name, is obtained by translating the type of corresponding variable in M in a sorting together with positive modality. Channel type of the access-name is the translation of the type of M together with a negative modality. The sorting associated to process variables is obtained by translating the type of corresponding stable variable. More formal details are in Theorem 4.2.1. If M is closed then P contains a unique free name, namely the access-name q typed by negative modality and the sorting $[\llbracket \sigma_1 \rrbracket, \dots, \llbracket \sigma_n \rrbracket, l]$ corresponding to the type of M . The translation of M can be questioned by the solo $\bar{q}(p_1, \dots, p_n, a)$ communicating a list of n question-names p_1, \dots, p_n where processes mimicking “actual arguments” can be questioned in its turn by P , and an answer-name a where P can communicate the computation result.

Definition 4.2.1. *We denote by $\llbracket M \rrbracket^q$ the process encoding a program $\Gamma \vdash M : \sigma$ on the access-name q . The recursive definition of the translation from $\mathcal{S}\ell\text{PCF}$ to ℓinProc is given in Table 4.3.*

We use the ground-prefix in order to model the causal dependency needed in order to respect call-by-value computations. As instances, we remark that $q_\epsilon(a_\epsilon) \bar{a}_\epsilon(\underline{n})$ is an abbreviation for $(\nu a_\epsilon^i)(q_\epsilon(a_\epsilon) \llbracket \bar{a}_\epsilon(\underline{n}) \rrbracket)$ and the translation of $\text{pred}^{l \rightarrow l}$ is an abbreviation for

$$(\nu q_1^{[l]})(\nu a_\epsilon^i)(q_\epsilon(q_1, a_\epsilon) \llbracket (\nu a_1^i)(\bar{q}_1(a_1) \llbracket (\nu q^{[l]}) a_1(x).(\bar{q}(a_\epsilon) \llbracket (\nu a^i)(q(a) \bar{a}(\text{pred}(x)))) \rrbracket) \rrbracket^{q_\epsilon})$$

Theorem 4.2.1 (Typing soundness). *Let $F_1^{\tau_1}, \dots, F_m^{\tau_m}, \mathbf{f}_1 : \sigma_1, \dots, \mathbf{f}_n : \sigma_n, \mathbf{x}_1^l, \dots, \mathbf{x}_h^l \vdash M : \sigma$ be a term of $\mathcal{S}\ell\text{PCF}$. Then $F_1 : \llbracket \tau_1 \rrbracket, \dots, F_m : \llbracket \tau_m \rrbracket \vdash \llbracket M \rrbracket^q \triangleright q : \llbracket \sigma \rrbracket^-, \mathbf{f}_1 : \llbracket \sigma_1 \rrbracket^+, \dots, \mathbf{f}_n : \llbracket \sigma_n \rrbracket^+$ where $\text{GFV}(\llbracket M \rrbracket^q) = \{\mathbf{x}_1, \dots, \mathbf{x}_h\}$.*

Proof. By induction on the derivation of $\Gamma \vdash M : \sigma$. □

We say that a translation from a calculus to another is *faithful* whenever each reduction on the first calculus can be mimicked by some reductions in the second one. We can prove that our translation is faithful, more precisely $M \rightarrow_{\mathcal{S}\ell} N$ implies $\llbracket M \rrbracket^q \rightarrow^* \llbracket N \rrbracket^q$.

Lemma 4.2.1. 1. $\llbracket \text{pred}(\text{succ } \underline{n}) \rrbracket^q \rightarrow^* \llbracket \underline{n} \rrbracket^q$.

2. Let $\Gamma \vdash \llbracket L \rrbracket^{q_2}, \llbracket R \rrbracket^{q_2} \triangleright q_2 : \llbracket \iota \rrbracket^-, A$.
Both $\llbracket \text{if } \emptyset L R \rrbracket^q \rightarrow^* \llbracket L \rrbracket^q$ and $\llbracket \text{if } \underline{n+1} L R \rrbracket^q \rightarrow^* \llbracket R \rrbracket^q$.

Proof.

1. $\llbracket \text{succ } \underline{n} \rrbracket^{q_s} = q_s(a_s)(\nu q'_s, q_n) \left(\begin{array}{l} q'_s(q'_n, a'_s) \bar{q}'_n(a'_n) (\nu q) a'_n(x).(\bar{q}(a'_s); q(a) \bar{a}(\text{succ}(x))) \parallel \\ \bar{q}'_s \langle q_n, a_s \rangle \parallel q_n(a_n) \bar{a}_n \langle n \rangle \end{array} \right)$
 $\llbracket \text{pred}(\text{succ } \underline{n}) \rrbracket^{q_e} = q_e(a_e) (\nu q_p^*, q_s) \left(\begin{array}{l} q_p^*(q_s^*, a_p^*) \bar{q}_s^*(a_s^*) (\nu q) a_s^*(y).(\bar{q}(a_p^*); q(a) \bar{a}(\text{pred}(y))) \parallel \\ \bar{q}_p^* \langle q_s, a_e \rangle \parallel \llbracket \text{succ } \underline{n} \rrbracket^{q_s} \end{array} \right)$
 $\rightarrow^* q_e(a_e) \bar{a}_e \langle n \rangle = \llbracket \underline{n} \rrbracket^{q_e}$
2. $\llbracket \text{if } \emptyset L R \rrbracket^{q_e} = q_e(a_e) (\nu q_1) \left(q_1(a_1) \bar{a}_1 \langle \emptyset \rangle \parallel \bar{q}_1(a_1) (\nu q_2) a_1(x) (\bar{q}_2(a_e); \text{if } x \text{ then } \llbracket L \rrbracket^{q_2} \text{ else } \llbracket R \rrbracket^{q_2}) \right) \rightarrow^* \llbracket L \rrbracket^{q_e}$.
The other case is similar to the previous one. □

Ground substitution and stable substitution does not present particular problems with respect to faithful translation.

Lemma 4.2.2. *If $\Gamma \vdash M : \sigma_1 \multimap \dots \sigma_n \multimap \iota$ ($n \geq 0$) then $q_e(q_1, \dots, q_n, a) (\nu p) (\bar{p} \langle q_1, \dots, q_n, a \rangle) \parallel \llbracket M \rrbracket^p \rightarrow^* \llbracket M \rrbracket^{q_e}$*

Proof. The proof is by cases on the definition of interpretation. All cases are straightforward, except the one dealing with recursion. In fact, for all M , the process $\llbracket M \rrbracket^p$ always start with a question on p , except for the case $M = \mu F.M'$. Thus let us consider the case $M = \mu F_1 \dots \mu F_m.M'$ ($m \geq 1$) where $M' \neq \mu F.M''$. We can check, by induction on m , that for an opportune process P we have $\llbracket \mu F_1 \dots \mu F_m.M' \rrbracket^p \equiv p(q_1, \dots, q_n, a) P$. So, we get

$$\begin{aligned} q_e(q_1, \dots, q_n, a) (\nu p) (\bar{p} \langle q_1, \dots, q_n, a \rangle) \parallel \llbracket \mu F_1 \dots \mu F_m.M' \rrbracket^p &\equiv \\ q_e(q_1, \dots, q_n, a) (\nu p) (\bar{p} \langle q_1, \dots, q_n, a \rangle) \parallel p(q_1, \dots, q_n, a) P & \\ \rightarrow q_e(q_1, \dots, q_n, a) P &\equiv \llbracket \mu F_1 \dots \mu F_m.M' \rrbracket^{q_e}. \end{aligned} \quad \square$$

Lemma 4.2.3. *If $\Gamma \vdash (\lambda x^t.M)\underline{n} : \sigma$ then $\llbracket (\lambda x^t.M)\underline{n} \rrbracket^q \rightarrow^* \llbracket M[n/x] \rrbracket^q$.*

Proof. $\llbracket M \rrbracket^q [n/x] \equiv \llbracket M[n/x] \rrbracket^q$ follows easily by interpretation, hence

$$\begin{aligned} \llbracket (\lambda x^t.M)\underline{n} \rrbracket^{q_e} &= q_e(\tilde{u}, a_e) (\nu p_M, q) \left(\begin{array}{l} \bar{p}_M \langle q, \tilde{u}, a_e \rangle \parallel q(a) \bar{a} \langle n \rangle \\ p_M(q, \tilde{u}, a_e) \bar{q}(a) (\nu p'_M) a(x).(\bar{p}'_M \langle \tilde{u}, a_e \rangle; \llbracket M \rrbracket^{p'_M}) \end{array} \right) \\ \rightarrow^* q_e(\tilde{u}, a_e) (\nu p'_M) (\bar{p}'_M \langle \tilde{u}, a_e \rangle) \parallel \llbracket M \rrbracket^{p'_M} [n/x] &\rightarrow^* \llbracket M \rrbracket^{q_e} [n/x] \rightarrow^* \llbracket M[n/x] \rrbracket^{q_e} \end{aligned}$$

where the second reduction follows by Lemma 4.2.2. □

Lemma 4.2.4. *Let $\Gamma \vdash M : \sigma$.*

1. *If $F \in \text{SFV}^t(M)$ and $_ \vdash N : \tau$ then $\llbracket M \rrbracket^q \{ \llbracket N \rrbracket^q / F \} \rightarrow^* \llbracket M[N/F] \rrbracket^q$.*
2. *If $M \rightsquigarrow_Y N$ then $\llbracket M \rrbracket^q \rightarrow^* \llbracket N \rrbracket^q$.*

Proof. 1. By induction on the derivation of $\Gamma \vdash M : \sigma$ and by Lemma 4.2.2.

2. Let $M = \mu F.M'$, thus $\llbracket \mu F.M' \rrbracket^a \equiv \llbracket M' \rrbracket^a \{ \llbracket \mu F.M' \rrbracket^a / F \} \rightarrow^* \llbracket M'[\mu F.M' / F] \rrbracket^a$.

□

To mimic substitutions of programs to higher-order variables, we need to use the ground prefix together with its structural rules.

Lemma 4.2.5. *Let $\Gamma, f : \sigma \multimap \tau \vdash M : \sigma'$ and $\Delta \vdash N : \sigma \multimap \tau$ two $\mathcal{S}\ell$ PCF-terms.*

1. $(\nu f)(\llbracket M \rrbracket^a \parallel \llbracket N \rrbracket^f) \rightarrow^* \llbracket M[N/f] \rrbracket^a$.
2. *If $(\lambda f^{\sigma \multimap \tau}.M)N \rightsquigarrow_\beta M[N/f]$ then $\llbracket (\lambda f^{\sigma \multimap \tau}.M)N \rrbracket^a \rightarrow^* \llbracket M[N/f] \rrbracket^a$.*

Proof. 1. The proof is by induction on the derivation of $\Gamma \vdash M : \tau$. For the base case $M = f$, we make use of Lemma 4.2.2. For the inductive step, there are two non-trivial cases, that are the ground λ -abstraction case and the ℓ if-case. In case of ground λ -abstraction, we need only to use the structural rule $P \parallel a(x).(\bar{q}\langle \bar{u} \rangle; Q) \equiv a(x).(\bar{q}\langle \bar{u} \rangle; P \parallel Q)$ providing that $x \notin \text{GFV}(P)$. In case of if, it is necessary to use also the distributive laws for the sum. All further cases are straightforward.

2. $\llbracket (\lambda f^{\sigma \multimap \tau}.M)N \rrbracket^{a_\epsilon} =$
 $a_\epsilon(\bar{u}, a_\epsilon)(\nu q, f)(\bar{q}\langle f, \bar{u}, a_\epsilon \rangle \parallel q(f, \bar{u}, a)(\nu p)(\bar{p}\langle \bar{u}, a \rangle \parallel \llbracket M \rrbracket^p) \parallel \llbracket N \rrbracket^f) \rightarrow^*$
 $(\nu f)(\llbracket M \rrbracket^{a_\epsilon} \parallel \llbracket N \rrbracket^f) \rightarrow^* \llbracket M[N/f] \rrbracket^{a_\epsilon}$.

□

Note that our translation actually maps a calculus (i.e. $\mathcal{S}\ell$ PCF with $\rightarrow_{\mathcal{S}\ell}$ -reduction) into another calculus (i.e. ℓ inProc with \rightarrow -reduction).

Theorem 4.2.2. *Our translation is faithful, i.e. if $M \rightarrow_{\mathcal{S}\ell} N$ then $\llbracket M \rrbracket^a \rightarrow^* \llbracket N \rrbracket^a$.*

Proof. Note that \rightarrow is closed under all context, by Table 4.2. Thus, the proof follows by previous lemmas. □

Example. Let us consider the evaluation of the term $(\lambda f^{\ell \multimap \ell}. \lambda x^{\ell}. f \underline{5})(\lambda z^{\ell}. z)(\text{pred} \underline{3})$ in $\mathcal{S}\ell$ PCF. We have the following derivation.

$$\frac{\frac{\frac{\underline{2} \Downarrow \underline{2}}{\text{pred} \underline{3} \Downarrow \underline{2}} \quad \frac{\frac{\underline{5} \Downarrow \underline{5}}{(\lambda z^{\ell}. z) \underline{5} \Downarrow \underline{5}}{\underline{5} \Downarrow \underline{5}}}}{(\lambda x^{\ell}. (\lambda z^{\ell}. z) \underline{5})(\text{pred} \underline{3}) \Downarrow \underline{5}}}{(\lambda f^{\ell \multimap \ell}. \lambda x^{\ell}. f \underline{5})(\lambda z^{\ell}. z)(\text{pred} \underline{3}) \Downarrow \underline{5}}$$

The process language ℓ inProc can simulate this reduction strategy. Observe in fact that

$$\begin{aligned} \llbracket (\lambda f^{\ell \multimap \ell}. \lambda x^{\ell}. f \underline{5})(\lambda z^{\ell}. z)(\text{pred} \underline{3}) \rrbracket^a &\rightarrow \llbracket (\lambda x^{\ell}. (\lambda z^{\ell}. z) \underline{5})(\text{pred} \underline{3}) \rrbracket^a \\ &\rightarrow^* \llbracket (\lambda x^{\ell}. (\lambda z^{\ell}. z) \underline{5})(\underline{2}) \rrbracket^a \\ &\rightarrow^* \llbracket ((\lambda z^{\ell}. z) \underline{5}) \rrbracket^a \\ &\rightarrow^* \llbracket \underline{5} \rrbracket^a \end{aligned}$$

But, *linProc* is also able to simulate other reduction strategies, as shown in the following.

$$\begin{aligned} \llbracket (\lambda f^{\iota \circ \iota} . \lambda x^{\iota} . f \underline{5})(\lambda z^{\iota} . z)(\text{pred} \underline{3}) \rrbracket^{\text{q}} &\rightarrow \llbracket (\lambda x^{\iota} . (\lambda z^{\iota} . z) \underline{5})(\text{pred} \underline{3}) \rrbracket^{\text{q}} \\ &\rightarrow \llbracket (\lambda x^{\iota} . \underline{5})(\text{pred} \underline{3}) \rrbracket^{\text{q}} \\ &\rightarrow \llbracket (\lambda x^{\iota} . \underline{5}) \underline{2} \rrbracket^{\text{q}} \\ &\rightarrow \llbracket \underline{5} \rrbracket^{\text{q}} \end{aligned}$$

The result established by Theorem 4.2.2 is very strong and it overcomes the traditional encodings from programs to processes [Sangiorgi and Walker, 2001]: in our encoding *no reduction strategy is determined in advance*. Gordon Plotkin in [Plotkin, 1975] remarked that the call-by-value parameter passing is hardly in accord with a strategy on (call-by-name) λ -calculus.

Corollary 4.2.1. *If $M \Downarrow \underline{n}$ then $\llbracket M \rrbracket^{\text{p}} \approx_{\text{E}} \llbracket \underline{n} \rrbracket^{\text{p}}$.*

Proof. Since $\rightarrow_{\subseteq} \approx_{\text{E}}$ by Lemma 4.1.4, the proof follows by Definition 4.1.5. \square

4.3 Soundness and correctness

An interpretation is said to be *adequate* when $\llbracket M \rrbracket \approx_{\text{E}} \llbracket \underline{n} \rrbracket$ and $M \Downarrow \underline{n}$ are logically equivalent for any program M , numeral \underline{n} . Actually, we prove a stronger form of adequacy result, namely that $M \rightarrow_{\mathcal{S}}^* \underline{n}$, $\llbracket M \rrbracket \rightarrow^* \llbracket \underline{n} \rrbracket$ and $\llbracket M \rrbracket \approx_{\text{E}} \llbracket \underline{n} \rrbracket$ are logically equivalent, for any program M and numeral \underline{n} .

In order to complete the proof of adequacy for our interpretation, we straightforwardly adapt the Tait's computability argument likewise to that done in [Paolini, 2006, Plotkin, 1977] for denotational semantics.

Definition 4.3.1. *The “computability predicate” is defined by the following cases.*

- *Case $FV(M) = \emptyset$.*
 - *Subcase $\sigma = \iota$. $\text{Comp}(M^{\iota})$ if and only if $\llbracket M \rrbracket^{\text{q}} \approx_{\text{E}} \llbracket \underline{n} \rrbracket^{\text{q}}$ implies $M \rightarrow_{\mathcal{S}}^* \underline{n}$.*
 - *Subcase $\sigma = \mu \multimap \tau$. $\text{Comp}(M^{\mu \multimap \tau})$ if and only if $\text{Comp}(M^{\mu \multimap \tau} N^{\mu})$ for each closed N^{μ} such that $\text{Comp}(N^{\mu})$.*
- *Case $FV(M^{\sigma}) = \{\chi_1^{\tau_1}, \dots, \chi_n^{\tau_n}\}$, for some $n \geq 1$.*
 $\text{Comp}(M^{\sigma})$ if and only if $\text{Comp}(M[N_1/\chi_1, \dots, N_n/\chi_n])$ for each closed $N_i^{\tau_i}$ such that $\text{Comp}(N_i^{\tau_i})$.

Lemma 4.3.1 states a standard equivalent formulation of computability predicate.

Lemma 4.3.1. *Let $M^{\tau_1 \multimap \dots \multimap \tau_m \multimap \iota} \in \mathcal{S}PCF$ and $FV(M) = \{\chi_1^{\mu_1}, \dots, \chi_n^{\mu_n}\}$ ($n, m \in \mathbb{N}$).*

$\text{Comp}(M)$ if and only if $\llbracket M[N_1/\chi_1, \dots, N_n/\chi_n] P_1 \dots P_m \rrbracket^{\text{q}} \approx_{\text{E}} \llbracket \underline{n} \rrbracket^{\text{q}}$ implies

$M[N_1/\chi_1, \dots, N_n/\chi_n] P_1 \dots P_m \rightarrow_{\mathcal{S}}^ \underline{n}$ for each closed terms $N_i^{\mu_i}$ and $P_j^{\tau_j}$ such that $\text{Comp}(N_i)$ and $\text{Comp}(P_j)$ where $i \leq n, j \leq m$.*

The proof is an adaptation of the proof given by Plotkin in [Plotkin, 1977].

Lemma 4.3.2. *Let $\Gamma \vdash M : \sigma$. $\llbracket (\lambda x^l.M)N \rrbracket^q \approx_E \llbracket M[\underline{n}/x] \rrbracket^q$ if and only if $\llbracket N \rrbracket^q \approx_E \llbracket \underline{n} \rrbracket^q$.*

Proof. The proof is done by induction on N . □

Lemma 4.3.3. *If $M^\sigma \in \mathcal{S}\ell\text{PCF}$ then $\text{Comp}(M^\sigma)$.*

Proof. Following the pattern of [Paolini, 2006, Plotkin, 1977], the proof is done by induction on the “untyped syntax shape” of M .

- $M = \underline{0}$ or $M = \underline{n}$. The only possible type is ι , so the proof is obvious.
- $M = \lambda$. Let $\sigma = \tau_1 \multimap \cdots \multimap \tau_m \multimap \iota$, where $m \in \mathbb{N}$. Let P^σ and $N_i^{\tau_i}$ for $1 \leq i \leq m$ be closed terms such that $\text{Comp}(P^\sigma)$ and $\text{Comp}(N_i^{\tau_i})$. By definition $\text{Comp}(P^\sigma)$ imply that, if $\llbracket PN_1 \cdots N_m \rrbracket^q \approx_E \llbracket \underline{n} \rrbracket^q$ then $PN_1 \cdots N_m \rightarrow_{\mathcal{S}\ell}^* \underline{n}$.
- $M = \text{succ}$. Clearly, $\iota \multimap \iota$ is the only possible type. Let P^l be a closed term such that $\text{Comp}(P^l)$. It is easy to check that $\llbracket \text{succ}(P^l) \rrbracket^q \approx_E \llbracket \underline{m} \rrbracket^q$ implies $\llbracket P^l \rrbracket^q \approx_E \llbracket \underline{m} - 1 \rrbracket^q$. Since $\text{Comp}(P^l)$ means that $\llbracket P^l \rrbracket^q \approx_E \llbracket \underline{n} \rrbracket^q$ implies $P^l \rightarrow_{\mathcal{S}\ell}^* \underline{n}$, the proof follows by Definition 3.1.3.
- $M = \text{pred}$. Similar to the previous case.
- $M = \text{lif}$ $N^l L^l R^l$ where $\text{FV}(M) = \{\mathcal{X}_1^{\mu_1}, \dots, \mathcal{X}_k^{\mu_k}\}$ for $k \geq 0$. Let $\text{Comp}(N_i)$ for some $N_1^{\mu_1}, \dots, N_k^{\mu_k}$ closed ($1 \leq i \leq k$) and $M' = (\text{lif } N L R)[N_1/\mathcal{X}_1, \dots, N_k/\mathcal{X}_k]$. Suppose $\llbracket M' \rrbracket^q \approx_E \llbracket \underline{n} \rrbracket^q$, so it is easy to check that either $\llbracket N[N_1/\mathcal{X}_1, \dots, N_k/\mathcal{X}_k] \rrbracket^q \approx_E \llbracket \underline{0} \rrbracket^q$ or $\llbracket N[N_1/\mathcal{X}_1, \dots, N_k/\mathcal{X}_k] \rrbracket^q \approx_E \llbracket \text{succ}(\underline{m}) \rrbracket^q$ by interpretation. In the former case, $\llbracket L[N_1/\mathcal{X}_1, \dots, N_n/\mathcal{X}_n] \rrbracket^q \approx_E \llbracket \underline{n} \rrbracket^q$, thus by induction hypothesis $L[N_1/\mathcal{X}_1, \dots, N_k/\mathcal{X}_k] \rightarrow_{\mathcal{S}\ell}^* \underline{0}$ and $N[N_1/\mathcal{X}_1, \dots, N_n/\mathcal{X}_n] \rightarrow_{\mathcal{S}\ell}^* \underline{n}$. So, $M' \rightarrow_{\mathcal{S}\ell}^* \underline{n}$. The other case is similar.
- $M = \text{NP}$. Assume $N^{\tau \multimap \sigma}$ and P^τ for types σ and τ . By induction hypothesis $\text{Comp}(N^{\tau \multimap \sigma})$ and $\text{Comp}(P^\tau)$ and the proof follows by Definition 4.3.1.
- $M = \lambda x.Q$. Assume x^μ and Q^τ for types μ and τ . Let $\text{FV}(M) = \{\mathcal{X}_1^{\mu_1}, \dots, \mathcal{X}_k^{\mu_k}\}$ for $k \geq 0$ and $\tau = \tau_1 \multimap \cdots \multimap \tau_h \multimap \iota$, where $h \geq 0$. Let $N_1^{\mu_1}, \dots, N_k^{\mu_k}, P_0^\mu, P_1^{\tau_1}, \dots, P_h^{\tau_h}$ be closed terms such that $\text{Comp}(N_i)$ and $\text{Comp}(P_j)$ for $1 \leq i \leq k$ and $0 \leq j \leq h$ respectively. Thus $\text{Comp}(Q^\tau[P_0/x][N_1/\mathcal{X}_1, \dots, N_k/\mathcal{X}_k]P_1 \dots P_h)$, since $\text{Comp}(Q^\tau)$ holds by induction hypothesis.
Consider the case $\mu \neq \iota$ and $\llbracket (\lambda x^\mu.Q^\tau)[N_1/\mathcal{X}_1, \dots, N_k/\mathcal{X}_k]P_0 \dots P_h \rrbracket^q \approx_E \llbracket \underline{n} \rrbracket^q$. Thus $\llbracket Q^\tau[P_0/x][N_1/\mathcal{X}_1, \dots, N_k/\mathcal{X}_k]P_1 \dots P_h \rrbracket^q \approx_E \llbracket \underline{n} \rrbracket^q$ by Lemma 4.2.5. Therefore $Q^\tau[P_0/x][N_1/\mathcal{X}_1, \dots, N_k/\mathcal{X}_k]P_1 \dots P_h \rightarrow_{\mathcal{S}\ell}^* \underline{n}$ by induction hypothesis. So, by a β -expansion, $(\lambda x^\mu.Q^\tau)[N_1/\mathcal{X}_1, \dots, N_k/\mathcal{X}_k]P_0 \dots P_h \rightarrow_{\mathcal{S}\ell}^* \underline{n}$.
Suppose $\mu = \iota$ and $\llbracket (\lambda x^\mu.Q^\tau)[N_1/\mathcal{X}_1, \dots, N_k/\mathcal{X}_k]P_0 \dots P_h \rrbracket^q \approx_E \llbracket \underline{n} \rrbracket^q$, thus $\llbracket P_0 \rrbracket^{q'} \approx_E \llbracket \underline{m} \rrbracket^{q'}$ for some \underline{m} by Lemma 4.3.2. But $\text{Comp}(P_0^\mu)$ and $\llbracket P_0 \rrbracket^{q'} \approx_E \llbracket \underline{m} \rrbracket^{q'}$ imply $P_0 \rightarrow_{\mathcal{S}\ell}^* \underline{m}$. Hence $\llbracket Q^\tau[\underline{m}/x][N_1/\mathcal{X}_1, \dots, N_k/\mathcal{X}_k]P_1 \dots P_h \rrbracket^q \approx_E \llbracket \underline{n} \rrbracket^q$, by Lemma 4.3.2. Therefore $Q^\tau[\underline{m}/x][N_1/\mathcal{X}_1, \dots, N_k/\mathcal{X}_k]P_1 \dots P_h \rightarrow_{\mathcal{S}\ell}^* \underline{n}$ by induction. So $(\lambda x^\mu.Q^\tau)[N_1/\mathcal{X}_1, \dots, N_k/\mathcal{X}_k]P_0, P_1 \dots P_h \rightarrow_{\mathcal{S}\ell}^* \underline{n}$ by some β -expansions.

- $M = \mu F.N$. Assume M^σ, N^σ for some type σ . Let $FV(M) = \{\mathcal{X}_1^{\mu_1}, \dots, \mathcal{X}_k^{\mu_k}\}$ for $k \geq 0$ and $\sigma = \tau_1 \multimap \dots \multimap \tau_h \multimap \iota$, where $h \geq 0$. By induction on h , likewise to the corresponding proof of [Plotkin, 1977]. The case $h = 0$ is trivial, so assume $h \geq 1$. Assume $N_1^{\mu_1}, \dots, N_k^{\mu_k}$ and $P_1^{\tau_1}, \dots, P_h^{\tau_h}$ be closed terms such that $\text{Comp}(N_i)$ and $\text{Comp}(P_j)$ for $1 \leq i \leq k$ and $1 \leq j \leq h$ respectively. Let $\llbracket (\mu F.N^\sigma[N_1/\mathcal{X}_1, \dots, N_k/\mathcal{X}_k])P_1 \dots P_h \rrbracket^q \approx_E \llbracket \underline{n} \rrbracket^q$. By Lemma 4.2.4, since $\rightarrow_{\subseteq} \approx_E$, $\llbracket (\mu F.N^\sigma[N_1/\mathcal{X}_1, \dots, N_k/\mathcal{X}_k])P_1 \dots P_h \rrbracket^q \approx_E \llbracket (\mu^k F.N^\sigma[N_1/\mathcal{X}_1, \dots, N_k/\mathcal{X}_k])P_1 \dots P_h \rrbracket^q$ for some $k \in \mathbb{N}$. Thus, $\mu^k F.N^\sigma[Q'/v_1, \dots, Q'/v_m]P_1 \dots P_h \rightarrow_{\mathcal{S}}^* \underline{n}$, by the previous points of this lemma. The proof follows by Lemma 3.2.3. □

Corollary 4.3.1 (Strong adequacy). *Let M be a program and \underline{n} be a numeral.*

$\llbracket M \rrbracket^q \approx_E \llbracket \underline{n} \rrbracket^q$, $M \rightarrow_{\mathcal{S}}^* \underline{n}$ and $\llbracket M \rrbracket^q \rightarrow^* \llbracket \underline{n} \rrbracket^q$ are logically equivalent.

Proof. $\llbracket M \rrbracket^q \approx_E \llbracket \underline{n} \rrbracket^q$ implies $M \rightarrow_{\mathcal{S}}^* \underline{n}$ by Lemma 4.3.3. Moreover, $M \rightarrow_{\mathcal{S}}^* \underline{n}$ implies $\llbracket M \rrbracket^q \rightarrow^* \llbracket \underline{n} \rrbracket^q$ by Theorem 4.2.2. Thus, since $\llbracket M \rrbracket^q \rightarrow^* \llbracket \underline{n} \rrbracket^q$ implies $\llbracket M \rrbracket^q \approx_E \llbracket \underline{n} \rrbracket^q$ the proof is done. □

Consequently, *linProc* give us a syntactical model where we can study the operational equivalence between *S*PCF-programs. Motivations are the game-semantics goals. To provide tools for proving properties of our language and programs. But also, to provide rigorous definitions of implementation instance with good parallel and optimal evaluation features.

Theorem 4.3.1 (Correctness). *If $\llbracket M^\sigma \rrbracket^q \approx_E \llbracket N^\sigma \rrbracket^q$ then $M \approx_\sigma N$.*

Proof. Let $B \vdash M : \sigma$ and $B \vdash N : \sigma$ such that $\llbracket M \rrbracket^q \approx_E \llbracket N \rrbracket^q$. If $C[\sigma]$ is a closing context such that both $C[M]$ and $C[N]$ are programs and $C[M] \rightarrow_{\mathcal{S}}^* \underline{n}$ for some value \underline{n} , then $\llbracket C[M] \rrbracket \rightarrow \llbracket \underline{n} \rrbracket$ by Corollary 4.3.1. So $\llbracket C[N] \rrbracket^q \approx_E \llbracket C[M] \rrbracket^q \approx_E \llbracket \underline{n} \rrbracket^q$, implies $\llbracket C[N] \rrbracket^q \rightarrow^* \llbracket \underline{n} \rrbracket^q$ by Corollary 4.3.1, which implies $C[N] \rightarrow_{\mathcal{S}}^* \underline{n}$ by strong adequacy. By definition of operational equivalence the proof is done. □

4.4 Conclusion

As a final remark, we proposed a process model of *S*PCF showing that, as a consequence of faithfulness, all evaluation strategies of *S*PCF-programs can be represented by our *linProc*-processes. Furthermore, we show that our process model is adequate w.r.t. the operational equivalence of *S*PCF.

One of the key point to obtain faithfulness is the introduction of ground-input prefix in *linProc*, in order to model the call-by value computation. In this way, we got a translation from a calculus (*S*PCF with $\rightarrow_{\mathcal{S}}$ -reduction) into an other calculus (*linProc* with \rightarrow -reduction). A reviewer asked whether it is possible to take a modification of *S*PCF with absolutely no constraint on β -reduction, i.e. a language where both ground and high-order arguments are treated using a call-by name policy, and get a

faithfulness result w.r.t. an opportune process language. We can answer positively to such a question; it is not difficult to see that a suitable process calculus could be a modified version of *linProc* in which ground-input prefix is replaced with a completely asynchronous construct. However we should observe that the so obtained source language does not enjoy denotational linearity in the sense of [Paolini and Piccolo, 2008]. Our purpose includes in fact to use processes to relate the classical denotational models focusing on functional aspects of computation and the new game models focusing on the dynamical (operational) aspects of computation. In particular, the proposed source language does not have a clear denotational status; however it could be an interesting example of functional calculus whose reduction can be mimicked by a fully asynchronous process calculus.

These results are the starting points for many further developments. We are characterising the relevant contexts of *linProc* (i.e. contexts that are able to separate processes corresponding to different programs) in order to tackle the full abstraction of our syntactical model. We are working on a characterisation of processes corresponding to the interpretation of programs, by adapting the proof-nets correctness criterion. We are already able to extend our results on the pair *SLPCF* and *linProc* at the price of some additional technicalities due to pairing-projections codifications. However, we plan to explore process-languages inducing similar results for language more complex languages such as PCF [Plotkin, 1977] or StPCF [Paolini, 2006]. We plan to interpret processes directly on linear coherence spaces, following techniques developed for proof-nets and proof-structures. We plan to define a new kind of game semantics with a more flexible structure, where useless sequentialisation is relaxed. Moreover, we want to explore the application of Levy's optimality theory to the evaluation of programs inside *linProc*. In particular, to tackle the relations between that theory with the notion of operational linearity.

5 Ludics Strategies and the π -calculus

ABSTRACT.

We show a precise correspondence between the finitary fragment of linear π -calculus, introduced by N. Yoshida, K. Honda and M. Berger and an opportune extension of the framework of Girard's Ludics. We need this extension, in order to validate the Mix rule which is not valid in the original Girard's framework and it is necessary to interpret soundly linear π -processes. We present a Ludics model of finitary linear π -calculus, being fully abstract with respect to a suitable notion of operational equivalence. Then, we address the problem of extending these result to an extension of linear π -calculus with recursive definitions.

Introduction

Ludics is a framework introduced by Girard in [Girard, 2001], as a as a foundational, pre-logical framework upon which ordinary logics and type systems are to be built. The basic entities in Ludics are called designs. They are an abstraction of sequent calculus proofs in the Multiplicative Additive fragment of Linear Logic (MALL). The central idea in Ludics is interaction, based on cut-elimination. All structures in Ludics are built in an interactive way and all properties has to be tested by interaction. These aspects bring a more symmetric vision on computation, viewed as an interaction between agents rather than the classical asymmetrical functional application. This point of view is then able to take into account the parallel and concurrent aspects of computation and again it is clearly near to the kind of vision of process calculi like π -calculus [Sangiorgi and Walker, 2001].

The problem we address in this chapter is to study possible connections between the setting of Ludics and process algebra in concurrency theory. In particular we propose Ludics as a tool to study some important properties of processes, like liveness or deadlock-freeness, which can be considered the concurrent counterpart of solvability in λ -calculus.

A connection between Ludics and process calculi appeared to exist since the beginning but a formal connection was not established yet, before this work. In

[Faggian and Maurel, 2005] Faggian and Maurel showed that it is possible to extend the notion of design to L-net, which can be considered an abstraction of a MALL proof-net. The introduction of L-nets took to a development of Ludics in a concurrent sense. In [Curien and Faggian, 2005] an equivalent of sequentialisation theorem for proof-nets was proved in the L-nets setting. Furthermore, some parallelization techniques were introduced in order to transform designs (which can be considered as a completely sequential structures) to L-nets (the maximal form of parallelism). In [Faggian and Piccolo, 2007a, Piccolo, 2006] Faggian and I showed that L-nets are a particular sub-class of Winskel's Event Structures called *confusion free event structures*. These structures model concurrent systems in which the non-deterministic choice is local and it does not depend on independent parts of the system.

In this chapter we propose Ludics as a model for linear π -calculus. Linear π -calculus is a simple typed calculus based on π -calculus, introduced by N. Yoshida, K. Honda and M. Berger. This calculus is flexible enough to study many interesting properties of processes, like strong normalisation [Yoshida et al., 2004], information flow security [Honda et al., 2000] and many others. Moreover it is expressive enough to provide fully abstract encodings of simply typed λ -calculus [Yoshida et al., 2004]. PCF [Berger et al., 2001], System F [Berger et al., 2003] and call-by-value $\lambda\mu$ -calculus [Honda et al., 2004].

We observe that a model of linear π -calculus should validate the Mix rule, i.e. it should be a *-autonomous category with the unit of tensor product as dualizing object. The original framework of Ludics, introduced by Girard, does not validate this rule. Before our work, an extension of Ludics, that validates a limited Mix rule with a polarity condition was proposed in [Curien and Faggian, 2005]. In this work, multiplicative designs may have many minimal actions, provided that these actions were of positive polarity.

In this chapter, we propose an extension of Ludics where the full Mix rule is valid. This means that there could be multiplicative multidesign having more than one minimal negative action. Here the interactive objects (here called *multidesigns*) are particular confusion free event structures (in the same spirit of [Faggian and Piccolo, 2007a]) and by defining normalisation of multidesigns as the categorical pull-back in the category of event structure. We show also that in the so obtained framework, a limited result of separation holds. Finally, to see that the Mix rule is validated, we show that the associated category of multidesigns is compact closed.

We consider then the finitary fragment of linear π -calculus and we give an interpretation of π -processes into multidesigns being fully abstract with respect to a suitable notion of observational equivalence. The full abstraction result is based on a definability argument together with the fact that a separation result holds in the model. Finally we address the problem of extending such a result to a wider and more expressive fragment of linear π -calculus. By using analogous techniques of the previous chapter, we show how to preserve linearity of the calculus during reduction, even in presence of recursion, and we give a sound interpretation of the so obtained calculus into the extended setting of Ludics. We left for future works the issue of showing adequacy and full abstraction.

This chapter is an extended version of [Faggian and Piccolo, 2007b]. The notion of multidesign instead comes from [Faggian and Piccolo, 2009]. Both articles are written with Claudia Faggian.

5.1 Event Structures

Event structures were introduced by Nielsen, Plotkin, and Winskel [Nielsen et al., 1981, Winskel, 1986, Winskel and Nielsen, 1995], as a theory combining Petri nets and domain theory.

Definition 5.1.1 (Event Structure). *An event structure is a triple $\mathcal{E} = \langle E, \leq, \smile \rangle$ such that*

- $\langle E, \leq \rangle$ is a partially ordered set, where the order relation is called causal order. It is such that E is at most countable and the sets $[e] = \{e' \mid e' \leq e\}$ are always finite for all $e \in E$.
- \smile is an irreflexive and symmetric relation, called conflict relation, which satisfies, for every $e_1, e_2, e_3 \in E$:

$$\text{if } e_1 \leq e_2 \text{ and } e_1 \smile e_3 \text{ then } e_2 \smile e_3.$$

We say that the conflict $e_2 \smile e_3$ is *inherited* from the conflict $e_1 \smile e_3$ if $e_1 < e_2$. If a conflict $e_1 \smile e_2$ is not inherited, we say that it is *immediate*, written $e_1 \smile_\mu e_2$. The reflexive closure of the conflict relation (resp. immediate conflict relation) is denoted with \asymp (resp. \asymp_μ). Note that, causal order and conflict are mutually exclusive. Two events e_1 and e_2 which are neither causally ordered nor in conflict are said to be *concurrent*. We will denote $[e) = [e] \setminus \{e\}$ and with $\text{parents}(e)$ the set of maximal elements of $[e)$. Finally, given two events e, d of \mathcal{E} , we will write $e \leftarrow_{\mathcal{E}} d$ and we say that e is an *immediate predecessor* of d , when $e \in \text{parents}(d)$. An event structure \mathcal{E} is said to be *arborescent* if for all $e \in \mathcal{E}$ the sets $[e]$ are totally ordered.

A *configuration* $x \subseteq E$ is a set satisfying

1. *downward closure* i.e. if $e \in x$ and $d \leq e$ then $d \in x$ and
2. *conflict freeness*, i.e. if $e, e' \in x$ then it is never the case that $e \smile e'$.

Therefore, two events of a configuration are either causally related or concurrent, i.e. a configuration represents a run of an event structure where events are partially ordered.

The set of configurations of \mathcal{E} , ordered by inclusion is denoted with $\mathcal{L}(\mathcal{E})$ and it is a coherent Scott domains¹ [Winskel and Nielsen, 1995], whose set of compact elements are finite configurations. A *labelled event structure* is an event structure E together with a labelling function $\lambda : E \rightarrow L$, where L is a set of labels. Events should be thought of as occurrences of actions. Labels allow us to identify events which are occurrences of the same action.

¹A Scott Domain D is said to be *coherent* if for all set $X \subseteq D$ such that for any $x, y \in X$ which are consistent (for the definition of *consistency*, see Section 2.3.2), we have that $\sqcup X \in D$

Notation 5.1.1. In the sequel, to lighten notation we will identify an event structure $\mathcal{E} = \langle E, \leq, \smile \rangle$ with its set of events. So we will write $e \in \mathcal{E}$ or $x \subseteq \mathcal{E}$ in place of $e \in E$ and $x \subseteq E$.

5.1.1 A category of event structures

Event structures form the class of objects of a category [Winskel and Nielsen, 1995]. The morphisms are defined as following.

Definition 5.1.2. Let $\mathcal{E}_1 = \langle E_1, \leq_1, \smile_1 \rangle$ and $\mathcal{E}_2 = \langle E_2, \leq_2, \smile_2 \rangle$ two event structures. A morphism $\varphi : \mathcal{E}_1 \rightarrow \mathcal{E}_2$ is any partial map $\varphi : E_1 \rightarrow E_2$ such that $x \in \mathcal{L}(\mathcal{E}_1)$ implies both $\varphi(x) \in \mathcal{L}(\mathcal{E}_2)$ and for all $e_1, e_2 \in x$, if $\varphi(e_1), \varphi(e_2)$ both defined and $\varphi(e_1) = \varphi(e_2)$ then $e_1 = e_2$.

An equivalent formulation of morphisms between event structures, described in term of the causality and conflict relation is given by the following proposition.

Proposition 5.1.1. Let $\mathcal{E}_1 = \langle E_1, \leq_1, \smile_1 \rangle$ and $\mathcal{E}_2 = \langle E_2, \leq_2, \smile_2 \rangle$ two event structures. A morphism $\varphi : \mathcal{E} \rightarrow \mathcal{E}'$ is any partial map $\varphi : E \rightarrow E'$ satisfying

- if $e' \leq \varphi(d)$ then there exists $e \leq d$ such that $\varphi(e) = e'$
- if $\varphi(e), \varphi(d)$ both defined and $\varphi(e) \smile \varphi(d)$ then $e \smile d$.

The initial object of the category is clearly the empty structure, which is also the terminal one. Furthermore, this category admits all finite limits [Winskel and Nielsen, 1995]. For what follows, it is useful to remind the definition of co-product between event structures, given in [Winskel and Nielsen, 1995]. Intuitively, it is obtained from the disjoint union of two event structure, by putting in conflict events coming from different event structures.

Definition 5.1.3 (Co-product). Let $\mathcal{E}_1 = \langle E_1, \leq_1, \smile_1 \rangle$ and $\mathcal{E}_2 = \langle E_2, \leq_2, \smile_2 \rangle$ be two event structures. We denote with $\mathcal{E}_1 + \mathcal{E}_2 = \langle E, \leq, \smile \rangle$ the co-product of \mathcal{E}_1 and \mathcal{E}_2 where

- $E = E_1 + E_2$.
- given $e, d \in E$, we let $e \leq d$ if either $e = in_1(e_1), d = in_2(d_1)$ and $e_1 \leq_1 d_1$, or $e = in_2(e_2), d = in_2(d_2)$ and $e_2 \leq_2 d_2$.
- given $e, d \in E$, we let $e \smile d$ if one of the following holds
 - $e = in_1(e_1)$ and $d = in_2(e_2)$ for some $e_1 \in E_1$ and $e_2 \in E_2$.
 - $e = in_1(e_1), d = in_1(d_1)$ and $e_1 \smile_1 d_1$.
 - $e = in_2(e_2), d = in_2(d_2)$ and $e_2 \smile_2 d_2$.

There are various way of dealing with labels. For a general treatment, we refer to [Winskel and Nielsen, 1995]. Here we present the simplest notion: take two labelled event structure $\mathcal{E}_1 = \langle E_1, \leq_1, \smile_1, \lambda_1 \rangle$ and $\mathcal{E}_2 = \langle E_2, \leq_2, \smile_2, \lambda_2 \rangle$ on the same set of labels L . A morphism $f : \mathcal{E}_1 \rightarrow \mathcal{E}_2$ is said to be *label preserving* if, whenever $f(e_1)$ is defined, $\lambda_2(f(e_1)) = \lambda_1(e_1)$.

5.1.2 Construction on event structures

We define several constructions on labelled event structures. See [Winskel, 1986] for more details.

Prefixing. Let $\mathcal{E} = \langle E, \leq, \smile, \lambda \rangle$ be a labelled event structure, and let $a \in L$ be a label. Then $a.\mathcal{E}$ is the labelled event structure obtained from \mathcal{E} by adding a new minimum event e (with respect to the causal order) which is labelled with a . Conflict and order on remaining events remains the same as \mathcal{E} .

Restriction. Let $\mathcal{E} = \langle E, \leq, \smile, \lambda \rangle$ be a labelled event structure and let $X \subseteq L$ be a set of labels. We denote with $\mathcal{E} \setminus X$ to be the labelled event structure obtained from \mathcal{E} by removing all the events having labels in X and all other events that are above one of those, according to the causal order.

Disjoint union. Let $\mathcal{E}_i = \langle E_i, \leq_i, \smile_i, \lambda_i \rangle$ be a family of event structures. We denote with $\uplus_i \mathcal{E}_i = \langle \uplus_i E_i, \uplus_i \leq_i, \uplus_i \smile_i, \uplus_i \lambda_i \rangle$.

Relabelling. Let $\mathcal{E} = \langle E, \leq, \smile, \lambda \rangle$ be a labelled event structure and let $\theta : L \rightarrow L$ be a relabelling function. We denote with $\theta(\mathcal{E}) = \langle E, \leq, \smile, \theta \circ \lambda \rangle$.

5.1.3 A cpo of event structures

The following definition has been taken in [Crafa et al., 2007], where it is said that it is equivalent to the ordering among event structures defined by Winskel in [Winskel, 1982].

Definition 5.1.4. An event structure \mathcal{E} is a prefix of an event structure \mathcal{E}' , denoted $\mathcal{E} \sqsubseteq \mathcal{E}'$ if there exists $\mathcal{E}'' \cong \mathcal{E}'$ such that $\mathcal{E} \subseteq \mathcal{E}''$ and for all $e \in \mathcal{E}$ there are no $d \in \mathcal{E}'' \setminus \mathcal{E}$ such that $d \leq e$.

Proposition 5.1.2. Let \mathcal{E}_1 and \mathcal{E}_2 be two event structures. Then $\mathcal{E}_1 \sqsubseteq \mathcal{E}_2$ if and only if \mathcal{E}_2 contains \mathcal{E}_1 as retract.

Proof. Suppose $\mathcal{E}_1 \sqsubseteq \mathcal{E}_2$. Then there is an event structure $\mathcal{E}'_2 \cong \mathcal{E}_2$ such that $\mathcal{E}_1 \subseteq \mathcal{E}'_2$. We let $i : \mathcal{E}_1 \rightarrow \mathcal{E}'_2$ to be the inclusion function. It is an event structure morphism since by definition $i(e) \asymp i(e') \iff e \asymp e'$. Moreover if $d \leq i(e)$, we cannot have $d \notin \mathcal{E}_1$ again by definition. Thus the functions $\varphi \circ i : \mathcal{E}_1 \rightarrow \mathcal{E}_2$ and $\varphi^{-1} \circ i^{-1} : \mathcal{E}_2 \rightarrow \mathcal{E}_1$ forms a retraction pair, by construction. To prove the converse, suppose that $\mathcal{E}_2 = \langle E_2, \leq_2, \smile_2 \rangle$ contains $\mathcal{E}_1 = \langle E_1, \leq_1, \smile_1 \rangle$ as retract through the retraction pair $\langle \varphi, \psi \rangle$, where $\varphi : \mathcal{E}_1 \rightarrow \mathcal{E}_2$ and $\psi : \mathcal{E}_2 \rightarrow \mathcal{E}_1$. Let us build the event structure $\mathcal{E} = \langle E, \leq, \smile \rangle$ as

- $E = E_1 \cup \{e_2 \in E_2 \mid e_2 \notin \varphi(E_1)\}$
- $e_1 \leq e_2$ if one of the following holds
 - $e_1 \in E_1, e_2 \in E_1$ and $e_1 \leq_1 e_2$
 - $e_1 \in E_2, e_2 \in E_2$ and $e_1 \leq_2 e_2$

- $e_1 \in E_1, e_2 \in E_2$ and there exists $e \leq_2 e_2$ such that $\psi(e) = e_1$
- $e_1 \smile e_2$ if one of the following holds
 - $e_1 \in E_1, e_2 \in E_1$ and $e_1 \smile_1 e_2$
 - $e_1 \in E_2, e_2 \in E_2$ and $e_1 \smile_2 e_2$
 - $e_1 \in E_1, e_2 \in E_2$ and there exists $e \smile_2 e_2$ such that $\psi(e) = e_1$

By construction $\mathcal{E} \cong \mathcal{E}_2$ and $\mathcal{E}_1 \subseteq \mathcal{E}$. Moreover, there is no $d \in \mathcal{E} \setminus \mathcal{E}_1$ such that $d \leq e$. \square

Winskel [Winskel, 1982] has shown that the class of event structures with the prefix order is a large CPO and thus the limits of countable increasing chain exists. The set of compact elements of the domain is given by *finite event structures* (an event structure is *finite* when its set of events is) and every event structure is the supremum of the set of finite event structures which are below it with respect to the prefix order.

Proposition 5.1.3 ([Winskel, 1982]). *Let \mathcal{E} be an event structure. Then*

$$\mathcal{E} = \bigsqcup \{ \mathcal{E}' \sqsubseteq \mathcal{E} \mid \mathcal{E}' \text{ finite} \}$$

Proposition 5.1.4 ([Winskel, 1982]). *Prefixing, restriction, disjoint union, relabelling, products and co-products are continuous with respect to the prefix order.*

5.1.4 Confusion free event structures

An interesting subclass of event structures is the following

Definition 5.1.5 (Conflict free event structure). *An event structure is conflict free when its conflict relation is empty.*

An other interesting subclass of event structure is the class of confusion free event structures. Confusion free event structures are a class of event structures where every choice is localised. In order to formalise such idea we need a notion of locality, given by the following definition.

Definition 5.1.6 (Cell). *A partial cell c is a set c of events such that $e, e' \in c$ implies $e \asymp_\mu e'$ and $[e] = [e']$. A maximal partial cell is called a cell.*

In general, two events in immediate conflict do not belong to the same cell. If a cell is thought of as a location, this means that not all conflicts are localised. This leads to the following definition.

Definition 5.1.7 (Confusion free event structure). *An event structure is confusion free if its cells are closed under immediate conflict.*

Equivalently, in a confusion free event structure, \asymp_μ is an equivalence relation having cells as its equivalence classes. More specifically we have the following

Proposition 5.1.5. *An event structure \mathcal{E} is confusion free when \asymp_μ is a transitive relation and such that, given any two events e_1, e_2 such that $e_1 \asymp_\mu e_2$ we have $[e_1] = [e_2]$.*

5.2 Ludics

The computational objects of Ludics are designs, which can be seen as a linear version of Hyland-Ong innocent strategies. Designs are given by a forest of actions and by an interface, playing the role of an arena.

In this section we introduce an immediate generalisation of the setting of Girard's Ludics [Girard, 2001], where the interactive objects (here called *multidesigns* instead of *designs*) are event structures. It is possible to show that this extension is conservative, in the sense that, when restricting to multidesigns which are also designs, the definitions we give are equivalent. Some indications can be found in [Faggian and Piccolo, 2007a, Piccolo, 2006].

In the following, given two string s, t on a given alphabet, we denote with $s \cdot t$ their concatenation and we write $s \leq t$ if s is a prefix of t .

5.2.1 Loci, actions and designs

A **locus** is a string of natural numbers. We use $\xi, \sigma, \tau \dots$ to range over loci.

If $\xi \leq \xi'$, we also say that ξ' is a **sub-locus** of ξ . We say that two loci ξ and ξ' are **disjoint** when none of them is prefix of the other.

A *base* is a finite set $\{\xi_1^{\epsilon_1}, \dots, \xi_n^{\epsilon_n}\}$ where ξ_i is a locus and $\epsilon_i \in \{+, -\}$ is a polarity, and it is such that for every two distinct i, j we have ξ_i disjoint with ξ_j . The polarity $+$ is said to be *positive* while the polarity $-$ is *negative*; they are mutually dual polarities. We use Greek capital letters Γ, Δ, \dots to range over bases. A base is *negative* when it contains at least a negative locus, positive otherwise. Observe that in particular the empty base \emptyset is positive. We will denote with Γ^e a base Γ having all loci of polarity e .

A base induces a *polarisation* of sub-loci ξ' of a given $\xi \in \Gamma$: if $\xi' = \xi \cdot \tau$, its polarity is the same as ξ , if τ is of even length, opposite otherwise. Given a base Γ , we denote with Γ^\perp the base obtained from Γ by exchanging all the polarities of its loci ($+$ with $-$ and $-$ with $+$). Given two bases Γ, Δ whose loci are pairwise disjoint, we denote with $\Gamma \otimes \Delta$ the base obtained from the union of Γ and Δ ; we also denote $\Gamma \multimap \Delta = \Gamma^\perp \otimes \Delta$.

Let us assume to have a an infinite (countable) set $Ram \subseteq \wp_{fin}(\mathbb{N})$ such that for all $i \in \mathbb{N}$, there is a unique $I \in Ram$ such that $i \in I$. Observe that this implies that, for any two distinct $I, J \in Ram$ we have $I \cap J = \emptyset$. An **action** κ is either the symbol \blacklozenge called *daimon*, or a pair (ξ, I) , called *proper action*, where ξ (**focus**) is a locus and $I \in Ram$.

We will write $\xi \cdot I$ for $\{\xi \cdot i \mid i \in I\}$, which are the loci generated by the action (ξ, I) . Given two actions $\kappa_1 = (\xi, I)$ and κ_2 we say that κ_1 *justifies* κ_2 (written $\kappa_1 \vdash \kappa_2$) when $\kappa_2 = (\sigma, J)$ and $\sigma \in \xi I$. Observe that

1. there is no action κ such that $\kappa \vdash \blacklozenge$ and $\blacklozenge \vdash \kappa$.
2. if $(\xi, I), (\xi, J)$ are two distinct actions then $\xi I \cap \xi J = \emptyset$. This means that, given κ_1, κ_2 having the same focus and given κ be such that both $\kappa_1 \vdash \kappa$ and $\kappa_2 \vdash \kappa$, we have that $\kappa_1 = \kappa_2$.

Given a base Γ we denote with \mathcal{A}_Γ the **arena of proper actions generated by Γ** and it is the event structures $\langle A_\Gamma, \leq_\Gamma, \smile_\Gamma \rangle$ where

- $A_\Gamma = \{\kappa \mid \kappa = (\sigma, I), \exists \xi \in \Gamma. \xi \leq \sigma\}$
- \leq_Γ is the reflexive and transitive closure of \vdash
- given two distinct κ_1, κ_2 we have $\kappa_1 \smile_\Gamma \kappa_2$ when there are $\kappa'_1 \leq \kappa_1$ and $\kappa'_2 \leq \kappa_2$ such that κ_1 and κ_2 have the same focus.

Observe that the relation \smile_Γ is a well defined conflict relation; in particular, it is anti-reflexive because of the point (2.) of the observations made above. Moreover, if $\kappa_1 \smile_\mu \kappa_2$ then κ_1 and κ_2 have the same focus and thus the same polarity. Moreover \mathcal{A}_Γ is confusion free and arborescent. We will denote $\mathcal{A}_\Gamma^\boxtimes = \mathcal{A}_\Gamma \uplus \{\boxtimes\}$.

Let $\kappa \in \mathcal{A}_\Gamma$, where Γ is a base. Its **polarity** is the one of its focus according to the base Γ . The **polarity** of \boxtimes is defined to be always positive. An action $\kappa \in A_\Gamma$ is said *initial* when there is no $\kappa' \in A_\Gamma$ such that $\kappa' \vdash \kappa$.

Definition 5.2.1 (Multidesign). *Let Γ be base. A multidesign \mathfrak{D} on the base Γ (written $\mathfrak{D} : \Gamma$) is an arborescent confusion free labelled event structure $\langle D, \leq_\mathfrak{D}, \smile_\mathfrak{D}, \lambda_\mathfrak{D} \rangle$ where $\lambda_\mathfrak{D} : \mathfrak{D} \rightarrow \mathcal{A}_\Gamma^\boxtimes$ is a total labelling function satisfying the following conditions.*

Alternation. *If $e_1, e_2 \in D$ are such that $e_1 \leftarrow_\mathfrak{D} e_2$ then $\lambda_\mathfrak{D}(e_1)$ and $\lambda_\mathfrak{D}(e_2)$ have opposite polarity.*

Justification. *If $\kappa \leq_\Gamma \lambda_\mathfrak{D}(e_2)$ then there is $e_1 \leq_\mathfrak{D} e_2$ such that $\lambda_\mathfrak{D}(e_1) = \kappa$.*

Linearity. *Let $e_1, e_2 \in \mathfrak{D}$ be such that $\lambda_\mathfrak{D}(e_1), \lambda_\mathfrak{D}(e_2)$ are proper actions. If $\lambda_\mathfrak{D}(e_1) \asymp \lambda_\mathfrak{D}(e_2)$ then $e_1 \asymp e_2$.*

Innocence. *Let $e \in \mathfrak{D}$ such that $\lambda_\mathfrak{D}(e)$ is a negative action. If it is initial, then e is minimal in \mathfrak{D} . Otherwise there is $e' \leftarrow_\mathfrak{D} e$ such that $\lambda_\mathfrak{D}(e') \vdash \lambda_\mathfrak{D}(e)$.*

Additive Coherence. *Let e_1, e_2 be such that $e_1 \smile_\mu e_2$. Then $\lambda_\mathfrak{D}(e_1) \smile_\mu \lambda_\mathfrak{D}(e_2)$ and their polarity is negative.*

Maximality. *If e is maximal in \mathfrak{D} then $\lambda_\mathfrak{D}(e)$ is positive.*

Let \mathfrak{D} be a multidesign on Γ . Observe that $\lambda_\mathfrak{D} : \mathfrak{D} \rightarrow \mathcal{A}_\Gamma^\boxtimes$ is not necessarily an event structure morphism, because there could be many events belonging to the same configuration of \mathfrak{D} which are labelled with \boxtimes . We also observe that if there is an $e \in \mathfrak{D}$ such that $\lambda_\mathfrak{D}(e) = \boxtimes$, then by Alternation and by Innocence e is forced to be maximal in \mathfrak{D} . Thus, we can state the following proposition.

Proposition 5.2.1. *Let $\lambda_\mathfrak{D}^A : \mathfrak{D} \rightarrow \mathcal{A}_\Gamma$ be the morphism defined as*

$$\lambda_\mathfrak{D}^A(e) = \begin{cases} \lambda_\mathfrak{D}(e) & \text{if } \lambda_\mathfrak{D}(e) \text{ is proper} \\ \text{undefined} & \text{otherwise} \end{cases}$$

Then $\lambda_\mathfrak{D}^A$ is an event structure morphism between \mathfrak{D} and \mathcal{A}_Γ .

Proof. By the observation above, we note that the condition Justification and Linearity exactly correspond to the two condition required to be a morphism between event structures, since Proposition 5.1.1. \square

Among multidesigns, we define an order, which is inherited from the prefix order among event structures (see Section 5.1.3).

Definition 5.2.2 (Stable order). *Let $\mathfrak{D}_1, \mathfrak{D}_2$ two multidesigns on the same base Γ . We say that \mathfrak{D}_1 is less or equal than \mathfrak{D}_2 according to the **stable order** and we write $\mathfrak{D}_1 \sqsubseteq_d \mathfrak{D}_2$, if \mathfrak{D}_2 contains as retract \mathfrak{D}_1 and the retraction pair consists of label preserving morphisms.*

Observe that, by Proposition 5.1.2, if $\mathfrak{D}_1 \sqsubseteq \mathfrak{D}_2$ then $\mathfrak{D}_1 \sqsubseteq_d \mathfrak{D}_2$. We will write $\mathfrak{D}_1 =_d \mathfrak{D}_2$ to denote $\mathfrak{D}_1 \sqsubseteq_d \mathfrak{D}_2$ and $\mathfrak{D}_2 \sqsubseteq_d \mathfrak{D}_1$ (i.e. when they are isomorphic). It is possible to prove that the class of multidesigns ordered according to \sqsubseteq_d is an ω -algebraic CPO whose compact elements are finite multidesigns.

We now give some example of designs which will be useful in the sequel.

Multidesigns on the empty base. Let $\mathfrak{D} : \emptyset$. Then it consists of a conflict free event structure $\mathfrak{D}\text{ai}^n$ ($n \in \mathbb{N} \cup \{\infty\}$), having $[1, n]$ as set of its events, where $i \leq_{\mathfrak{D}\text{ai}^n} j$ iff $i = j$ and $\lambda_{\mathfrak{D}\text{ai}^n}(i) = \blacktriangleright$ for all $i \in [1, n]$. For example, the least multidesign on the empty base is $\mathfrak{D}\text{ai}^0$ which is sometimes denoted with $\mathfrak{F}\text{id}$, while the multidesign $\mathfrak{D}\text{ai}^\infty$ is the maximum multidesign on the empty base; it has infinite countable minimal events, all labelled with \blacktriangleright .

The skunk. Let Γ be a negative base. The empty event structure is a multidesign on Γ and it is denoted with $\mathfrak{S}\text{funf}$.

Constructions on multidesigns

We define several construction on designs.

- Multidesign prefixing.**
1. Let \mathfrak{D} be a multidesign on the base $\{\sigma^+ \mid \sigma \in \xi I\} \otimes \Gamma^+$. We denote $(\xi, I) \ominus \mathfrak{D} = \mathfrak{C}$ on base $\{\sigma^-\} \otimes \Gamma^+$, where $\mathfrak{C} = \mathfrak{S}\text{funf}$ if $\mathfrak{D} = \mathfrak{F}\text{id}$, while \mathfrak{C} is obtained by the prefixing $(\xi, I).\mathfrak{D}$ of the underlying labelled event structure.
 2. Let \mathfrak{D} be a multidesign on the base $\{\sigma^- \mid \sigma \in \xi I\} \otimes \Gamma^+$, such that all its minimal events are labelled with actions having focus in ξI . Then we define $(\xi, I) \ominus \mathfrak{D} = (\xi, I).\mathfrak{D}$ which is a multidesign on base $\{\xi^+\} \otimes \Gamma^+$.

Sum. Let $\{\mathfrak{D}_i\}$ be a family of multidesigns on the base $\{\xi^-\} \otimes \Gamma^+$, having all their minimal events labelled with actions having focus ξ . We denote $R_\xi(\mathfrak{D}_i) = \{I \mid \exists d \in \mathfrak{D}_i. \lambda_{\mathfrak{D}_i}(d) = (\xi, I)\}$. Under the hypothesis that, given two distinct i, j , we have $R_\xi(\mathfrak{D}_i) \cap R_\xi(\mathfrak{D}_j) = \emptyset$, we define $\sum_i \mathfrak{D}_i$ to be the co-product of the underlying event structures. When the sum is binary, we denote it with $\mathfrak{D}_1 + \mathfrak{D}_2$. Observe that $\mathfrak{S}\text{funf}$ is the neutral element of the sum.

It is not difficult to see that both the operations are continuous with respect to the stable ordering among multidesigns.

5.2.2 Normalisation

In this section we define the notion of normalisation of designs. It is obtained as the composition of two operations, called respectively *parallel composition* and *hiding*.

Parallel Composition

Parallel composition will be defined as an opportune restriction of the pull back of the two underlying event structure being composed. Observe that the category of event structure admits pull-back for every pair of morphisms, since the category have all finite limits [Winskel and Nielsen, 1995].

Let $\mathfrak{D}_1 : \Gamma_1 \multimap \Lambda$ and $\mathfrak{D}_2 : \Lambda \multimap \Gamma_2$ be two multidesigns. Let $\varphi_i : \mathfrak{D}_i \rightarrow \mathcal{A}_\Lambda$ ($i \in \{1, 2\}$) to be such that

$$\varphi_i(e) = \begin{cases} \lambda_{\mathfrak{D}_i}^A(e) & \text{if } \lambda_{\mathfrak{D}_i}^A(e) \in \mathcal{A}_\Lambda \\ \text{undefined} & \text{otherwise} \end{cases}$$

We denote with $\mathfrak{D}_1 * \mathfrak{D}_2$ the event structure obtained from the pull back of \mathfrak{D}_1 and \mathfrak{D}_2 along φ_1 and φ_2 where projections are respectively denoted with $p_1 : \mathfrak{D}_1 * \mathfrak{D}_2 \rightarrow \mathfrak{D}_1$ and $p_2 : \mathfrak{D}_1 * \mathfrak{D}_2 \rightarrow \mathfrak{D}_2$. We call the morphisms φ_i *synchronisation morphisms*. A useful property of the pull-back is the following, which will be used many times in the following.

Lemma 5.2.1. *Let $e_1, e_2 \in \mathfrak{D}_1 * \mathfrak{D}_2$, such that $e_1 \leftarrow e_2$. Then either $p_1(e_1) \leftarrow p_1(e_2)$ or $p_2(e_1) \leftarrow p_2(e_2)$.*

Proof. Let $e_1, e_2 \in \mathfrak{D}_1 * \mathfrak{D}_2$. First of all, observe that either $p_1(e_1), p_1(e_2)$ are both defined or $p_2(e_1), p_2(e_2)$ are both defined. To see this we reason by contradiction, showing that $\mathfrak{D}_1 * \mathfrak{D}_2$ does not enjoy the universal property. More specifically, if this does not hold, we can take an event structure \mathcal{E} whose events, causal order and conflict are the same as $\mathfrak{D}_1 * \mathfrak{D}_2$ except for the fact that e_2 is concurrent with e_1 (so all events above e_2 are not necessarily causally related with e_1). If we take the same projection morphisms for \mathcal{E} , we have that the pull back diagram still commutes by construction, but we are not able to construct the morphism $f : \mathcal{E} \rightarrow \mathfrak{D}_1 * \mathfrak{D}_2$, since the only candidate (the identity function) is not an event structure morphism, reaching a contradiction. Suppose that $p_1(e_1), p_1(e_2)$ are both defined. So $p_1(e_1) \leq p_1(e_2)$ since p_1 is an event structure morphism. If $p_1(e_1) \leftarrow p_1(e_2)$ then we are done. Otherwise there is d_1 such that $p_1(e_1) \leq d_1 \leftarrow p_1(e_2)$. Since p_1 is an event structure morphism there is $e \leq e_2$ such that $p_1(e) = d_1$. Observe that, for the same reason $e_1 \leq e$, so we can conclude. The other case is similar. \square

Let $e \in \mathfrak{D}_1 * \mathfrak{D}_2$. If $p_1(e), p_2(e)$ are both defined, then we can think of e as the result of the synchronisation of $p_1(e) = e_1$ and $p_2(e) = e_2$. If the synchronisation morphisms are both defined on e_1 and e_2 then the two events are labelled with the same action (with opposite polarity) in \mathfrak{D}_1 and \mathfrak{D}_2 , by construction. In this case we will talk of *good synchronisation*. If the synchronisation morphisms are not defined on e_1 and e_2 then we will talk of *bad synchronisation*.

$\mathfrak{D}_1 * \mathfrak{D}_2$ is a labelled event structure by defining the labelling function $\lambda_{\mathfrak{D}_1 * \mathfrak{D}_2} : \mathfrak{D}_1 * \mathfrak{D}_2 \rightarrow L$, where $L = \mathcal{A}_{\Gamma_1} \cup \mathcal{A}_{\Gamma_2} \cup \mathcal{A}_{\Lambda} \cup \{\mathbf{X}\} \cup \{undef\}$, as follows,

$$\lambda_{\mathfrak{D}_1 * \mathfrak{D}_2}(e) = \begin{cases} \lambda_{\mathfrak{D}_1}(e) & \text{if } p_1(e) \text{ defined, } p_2(e) \text{ undefined} \\ \lambda_{\mathfrak{D}_2}(e) & \text{if } p_2(e) \text{ defined, } p_1(e) \text{ undefined} \\ \lambda_{\mathfrak{D}_1}^A(p_1(e)) = \lambda_{\mathfrak{D}_2}^A(p_2)(e) & \text{if } p_1(e), p_2(e), \lambda_{\mathfrak{D}_1}^A(p_1(e)) \text{ all defined} \\ undef & \text{otherwise} \end{cases}$$

Note that all events which are obtained as result of a bad synchronisation are labelled with *undef*. Let us see now an example of bad synchronisation.

Example. Let $\Gamma = \{\xi^+\}$ and $\Delta = \{\sigma^+\}$. Let $\mathfrak{D}_1 : \Gamma$ and $\mathfrak{D}_2 : \Delta$ be two multidesigns such that

- $\mathfrak{D}_1 = \{e_1\}$ with $\lambda_{\mathfrak{D}_1}(e_1) = (\xi, I)$
- $\mathfrak{D}_2 = \{e_2\}$ with $\lambda_{\mathfrak{D}_2}(e_2) = (\sigma, J)$

Then $\mathfrak{D}_1 * \mathfrak{D}_2$ is the event structure such that

- its set of events is $\{d_1, d_2, d_3\}$ where $p_1(d_1) = e_1$, $p_2(d_1)$ is undefined, $p_1(d_2) = e_1$, $p_2(d_2) = e_2$, $p_1(d_3)$ is undefined and $p_2(d_3) = e_2$
- d_1, d_2, d_3 are incomparable while we have $d_1 \smile d_2$ and $d_3 \smile d_2$.

Observe that $\lambda_{\mathfrak{D}_1 * \mathfrak{D}_2}(d_3) = undef$ since e_1 and e_2 are not labelled with the same label, thus d_3 is a bad synchronisation. Observe that, in this case, since the pull-back of \mathfrak{D}_1 and \mathfrak{D}_2 is made along the terminal object, we have that $\mathfrak{D}_1 * \mathfrak{D}_2$ is equivalent to the categorical product of $\mathfrak{D}_1 \times \mathfrak{D}_2$ of the underlying event structures.

Parallel composition of two multidesigns is obtained by removing from the pull back all events coming from a bad synchronisation.

Definition 5.2.3 (Parallel composition). Let $\mathfrak{D}_1 : \Gamma_1 \multimap \Lambda$ and $\mathfrak{D}_2 : \Lambda \multimap \Gamma_2$ be two multidesigns. We define $\mathfrak{D}_1 \parallel \mathfrak{D}_2 = \mathfrak{D}_1 * \mathfrak{D}_2 \setminus \{undef\}$.

Remark 5.2.1. Notice that also Girard uses the notion of “pull-back” in [Girard, 2001], related to the definition of parallel composition. However the term “pull back” used in [Girard, 2001] does not correspond to the usual categorical construction. When in [Girard, 2001], Girard talk about the pull back of $\mathfrak{D}_1 \parallel \mathfrak{D}_2$ he refers to the two designs \mathfrak{D}_1 and \mathfrak{D}_2 .

Remark 5.2.2. We would like to define a notion of polarity for the actions in L , in the analogous way we have done for multidesigns. Unfortunately, there is an ambiguity on what is the polarity of the actions of \mathcal{A}_{Λ} for the event structure $\mathfrak{D}_1 \parallel \mathfrak{D}_2$, since they are both positive and negative. Thus, we introduce an additional polarity, called neutral (denoted with \pm) for all actions belonging to \mathcal{A}_{Λ} . The polarity of actions in \mathcal{A}_{Γ_1} and \mathcal{A}_{Γ_2} are defined as usual according to the bases Γ_1^+ and Γ_2 .

Let us see an example of parallel composition.

Example. Let $\Gamma = \{\xi^-\}$ and $\Delta = \{\sigma^-\}$. Let $\mathfrak{D}_1 : \Delta$ and $\mathfrak{D}_2 : \Delta \multimap \Gamma$ be two multidesigns such that

- $\mathfrak{D}_1 = \{e_1, e_2\}$ with $e_1 \leq e_2$, the empty conflict relation and $\lambda_{\mathfrak{D}_1}(e_1) = (\sigma, I)$ and $\lambda_{\mathfrak{D}_1}(e_2) = \mathbf{X}$.
- $\mathfrak{D}_2 = \{d_1, d_2\}$ with $d_1 \leq d_2$, the empty conflict relation and $\lambda_{\mathfrak{D}_2}(d_1) = (\xi, J)$ and $\lambda_{\mathfrak{D}_2}(d_2) = (\sigma, I)$

Then $\mathfrak{D}_1 \parallel \mathfrak{D}_2$ is the event structure such that

- its set of events is $\{e, e', e''\}$ where $p_1(e)$ is undefined, $p_2(e) = e_1$, $p_1(e') = e_1, p_2(e') = d_2$, $p_1(e'') = e_2$, $p_2(e'')$ is undefined.
- the causal order is $e \leq e' \leq e''$ and the conflict relation is empty.

Observe that e' is labelled with the action (σ, I) , whose polarity is neutral.

About parallel composition, we can prove the following proposition.

Theorem 5.2.1 (Associativity). $\mathfrak{D}_1 \parallel (\mathfrak{D}_2 \parallel \mathfrak{D}_3) = (\mathfrak{D}_1 \parallel \mathfrak{D}_2) \parallel \mathfrak{D}_3$.

Proof. Let us observe that $(\mathfrak{D}_1 * (\mathfrak{D}_2 * \mathfrak{D}_3)) \setminus \{undef\}$ is isomorphic to $(\mathfrak{D}_1 * ((\mathfrak{D}_2 * \mathfrak{D}_3) \setminus \{undef\})) \setminus \{undef\}$. In the same way, we have $((\mathfrak{D}_1 * \mathfrak{D}_2) * \mathfrak{D}_3) \setminus \{undef\}$ is isomorphic to $((\mathfrak{D}_1 * \mathfrak{D}_2) \setminus \{undef\}) * \mathfrak{D}_3 \setminus \{undef\}$. This can be shown by observing that by construction, when computing the pull-back, events labelled with *undef* are never in the domain of definition of the synchronisation morphisms. Thus we can conclude by the universal property of the pull-back. \square

Theorem 5.2.2. Let $\mathfrak{D}_1 : \Gamma_1 \multimap \Lambda$ and $\mathfrak{D}_2 : \Lambda \multimap \Gamma_2$ be two multidesigns. Let $\mathfrak{D}_1 \parallel \mathfrak{D}_2$ be the parallel composition. Then

1. $\mathfrak{D}_1 \parallel \mathfrak{D}_2$ is a confusion free arborescent event structure.
2. $\lambda_{\mathfrak{D}_1 \parallel \mathfrak{D}_2}$ satisfies Innocence, Additive Coherence, Justification and Linearity.
3. Let $e_1, e_2 \in \mathfrak{D}_1 \parallel \mathfrak{D}_2$ such that $e_1 \leftarrow_{\mathfrak{D}_1 \parallel \mathfrak{D}_2} e_2$. Then either $\lambda_{\mathfrak{D}_1 \parallel \mathfrak{D}_2}(e_1)$ and $\lambda_{\mathfrak{D}_1 \parallel \mathfrak{D}_2}(e_2)$ are opposite, either they are both \pm , either the former is negative and the latter is neutral or the former is neutral and the latter is positive.

Proof. The proof will be deferred to Section 5.2.5. \square

As the previous proposition shows, parallel composition is associative and almost all properties of multidesigns, except Alternation and Maximality. However, point (3.) of proposition above shows that Alternation is also preserved, modulo the event labelled with neutral actions. The only property which is not preserved under parallel composition is Maximality, as shown by the following example.

Example. Let $\Gamma = \{\xi^+\}$ and $\Lambda = \{\sigma^+\}$. Let $\mathfrak{D}_1 : \Gamma \multimap \Lambda$ and $\mathfrak{D}_2 : \Lambda^\perp$ such that

- $\mathfrak{D}_1 = \{e_1, d_1\}$ with $e_1 \leq_{\mathfrak{D}_1} d_1$, the empty conflict relation and the labelling given by $\lambda_{\mathfrak{D}_1} = (\xi, \emptyset)$ and $\lambda_{\mathfrak{D}_1} = (\sigma, \emptyset)$.
- $\mathfrak{D}_2 = \text{Gfunf.}$

Then $\mathfrak{D}_1 \parallel \mathfrak{D}_2 = \{e\}$ with $\lambda_{\mathfrak{D}_1 \parallel \mathfrak{D}_2} = (\xi, \emptyset)$ and the polarity of e is negative.

This motivates the following definition.

Definition 5.2.4. We define $(\mathfrak{D}_1 \parallel \mathfrak{D}_2)^\bullet$ to be the maximum event structure being prefix of $\mathfrak{D}_1 \parallel \mathfrak{D}_2$ and satisfying maximality, i.e.

$$(\mathfrak{D}_1 \parallel \mathfrak{D}_2)^\bullet = \bigsqcup \{ \mathfrak{D} \mid \mathfrak{D} \sqsubseteq_d \mathfrak{D}_1 \parallel \mathfrak{D}_2, \mathfrak{D} \text{ enjoys Maximality} \}$$

Observe that by construction, $\mathfrak{D}_1 \parallel \mathfrak{D}_2$ contains as retract $(\mathfrak{D}_1 \parallel \mathfrak{D}_2)^\bullet$. Thus, it is not difficult to extend Theorem 5.2.2 also for the operation $(- \parallel -)^\bullet$. The most delicate result to extend is Associativity (Theorem 5.2.1), which holds also for the extended operation.

Lemma 5.2.2. Let $\mathfrak{D}_1 : \Gamma \multimap \Lambda$, $\mathfrak{D}_2 : \Lambda \multimap \Delta$ and $\mathfrak{D}_3 : \Delta \multimap \Xi$. Then

1. $((\mathfrak{D}_1 \parallel \mathfrak{D}_2)^\bullet \parallel \mathfrak{D}_3)^\bullet =_d ((\mathfrak{D}_1 \parallel \mathfrak{D}_2) \parallel \mathfrak{D}_3)^\bullet$.
2. $(\mathfrak{D}_1 \parallel (\mathfrak{D}_2 \parallel \mathfrak{D}_3)^\bullet)^\bullet =_d (\mathfrak{D}_1 \parallel (\mathfrak{D}_2 \parallel \mathfrak{D}_3))^\bullet$.

Proof. (1.) It is not so difficult to see that by construction $((\mathfrak{D}_1 \parallel \mathfrak{D}_2) \parallel \mathfrak{D}_3)^\bullet$ contains as retract $((\mathfrak{D}_1 \parallel \mathfrak{D}_2)^\bullet \parallel \mathfrak{D}_3)^\bullet$ through the retraction pair $\langle f, g \rangle$, with $f : ((\mathfrak{D}_1 \parallel \mathfrak{D}_2) \parallel \mathfrak{D}_3)^\bullet \rightarrow ((\mathfrak{D}_1 \parallel \mathfrak{D}_2)^\bullet \parallel \mathfrak{D}_3)^\bullet$ and $g : ((\mathfrak{D}_1 \parallel \mathfrak{D}_2)^\bullet \parallel \mathfrak{D}_3)^\bullet \rightarrow ((\mathfrak{D}_1 \parallel \mathfrak{D}_2) \parallel \mathfrak{D}_3)^\bullet$. Thus $f \circ g = id_{((\mathfrak{D}_1 \parallel \mathfrak{D}_2)^\bullet \parallel \mathfrak{D}_3)^\bullet}$. To prove the converse, let us suppose $g \circ f \neq id_{((\mathfrak{D}_1 \parallel \mathfrak{D}_2) \parallel \mathfrak{D}_3)^\bullet}$; then by construction there is an $e \in ((\mathfrak{D}_1 \parallel \mathfrak{D}_2) \parallel \mathfrak{D}_3)^\bullet$ such that $f(e)$ is undefined. This happens only when (see Section 5.2.5 for the concrete construction of parallel composition) there is $e' \leq e$ such that

1. $p_1(e') \in \mathfrak{D}_1 \parallel \mathfrak{D}_2$ is maximal and $\lambda_{\mathfrak{D}_1 \parallel \mathfrak{D}_2}(p_1(e')) = \kappa'$ is negative
2. $p_2(e') \in \mathfrak{D}_3$ is positive

Observe that $p_2(e')$ cannot be maximal, by Lemma 5.2.1 and by observing that by hypothesis e' has an immediate successor $e'' \leq e$. Thus $p_2(e')$ has at least an immediate successor d and by Innocence it is labelled with an action κ'' justified by κ' . Since $e' \leftarrow e''$ again by Lemma 5.2.1 and by (1.) we have $p_2(e'') = d$. By the commutativity of pull-back square we have that $p_1(e'') = \kappa''$; moreover, since p_1 is an event structure morphism, it is not the case that $p_1(e'') \smile p_1(e')$ and since $\lambda_{\mathfrak{D}_1 \parallel \mathfrak{D}_2}$ enjoys Justification, we cannot have neither $p_1(e'')$ concurrent with $p_1(e')$ nor $p_1(e'') \leq p_1(e')$. Thus $p_1(e') \leq p_1(e'')$, but this contradicts (1.). (2.) can be proved in a similar way. \square

Theorem 5.2.3 (Associativity). $((\mathfrak{D}_1 \parallel \mathfrak{D}_2)^\bullet \parallel \mathfrak{D}_3)^\bullet =_d (\mathfrak{D}_1 \parallel (\mathfrak{D}_2 \parallel \mathfrak{D}_3)^\bullet)^\bullet$.

Proof.

$$\begin{aligned}
((\mathcal{D}_1 \parallel \mathcal{D}_2)^\bullet \parallel \mathcal{D}_3)^\bullet &=_{d} ((\mathcal{D}_1 \parallel \mathcal{D}_2) \parallel \mathcal{D}_3)^\bullet && \text{Lemma 5.2.2 point (1.)} \\
&=_{d} (\mathcal{D}_1 \parallel (\mathcal{D}_2 \parallel \mathcal{D}_3))^\bullet && \text{Theorem 5.2.1} \\
&=_{d} (\mathcal{D}_1 \parallel (\mathcal{D}_2 \parallel \mathcal{D}_3)^\bullet)^\bullet && \text{Lemma 5.2.2 point (2.)}
\end{aligned}$$

□

Hiding

In this section we define the core operation between multidesigns, which is *normalisation*. This operation is based on the operation of parallel composition, defined in previous section; essentially normalisation of designs consists of parallel composition plus an additional operation called *hiding*, which erases all neutral actions appearing in the design.

Definition 5.2.5 (Normalisation). *Let $\mathcal{D}_1 : \Gamma_1 \multimap \Lambda$ and $\mathcal{D}_2 : \Lambda \multimap \Gamma_2$ be two designs. We define $\mathcal{D}_1 \circ \mathcal{D}_2 = \langle X, \leq, \smile, \lambda \rangle$ where*

- $X = \{e \in (\mathcal{D}_1 \parallel \mathcal{D}_2)^\bullet \mid \lambda_{\mathcal{D}_1 \parallel \mathcal{D}_2}(e) \in (\mathcal{A}_{\Gamma_1^\perp \otimes \Gamma_2} \cup \{\mathbf{X}\})\}$
- \leq, \smile, λ are the restriction of $\leq_{\mathcal{D}_1 \parallel \mathcal{D}_2}, \smile_{\mathcal{D}_1 \parallel \mathcal{D}_2}, \lambda_{\mathcal{D}_1 \parallel \mathcal{D}_2}$ to the set X .

The class of multidesigns is closed under normalisation, as shown by the following theorem.

Theorem 5.2.4. $\mathcal{D}_1 \circ \mathcal{D}_2 : \Gamma_1 \multimap \Gamma_2$ is a multidesign.

Proof. It follows from Theorem 5.2.2. □

Moreover \circ is associative, since Theorem 5.2.3. Moreover it is continuous, since pull-back and all the other operations are [Winskel, 1982].

Lemma 5.2.3. *Let $\mathcal{D}_1 : \Gamma_1 \multimap \Lambda$ and $\mathcal{D}_2 : \Lambda \multimap \Gamma_2$ be two multidesigns and let $\xi^+ \in \Lambda$. If $\mathcal{D}_1 = (\xi, I) \ominus \mathcal{E}$ and $\mathcal{D}_2 = \sum_{J \in \mathcal{F}} (\xi, J) \ominus \mathcal{E}_J$ with $I \in \mathcal{F}$, then $\mathcal{D}_1 \circ \mathcal{D}_2 =_{d} \mathcal{E} \circ \mathcal{E}_I$*

Proof. First of all, observe that, by Linearity, there is no event $e \in \mathcal{E}$ such that $\lambda_{\mathcal{D}_1}(e)$ has ξ as focus. So let $e_1 \in \mathcal{D}_1$ be the minimal event labelled with (ξ, I) . Observe also that, by Additive Coherence there is only an event $e_2 \in \mathcal{D}_2$ labelled with (ξ, I) and this event is minimal. This implies by construction that there exists a unique event $d \in \mathcal{D}_1 \parallel \mathcal{D}_2$ such that $p_1(d) = e_1$ and $p_2(d) = e_2$. Observe that, again by construction, such an event is minimal. Moreover, no other event in $\mathcal{D}_1 \parallel \mathcal{D}_2$ different from d is labelled with an action having focus ξ . Thus we conclude. □

5.2.3 A category of multidesigns

In this section, we show that multidesigns are morphisms of a well established category, in which composition is given by normalisation of multidesigns. In order to do this, we need to find a multidesign which behaves like the identity. This multidesign corresponds to the $\mathfrak{F}\alpha$ in [Girard, 2001].

A **delocation** from locus ξ to locus ξ' is an injective map $\theta : \mathbb{N}^* \rightarrow \mathbb{N}^*$ which behaves like the identity on those loci which are not subloci of ξ , while maps all the subloci of ξ to the subloci of ξ' such that $\theta(\xi \cdot s) = \theta(\xi' \cdot s)$ for all suffix s ². This definition can be extended to actions as $\theta(\sigma, I) = (\theta(\sigma), I)$ and $\theta(\mathfrak{X}) = \mathfrak{X}$.

Delocations are useful in order to model exchanging of data. Let us consider a multidesign \mathfrak{D} on the base $\{\xi^-, \sigma^+\}$. We would like to normalise it with an other design \mathfrak{C} . Suppose that, unfortunately \mathfrak{C} is on the base $\{\sigma'^-\}$ with σ' disjoint from σ . Hence, it is not possible to compose the two multidesigns. Delocation can be used in order to overcome this problem. Let θ be a delocation from σ' to σ . We have that $\theta(\mathfrak{C})$ is a multidesign on the base $\{\sigma^-\}$ and it can be normalised with \mathfrak{D} .

We would like to address the following problem: is there a multidesign which is able to perform delocation in an interactive way? More formally let $\mathfrak{D}, \mathfrak{C}$ two designs of respective bases $\{\sigma^-\}$ and $\{\sigma^+\}$. Let θ_1 be a delocation from σ to ξ and θ_2 be a delocation from σ to ξ' . We would like to know if there exists a multidesign \mathfrak{F} on base $\{\xi^-, \xi'^+\}$ satisfying the following

$$\theta_2(\mathfrak{D}) \circ \mathfrak{F} = \theta_1(\mathfrak{D}) \qquad \mathfrak{F} \circ \theta_1(\mathfrak{C}) = \theta_2(\mathfrak{C})$$

Such a design is characterised in [Girard, 2001], where is called $\mathfrak{F}\alpha_{\xi, \xi'}$.

Proposition 5.2.2. *$\mathfrak{F}\alpha_{\xi, \xi'}$ is the minimum multidesign $\mathfrak{D}_{\xi, \xi'}$ on the base $\{\xi^-, \xi'^+\}$ satisfying the following equation*

$$\mathfrak{D}_{\xi, \xi'} = \sum_{I \in \text{Ram}} (\xi, I) \ominus (\xi', I) \ominus \left(\bigsqcup_{i \in I} \mathfrak{D}_{\xi', \xi, i} \right)$$

Proof. Let $\mathfrak{D}, \mathfrak{C}, \theta_1$ and θ_2 as above. We can define by induction on n , the n -th approximant of $\mathfrak{F}\alpha_{\xi, \xi'}$ as $\mathfrak{F}\alpha_{\xi, \xi'}^0 = \mathfrak{S}\text{funf}$ and $\mathfrak{F}\alpha_{\xi, \xi'}^{n+1} = \sum_{I \in \text{Ram}} (\xi, I) \ominus (\xi', I) \ominus \left(\bigsqcup_{i \in I} \mathfrak{F}\alpha_{\xi', \xi, i}^n \right)$. We can prove by mutual induction on n that for all n , we have both $\theta_2(\mathfrak{D}) \circ \mathfrak{F}\alpha_{\xi, \xi'}^n \sqsubseteq_d \theta_1(\mathfrak{D})$ and $\mathfrak{F}\alpha_{\xi, \xi'}^n \circ \theta_1(\mathfrak{C}) \sqsubseteq_d \theta_2(\mathfrak{C})$. Then we can conclude, by continuity of normalisation that both $\theta_2(\mathfrak{D}) \circ \mathfrak{F}\alpha_{\xi, \xi'} \sqsubseteq_d \theta_1(\mathfrak{D})$ and $\mathfrak{F}\alpha_{\xi, \xi'} \circ \theta_1(\mathfrak{C}) \sqsubseteq_d \theta_2(\mathfrak{C})$. To get the converse it suffices to show that for all $\mathfrak{D}' \sqsubseteq_d \theta_1(\mathfrak{D})$ finite, there exists n such that $\theta_2(\mathfrak{D}) \circ \mathfrak{F}\alpha_{\xi, \xi'}^n$ and for all $\mathfrak{C}' \sqsubseteq_d \theta_2(\mathfrak{C})$ finite, there exists m such that $\mathfrak{F}\alpha_{\xi, \xi'}^m \circ \theta_1(\mathfrak{C}) = \mathfrak{C}'$. This can be proved by induction on the number of cells of \mathfrak{D}' and \mathfrak{C}' . Thus we can conclude by Proposition 5.1.3 that $\bigsqcup_n \theta_2(\mathfrak{D}) \circ \mathfrak{F}\alpha_{\xi, \xi'}^n = \theta_1(\mathfrak{D})$ and $\bigsqcup_m \mathfrak{F}\alpha_{\xi, \xi'}^m \circ \theta_1(\mathfrak{C}) = \theta_2(\mathfrak{C})$ as required. \square

We conclude this section by observing that (equivalence classes of) multidesigns form a classes of morphisms of a well established category.

²here we use a simplification of the definition given in [Girard, 2001]

A *signature* S is a function $[1, n] \rightarrow \{+, -\}$. A signature $S : \{1\} \rightarrow \{+, -\}$ is said *atomic*. The empty signature is denoted with $\mathbf{1}$. Given a bases $\Gamma = \{\xi_1^{\epsilon_1}, \dots, \xi_n^{\epsilon_n}\}$ we define the signature $\text{sign}_\Gamma : [1, n] \rightarrow \{+, -\}$ as $\text{sign}_\Gamma(1) = \epsilon_1, \dots, \text{sign}_\Gamma(n) = \epsilon_n$. Observe that a canonical indexing is fixed for each base; this will be taken to be such that, for all signature S , there exists a base Γ such that $\text{sign}_\Gamma = S$. Given a signature S , we denote with S^\perp the signature obtained from S by reversing the target polarity of each number. Given two signature $S_1 : [1, n_1] \rightarrow \{+, -\}, S_2 : [1, n_2] \rightarrow \{+, -\}$ we denote with $S_1 \otimes S_2 : [1, n_1 + n_2] \rightarrow \{+, -\}$ which is such that

$$S_1 \otimes S_2(m) = \begin{cases} S_1(m) & \text{if } m \in [1, n_1] \\ S_2(m - n_1) & \text{otherwise} \end{cases}$$

Finally we define $S_1 \multimap S_2 = S_1^\perp \otimes S_2$. We can observe that every signature S is equal to $\bigotimes S_i$ where each S_i is atomic.

We define the *equality up to delocation* to be the smallest transitive relation \sim_d between two multidesigns $\mathfrak{D}_1, \mathfrak{D}_2$ such that \mathfrak{D}_1 is equal up to delocation to \mathfrak{D}_2 when there is a delocation θ such that $\mathfrak{D}_1 = \theta(\mathfrak{D}_2)$. We denote with $[\mathfrak{D}]_d$ the class of equivalence containing the multidesign \mathfrak{D} .

Definition 5.2.6. Let **Lud** be the category obtained as following

- The class of object is given by the collection of all signature.
- Let Γ, Δ be two bases and let $S_1 = \text{sign}_\Gamma$ and $S_2 = \text{sign}_\Delta$ the corresponding signature. Then $\mathbf{Lud}(S_1, S_2) = \{[\mathfrak{D}]_d \mid \mathfrak{D} : \Gamma \multimap \Delta\}$.
- Let $[\mathfrak{D}_1]_d : S_1 \rightarrow S_2, [\mathfrak{D}_2]_d : S_2 \rightarrow S_3$ be two morphisms. Let us assume $\mathfrak{D}_1 : \Gamma \multimap \Delta$ and $\mathfrak{D}_2 : \Delta \multimap \Xi$ (if this is not the case, we take an opportune delocation of one of the two designs). Then we define $[\mathfrak{D}_1]_d \circ [\mathfrak{D}_2]_d = [\mathfrak{D}_1 \circ \mathfrak{D}_2]_d$
- Let $[\mathfrak{D}_1]_d : S_1 \rightarrow S_2$ and $[\mathfrak{D}_2]_d : S_3 \rightarrow S_4$ such that $\mathfrak{D}_1 : \Gamma$ and $\mathfrak{D}_2 : \Delta$ such that all loci in $\Gamma \cup \Delta$ are pairwise disjoint (if this is not the case, we take an opportune delocation of one of the two designs). Then we define $[\mathfrak{D}_1]_d \otimes [\mathfrak{D}_2]_d = [\mathfrak{D}_1 \uplus \mathfrak{D}_2]_d : S_1 \otimes S_3 \rightarrow S_2 \otimes S_4$.
- Let S be a signature. If it is atomic, then we define $\text{id}_S = [\mathfrak{F}\alpha_{\xi, \xi'}]_d$. Otherwise $S = \bigotimes S_i$ with S_i atomic and $\text{id}_S = \bigotimes \text{id}_{S_i}$.

Theorem 5.2.5. **Lud** is a compact closed category.

Proof. The operator \otimes is a bifunctor being commutative and associative, with $\mathbf{1}$ as neutral element. So the required natural transformations are natural isomorphisms and satisfies the required axioms (since they are all identities). The category is symmetric monoidal closed, by showing that $S \multimap -$ is the right adjoint of $S \otimes -$. It is a compact closed category since it is $*$ -autonomous with $\mathbf{1}$ as dualizing object and $(S_1 \otimes S_2)^\perp = S_1^\perp \otimes S_2^\perp$. \square

5.2.4 Analytical Theorems

The name “analytical theorems” is a name given by Girard in [Girard, 2001], to denote some essential properties of designs, like associativity, separation, stability and so on.

In this section we discuss the analytical properties of multidesigns, as interactive objects. In particular, we prove a separation result for a class of multidesigns, and we prove an analogous of Proposition 2.3.8, namely we prove that when a multidesign normalises to a daimon, then only a finite part of it is explored.

Operational equivalence and separation

Let \mathfrak{D} be a multidesign on the empty base. Then \mathfrak{D} is either \mathfrak{Fid} or $\mathfrak{D} = \mathfrak{Dai}^n$ for some n . Let us define a notion of operational equivalence between multidesigns.

Definition 5.2.7 (Operational equivalence). *Let $\mathfrak{D}_1, \mathfrak{D}_2 : \Gamma$ be two multidesigns. We write $\mathfrak{D}_1 \simeq \mathfrak{D}_2$ when for all designs $\mathfrak{E} : \Gamma^\perp$ we have $\mathfrak{D}_1 \circ \mathfrak{E} = \mathfrak{Dai}^{n+1}$ if and only if $\mathfrak{D}_2 \circ \mathfrak{E} = \mathfrak{Dai}^{n+1}$.*

The following property is called **separation** in [Girard, 2001] and tells that any two designs are isomorphic if and only if they are operationally equivalent.

$$\mathfrak{D}_1 =_d \mathfrak{D}_2 \iff \mathfrak{D}_1 \simeq \mathfrak{D}_2$$

This property has some similarity with a *well-pointed* condition on a Cartesian Closed Category, where the role of the terminal object is played here by the empty base. The \Rightarrow direction holds trivially. The other direction is not obvious at all: in particular, it does not hold for all \mathfrak{D}_1 and \mathfrak{D}_2 .

We first need to characterise isomorphism between multidesign, to adapt the Faggian’s proof technique [Faggian and Maurel, 2005], which is a simplification of the original proof of separation made in [Girard, 2001]. The idea is to say that two multidesigns \mathfrak{D}_1 and \mathfrak{D}_2 are isomorphic when $\{\lambda_{\mathfrak{D}_1}(\ulcorner e \urcorner) \mid e \in \mathfrak{D}_1\} = \{\lambda_{\mathfrak{D}_2}(\ulcorner e \urcorner) \mid e \in \mathfrak{D}_2\}$. This fact is not true since the labelling function is not an event structure morphism, thus we cannot identify an event through the labels of the events that are below it. This is due to the fact that there could be many events belonging to the same configuration of a multidesign \mathfrak{D} which are labelled with daimon. So for example the multidesign \mathfrak{Dai}^1 would be equated to \mathfrak{Dai}^2 .

Definition 5.2.8. *Let \mathfrak{D} be a multidesign on Γ and let $\lambda_{\mathfrak{D}}$ its labelling function. We denote with $\lambda_{\mathfrak{D}}^*$ to be the labelling function obtained from $\lambda_{\mathfrak{D}}$ by relabelling all events e labelled with \mathfrak{X} with a new fresh label \mathfrak{X}_e .*

Observe that now $\lambda_{\mathfrak{D}}^* : \mathfrak{D} \rightarrow \mathcal{A}_\Gamma \uplus \{\mathfrak{X}_e \mid e \in \mathfrak{D}\}$ is an event structure morphism. Thus we can state the following.

Lemma 5.2.4. *Let $\mathfrak{D}_1, \mathfrak{D}_2$ be two multidesign on the same base Γ . $\mathfrak{D}_1 =_d \mathfrak{D}_2$ if and only if $\{\lambda_{\mathfrak{D}_1}^*(\ulcorner e \urcorner) \mid e \in \mathfrak{D}_1\} = \{\lambda_{\mathfrak{D}_2}^*(\ulcorner e \urcorner) \mid e \in \mathfrak{D}_2\}$.*

Proof. The \Rightarrow direction is straightforward. For the opposite direction, define $f : \mathfrak{D}_1 \rightarrow \mathfrak{D}_2$ as $f(e)$ equal to the event d such that $\lambda_{\mathfrak{D}_2}^*(\llbracket d \rrbracket) = \lambda_{\mathfrak{D}_1}^*(\llbracket e \rrbracket)$. To see that f is well defined, suppose that there are $d_1, d_2 \in \mathfrak{D}_2$ such that $\lambda_{\mathfrak{D}_2}^*(\llbracket d_1 \rrbracket) = \lambda_{\mathfrak{D}_2}^*(\llbracket d_2 \rrbracket) = \lambda_{\mathfrak{D}_1}^*(\llbracket e \rrbracket)$, for an event $e \in \mathfrak{D}_1$. Since $\lambda_{\mathfrak{D}_2}^*$ is an event structure morphism, we would have $d_1 \asymp d_2$. If $d_1 \smile d_2$, by Additive Coherence of $\lambda_{\mathfrak{D}_2}^*$ there are $d'_1 \leq d_1$ and $d'_2 \leq d_2$ such that $d'_1 \smile_{\mu} d'_2$ and labelled in a different way. But this would contradict $\lambda_{\mathfrak{D}_2}^*(\llbracket d_1 \rrbracket) = \lambda_{\mathfrak{D}_2}^*(\llbracket d_2 \rrbracket)$. So $d_1 = d_2$. By a similar reasoning, we can prove that f is also surjective (now we need the fact that $\lambda_{\mathfrak{D}_1}^*$ is a event structure morphism). It is an event structure morphism since the labelling functions $\lambda_{\mathfrak{D}_1}^*$ and $\lambda_{\mathfrak{D}_2}^*$ are. To prove injectivity suppose that there are two events $e, e' \in \mathfrak{D}_1$ such that $f(e) = f(e')$. Thus $e \asymp_{\mathfrak{D}_1} e'$, but again they cannot be distinct since $\lambda_{\mathfrak{D}_1}^*$ satisfies Additive Coherence. \square

Let \mathfrak{D} be a multidesign and let e_1, \dots, e_n be a non-empty sequence of events such that e_1 is minimal and for all $i \in [1, n-1]$ we have $e_i \in \text{parents}(e_{i+1})$. We call the string $\lambda_{\mathfrak{D}}^*(e_1) \cdots \lambda_{\mathfrak{D}}^*(e_n)$ a *chronicle* of \mathfrak{D} and e_n is called the *event associated to the chronicle* c ; observe that, by the above lemma, such an event is always unique. We denote with $Ch(\mathfrak{D})$ the set of all chronicles of \mathfrak{D} and we use c, \mathfrak{d}, \dots to range over chronicles in $Ch(\mathfrak{D})$.

Corollary 5.2.1. $\mathfrak{D}_1 \equiv_d \mathfrak{D}_2$ if and only if $Ch(\mathfrak{D}_1) = Ch(\mathfrak{D}_2)$.

We denote with $|c|$ the set of proper actions appearing in c .

Lemma 5.2.5. Let \mathfrak{D} be a multidesign and let $c, \mathfrak{d} \in Ch(\mathfrak{D})$. If $|c| \subseteq |\mathfrak{d}|$ then $c \leq \mathfrak{d}$.

Proof. Let us observe that $|\mathfrak{d}|$ does not contain any two action having the same focus by Linearity. Assume that $s \cdot \kappa$ is the minimum prefix of c which is not prefix of \mathfrak{d} . Thus $s \cdot \kappa \leq c$ and $s \cdot t \cdot \kappa \leq \mathfrak{d}$ with t non empty. Let $e_1, e_2 \in \mathfrak{D}$ be respectively the events associated to the chronicle $s \cdot \kappa$ and $s \cdot t \cdot \kappa$. By construction we have $e_1 \smile e_2$, thus there are $d_1 \leq e_1$ and $d_2 \leq e_2$ such that $d_1 \smile_{\mu} d_2$. By Additive Coherence, they are labelled with different negative labels κ_1, κ_2 , different from κ and they have the same predecessors. But this cannot be by the above hypothesis. \square

Let \mathfrak{D} a multidesign and let $c \in Ch(\mathfrak{D})$ and let e be the event associated to the chronicle c . We define $dai_{\mathfrak{D}}(c)$ to be the cardinality of the set $\{d \in \mathfrak{D} \mid \lambda_{\mathfrak{D}}(d) = \blackbox, \exists e' \in [e]. e' \leftarrow_{\mathfrak{D}} d\}$ and $size(c)$ to be the cardinality of $|c|$.

Theorem 5.2.6 (Finite separation). Let $\mathfrak{D}_1, \mathfrak{D}_2 : \Gamma$ two finite multidesigns. If $\mathfrak{D}_1 \not\equiv_d \mathfrak{D}_2$ then there exists a finite multidesign $\mathfrak{E} : \Gamma^{\perp}$ such that $\mathfrak{D}_1 \circ \mathfrak{E} \neq \mathfrak{D}_2 \circ \mathfrak{E}$

Proof. Let $c = \kappa_1 \cdot \dots \cdot \kappa_n$ be a minimal chronicle of \mathfrak{D}_1 (with respect to $size(c)$) which is such that, its associated event e is labelled with a positive action and either $c \notin Ch(\mathfrak{D}_2)$ or $c \in Ch(\mathfrak{D}_2)$ with $dai_{\mathfrak{D}_1}(c) > dai_{\mathfrak{D}_2}(c)$. Let us consider the case that e is labelled with an action different from \blackbox , first. Let $Opp(c) = \langle E, \leq, \smile, \lambda_{Opp(c)} \rangle$ be the multidesign on Γ^{\perp} such that $E = \{\kappa_1, \dots, \kappa_n, \blackbox\}$ with $\lambda_{Opp(c)}$ defined as the identity function and the order defined as the reflexive transitive closure of the justification relation among actions. Let us define $Vis(\mathfrak{D}_1, Opp(c))$ to be the event structure having as set of event $\{e \in$

$\mathcal{D}_1 \parallel \text{Opp}(c) \mid p_1(e), p_2(e)$ both defined} with the inherited order and labelling function. Let us observe that $\text{Vis}(\mathcal{D}_1, \text{Opp}(c))$ is a totally ordered set and its set of chronicles have a maximum chronicle³ equal to c . Moreover $\mathcal{D}_1 \circ \text{Opp}(c) = \mathcal{D} \text{ai}^m$ where $m = \text{dai}_{\mathcal{D}_1}(c) + 1 + m'$ where m' is the number of minimal events of \mathcal{D}_1 labelled with \blackboxtimes . By construction, observe that m is finite since \mathcal{D}_1 is. Let us consider now $\text{Vis}(\mathcal{D}_2, \text{Opp}(c))$: let $\mathfrak{d}_1, \mathfrak{d}_2$ two chronicles of $\text{Vis}(\mathcal{D}_1, \text{Opp}(c))$: by construction we have $|\mathfrak{d}_1| \subseteq |c|$ and $|\mathfrak{d}_2| \subseteq |c|$, thus \mathfrak{d}_1 and \mathfrak{d}_2 are in $\text{Ch}(\mathcal{D}_1)$ since the minimality condition of c ; hence, $\mathfrak{d}_1 \leq c$ and $\mathfrak{d}_2 \leq c$ by Lemma 5.2.5, allowing to conclude that either $\mathfrak{d}_1 \leq \mathfrak{d}_2$ or $\mathfrak{d}_2 \leq \mathfrak{d}_1$. So $\text{Vis}(\mathcal{D}_2, \text{Opp}(c))$ is also totally ordered, so there is a maximum chronicle of $\text{Vis}(\mathcal{D}_2, \text{Opp}(c))$ which we denote with \mathfrak{d} . Let us observe that $\mathfrak{d} \in \text{Ch}(\mathcal{D}_2)$ and by construction $\mathcal{D}_2 \circ \text{Opp}(c) = \mathcal{D} \text{ai}^k$ with $k = \text{dai}_{\mathcal{D}_2}(\mathfrak{d}) + k'$ where k' is the number of minimal action of \mathcal{D}_2 labelled with \blackboxtimes . Observe that if $c = \mathfrak{d}$ we can conclude since $\text{dai}_{\mathcal{D}_1}(c) > \text{dai}_{\mathcal{D}_2}(\mathfrak{d})$ and $m' = k'$ by the minimality condition. Otherwise $\mathfrak{d} \leq c$ and we still conclude since by construction $\text{dai}_{\mathcal{D}_2}(\mathfrak{d})$ is at most equal to $\text{dai}_{\mathcal{D}_1}(c)$. The case that $c = \kappa_1 \dots, \kappa_{n-1} \cdot \blackboxtimes$ action can be obtained in a similar way, by taking $\text{Opp}(c) = \{\kappa_1, \dots, \kappa_{n-1}\}$ with the order and the labelling function defined as above. \square

Unfortunately the above result does not extend to non-finite multidesigns. To see this consider the following two multidesigns, both on base $\{\xi^+\}$

- $\mathcal{D}_1 = \mathcal{D} \text{ai}^\infty \uplus \mathcal{E}_1$ where $\mathcal{E}_1 = \{d_1\}$ with $\lambda_{\mathcal{E}_1}(d_1) = (\xi, \{1\})$
- $\mathcal{D}_2 = \mathcal{D} \text{ai}^\infty \uplus \mathcal{E}_2$ where $\mathcal{E}_2 = \{d_2\}$ with $\lambda_{\mathcal{E}_2}(d_2) = (\xi, \{2\})$

There is no way to find a counter-multidesigns which separates \mathcal{D}_1 and \mathcal{D}_2

Materiality

In the previous section, we showed that normalisation is a continuous operation. Thus we can show an analogous of Proposition 2.3.8, which tells that when the result of normalisation is $\mathcal{D} \text{ai}^{n+1}$, then there is a finite part of the design being explored. More formally we have the following

Theorem 5.2.7 (Materiality). *Let $\mathcal{D} : \Gamma$ be a multidesign. If $\mathcal{D} \circ \mathcal{E} = \mathcal{D} \text{ai}^{n+1}$ for some n then there exists a finite $\mathcal{D}' \sqsubseteq_d \mathcal{D}$ such that $\mathcal{D}' \circ \mathcal{E} = \mathcal{D} \text{ai}^{n+1}$.*

Proof.

$$\begin{aligned} \mathcal{D} \circ \mathcal{E} &= \bigsqcup \{ \mathcal{D}' \sqsubseteq_d \mathcal{D} \mid \mathcal{D}' \text{ finite} \} \circ \mathcal{E} && \text{Proposition 5.1.3} \\ &= \bigsqcup \{ \mathcal{D}' \circ \mathcal{E} \mid \mathcal{D}' \text{ finite}, \mathcal{D}' \sqsubseteq_d \mathcal{D} \} && \text{Continuity of } \circ \\ &= \mathcal{D} \text{ai}^{n+1} \end{aligned}$$

Thus for all finite $\mathcal{D}' \sqsubseteq \mathcal{D}$, we have $\mathcal{D}' \circ \mathcal{E} \sqsubseteq_d \mathcal{D} \text{ai}^{n+1}$. But since $\mathcal{D} \text{ai}^{n+1}$ is a compact element, we have that there exists $\mathcal{D}^* \sqsubseteq_d \mathcal{D}$ finite such that $\mathcal{D} \text{ai}^{n+1} \sqsubseteq_d \mathcal{D}^* \circ \mathcal{E}$. So we can conclude. \square

³we take the same definition of chronicle as above also for $\text{Vis}(\mathcal{D}_1, \text{Opp}(c))$.

5.2.5 Technical characterisation of parallel composition

In this section, we will give a concrete characterisation of parallel composition and we will use it to prove Theorem 5.2.2.

In the following we give a concrete construction of the pull-back of two event structures $\mathcal{E}_1 = \langle E_1, \leq_1, \smile_1 \rangle, \mathcal{E}_2 = \langle E_2, \leq_2, \smile_2 \rangle$ along the morphisms $\varphi_1 : \mathcal{E}_1 \rightarrow \mathcal{A}$ and $\varphi_2 : \mathcal{E}_2 \rightarrow \mathcal{A}$. This construction is an opportune modification of the concrete construction of categorical product presented in [Varacca and Yoshida, 2006].

Pull-back. We take $\mathcal{E}_i = \langle E_i, \leq_i, \smile_i \rangle, \varphi_i$ ($i \in \{1, 2\}$) as above. Let $E_i^* = E_i \uplus \{\star\}$. Consider the set \tilde{E} obtained by initial equation $X = \wp_{fin}(X) \times E_1^* \times E_2^*$. Its elements has the form (x, e_1, e_2) for $x \subseteq_{fin} \tilde{E}$. We define a notion of *height* of an element of \tilde{E} as

$$h(\emptyset, e_1, e_2) = 0 \quad h(x, e_1, e_2) = \max\{h(e) \mid e \in x\} + 1$$

We now define $\mathcal{E}_1 *_{\mathcal{A}} \mathcal{E}_2 = \langle E, \leq, \smile \rangle$ where $E \subseteq \tilde{E}$ by defining the membership predicate $e \in E$ by induction on $h(e)$; we define at the same time \leq and \smile .

Base case:

1. $(\emptyset, e_1, \star) \in D$ if $\varphi_1(e_1)$ is undefined and e_1 minimal.
2. $(\emptyset, \star, e_2) \in E$ if $\varphi_2(e_2)$ is undefined and e_2 minimal.
3. $(\emptyset, e_1, e_2) \in E$ if e_1, e_2 are minimal and either $\varphi_1(e_1), \varphi_2(e_2)$ are both undefined or $\varphi_1(e_1) = \varphi_2(e_2)$.

The elements of height 0 are incomparably. Instead, for the conflict we have $(\emptyset, e_1, e_2) \smile (\emptyset, d_1, d_2)$ if $e_1 \smile_1 d_1$ or $e_2 \smile_2 d_2$.

Inductive case: Let us assume that all events having height $\leq n$ have been already defined. Let (x, e_1, e_2) be an event having height $n + 1$. Let y the set of maximal elements of x , let $y_1 = \{d_1 \in E_1 \mid (x, d_1, d_2) \in y\}$ and $y_2 = \{d_2 \in E_2 \mid (x, d_1, d_2) \in y\}$. We have that $(x, e_1, e_2) \in E$ when x is downward closed, conflict free and

1. suppose $e_1 \in E_1$ and $e_2 = \star$. Then $\varphi_1(e_1)$ is undefined and $y_1 = \text{parents}(e_1)$.
2. suppose $e_2 \in E_2$ and $e_1 = \star$. Then $\varphi_2(e_2)$ is undefined and $y_2 = \text{parents}(e_2)$.
3. suppose $e_1 \in E_1$ and $e_2 \in E_2$. Then
 - a) either both $\varphi_1(e_1)$ and $\varphi_2(e_2)$ are undefined or $\varphi_1(e_1) = \varphi_2(e_2)$.
 - b) if $(z, d_1, d_2) \in y$ then $d_1 \in \text{parents}(e_1)$ or $d_2 \in \text{parents}(e_2)$.
 - c) for all $d_1 \in \text{parents}(e_1)$ there exists $(z, d_1, d_2) \in x$.
 - d) for all $d_2 \in \text{parents}(e_2)$ there exists $(z, d_1, d_2) \in x$.
4. let $x_1 = \{d_1 \in E_1 \mid (z, d_1, d_2) \in x\}$ and $x_2 = \{d_2 \in E_2 \mid (z, d_1, d_2) \in x\}$. Then there is no $d_1 \in x_1$ and $d_2 \in x_2$ such that $d_1 \smile_1 e_1$ or $d_2 \smile_2 e_2$.

The partial order is extended with $e \leq (x, e_1, e_2)$ when $e \in x$ or $e = (x, e_1, e_2)$. Note that if $e < e'$ then $h(e) < h(e')$. Finally to define conflicts take $e = (x, e_1, e_2)$ and $d = (z, d_1, d_2)$ where either $h(e) = n + 1$ or $h(d) = n + 1$ or both. Then $e \smile d$ if one of the following hold

1. $e_1 \smile_1 d_1$ or $e_2 \smile_2 d_2$ with $e \neq d$.

2. there exists $e' = (x', e'_1, e'_2) \in x$ such that $e'_1 \asymp_1 d_1$ or $e'_2 \asymp_2 d_2$ and $e \neq d$.
3. there exists $d' = (z', d'_1, d'_2) \in z$ such that $d'_1 \asymp_1 e_1$ or $d'_2 \asymp_2 e_2$ and $e \neq d$.
4. exists $e' \in x, d' \in z$ such that $e' \smile d'$.

As the following lemma shows, some of the above conditions are redundant, but they are kept for sake of simplicity.

Lemma 5.2.6. *Let $(x, e_1, e_2), (x', e_1, e_2)$ be two events in $\mathcal{E}_1 *_{\mathcal{A}} \mathcal{E}_2$ such that $x \neq x'$. Then there exists $e \in x, e' \in x$ such that $e \smile e'$.*

Proof. We prove it by induction on the joint size of x, x' . The base case is vacuously true. Now take $(x, e_1, e_2), (x', e_1, e_2)$ with $x \neq x'$. Since x, x' are downward closed sets, if their maximal elements coincide, they coincide. Therefore, without loss of generality, there must be a maximal element $(y, d_1, d_2) \in x$ such that $(y, d_1, d_2) \notin x'$. Let us assume without loss of generality that $d_1 \in \text{parents}(e_1)$ (the other case is symmetrical). Therefore by definition there must be $(y', d_1, d'_2) \in x'$. Suppose $d_2 \neq d'_2$. Then by definition of conflict $(y, d_1, d_2) \smile (y', d_1, d'_2)$. If $d_2 = d'_2$ then we conclude by induction. and by definition of conflict. \square

We now show that the construction we defined is the pull back of \mathcal{E}_1 and \mathcal{E}_2 along φ_1 and φ_2 . First of all, we show that $\mathcal{E}_1 *_{\mathcal{A}} \mathcal{E}_2$ is a well defined event structure.

Lemma 5.2.7. *$\mathcal{E}_1 *_{\mathcal{A}} \mathcal{E}_2 = \langle E, \leq, \smile \rangle$ is an event structure.*

Proof. First of all notice that for every $e = (x, e_1, e_2)$, $[e]$ is finite and it coincides with x . We can prove that conflict is hereditary and anti-reflexive. It is hereditary essentially by definition: suppose $e = (x, e_1, e_2) \smile d = (y, d_1, d_2)$ and let $d \leq d' = (y', d'_1, d'_2)$. By considering all the cases of the definition of $e \smile d$, we derive $e \smile d$, we derive $e \smile d'$. For instance suppose there exists $e' = (x', e'_1, e'_2) \leq e$ such that $e'_1 \asymp d_1$ and $e' \neq d$. This means that $e' \smile d$. Note that $e' \leq e$ and $d \leq d'$. Then $e \smile d'$ by the fourth case. Furthermore the conflict relation is irreflexive. Suppose $(x, e_1, e_2) \smile (x, e_1, e_2)$. Note that there are no $e, d \in x$ such that $e \smile d$ since x is a configuration. Therefore there must exist (x', e'_1, e'_2) such that $(x', e'_1, e'_2) \smile (x, e_1, e_2)$. Take the minimum such. This means $e'_1 \asymp e_1$ or $e'_2 \asymp e_2$. But this contradicts condition (4) of the inductive case. \square

We now provide two projection morphisms and we show that the pull-back diagram commutes.

Lemma 5.2.8. *Let $p_1 : E \rightarrow E_1$ and $p_2 : E \rightarrow E_2$ two maps defined as*

$$p_1(x, e_1, e_2) = \begin{cases} e_1 & \text{if } e_1 \in E_1 \\ \text{undefined} & \text{otherwise} \end{cases} \quad p_2(x, e_1, e_2) = \begin{cases} e_2 & \text{if } e_2 \in E_2 \\ \text{undefined} & \text{otherwise} \end{cases}$$

*Then $p_1 : \mathcal{E}_1 *_{\mathcal{A}} \mathcal{E}_2 \rightarrow \mathcal{E}_1$ and $p_2 : \mathcal{E}_1 *_{\mathcal{A}} \mathcal{E}_2 \rightarrow \mathcal{E}_2$ are event structure morphisms.*

Proof. We first show that p_1 is a morphism. The proof for p_2 is similar. We use Proposition 5.1.1.

- Take $e, e' \in D$ and suppose $p_1(e) \asymp p_2(e')$. Then by definition $e \asymp e'$.
- To show that p_1 preserves downward closure, let $e = (x, e_1, e_2)$, suppose $e'_1 \leq e_1 = p_1(e)$. Then we show that there is $e' \leq e$ such that $p_1(e') = e'_1$. By induction on the height of e : the basis is vacuously true since e_1 is minimal. For the step, consider first the case where $e'_1 \in \text{parents}(e_1)$. Then by definition of D we have that there exists $e' = (x', e'_1, e'_2) \in x$ and we conclude. If $e'_1 \notin \text{parents}(e_1)$, then there is a $e''_1 \in \text{parents}(e_1)$ such that $e'_1 \leq e''_1 \leq e_1$ so that there is $e'' = (x'', e''_1, e''_2) \in x$. By induction hypothesis, there is $e' \in x''$ such that $\theta_1(e') = e'_1$. And by transitivity, $e' \leq e$.

□

Lemma 5.2.9. *The pull-back diagram commutes, i.e. for all $e \in E$, $\varphi_1(p_1(e)) = \varphi_2(p_2(e))$.*

Proof. We proceed by cases. Let us consider first the case $e = (x, e_1, \star)$: then $\varphi_1(p_1(e))$ is undefined; note also that $\varphi_2(p_2(e))$ is undefined, so we conclude. The case $e = (x, \star, e_2)$ is symmetrical. Last consider $e = (x, e_1, e_2)$ with $e_1 \in E_1, e_2 \in E_2$. Then $\varphi_1(e_1) = \varphi_2(e_2)$ by construction. So the proof is done. □

We conclude by showing the universal property of pull-back.

Lemma 5.2.10. *Let \mathcal{B} an event structure. Let $p'_1 : \mathcal{B} \rightarrow \mathcal{E}_1$ and $p'_2 : \mathcal{B} \rightarrow \mathcal{E}_2$ two morphisms such that $\varphi_1 \circ p'_1 = \varphi_2 \circ p'_2$. Then there exists a unique morphism $\theta : \mathcal{B} \rightarrow \mathcal{E}_1 *_{\mathcal{A}} \mathcal{E}_2$ such that $p'_1 = p_1 \circ \theta$ and $p'_2 = p_2 \circ \theta$.*

Proof. Clearly, if θ exists, it must be defined as $\theta(d) = (x, p'_1(d), p'_2(d))$ for some x . By this, we mean $\theta(d) = (x, p'_1(d), \star)$ if $p'_2(d)$ is undefined, $\theta(d) = (x, \star, p'_2(d))$ if $p'_1(d)$ is undefined and undefined if both are undefined. Under this condition the requirements $p'_1 = p_1 \circ \theta$ and $p'_2 = p_2 \circ \theta$ are satisfied. We now define x by induction on the cardinality of $[d]$. Suppose d minimal. Then, since p'_1, p'_2 are morphisms and in particular they preserve downward closure, we have that $p'_1(d), p'_2(d)$ are both minimal. Since every maximal element of x must contain the parent of at least one of them, the only possibility is that x be empty. Putting $\theta(d) = (\emptyset, \theta'_1(d), \theta'_2(d))$, we obtain, that, on element of height 0,

- $\theta(d)$ is uniquely defined: we have seen that all choices are forced.
- θ reflects reflexive conflicts: suppose $(\emptyset, p'_1(d), p'_2(d)) \asymp (\emptyset, p'_1(d'), p'_2(d'))$. Then either $p'_1(d) \asymp p'_1(d')$ or $p'_2(d) \asymp p'_2(d')$. In both cases $d \asymp d'$ since p'_1, p'_2 are morphisms.
- θ preserves downward closure vacuously.

Now suppose θ to be uniquely defined for all elements d' having the size of $[d']$ less or equal than n , it reflects reflexive conflict and preserves downward closure. Consider d having the size of $[d]$ equal to $n + 1$. We want to define $\theta(d) = (x, \theta'_1(d), \theta'_2(d))$. Define x as follows. For a set A , let $\lceil A \rceil = \{a' \mid a' \leq a, a \in A\}$ its downward closure. Let $X = \{\theta(d') \mid d' < d \wedge [\theta'_1(d') \in \text{parents}(\theta'_1(d)) \vee \theta'_2(d') \in \text{parents}(\theta'_2(d))]\}$ and define

x as $\lceil X \rceil$. We first check that this is indeed an element of E . x is downward closed by definition. It is finite because X is and each element of X has finitely many predecessors. Suppose there are $d', d'' < d$ such that $\theta(d') \smile \theta(d'')$. We now by inductive hypothesis that θ reflects reflexive conflict on element strictly less than d , which means $d' \smile d''$, contradiction. Now the maximal elements of x contain either a parent of $p'_1(d)$ or a parent of $p'_2(d)$ by construction. Take a parent e_1 of $p'_1(d)$. Since p'_1 is a morphism then there exists $d' \leq d$ such that $p'_1(d') = e_1$. So all parents are represented in X . Finally suppose there exists $(z, e_1, e_2) \in x$ such that $e_1 \asymp p'_1(d)$ or $e_2 \asymp p'_2(d)$. If $(z, e_1, e_2) \in X$ there is $d' < d$ such that $\theta(d') = (z, e_1, e_2)$. Then $e_1 = \theta'_1(d')$ or $e_2 = \theta'_2(d')$. Since p'_1, p'_2 are morphisms then we would have $d \smile d'$, contradiction. Otherwise there must be $\theta(d') \in X$ such that $(z, e_1, e_2) < \theta(d')$. Since θ preserve downward closure by induction, there is $d'' < d'$ such that $\theta(d'') = (z, e_1, e_2)$. Then by using a same reasoning as above, we conclude $d'' \smile d$, getting a contradiction.

Thus, putting $\theta(d) = (x, \theta'_1(d), \theta'_2(d))$ we have θ is well defined on d . Moreover

- $\theta(d)$ is uniquely defined: suppose we have another possible x . Since θ preserves downward closure by induction, for all $e \in x$ we have $e = \theta(d')$, for some $d' < d$. Now suppose there is an element $\theta(d') \in X$ which is not in x . Without loss of generality, assume that $p'_1(d') \in \text{parents}(p'_1(d))$. Then by construction there is an element $e' = (y, p'_1(d'), d'_2)$ maximal in x . By the observation above, we must have $e' = \theta(d')$. Contradiction
- θ is a morphism: we can prove it using Proposition 5.1.1 and by a case reasoning. □

Theorem 5.2.8. $\mathcal{E}_1 *_{\mathcal{A}} \mathcal{E}_2 = \langle E, \leq, \smile \rangle$ is the pull-back of \mathcal{E}_1 and \mathcal{E}_2 along φ_1 and φ_2 .

Proof. It follows from Lemmas 5.2.7, 5.2.8, 5.2.9 and 5.2.10. □

Now we will prove Theorem 5.2.2. Namely, we study properties of the event structure obtained from the parallel composition of two multidesigns. More specifically we will focus on what kind of properties of multidesigns (alternation, justification, linearity, innocence, additive coherence, maximality) are preserved under parallel composition.

Justification and Linearity. The properties of Justification and Linearity can be easily proved again by using the abstract definition. They follows since the commutativity of the pull-back square.

Proposition 5.2.3. $\lambda_{\mathfrak{D}_1 \parallel \mathfrak{D}_2}$ satisfies Justification and Linearity.

Proof. It is easy to see, by the property of pull-back, that

$$\lambda_{\mathfrak{D}_1 \parallel \mathfrak{D}_2}(e) = \begin{cases} \lambda_{\mathfrak{D}_1}(p_1(e)) & \text{if } p_1(e) \text{ is defined} \\ \lambda_{\mathfrak{D}_2}(p_2(e)) & \text{otherwise} \end{cases}$$

Then we can conclude by Proposition 5.1.1. □

Arborescence. We prove that the parallel composition of two multidesign is an arborescent event structure. In particular we prove that every non-minimal event $e \in \mathfrak{D}_1 \parallel \mathfrak{D}_2$ has exactly one immediate predecessor. The proof makes use of the Innocence and Justification properties of both \mathfrak{D}_1 and \mathfrak{D}_2 .

Proposition 5.2.4. $\mathfrak{D}_1 \parallel \mathfrak{D}_2$ is arborescent.

Proof. We prove that for all $e \in D$, if both $e_1 \leftarrow_{\mathfrak{D}_1 \parallel \mathfrak{D}_2} e$ and $e_2 \leftarrow_{\mathfrak{D}_1 \parallel \mathfrak{D}_2} e$ then $e_1 = e_2$. Let $e_1, e_2 \in D$ such that $e_1 \leftarrow_{\mathfrak{D}_1 \parallel \mathfrak{D}_2} e$ and $e_2 \leftarrow_{\mathfrak{D}_1 \parallel \mathfrak{D}_2} e$. By construction, we have four cases.

1. Case $p_1(e_1) \leftarrow_{\mathfrak{D}_1} p_1(e)$ and $p_1(e_2) \leftarrow_{\mathfrak{D}_1} p_1(e)$. In this case $p_1(e_1) = p_1(e_2)$ by arborescence of \mathfrak{D}_1 . Then $e_1 \asymp e_2$ since p_1 is a morphism. But it cannot be the case that $e_1 \smile e_2$, since $\mathfrak{D}_1 \parallel \mathfrak{D}_2$ is an event structure by Lemma 5.2.7. Thus it remains $e_1 = e_2$ as required.
2. Case $p_2(e_1) \leftarrow_{\mathfrak{D}_2} p_2(e)$ and $p_2(e_2) \leftarrow_{\mathfrak{D}_2} p_2(e)$. Similar to the previous one.
3. Case $p_1(e_1) \leftarrow_{\mathfrak{D}_1} p_1(e)$ and $p_2(e_2) \leftarrow_{\mathfrak{D}_2} p_2(e)$. Then, by construction $e = (y, d_1, d_2)$ with both $d_1 \neq \star$ and $d_2 \neq \star$ and $\varphi_1(p_1(e)) = \varphi_2(p_2(e)) \in \mathcal{A}_\Lambda$. Without loss of generality, we assume that the polarity of $\lambda_{\mathfrak{D}_1}(p_1(e))$ to be negative (and thus the polarity of $\lambda_{\mathfrak{D}_2}(p_2(e))$ is positive); the opposite case is easy. Then by Innocence of \mathfrak{D}_1 we have $\varphi_1(p_1(e_1)) \vdash \varphi_1(p_1(e))$ and $p_2(e_1)$ defined. Then, by Justification there must exist $e' \leq_{\mathfrak{D}_2} p_2(e)$ such that $\lambda_{\mathfrak{D}_2}(e') = \varphi_1(p_1(e_1))$ and by construction we have $e' = p_2(e_1)$. The case $p_2(e_1) < p_2(e_2) \leq p_2(e)$ is not possible since p_2 is an event structure morphism and if this holds then it would contradict $e_1 \leftarrow_{\mathfrak{D}_1 \parallel \mathfrak{D}_2} e$. By the arborescence of \mathfrak{D}_2 we conclude $p_2(e_1) = p_2(e_2)$; thus we conclude using a similar reason of the previous point.
4. Case $p_2(e_1) \leftarrow_{\mathfrak{D}_2} p_2(e)$ and $p_1(e_2) \leftarrow_{\mathfrak{D}_1} p_1(e)$. Similar to the previous point.

□

Confusion freeness and Additive Coherence. We prove that both the properties of confusion freeness and additive coherence are preserved under parallel composition.

We first prove preservation of confusion freeness. The key lemma is the following, which relates the conflict of events in $\mathfrak{D}_1 \parallel \mathfrak{D}_2$ to the immediate conflict of \mathfrak{D}_1 and \mathfrak{D}_2 .

Lemma 5.2.11. Let $(x, e_1, e_2), (y, d_1, d_2)$ be two events in $\mathfrak{D}_1 \parallel \mathfrak{D}_2$. Suppose $(x, e_1, e_2) \smile (y, d_1, d_2)$. Then there exists $(x', e'_1, e'_2) \in x$ and (y', d'_1, d'_2) such that either $e_1 \smile_\mu d_1$ or $e_2 \smile_\mu d_2$.

Proof. We check this by cases on the definition of conflict.

- $e_1 \smile d_1$. In this case there must exist $e'_1 \leq e_1$ and $d'_1 \leq d_1$ such that $e'_1 \smile_\mu d'_1$. Since projections are event structure morphism and since they preserve configuration, we have that there must exist $(x', e'_1, e'_2) \in x$ and $(y', d'_1, d'_2) \in y$, for some x', y', e'_2, d'_2 , as required.

- $e_2 \smile d_2$. Symmetrical to the above one.
- $e_1 = d_1$ and $e_2 \neq d_2$. Let us observe that it is not possible that $e_2 = \star$ and $d_2 \neq \star$ (nor symmetrically). If $\lambda_{\mathfrak{D}_2}(e_2) \in A(\Gamma_2)$ then by construction we have $e_2 = d_2 = \star$. Otherwise, if $\lambda_{\mathfrak{D}_2}(e_2) \in A(\Lambda)$ it is not possible that the event (x, e_1, \star) is in $\mathfrak{D}_1 \parallel \mathfrak{D}_2$, since the commutativity of the pull back square. Always by commutativity of pull-back square we need to have $\lambda_{\mathfrak{D}_2}(e_2) = \lambda_{\mathfrak{D}_2}(d_2)$ so they must be in conflict. Hence we reason as above and we conclude.
- $e_2 = d_2$ and $e_1 \neq d_1$. Symmetrical to the above one.
- $e_1 = d_1$ and $e_2 = d_2$. Then the conclusion follows by Lemma 5.2.6.

All the other cases are straightforward. \square

We make use of the above lemma to prove that any two events in immediate conflict share the same predecessors in $\mathfrak{D}_1 \parallel \mathfrak{D}_2$.

Lemma 5.2.12. *If $(x, e_1, e_2) \asymp_\mu (y, d_1, d_2)$ then $x = y$.*

Proof. We first prove that $(x, d_1, d_2) \in D$. Suppose first $e_2 = d_2 = \star$: then $e_1 \asymp d_1$ by previous lemma and $\text{parents}(e_1) = \text{parents}(d_1)$ by confusion freeness of \mathfrak{D}_1 . This implies the conclusion, since $(y, d_1, d_2) \in D$. Dually when $e_1 = d_1 = \star$. In case $e_1 \asymp_\mu d_1$ we must have $e_2 \asymp d_2$ the commutativity of pull-back square. But $e_2 \asymp_\mu d_2$ since the opposite would contradict $(x, e_1, e_2) \asymp_\mu (y, d_1, d_2)$. Now that we prove $(x, d_1, d_2) \in D$, let us suppose $x \neq y$. But by Lemma 5.2.6 there are $e \in x, e' \in y$ such that $e \smile e'$. But this contradicts $(x, e_1, e_2) \asymp_\mu (y, d_1, d_2)$. \square

We now prove that the conflict relation \asymp_μ is an equivalence relation.

Lemma 5.2.13. *\asymp_μ is transitive.*

Proof. Let us suppose $(x, e_1, e_2) \asymp_\mu (y, d_1, d_2)$ and $(y, d_1, d_2) \asymp_\mu (z, g_1, g_2)$. Thus reasoning as above, we can prove $e_1 \asymp_\mu d_1 \asymp_\mu g_1$ and $e_2 \asymp_\mu d_2 \asymp_\mu g_2$, since the confusion freeness of \mathfrak{D}_1 and \mathfrak{D}_2 . Hence we conclude. \square

Proposition 5.2.5. *$\mathfrak{D}_1 \parallel \mathfrak{D}_2$ is confusion free.*

Proof. By Lemma 5.2.12 and Lemma 5.2.13. \square

From confusion freeness and by Additive Coherence of both \mathfrak{D}_1 and \mathfrak{D}_2 follows that $\mathfrak{D}_1 \parallel \mathfrak{D}_2$ satisfies Additive Coherence.

Proposition 5.2.6. *$\lambda_{\mathfrak{D}_1 \parallel \mathfrak{D}_2}$ satisfies Additive Coherence.*

Proof. Suppose there are $e = (x, e_1, e_2)$ and $d = (y, d_1, d_2)$ in $\mathfrak{D}_1 \parallel \mathfrak{D}_2$ such that $e \smile_\mu d$. By reasoning as above we have that their polarity is always negative since both \mathfrak{D}_1 and \mathfrak{D}_2 satisfies Additive Coherence. \square

Innocence and Alternation. Showing the preservation of Innocence and Alternation is more difficult, since we need to speak about polarities of action which are labels of events in $\mathfrak{D}_1 \parallel \mathfrak{D}_2$. We note that there is an ambiguity for those events e which are such that $\lambda_{\mathfrak{D}_1 \parallel \mathfrak{D}_2}(e) \in A(\Lambda)$; to overcome it, we use an auxiliary polarity denoted with \pm (the *neutral polarity*) to denote the fact that, given $e \in \mathfrak{D}_1 \parallel \mathfrak{D}_2$, $\lambda_{\mathfrak{D}_1 \parallel \mathfrak{D}_2}(e) \in A(\Lambda)$. About innocence, we can show the following.

Lemma 5.2.14. *Suppose $e_1, e_2 \in \mathfrak{D}_1 \parallel \mathfrak{D}_2$ such that $e_1 \leftarrow_{\mathfrak{D}_1 \parallel \mathfrak{D}_2} e_2$ and either the polarity of $\lambda_{\mathfrak{D}_1 \parallel \mathfrak{D}_2}(e_1)$ is positive or the polarity of $\lambda_{\mathfrak{D}_1 \parallel \mathfrak{D}_2}(e_2)$ is negative. Then $\lambda_{\mathfrak{D}_1 \parallel \mathfrak{D}_2}(e_1) \vdash \lambda_{\mathfrak{D}_1 \parallel \mathfrak{D}_2}(e_2)$.*

Proof. It follows immediately by Innocence of \mathfrak{D}_1 and \mathfrak{D}_2 that for all $i \in \{1, 2\}$, if $e_1 \leftarrow_{\mathfrak{D}_i} e_2$ and either the polarity of $\lambda_{\mathfrak{D}_i}(e_1)$ is positive or the polarity of $\lambda_{\mathfrak{D}_i}(e_2)$ is negative. Then $\lambda_{\mathfrak{D}_i}(e_1) \vdash \lambda_{\mathfrak{D}_i}(e_2)$. Thus, suppose $e_1 \leftarrow_{\mathfrak{D}_1 \parallel \mathfrak{D}_2} e_2$ in $\mathfrak{D}_1 \parallel \mathfrak{D}_2$. By construction, either $p_1(e_1) \leftarrow_{\mathfrak{D}_1} p_1(e_2)$ or $p_2(e_1) \leftarrow_{\mathfrak{D}_2} p_2(e_2)$. In both cases, we conclude since the above observation. \square

The following Lemma talk about the preservation of alternation. The previous lemma will be crucial to exclude some undesired cases.

Lemma 5.2.15. *Let $e_1, e_2 \in \mathfrak{D}_1 \parallel \mathfrak{D}_2$ such that $e_1 \leftarrow_{\mathfrak{D}_1 \parallel \mathfrak{D}_2} e_2$. Then either $\lambda_{\mathfrak{D}_1 \parallel \mathfrak{D}_2}(e_1)$ and $\lambda_{\mathfrak{D}_1 \parallel \mathfrak{D}_2}(e_2)$ are opposite, either they are both \pm , either the former is negative and the latter is neutral or the former is neutral and the latter is positive.*

Proof. Let $e_1, e_2 \in \mathfrak{D}_1 \parallel \mathfrak{D}_2$ such that $e_1 \leftarrow_{\mathfrak{D}_1 \parallel \mathfrak{D}_2} e_2$. It follows by construction and by alternation of \mathfrak{D}_1 and \mathfrak{D}_2 that either $\lambda_{\mathfrak{D}_1 \parallel \mathfrak{D}_2}(e_1)$ and $\lambda_{\mathfrak{D}_1 \parallel \mathfrak{D}_2}(e_2)$ are opposite, either they are both \pm , either the former is negative and the latter is neutral, either the former is neutral and the latter is positive, either the former is positive and the latter is neutral or either the former is neutral and the latter is negative. However the last two cases are not possible since Lemma 5.2.14. So we conclude. \square

5.3 Finitary linear π -calculus

The linear π -calculus is a process language based on π -calculus. The syntax we are going to introduce in this section is the asynchronous version without replication of linear π -calculus with free asynchronous output, given in [Yoshida et al., 2004]. This calculus is augmented with a success signal, denoted with \surd .

As usual, we assume an infinite numerable set of *names* of channel, ranging over by $a, b, c, u, v, x, y, z, \dots$. We denote \tilde{x} to be a (possibly empty) finite sequence of pairwise distinct names, and we write $|\tilde{x}|$ for its length. The untyped syntax of finitary linear π -calculus is generated by the following grammar.

$$P, Q ::= a(\tilde{x}).P \mid \bar{a}\langle\tilde{x}\rangle \mid P \parallel P \mid (x)P \mid \emptyset \mid \surd$$

The syntax deserves some explanation. $a(\tilde{x}).P$ is the usual **prefixed input** construct: it denotes the reception of some names \tilde{x} on the channel a ; these names can be

eventually used in the body P , which is blocked until an input on a is performed. This construct binds all the free occurrences of the names \tilde{x} . $\bar{a}\langle\tilde{x}\rangle$ is the **asynchronous free output** construct: it denotes the emission of the names \tilde{x} along the channel a . In both input and output, we call a the *subject of the action*, while \tilde{x} are called the *objects of the action*. $P\|Q$ is the **parallel composition** of processes. Sometimes, we use the infix notation $\prod_{i=1}^n P_i$ to denote the process $P_1\|P_2\|\dots\|P_n$. $(x)P$ denotes the **restriction** of the name x in the process P . It binds all the free occurrences of the name x in P . There are two kinds of **termination** constructs in this calculus. The first one is the usual \emptyset which is the neutral element of parallel composition. The second one \surd denotes a good termination or a success. We need the success signal because it is the syntactical counterpart of the daimon action \star .

Given a process P , we denote with $\text{FN}(P)$ the set of free names of P , defined in the standard way. A process P is *closed* when $\text{FN}(P) = \emptyset$. Further $P[y/x]$ denotes the substitution to all free occurrences of x with y in P .

Since our language is conceived as the syntactical counterpart of designs, on the untyped calculus, we impose the following syntactical constraint:

IO - alternation in all processes under the form $a(\tilde{x}).P$, all free names of P are never used as subject of an input action.

Definition 5.3.1 (Structural congruence). \equiv is the smallest congruence over processes containing α -equivalence and the following axioms

1. $P\|\emptyset \equiv P$, $P\|Q \equiv Q\|P$, $P\|(Q\|R) \equiv (P\|Q)\|R$.
2. $(x)(y)P \equiv (y)(x)P$, $(x)(P\|Q) \equiv P\|(x)Q$ if $x \notin \text{FN}(P)$.
3. $(x)\emptyset \equiv \emptyset$.

Axioms in (1) tell us that parallel composition is commutative and has \emptyset as neutral element. Axioms in (2) are the usual scope-extrusion rules. Axioms in (3) are the structural rules dealing with the termination constructs.

Proposition 5.3.1. Every process P is structurally equivalent to one under the form $(\tilde{x})\prod_{i=1}^n P_i$ where either $P_i = a(\tilde{y}).Q$, $P_i = \bar{a}\langle\tilde{y}\rangle$ or $P_i = \surd$ (we call **atomic** such processes P_i).

Proof. By structural induction on P . □

The operational semantics is given in term of reduction rules.

Definition 5.3.2 (Reduction). \longrightarrow is the least relation between processes satisfying the following rules:

$ \tilde{x} = \tilde{y} \Rightarrow a(\tilde{x}).P\ \bar{a}\langle\tilde{y}\rangle \longrightarrow P[\tilde{y}/\tilde{x}]$	$P \equiv P' \longrightarrow Q' \equiv Q \Rightarrow P \longrightarrow Q$
$P \longrightarrow P' \Rightarrow P\ Q \longrightarrow P'\ Q$	$P \longrightarrow P' \Rightarrow (x)P \longrightarrow (x)P'$

As usual, we denote with \Longrightarrow the reflexive transitive closure of \longrightarrow .

$\frac{}{\emptyset \triangleright \emptyset} \text{ (z)}$	$\frac{}{\sqrt{\triangleright} \emptyset} \text{ (s)}$	$\frac{P \triangleright \mathcal{I} \quad Q \triangleright \mathcal{J}}{P \parallel Q \triangleright \mathcal{I} \odot \mathcal{J}} \text{ (par)}$	$\frac{P \triangleright \mathcal{I}}{P \triangleright \mathcal{I}, a^\uparrow} \text{ (w)}$
$\frac{P \triangleright x^\uparrow, \mathcal{I}}{(x)P \triangleright \mathcal{I}} \text{ (res)}$	$\frac{P \triangleright x_1^-, \dots, x_n^-, \mathcal{I}^-}{a(\bar{x}).P \triangleright a^+, \mathcal{I}^-} \text{ (in)}$	$\frac{}{\bar{a}(\bar{x}) \triangleright a^-, x_1^-, \dots, x_n^-} \text{ (out)}$	

 Table 5.1: Typing rules for finitary π -calculus

5.3.1 Typing system

In this section we introduce a typing discipline for our processes. The main purpose is to extract those processes which are linear in a syntactic sense, since only those processes can be soundly interpreted into a Multidesign: more specifically a process P which can be reasonably interpreted into a design should satisfy the following properties: every free name of P should appear exactly once either as subject of an input, or as subject or object of an output. Observe that the typing discipline we use checks only this property, so for example there is no arity check made by types.

The main idea is to use a very elementary form of typing for names in processes, namely **action modalities** ranged over p, q, \dots : $+$ is the *input modality*, $-$ is the *output modality*, while \uparrow corresponds to a neutral modality denoting a match between dual names. On modalities we define \odot to be a partial operator such that $+\odot- = -\odot+ = \uparrow$.

An **interface** is a finite set of pairs consisting of a name together with a modality (denoted x^p), where every name appears at most once and there is at most one name having modality $+$. We use $\mathcal{I}, \mathcal{J}, \dots$ to range over interfaces. We write \mathcal{I}^p to denote the interface \mathcal{I} where all names have modality p (where p could be either $-$ or \uparrow by definition). We denote with $\text{FN}(\mathcal{I})$ the set of names appearing in the interface \mathcal{I} and we denote with $\mathcal{I}(a)$ to be the modality assigned to a by the interface \mathcal{I} (which is unique, by definition). We use to write an interface as a decorated sequent, so we denote with \mathcal{I}, x^p the interface $\mathcal{I} \uplus \{x^p\}$. We also define $\mathcal{I} \setminus \mathcal{J} = \{x^p \in \mathcal{I} \mid x \notin \text{FN}(\mathcal{J})\}$

We extend the partial operator \odot to interfaces in the following way.

$$\mathcal{I} \odot \mathcal{J} = (\mathcal{I} \setminus \mathcal{J}) \cup (\mathcal{J} \setminus \mathcal{I}) \cup \{x^{p \odot q} \mid x^p \in \mathcal{I}, x^q \in \mathcal{J}\}$$

It is easy to see that \odot is a partial commutative associative operator and it is undefined when there is $x^p \in \mathcal{I}$ and $x^q \in \mathcal{J}$ such that $p \odot q$ is undefined.

Valid typing judgements have the shape $P \triangleright \mathcal{I}$, where P is a process and \mathcal{I} is an interface.

Definition 5.3.3. A typing judgement is valid when it is the conclusion of a derivation respecting the typing rules given in Table 5.1.

Theorem 5.3.1 (Subject reduction). Let $P \triangleright \mathcal{I}$. Then, if $P \longrightarrow Q$ then $Q \triangleright \mathcal{I}$.

Proof. The proof of Subject Reduction Theorem is just an adaptation of the ones given in [Berger et al., 2001, Yoshida et al., 2004, Paolini and Piccolo, 2009]. As usual we

will first prove that the set of typed term is closed under structural congruence. The proof is done by cases on the structural congruence axiom used. For axioms in (1) we can conclude since the operator \odot on interfaces is commutative, associative and it has the empty interface as neutral element. For axioms in (2), we can conclude just by manipulating opportunely the type derivation. For axioms in (3), for the first one $\emptyset \equiv (\mathbf{x})\emptyset$ we can conclude since occurrences of names with neutral modality can be weakened. After that we can prove that the set of typed term is closed under reduction. The most interesting case is whenever $a(\tilde{\mathbf{x}}).P \parallel \bar{a}(\tilde{\mathbf{y}}) \triangleright \mathcal{I}$ reduces to $P[\tilde{\mathbf{y}}/\tilde{\mathbf{x}}]$. But since $\mathcal{I} = a^\dagger, \tilde{\mathbf{y}}^-, \mathcal{I}'$ and by typability of $P \triangleright \mathcal{I}', \mathbf{x}^-$, we can easily conclude $P[\tilde{\mathbf{y}}/\tilde{\mathbf{x}}]$ as required. \square

Typing rules are very elementary. They do not perform any kind of arity check. However they check whether a free name appearing in a process is used exactly once as subject of an input and at most once as subject or object of an output. The key rules are **(in)** that checks *alternation* and **(par)** that checks composability of processes, ensuring linearity policy.

Proposition 5.3.2 (Linearity). *Let $P \triangleright \mathcal{I}, \mathbf{x}^p$. Then*

- if $p = +$ then \mathbf{x} appears exactly once in P as subject of an input.
- if $p = -$ then \mathbf{x} appears exactly once in P either as subject of or as object of an output.

Proof. By induction on the derivation of $P \triangleright \mathcal{I}, \mathbf{x}^p$. \square

Linearity implies strong confluence and strong normalisation in our linear setting

Theorem 5.3.2. *Let $P \triangleright \mathcal{I}$.*

Confluence *If $P \longrightarrow Q_i$ ($i \in \{1, 2\}$) then there exists R such that $Q_i \longrightarrow R$*

Strong normalisation *For all possible strategies over \longrightarrow , P reduces to a normal form in a finite number of steps.*

Proof. Confluence can be proved by a straightforward case analysis, as in [Yoshida et al., 2004], observing that there are no critical pairs. Concerning strong normalisation, notice that one \longrightarrow -step reduction reduces a process P to a process P' having a less number of syntactical constructs than P . So termination holds trivially. \square

An interface is **neutral free** when it does not contain any name with neutral modality. Notice that if P is a process such that $P \triangleright \mathcal{I}$ with \mathcal{I} neutral free, then, by construction, if P contains a sub-process under the form $a(\tilde{\mathbf{x}}).Q \parallel \bar{a}(\tilde{\mathbf{y}})$, then in P the name a is restricted somewhere before such sub-process.

5.3.2 Observational equivalence

We define a notions of typed behavioural equivalences between π -processes, based on a predicate of success. It is called *testing equivalence* and it is similar to the one presented in [Hennessy, 1991]. Let $n \in \mathbb{N}$; we define \sqrt{n} as $\sqrt{0} = \emptyset$ and $\sqrt{n+1} = \sqrt{n} \parallel \sqrt{n}$.

Definition 5.3.4 (Testing equivalence). $\approx_{\mathcal{I}}$ is the greatest equivalence over processes which are well typed on the same interface \mathcal{I} and which is such that if $P \approx_{\mathcal{I}} Q$ then, for all context C such that $C[P], C[Q] \triangleright \emptyset$, $C[P] \Longrightarrow \sqrt{n+1} \parallel \mathbb{R}_1$ if and only if $C[Q] \Longrightarrow \sqrt{n+1} \parallel \mathbb{R}_2$.

We write $P \approx Q$ whenever there exists \mathcal{I} such that $P \approx_{\mathcal{I}} Q$. Following [Berger et al., 2001, Honda and Yoshida, 1995] we can prove that \approx is consistent (i.e. it does not equate all processes), it is reduction closed, it is maximally consistent (i.e. the only typed congruence which strictly include \approx is the universal relation) and it equates all insensitive processes (i.e. processes that does not produce any observation) with the same type. The following lemma will be useful in the following.

Lemma 5.3.1. $(a)(\bar{a}(\tilde{x}).P \parallel C[\bar{a}(\tilde{y})]) \approx C[P[\tilde{y}/\tilde{x}]]$, under the hypothesis of typability of the two processes

Proof. The proof is by induction on C , noting that C contains exactly one hole, by typability. The only interesting case is $C = b(\tilde{z}).C'$. We first show that $(a)(\bar{a}(\tilde{x}).P \parallel b(\tilde{z}).Q) \approx b(\tilde{z}).(a)(\bar{a}(\tilde{x}).P \parallel Q)$. This is true since for both terms, the first reduction involves the name b for both terms. Then, by applying induction, we have $(a)(\bar{a}(\tilde{x}).P \parallel b(\tilde{z}).C'[\bar{a}(\tilde{y})]) \approx b(\tilde{z}).(a)(\bar{a}(\tilde{x}).P \parallel C'[\bar{a}(\tilde{y})]) \approx b(\tilde{z}).C'[P[\tilde{y}/\tilde{x}]]$ as required. \square

5.4 Interpreting finitary linear π -calculus

In this section we define an interpretation from π -processes into multidesigns (denoted with $\llbracket \cdot \rrbracket$) and we will show that it is fully abstract with respect to testing equivalence. The interpretation is defined only on processes typed in a neutral free interface.

5.4.1 Defining the model

A **loci-environment** ϕ is a function associating to each name x an address ξ , such that if $x \neq y$, then $\phi(x)$ and $\phi(y)$ are two disjoint loci.

Let ϕ be a loci environment and let ξ be a locus. Then $\phi[\xi/x]$ is the environment such that $\phi[\xi/x](x) = \xi$, but if $y \neq x$ then $\phi[\xi/x](y) = \phi(y)$. Furthermore, given a finite set $I \subset_{fin} \mathbb{N}$ such that we always assume a canonical indexing of the elements of $I = \{i_1, \dots, i_n\}$ (which could be for instance the indexing induced by putting the element in increasing order). We denote $\phi[\xi \cdot I/\tilde{x}]$ (where $I = \{i_1, \dots, i_n\}$ with $n = |\tilde{x}|$) to be the loci environment $\phi[\xi \cdot i_1/x_1] \dots [\xi \cdot i_n/x_n]$.

An **action-environment** δ is a function taking a locus ξ and an arity n and giving back an action (ξ, I) such that $|I| = n$.

We first define the **mapping from neutral free interfaces to bases**. Let $\mathcal{I} = \{x_i^+ | i \in [1, n]\} \cup \{y_j^- | j \in [1, m]\}$ be a neutral free interface and let ϕ be a loci-environment. Then we define

$$\llbracket \mathcal{I} \rrbracket^\phi = \{\phi(x_i)^+ | i \in [1, n]\} \cup \{\phi(y_j)^- | j \in [1, m]\}$$

The **interpretation of typed process into multidesigns** is defined in the table below.

$\llbracket \sqrt{} \rrbracket^{\phi, \delta} = \mathfrak{D} \text{ai}$	$\llbracket \emptyset \rrbracket^{\phi, \delta} = \mathfrak{F} \text{id}$
$\llbracket a(\tilde{x}).P \rrbracket^{\phi, \delta} = (\xi, J) \ominus \llbracket P \rrbracket^{\phi[\xi:J/\tilde{x}], \delta}$ where $(\xi, J) = \delta(\phi(a), \tilde{x})$	
$\llbracket \bar{a}(\tilde{x}) \rrbracket^{\phi, \delta} = (\xi, J) \ominus \bigoplus_{j_k \in J} \mathfrak{F} \alpha_{\xi, j_k, \phi(x_k)}$ where $(\xi, J) = \delta(\phi(a), \tilde{x})$	
$\llbracket (\tilde{x}) \prod_{i=1}^n P_i \rrbracket^{\phi, \delta} = \llbracket P_1 \rrbracket^{\phi, \delta} \circ \dots \circ \llbracket P_n \rrbracket^{\phi, \delta}$	

Interpretation deserves some explanations. To define the interpretation, we make use of Proposition 5.3.1, which tells that it is possible to decompose a process into a parallel composition of atomic processes, taking all the restrictions in the leftmost position. Let us begin from the **interpretation of a single atomic process**. The success signal $\sqrt{}$ is interpreted into the multidesign $\mathfrak{D} \text{ai}$. The input construct $a(\tilde{x}).P$ is interpreted by prefixing the action $\kappa = (\phi(a), J)$ (with $|J| = |\tilde{x}|$) to the multidesign corresponding to P , where to name \tilde{x} are associated the loci $\phi(a) \cdot J$. Note that the polarity of κ is negative. The asynchronous output construct $\bar{a}(\tilde{x})$ is interpreted by prefixing the action $\kappa = (\phi(a), J)$ (with $|J| = |\tilde{x}|$) to the multidesigns which perform the delocation from each locus $\phi(a) \cdot j_k$ ($j_k \in J$) to the locus $\phi(x_k)$. Note that the polarity of κ is positive. This allow us to interpret correctly the renaming induced by the contraction of the redex $a(\tilde{y}).P \llbracket \bar{a}(\tilde{x}) \rrbracket$. Finally, the process $Q \equiv (\tilde{x}) \prod_{i=1}^n P_i$ which is the **restriction of parallel composition of atomic processes** by taking the composition of multidesigns corresponding to the atomic processes P_i . Notice that the definition of interpretation is exhaustive by Proposition 5.3.1.

Example. Let $P_1 = a().\bar{b}(\langle \rangle)$ and $P_2 = b().\sqrt{}$ be two processes. Let ϕ be a loci environment such that $\phi(a) = \xi$ and $\phi(b) = \sigma$. Given any action environment δ we have that $\llbracket P_1 \rrbracket^{\phi, \delta} = \mathfrak{D}_1$, $\llbracket P_2 \rrbracket^{\phi, \delta} = \mathfrak{D}_2$ and $\llbracket (b)(P_1 \llbracket P_2 \rrbracket) \rrbracket^{\phi, \delta} = \mathfrak{D}$ where

- $\mathfrak{D}_1 = \{e_1, d_1\}$ with $e_1 \leq d_1$, $\lambda_{\mathfrak{D}_1}(e_1) = (\xi, \emptyset)$ and $\lambda_{\mathfrak{D}_1}(d_1) = (\sigma, \emptyset)$.
- $\mathfrak{D}_2 = \{e_2, d_2\}$ with $e_2 \leq d_2$, $\lambda_{\mathfrak{D}_2}(e_2) = (\sigma, \emptyset)$ and $\lambda_{\mathfrak{D}_2}(d_2) = \mathfrak{F}$.
- $\mathfrak{D} = \{e, d\}$ with $e \leq d$, $\lambda_{\mathfrak{D}}(e) = (\xi, \emptyset)$ and $\lambda_{\mathfrak{D}}(d) = \mathfrak{F}$.

Theorem 5.4.1 (Soundness). *Let P be a process such that $P \triangleright \mathcal{I}$ with \mathcal{I} neutral free. Then $\llbracket P \rrbracket^{\phi, \delta}$ is a multidesign on base $\llbracket \mathcal{I} \rrbracket^\phi$*

Proof. The proof is an easy induction on the structure of P , using the alternative definition of the syntax, given in Proposition 5.3.1. Proposition 5.3.2 plays a key role in the case $P \equiv (\tilde{x}) \prod_{i=1}^n Q_i$, since this guarantees that the labelling morphism satisfies Linearity. \square

Lemma 5.4.1. *Let P such that $P \triangleright \mathcal{I}$ with \mathcal{I} non neutral and let $\mathbf{x} \in \text{FN}(\mathcal{I})$. Let ϕ be a loci environment and let θ be a delocation from $\phi(\mathbf{x})$ to $\phi(\mathbf{y})$. Then $\llbracket P[y/\mathbf{x}] \rrbracket^{\phi, \delta} = \theta(\llbracket P \rrbracket^{\phi, \delta})$.*

Proof. By induction on the structure of P , using the alternative definition of the syntax given by Proposition 5.3.1. \square

Theorem 5.4.2 (Weak Adequacy). *Let P such that $P \triangleright \mathcal{I}$ with \mathcal{I} neutral free. If $P \longrightarrow Q$ then $\llbracket P \rrbracket = \llbracket Q \rrbracket$*

Proof. Let P a process typed in a neutral free interface. We know, by definition of interpretation that if $P \equiv P'$ then $\llbracket P \rrbracket = \llbracket P' \rrbracket$. Furthermore we know that $P \equiv (\tilde{\mathbf{x}}) \prod_{i=1}^n P_i$. Since $P \longrightarrow Q$, we know that there exists j, k distinct such that $P_i = a(\tilde{\mathbf{y}}).R$ and $P_j = \bar{a}(\tilde{\mathbf{z}})$ and $Q \equiv (\tilde{\mathbf{x}})(\prod_{i \neq j, k} P_i \parallel R[\tilde{\mathbf{z}}/\tilde{\mathbf{y}}])$. Notice that, by Proposition 5.3.2 such j, k are also unique and without loss of generality, we assume $j < k$. Now let $\mathfrak{D} = \llbracket P_1 \rrbracket^{\phi, \delta} \circ \dots \circ \llbracket P_{j-1} \rrbracket^{\phi, \delta} \circ \llbracket P_{j+1} \rrbracket^{\phi, \delta} \circ \dots \circ \llbracket P_{k-1} \rrbracket^{\phi, \delta} \circ \llbracket P_{k+1} \rrbracket^{\phi, \delta} \circ \dots \circ \llbracket P_n \rrbracket^{\phi, \delta}$, let $(\xi, I) = \delta(\phi(a), |\tilde{\mathbf{y}}|)$, let $\mathfrak{C} = \llbracket R \rrbracket^{\phi[\xi \cdot I/\tilde{\mathbf{y}}], \delta}$ and let θ be a delocation from each $\xi \cdot I$ respectively to each $\phi(\tilde{\mathbf{z}})$. Then

$$\begin{aligned} \llbracket P \rrbracket^{\phi, \delta} &= \mathfrak{D} \circ ((\xi, I) \ominus \mathfrak{C}) \circ ((\xi, I) \ominus \biguplus_{i \in I} \mathfrak{F}ax_{\xi \cdot i, \phi(z_i)}) \\ &= \mathfrak{D} \circ \mathfrak{C} \circ \biguplus_{i \in I} \mathfrak{F}ax_{\xi \cdot i, \phi(z_i)} && \text{Lemma 5.2.3} \\ &= \mathfrak{D} \circ \theta(\mathfrak{C}) && \text{Lemma 5.2.2} \\ &= \mathfrak{D} \circ \llbracket R[\tilde{\mathbf{z}}/\tilde{\mathbf{y}}] \rrbracket^{\phi, \delta} && \text{Lemma 5.4.1} \\ &= \llbracket Q \rrbracket^{\phi, \delta} \end{aligned}$$

\square

5.4.2 Adequacy and correctness

We prove the correctness of $\llbracket \cdot \rrbracket$ with respect to testing equivalence by showing that the interpretation is adequate with respect to the operational semantics.

First of all, we define $Dead = \{P \mid P \triangleright \emptyset, \forall n \neq 0. P \not\equiv \sqrt{n}, P \not\rightarrow\}$. Observe that for all $P \in Dead$ we have $\llbracket P \rrbracket^{\phi, \delta} = \mathfrak{F}id$. We can prove the following

Proposition 5.4.1. *Let $P \triangleright \emptyset$. If $P \Longrightarrow \sqrt{n+1} \parallel Q$, with $Q \in Dead$, then $\llbracket P \rrbracket = \llbracket \sqrt{n+1} \rrbracket$*

Proof. The result follows by the observation above and Theorem 5.4.2. \square

To prove the other direction, we straightforwardly adapt a proof of Plotkin [Plotkin, 1977] for Scott-continuous domains, based on a computability argument in a Tait's style.

Definition 5.4.1. *Let P a process such that $P \triangleright \mathcal{I}$ with \mathcal{I} neutral free. The **computability predicate** is defined by the following cases*

- Case $\mathcal{I} = \emptyset$. Then $\text{Comp}(P)$ if and only if $\llbracket P \rrbracket =_d \llbracket \sqrt{n+1} \rrbracket$ implies $P \Longrightarrow \sqrt{n+1} \parallel Q$ with $Q \in Dead$.
- Case $\mathcal{I} \neq \emptyset$. $\text{Comp}(P)$ if and only if $\text{Comp}((\mathbf{x}_1) \dots (\mathbf{x}_k)(P \parallel Q_1 \parallel \dots \parallel Q_n))$ for each Q_i such that $\text{Comp}(Q_i)$, $Q_i \Longrightarrow R_i$ with R_i atomic and $\prod Q_i \triangleright \mathcal{J}$ with $\mathcal{I} \odot \mathcal{J} = \{\mathbf{x}_1^\uparrow, \dots, \mathbf{x}_k^\uparrow\}$

Observe that if then for all $P \in Dead$ we have vacuously $Comp(P)$. Moreover, for all $P \in Dead$ we have $\llbracket P \rrbracket = \mathfrak{F}id$.

Lemma 5.4.2. *Let P a process such that $P \triangleright \mathcal{I}$, with \mathcal{I} neutral free. Then $Comp(P)$.*

Proof. The proof is by structural induction on P , using the alternative syntax provided by Proposition 5.3.1.

- $P = \emptyset$. Obvious
- $P = \sqrt{\quad}$. Obvious.
- $P = a(\tilde{x}).P'$. Suppose $\llbracket (a)(b_1) \dots (b_n)(a(\tilde{x}).P||Q_1|| \dots ||Q_k) \rrbracket =_d \llbracket \sqrt{\quad} \rrbracket$. Let us proceed by cases.
 - if there is $Q_j \implies \bar{a}\langle \tilde{z} \rangle$ for some R, \tilde{y} and \tilde{z} . If $|\tilde{z}| = |\tilde{x}|$, then by applying Theorem 5.4.2 we have

$$\begin{aligned} \llbracket (a)(b_1) \dots (b_n)(a(\tilde{x}).P'||Q_1|| \dots ||Q_k) \rrbracket &=_d \llbracket (a)(b_1) \dots (b_n)(a(\tilde{x}).P'|\bar{a}\langle \tilde{z} \rangle|| \prod_{i \neq j} Q_i) \rrbracket \\ &=_d \llbracket (a)(b_1) \dots (b_n)(\tilde{y})(P'[\tilde{z}/\tilde{x}]|| \prod_{i \neq j} Q_i) \rrbracket \\ &=_d \llbracket \sqrt{\quad} \rrbracket \end{aligned}$$

and we conclude by applying inductive hypothesis. If $|\tilde{z}| < |\tilde{x}|$, we observe that $a(\tilde{x}).P|\bar{a}\langle \tilde{z} \rangle \in Dead$. Then $\llbracket a(\tilde{x}).P||Q_j \rrbracket = \llbracket a(\tilde{x}).P|\bar{a}\langle \tilde{z} \rangle \rrbracket = \mathfrak{F}id$ and we conclude by definition of $Comp$.

- there is no $Q_j \implies \bar{a}\langle \tilde{z} \rangle$. This means that there is $Q_j \implies c(\tilde{y}).Q$ with $a \in FN(Q)$. If $Q_1|| \dots ||Q_j|| \dots ||Q_k \implies \bar{a}\langle \tilde{z} \rangle||R$, then we can reason as the previous point. If this is not the case, then the output on a is blocked somewhere by some prefix. This means that, by definition of $Comp$ that $(a)(b_1) \dots (b_n)(a(\tilde{x}).P'||Q_1|| \dots ||Q_k) \implies \sqrt{\quad}||R$ where $a(\tilde{x}).P$ is a subprocess of R .
- $P = \bar{a}\langle \tilde{y} \rangle$. Suppose $\llbracket (a)(\tilde{x})(\bar{a}\langle \tilde{y} \rangle||Q) \rrbracket = \llbracket \sqrt{\quad} \rrbracket$. Observe that by typing $Q \equiv (\tilde{w})(a(\tilde{z}).R|| \prod Q_i)$. If $|\tilde{y}| = |\tilde{z}|$, this implies by Theorem 5.4.2 $\llbracket (\tilde{x})(R[\tilde{y}/\tilde{z}]|| \prod Q_i) \rrbracket = \llbracket \sqrt{\quad} \rrbracket$ and we conclude by the previous point and by hypothesis. Otherwise, we use a similar reasoning as the second case of the previous point and we conclude.
- $P \equiv (\tilde{x}) \prod_{i=1}^n P_i$, we observe that by Proposition 5.3.1, we can assume without loss of generality that P_i is atomic. Since $Comp(P_i)$ by inductive hypothesis, we conclude immediately by definition.

□

Proposition 5.4.2 (Computational Adequacy). *Let $P \triangleright \emptyset$. $\llbracket P \rrbracket = \llbracket \sqrt{\quad} \rrbracket$ if and only if $P \implies \sqrt{\quad}||Q$, with $Q \in Dead$.*

Proof. By Proposition 5.4.1 and Lemma 5.4.2. □

Lemma 5.4.3. $\llbracket P \rrbracket = \llbracket Q \rrbracket$ implies that, for all context C $C[P] \Longrightarrow \sqrt{n+1} \parallel R_1$ if and only if $C[Q] \Longrightarrow \sqrt{n+1} \parallel R_2$, for all $R_1, R_2 \in \text{Dead}$.

Proof. Let P, Q such that $\llbracket P \rrbracket = \llbracket Q \rrbracket$ and let C such that $C[P] \Longrightarrow \sqrt{n+1} \parallel R$ with $R \in \text{Dead}$. Then by Proposition 5.4.2 $\llbracket C[P] \rrbracket = \llbracket \sqrt{n+1} \rrbracket$. But by hypothesis, we have $\llbracket C[Q] \rrbracket = \llbracket \sqrt{n+1} \rrbracket$. Then again by Proposition 5.4.2 we can conclude $C[Q] \Longrightarrow \sqrt{n+1} \parallel R_2$ with $R_2 \in \text{Dead}$, as required. In the same way, we can show that, if $C[Q] \Longrightarrow \sqrt{n+1} \parallel R_2$ then $C[P] \Longrightarrow \sqrt{n+1} \parallel R_1$, with $R_1, R_2 \in \text{Dead}$. \square

Lemma 5.4.4. If for all context C $C[P] \Longrightarrow \sqrt{n+1} \parallel R_1$ if and only if $C[Q] \Longrightarrow \sqrt{n+1} \parallel R_2$, for all $R_1, R_2 \in \text{Dead}$, then $P \approx Q$.

Proof. Let P such that $C[P] \Longrightarrow \sqrt{n+1} \parallel R_1$. Since the calculus is strongly normalising (Theorem 5.3.2) we have that exists R'_1 such that $R_1 \Longrightarrow R'_1 \not\rightarrow$. By construction $R'_1 \equiv \sqrt{m} \parallel R''_1$ with $R''_1 \in \text{Dead}$, obtaining $C[P] \Longrightarrow \sqrt{n+m+1} \parallel R''_1$. Thus also $C[Q] \Longrightarrow \sqrt{n+m+1} \parallel R''_2$ with $R''_2 \in \text{Dead}$. Thus we conclude. The converse can be obtained in a similar way. \square

Corollary 5.4.1 (Correctness). $\llbracket P \rrbracket =_d \llbracket Q \rrbracket$ implies $P \approx Q$

5.4.3 Completeness and full abstraction

We prove the completeness of $\llbracket \cdot \rrbracket$ with respect to testing equivalence. We use a definability argument, namely we show that all finite conflict free multidesigns are definable by a term of the language. Thus, given two terms interpreted into two different multidesigns, we can build a separating context which allows us to conclude that they are operationally distinct.

Proposition 5.4.3 (Finite Definability). Let \mathcal{D} a finite multidesign on interface Γ which is also a conflict free event structure. Then there exists a P such that $P \triangleright \mathcal{I}$ with $\llbracket \mathcal{I} \rrbracket^\phi = \Gamma$ and $\llbracket P \rrbracket^{\phi, \delta} = \mathcal{D}$

Proof. We say that a process P defines a design $\mathcal{D} : \Gamma$ when there exists $\phi, \delta, \mathcal{I}$ such that $P \triangleright \mathcal{I}$ with $\llbracket \mathcal{I} \rrbracket^\phi = \Gamma$ and $\llbracket P \rrbracket^{\phi, \delta} = \mathcal{D}$. We show by induction on the size of \mathcal{D} that there is P defining the design \mathcal{D} . If \mathcal{D} is empty, then we consider two cases:

- Γ is positive. Then $\mathcal{D} = \mathfrak{F}\text{id}$. If Γ is empty, then if we let $P = \emptyset$, then P defines $\mathfrak{F}\text{id}$ for any ϕ and δ . Otherwise $\Gamma = \{\xi_1^+, \dots, \xi_k^+\}$. and it is easy to check that the process

$$(a)(b)\left(a().\bar{b}\langle \rangle \parallel b().(\bar{a}\langle \rangle \parallel \bar{a}_1\langle \rangle) \parallel \dots \parallel \bar{a}_k\langle \rangle\right)$$

defines $\mathfrak{F}\text{id}$. It suffices to take a ϕ such that $\xi_1 = \phi(a_1), \dots, \xi_k = \phi(a_k)$ and any δ .

- Γ is negative. Then $\Gamma = \{\sigma_1^-, \dots, \sigma_n^-, \dots, \xi_1^+, \dots, \xi_m^+\}$ and $\mathcal{D} = \mathfrak{S}\text{funf}$. Let Q be the process defining $\mathfrak{F}\text{id}$ on the interface $\{\xi_1^+, \dots, \xi_m^+\}$ as in previous point. Then the process

$$c_1().\emptyset \parallel \dots \parallel c_m().\emptyset \parallel Q$$

defines $\mathfrak{S}\text{funf}$. It suffice to take a ϕ such that $\phi(c_1) = \sigma_1, \dots, \phi(c_m) = \sigma_m$.

If \mathfrak{D} is not empty, then also Γ is. Let us choose $\xi^\varepsilon \in \Gamma$ such that there is an event $e \in \mathfrak{D}$ labelled with an action having focus ξ (observe that such a locus always exists in Γ)
 If \mathfrak{D} is not empty, then also Γ is. Let us choose $\xi^\varepsilon \in \Gamma$ such that there is an event $e \in \mathfrak{D}$ labelled with an action having focus ξ (observe that such a locus always exists in Γ by Justification).

- $\varepsilon = +$. Then $\mathfrak{D} = ((\xi, I) \ominus \mathfrak{C} \uplus \mathfrak{D}',$ where $\mathfrak{C} : \{\sigma^- \mid \sigma \in \xi I\} \otimes \Gamma_1$ and $\mathfrak{D}' : \Gamma_2$ for some Γ_1, Γ_2 such that $\Gamma = \Gamma_1 \otimes \Gamma_2$, since \mathfrak{D} is conflict free. By induction we know that there are $P : \mathcal{I}$ and $Q : \mathcal{J}$ defining \mathfrak{C} and \mathfrak{D}' . Let $\mathbf{x}_1, \dots, \mathbf{x}_n \in \text{FN}(\mathcal{I})$ be the names assigned to $\xi \cdot i_1, \dots, \xi \cdot i_n$ ($i_j \in I$) by the loci environment. Then

$$(\mathbf{x}_1) \dots (\mathbf{x}_n) (\bar{a} \langle \mathbf{x}_1, \dots, \mathbf{x}_n \rangle \| P) \| Q$$

defines \mathfrak{D} , by setting $\phi(a) = \xi$.

- $\varepsilon = -$. Then $\mathfrak{D} = ((\xi, I) \ominus \mathfrak{C} \uplus \mathfrak{D}',$ where $\mathfrak{C} : \{\sigma^+ \mid \sigma \in \xi I\} \otimes \Gamma_1^+$ and $\mathfrak{D}' : \Gamma_2$ for some Γ_1, Γ_2 such that $\Gamma = \Gamma_1 \otimes \Gamma_2$, since \mathfrak{D} is conflict free. By induction we know that there are $P : \mathcal{I}$ and $Q : \mathcal{J}$ defining \mathfrak{C} and \mathfrak{D}' . Let $\mathbf{x}_1, \dots, \mathbf{x}_n \in \text{FN}(\mathcal{I})$ be the names assigned to $\xi \cdot i_1, \dots, \xi \cdot i_n$ ($i_j \in I$) by the loci environment. Then

$$a(\mathbf{x}_1, \dots, \mathbf{x}_n).P \| Q$$

defines \mathfrak{D} , by setting $\phi(a) = \xi$.

□

Theorem 5.4.3 (Completeness). $P \approx Q \Rightarrow \llbracket P \rrbracket =_d \llbracket Q \rrbracket$.

Proof. We show the contra-positive. Let $P, Q \triangleright \mathcal{I}$ such that $\llbracket P \rrbracket^{\phi, \delta} \neq_d \llbracket Q \rrbracket^{\phi, \delta}$. Let $\Gamma = \llbracket \mathcal{I} \rrbracket^{\phi, \delta}$. Then, since $\llbracket P \rrbracket^{\phi, \delta}$ and $\llbracket Q \rrbracket^{\phi, \delta}$ are finite multidesigns, then by Theorem 5.2.6, there is a finite multidesign $\mathfrak{C} : \Gamma^\perp$ such that $\llbracket P \rrbracket^{\phi, \delta} \circ \mathfrak{C} \neq \llbracket Q \rrbracket^{\phi, \delta} \circ \mathfrak{C}$, which is also conflict free. By Proposition 5.4.3, there is a closed process R such that $\llbracket R \rrbracket^{\phi, \delta} = \mathfrak{C}$. Thus we conclude by adequacy that $P \not\approx Q$, just by taking $(\check{z})([\cdot] \| R)$ as separating context (where $\check{z} = \text{FN}(\mathcal{I})$). □

Corollary 5.4.2 (Full abstraction). $P \approx Q \iff \llbracket P \rrbracket =_d \llbracket Q \rrbracket$.

5.5 Extending the finitary π -calculus

In this section we provide two extensions of the finitary π -calculus that again preserve linearity. The first one extends the finitary linear π -calculus with two conditionals constructs called **branching** and **selection**. The second one extends the calculus with **recursive definitions** allowing to model infinitary behaviour.

The new calculus is called **linear π -calculus** whose syntax is given by the following grammar.

$$\begin{array}{l}
 P, Q ::= M \quad | \quad \bar{a} \text{in}_j \langle \tilde{x} \rangle \quad | \quad \bar{a} \langle \tilde{x} \rangle \quad | \quad P \parallel Q \quad | \quad (x) P \\
 \quad \quad F \langle \tilde{x} \rangle \quad | \quad \text{let } F(\tilde{x}) = P \text{ in } [P] \quad | \quad \sqrt{\quad} \quad | \quad \emptyset \\
 M ::= a(\tilde{x}).P \quad | \quad \sum_{i=1}^n a \text{in}_i(\tilde{x}_i).P_i
 \end{array}$$

The **branching input** construct $\sum_{i=1}^n a \text{in}_i(\tilde{x}_i).P_i$ corresponds to an external choice construct, where each branch is under the form $a \text{in}_i(\tilde{x}_i).P_i$. The name a is called the *guard of the choice*, while the index i is called the *discriminator of the choice*. In the branch $a \text{in}_i(\tilde{x}_i).P_i$ the occurrences of the names \tilde{x}_i in P_i are bounded. Branches are selected using the selective output construct $\bar{a} \text{in}_j \langle \tilde{x} \rangle$. The reduction rule dealing with sum is the following

$$j \in [1, n] \Rightarrow \sum_{i=1}^n a \text{in}_i(\tilde{x}_i).P_i \parallel \bar{a} \text{in}_j \langle \tilde{y} \rangle \longrightarrow P_j[\tilde{y}/\tilde{x}_j]$$

It is standard to model infinitary behaviour in π -calculus by using recursive definition [Sangiorgi and Walker, 2001]. $F \langle \tilde{x} \rangle$ is the **invocation** of a definition F , where the names \tilde{x} are called *actual parameters*. $\text{let } F(\tilde{x}) = P \text{ in } [Q]$ is the **recursive definition**. Here names \tilde{x} are called *formal parameters* and their occurrences are bounded in P . The process variable F is bounded in Q . We denote $\text{FPV}(P)$ the set of free process variables of P defined in the standard way. Let \vec{P} a sequence of processes such that \tilde{x}_i are the free names of the process P_i . We denote $\text{let } \vec{F}(\vec{x}) = \vec{P} \text{ in } [Q]$ the process $\text{let } F_1(\tilde{x}_1) = P_1 \text{ in } [\dots \text{let } F_n(\tilde{x}_n) = P_n \text{ in } [Q] \dots]$

Structural congruence rules and reduction rules dealing with recursive definitions are the following

- $\text{let } F(\tilde{x}) = P \text{ in } [Q \parallel R] \equiv Q \parallel \text{let } F(\tilde{x}) = P \text{ in } [R]$ if $F \notin \text{FPV}(Q)$
- $\text{let } F(\tilde{x}) = P \text{ in } [(x) Q] \equiv (x) \text{let } F(\tilde{x}) = P \text{ in } [Q]$ if $x \notin \text{FN}(P)$
- $\text{let } F_1(\tilde{x}) = P \text{ in } [\text{let } F_2(\tilde{y}) = Q \text{ in } [R]] \equiv \text{let } F_2(\tilde{x}) = Q \text{ in } [\text{let } F_1(\tilde{y}) = P \text{ in } [R]]$ if $F_1 \notin \text{FPV}(Q)$ and $F_2 \notin \text{FPV}(P)$
- $\text{let } F(\tilde{x}) = P \text{ in } [\emptyset] \equiv \emptyset$
- $\text{let } F(\tilde{x}) = P \text{ in } [F \langle \tilde{y} \rangle \parallel Q] \longrightarrow \text{let } F(\tilde{x}) = P \text{ in } [P[\tilde{y}/\tilde{x}] \parallel Q]$

By analogy with the finitary case, we have also here a notion of process in canonical form, with respect to structural equivalence. So, an **atomic process** is either an input construct, a branching input, an output, a branching output, a success or an invocation of a recursive definition. We have the following.

Proposition 5.5.1. *Every process P is structurally equivalent to one of the form $(\vec{y}) \left(\text{let } \vec{F}(\vec{x}) = \vec{Q} \text{ in } [\prod_{i=1}^n P_i] \right)$ where each P_i is an atomic process.*

$\frac{}{\Gamma \vdash \emptyset \triangleright \emptyset} \text{ (z)}$	$\frac{}{\Gamma \vdash \sqrt{} \triangleright \emptyset} \text{ (s)}$	$\frac{\Gamma \vdash P \triangleright x^\downarrow, \mathcal{I}}{\Gamma \vdash (x)P \triangleright \mathcal{I}} \text{ (res)}$	$\frac{\Gamma \vdash P \triangleright \mathcal{I}}{\Gamma \vdash P \triangleright \mathcal{I}, a^\downarrow} \text{ (w)}$
$\frac{\Gamma \vdash P \triangleright x_1^-, \dots, x_n^-, \mathcal{I}^-}{\Gamma \vdash a(\tilde{x}).P \triangleright a^+, \mathcal{I}^-} \text{ (in)}$	$\frac{}{\Gamma \vdash \bar{a}(\tilde{x}) \triangleright a^-, \tilde{x}^-} \text{ (out)}$	$\frac{}{\Gamma \vdash \bar{a}in_j(\tilde{x}) \triangleright a^-, \tilde{x}^-} \text{ (sel)}$	
$\frac{\Gamma \vdash P \triangleright \mathcal{I} \quad \Gamma \vdash Q \triangleright \mathcal{J}}{\Gamma \vdash P \parallel Q \triangleright \mathcal{I} \odot \mathcal{J}} \text{ (par)}$		$\frac{\forall i \in [1, n] \Gamma \vdash a(\tilde{x}_i).P_i \triangleright a^+, \mathcal{I}^-}{\Gamma \vdash \sum_{i=1}^n a in_i(\tilde{x}_i).P_i \triangleright a^+, \mathcal{I}^-} \text{ (bra)}$	
$\frac{}{\Gamma, F : \lambda \tilde{x}. \mathcal{I} \vdash F(\tilde{y}) \triangleright \mathcal{I}[\tilde{y}/\tilde{x}]} \text{ (v)}$		$\frac{\Gamma, F : \lambda \tilde{x}. \mathcal{I} \vdash P \triangleright \mathcal{I} \quad \Gamma, F : \lambda \tilde{x}. \mathcal{I} \vdash Q \triangleright \mathcal{J}}{\Gamma \vdash \text{let } F(\tilde{x}) = P \text{ in } [Q] \triangleright \mathcal{J}} \text{ (def)}$	

 Table 5.2: Typing rule for the linear π -calculus

5.5.1 Recursion and linearity

To preserve linearity also in presence of recursion, we use ideas already developed in [Alves et al., 2006, Dal Lago, 2005, Paolini and Piccolo, 2008], by adapting the linear typing discipline already introduced in [Paolini and Piccolo, 2009].

The idea is to use **high order interfaces**, ranging over $\mathcal{H}, \mathcal{H}_1, \mathcal{H}_2, \dots$, which are under the form $\lambda \tilde{x}. \mathcal{I}$ where \mathcal{I} is a *neutral free interface* and \tilde{x} is a sequence of *abstracted names*. We impose that the set of names appearing in \mathcal{I} is exactly \tilde{x} . The occurrences of \tilde{x} are considered to be bounded in \mathcal{I} . In the following, we denote $\mathcal{I}[y/x]$ the interface obtained from \mathcal{I} by replacing the name x with the name y .

Process environments are set of pairs process variable plus high order interface (denoted $F : \mathcal{H}$) and they range over Γ, Δ, \dots . Given a process environment Γ , we denote with $FPV(\Gamma)$ the set of variable appearing in Γ and with $\Gamma(F)$ the high order interface assigned to F in Γ . Valid typing judgements have the form $\Gamma \vdash P \triangleright \mathcal{I}$, where Γ is a process environment, P is a process and \mathcal{I} is an interface.

Definition 5.5.1. A typing judgement $\Gamma \vdash P \triangleright \mathcal{I}$ is valid if it the conclusion of a derivation respecting rules in Table 5.2

Theorem 5.5.1. Let $\Gamma \vdash P \triangleright \mathcal{I}$. Then

Weakening $\Gamma, F : \mathcal{H} \vdash P \triangleright \mathcal{I}$

Strengthening if $F \notin FPV(P)$ then $\Gamma \setminus \{F : \Gamma(F)\} \vdash P \triangleright \mathcal{I}$.

Subject reduction If $P \longrightarrow Q$ then $\Gamma \vdash Q \triangleright \mathcal{I}$.

Proof. All the proofs can be done by induction on the derivation of $\Gamma \vdash P \triangleright \mathcal{I}$. The only non-obvious one is the proof of subject reduction when $P = \text{let } F(\tilde{x}) = P_1 \text{ in } [F(\tilde{y}) \parallel P_2]$ and $Q = \text{let } F(\tilde{x}) = P_1 \text{ in } [P_1[\tilde{y}/\tilde{x}] \parallel P_2]$. Observe that, since $\Gamma \vdash P_1 \triangleright \mathcal{I}$ then $\Gamma \vdash P_1[\tilde{y}/\tilde{x}] \triangleright \mathcal{I}[\tilde{y}/\tilde{x}]$. Moreover we can prove by an easy induction on the derivation of $\Gamma, F : \lambda \tilde{x}. \mathcal{I} \vdash F(\tilde{y}) \parallel P_2 \triangleright \mathcal{J}$, that $\Gamma, F : \lambda \tilde{x}. \mathcal{I} \vdash P_1[\tilde{y}/\tilde{x}] \parallel P_2 \triangleright \mathcal{J}$. Thus we can conclude simply by applying the rule **(def)**. \square

We allow weakening and contraction on process environments, while we treat linearly interfaces. Notice that, in rule **(bra)** interfaces are treated in an additive way, and rules **(def)** and **(v)** impose the use of F according to the type information given by the process environment. Notice that in rule **(def)** we impose that in the process $\text{let } F(\vec{x}) = P \text{ in } [Q]$ the only free names of P are \vec{x} . This is crucial in order to preserve typability by reduction, allowing safe duplication of the process P without duplicating free names. Due to this observation, for typed processes we can reformulate Proposition 5.5.1 in the following way. We will use such reformulation in the following section, in order to define the interpretation of typed processes into multidesigns.

Proposition 5.5.2. *Let P such that $\Gamma \vdash P \triangleright \mathcal{I}$. Then P is structurally equivalent to one of the form $\text{let } \vec{F}(\vec{x}) = \vec{Q} \text{ in } [(\vec{y}) \prod_{i=1}^n P_i]$ where each P_i is an atomic process*

Proof. By Proposition 5.5.1 we have that P is structurally equivalent to a process under the form $(\vec{y})(\text{let } \vec{F}(\vec{x}) = \vec{Q} \text{ in } [\prod_{i=1}^n P_i])$ where each P_i is an atomic process. Now, by typability of P and by subject reduction, we have that the only free name of each Q_j are only the formal parameter \vec{x}_j . So by applying the appropriate structural congruence axioms, we get the result. \square

Given a process P , a *slice* of P is a process obtained from P replacing all the branching input constructs appearing in P with one of their branches and repeating the procedure until they are all disappeared. We have the following.

Proposition 5.5.3 (Linearity). *Let $\Gamma \vdash P \triangleright \mathcal{I}, \mathbf{x}^p$. Then*

- *if $p = +$ then \mathbf{x} appears free in P exactly once either as subject of an input action, as guard of the choice in a branching input construct, or as parameter of a definition invocation.*
- *if $p = -$ then for all slices of P , \mathbf{x} appears exactly once either as a subject or object of an output or selective output or as parameter of a definition invocation.*

Proof. By induction on the derivation of $\Gamma \vdash P \triangleright \mathcal{I}, \mathbf{x}^p$. \square

Clearly, in presence of recursion, we loose **strong normalisation**. However **confluence** still holds in this linear calculus. Finally, the definition of **testing equivalence** is the straightforward extension of the definition given for the finitary case.

5.5.2 Encoding data and functions

The linear π -calculus is expressive enough to encode some basics data types like integers and also some functions able to manipulate them. Our aim is to show that our linear typed language is Turing-complete, i.e. it is able to represent all first-order computable functions.

We will use $\langle d \rangle$ to denote the encoding into a process of a datum (or a function) d . We will write $\bar{a}(\vec{x})P$ and $\bar{a}\text{in}_j(\vec{x})P$ as an abbreviation for the respective processes $(\vec{x})(\bar{a}(\vec{x})\|P)$ and $(\vec{x})(\bar{a}\text{in}_j(\vec{x})\|P)$.

Natural numbers To begin, let us consider the data type representing natural numbers, i.e. the type $\mathbb{N} = \{0, 1, \dots\}$. We can encode them in the following way:

$$\langle 0 \rangle^u = u(a).\bar{a}in_1\langle \rangle \quad \langle n + 1 \rangle^u = u(a).\bar{a}in_2(v) \langle n \rangle^v$$

By analogy with similar encodings [Hyland and Ong, 1995, Berger et al., 2001, Paolini and Piccolo, 2009] the parameter u is a name used as subject of an input action in the corresponding process and it could be thought as a name corresponding to the access point of the datum. We can prove, by induction on n that $\vdash \langle n \rangle^u \triangleright u^+$.

Operators on integers can be easily expressed by processes. Let us begin from the successor function *succ*

$$\langle succ \rangle_v^u = u(a).\bar{a}in_2(u') u'(a').\bar{v}\langle a' \rangle$$

Now, $\langle \cdot \rangle$ takes two name: the first one u is used as subject of an input action and as usual it is the access point where we can get the result; the second one v is used as subject of an output action and it is the access point the process uses in order to ask for the input. We have that

Proposition 5.5.4. 1. $\vdash \langle succ \rangle_v^u \triangleright u^+, v^-$

2. $(v)(\langle succ \rangle_v^u \parallel \langle n \rangle^v) \approx \langle n + 1 \rangle^u$

Proof. (1.) can be easily proved by applying rules **(out)**, **(sel)** and **(in)**. To prove (2.), we proceed by induction. The base case is easy, and it can be obtained simply by applying Lemma 5.3.1. For the inductive case we have

$$\begin{aligned} (v)(\langle succ \rangle_v^u \parallel \langle n + 1 \rangle^v) &\approx (v)(\langle succ \rangle_v^u \parallel v(a).\bar{a}in_2(w) \langle n \rangle^w) \\ &\approx (v)(\langle succ \rangle_v^u \parallel (w)(\langle succ \rangle_w^v \parallel \langle n \rangle^w)) && \text{Induction} \\ &\approx \langle n + 2 \rangle^u && \text{Lemma 5.3.1} \end{aligned}$$

□

By using the successor function, it is possible more complex operations between natural numbers, like the addition.

$$\langle add \rangle_{v,w}^u =$$

$$\text{let } F(u, v, w) = u(a).\bar{v}(z) \sum \left[\begin{array}{l} z in_1().\bar{w}\langle a \rangle \\ z in_2(b).(u')(v')(\bar{u}'\langle a \rangle \parallel \langle succ \rangle_{v'}^{u'} \parallel F(v', b, w)) \end{array} \right]$$

$$\text{in } F\langle u, v, w \rangle$$

Proposition 5.5.5. $(v)(w)(\langle add \rangle_{v,w}^u \parallel \langle n \rangle^v \parallel \langle m \rangle^w) \approx \langle n + m \rangle^u$

Proof. By induction on n . The base case is easy, while the inductive step can be obtained by applying Lemma 5.3.1. □

Pairs, duplicators and projections: Now that we have integers, we can build pairs of natural numbers. Given two natural number n, m the encoding of a pair $\llbracket n, m \rrbracket$ of type \mathbb{N}^2 is given by

$$\llbracket \llbracket n, m \rrbracket \rrbracket^u = u(a).\bar{a}(v_1, v_2) (\llbracket n \rrbracket^{v_1} \parallel \llbracket m \rrbracket^{v_2})$$

A remarkable feature of any finite datum, like an integer, is that it can be duplicated by a linear process. The duplicator process is defined in the following way

$$\begin{aligned} \llbracket dup \rrbracket_v^u = \\ \text{let } F(u, v) = u(a).\bar{v}(b) \sum \left[\begin{array}{l} b \text{ in}_1().\bar{a}(u_1, u_2) (\llbracket 0 \rrbracket^{u_1} \parallel \llbracket 0 \rrbracket^{u_2}) \\ b \text{ in}_2(v').(u')(F\langle u', v' \rangle \parallel \bar{u}'(a') a'(w_1, w_2).\bar{a}(v_1, v_2)) \\ \quad \quad \quad (\llbracket succ \rrbracket_{w_1}^{v_1} \parallel \llbracket succ \rrbracket_{w_2}^{v_2}) \end{array} \right] \\ \text{in } F\langle u, v \rangle \end{aligned}$$

It is also possible to define projections. For instance, the first projection is defined in the following way:

$$\begin{aligned} \llbracket \pi_1 \rrbracket_v^u = \text{let } F(u, v) = u(a).(b)(\bar{v}\langle b \rangle \parallel b(x, y).(z)(\bar{y}\langle z \rangle \parallel \\ \sum \left[\begin{array}{l} z \text{ in}_1().\bar{x}\langle a \rangle \\ z \text{ in}_2(w).(u')(v')(F\langle u', v' \rangle \parallel v'(b').\bar{b}'\langle x, w \rangle \parallel \bar{u}'\langle a \rangle) \end{array} \right] \parallel \\ \text{in } F\langle u, v \rangle) \end{aligned}$$

Proposition 5.5.6. 1. $(v)(\llbracket dup \rrbracket_v^u \parallel \llbracket n \rrbracket^v) \approx \llbracket \llbracket n, n \rrbracket \rrbracket^u$

2. $(v)(\llbracket \pi_1 \rrbracket_v^u \parallel \llbracket \llbracket n, m \rrbracket \rrbracket^v) \approx \llbracket n \rrbracket^u$

Turing completeness Generalised n-uple, generalised projections and generalised duplicators can be defined easily using linear processes as the straightforward extension of the binary case. Thus, we can state the following definition.

Definition 5.5.2. Let P be a process such that $\vdash P \triangleright u^+, v^-$. Let $f : \mathbb{N}^k \rightarrow \mathbb{N}$ be a function. We say that P defines the function f when for all $n_1, \dots, n_k \in \mathbb{N}$ we have

$$(v)(P \parallel \llbracket \llbracket n_1, \dots, n_k \rrbracket \rrbracket^v) \approx \llbracket f(n_1, \dots, n_k) \rrbracket^u$$

It is possible to prove that linear π -calculus is Turing-complete, by showing that Kleene's recursive functions are definable by opportune well typed π -processes.

Theorem 5.5.2 (Turing completeness). Let $f : \mathbb{N}^k \rightarrow \mathbb{N}$ be a computable function. Then there exists a process P such that $\vdash P \triangleright u^+, v^-$ which defines the function f .

Proof. The definability of zero, successor function and projections have been already showed above. Let us develop the remaining cases

Composition Let $h : \mathbb{N}^n \rightarrow \mathbb{N}$ and $g_1, \dots, g_n : \mathbb{N}^m \rightarrow \mathbb{N}$ be primitive recursive function and let $f : \mathbb{N}^m \rightarrow \mathbb{N}$ defined as

$$f(x_1, \dots, x_m) = h(g_1(x_1, \dots, x_m), \dots, g_n(x_1, \dots, x_m))$$

Let us develop the case $n, m = 2$: the generalisation is straightforward. Let $\vdash P \triangleright u^+, v^-$ be the process defining the function h and let $\vdash Q_1 \triangleright u_1^+, v_1^-$ and $\vdash Q_2 \triangleright u_2^+, v_2^-$ be the two processes defining respectively the functions g_1, g_2 . The function f is defined by the following process R:

$$R = (v) \left(P \| v(x). \bar{b}(y) y(y_1, y_2). (v y'_1) \left((dup)_{y'_1}^{y_1} \| \bar{y}'_1(z_1) z_1(c_1, d_1). (v y'_2) \left((dup)_{y'_2}^{y_2} \| \bar{y}'_2(z_2) z_2(c_2, d_2). (v v_1, v_2) (\bar{x}(u_1, u_2) (Q_1 \| Q_2 \| v_1(h_1). \bar{h}_1 \langle c_1, c_2 \rangle \| v_2(h_2). \bar{h}_2 \langle d_1, d_2 \rangle)) \right) \right) \right)$$

It is easy to see that $(b)(R \| (\llbracket n, m \rrbracket)^b) \approx (h(g_1(n, m), g_2(n, m)))^u$

Primitive recursion Let $h : \mathbb{N}^{n+2} \rightarrow \mathbb{N}$ and $g : \mathbb{N}^n \rightarrow \mathbb{N}$ be two primitive recursive functions and let $f : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ be the primitive recursive function defined as

$$f(k, x_1, \dots, x_n) = \begin{cases} g(x_1, \dots, x_n) & \text{if } k = 0 \\ h(f(k-1, x_1, \dots, x_n), k-1, x_1, \dots, x_n) & \text{otherwise} \end{cases}$$

Let us develop the case $n = 1$. The generalisation is straightforward. Let $\vdash P \triangleright u_1^+, v_1^-$ be the process defining the function h and let $\vdash Q \triangleright u_2^+, v_2^-$ be the process defining the function g . The process R defining the function f is the following

$$\text{let } F(u, v) = u(x). \bar{v}(y) y(y_1, y_2). \bar{y}_1(z_1) \left[z_1 \text{ in}_1(). R_1 + z_1 \text{ in}_2(k). R_2 \right] \text{in } F \langle u, v \rangle$$

where

$$R_1 = (u_2)(v_2) \left(Q \| v_2(w). \bar{y}_2 \langle w \rangle \| \bar{u}_2 \langle x \rangle \right)$$

and

$$R_2 = (y'_2) \left(\begin{array}{l} (dup)_{y'_2}^{y_2} \| \bar{y}'_2(z'_2) z'_2(c_1, d_1). (k') \left((dup)_k^{k'} \| \bar{k}'(z'_3) z'_3(c_2, d_2). (u_1)(v_1)(a)(b) \right) \\ (P \| v_1(h_1). \bar{h}_1 \langle a, c_2, c_1 \rangle \| F \langle a, b \rangle \| b(w). \bar{w} \langle d_2, d_1 \rangle) \end{array} \right)$$

We can show that $(v)(R \| (\llbracket 0, n \rrbracket)^v) \approx (g(n))^u$ and $(v)(R \| (\llbracket k+1, n \rrbracket)^v) \approx (h(f(k, n), k, n))^u$

Minimisation Let $h : \mathbb{N}^2 \rightarrow \mathbb{N}$ be a total recursive function and let $f : \mathbb{N} \rightarrow \mathbb{N}$ be the function defined as

$$f(x) = \begin{cases} \min\{k \in \mathbb{N} | h(x, k) = 0\} & \text{if such a } k \in \mathbb{N} \text{ exists} \\ \text{undefined} & \text{otherwise} \end{cases}$$

Let $\vdash P \triangleright u_1^+, v_1^-$ be the process defining the function h . The function f is defined by the following process R.

$$\text{let } F(u, v, w) = R_1 \text{ in } (w)(F \langle u, v, w \rangle \| (\llbracket 0 \rrbracket)^w)$$

where

$$\begin{aligned} & u(\mathbf{x}).(w')\left(\left(\text{dup}\right)_w^{w'} \|\bar{w}'(w'')\ w''(c_1, d_1).(v')\left(\left(\text{dup}\right)_v^{v'} \|\bar{v}'(v'')\ v''(c_2, d_2).\right.\right. \\ R_1 = & \left.\left. (u_1)(v_1)(P\|v_1(\mathbf{x}_1).\bar{x}_1\langle c_1, c_2\rangle\|\bar{u}_1(z_1)\left[z_1 \text{in}_1().(a)(b)(a(\mathbf{h}).\bar{h}\langle d_1, d_2\rangle\|(\pi_1)_a^b\|\bar{b}\langle \mathbf{x}\rangle\right)+\right.\right. \\ & \left.\left. z_1 \text{in}_2(z_2).(a)(b)(h_1)(\mathbf{k})\left(\left(\pi_1\right)_a^b\|\bar{b}\langle \mathbf{x}\rangle\|a(\mathbf{h}).\bar{h}\langle h_1, z_2\rangle\|F\langle h_1, d_2, \mathbf{k}\rangle\|(\text{succ})_{d_1}^k\right)\right)\right) \end{aligned}$$

It is easy to see that, if there exists $k \in \mathbb{N}$ such that $h(n, k) = 0$ then $(v)(R\|(\|n\|)^v) \approx (\|k'\|)^u$ where k' is the minimum such that $h(n, k') = 0$.

□

5.6 Interpreting linear π -calculus

In this section we extend the mapping $\llbracket \cdot \rrbracket$ to processes of linear π -calculus. We will prove that the equational theory induced on π -terms by $\llbracket \cdot \rrbracket$ contains the equalities induced by reduction rules of the calculus. We left the issue of studying correctness and completeness for future developments.

5.6.1 Defining the model

We need to define some additional definitions and notion with respect to those already introduced in Section 5.4.1. In particular we need to extend the notion of action environment, in order to deal with the new constructs of branching input and selective output. Furthermore we will introduce the notion of process variable environment, which is a function associating a multidesign to a process variable, to deal with recursive definitions and invocations.

So, an **action environment** δ is now a function taking a ludics address ξ , an arity n , a discriminator k and it gives back an action (ξ, I_k) such that $|I_k| = n$ and for two distinct discriminators k_1, k_2 we have $I_{k_1} \cap I_{k_2} = \emptyset$.

A **process variable environment** ρ is a function taking a process variable F and giving back a multidesign \mathcal{D} .

We denote with $\rho[\mathcal{D}/F]$ the process variable environment such that $\rho[\mathcal{D}/F](F) = \mathcal{D}$ while $\rho[\mathcal{D}/F](F') = \rho(F')$ for all $F' \neq F$. Given a loci environment ϕ and a process variable environment Γ , we say that ρ **respects** Γ in ϕ if for all F , if $\Gamma(F) = \lambda \bar{x}. \mathcal{I}$ then $\rho(F)$ is a multidesign on the base $\llbracket \mathcal{I} \rrbracket_\phi$.

The **interpretation of typed processes into multidesigns** is defined by using Proposition 5.5.2 and it is given below. The interpretation is defined only on processes typed on neutral free interfaces. Let P such that $\Gamma \vdash P \triangleright \mathcal{I}$, let ϕ be a loci environment, let δ be an action environment and let ρ be a process variable environment which respects Γ in ϕ . Then $\llbracket P \rrbracket_\rho^{\phi, \delta}$ is the multidesign defined in the following way:

$\llbracket \sqrt{} \rrbracket_\rho^{\phi, \delta} = \mathfrak{D} \text{ai}$	$\llbracket \mathcal{O} \rrbracket_\rho^{\phi, \delta} = \mathfrak{F} \text{id}$
$\llbracket \mathbf{a}(\tilde{\mathbf{x}}).P \rrbracket_\rho^{\phi, \delta} = (\xi, J) \ominus \llbracket P \rrbracket_\rho^{\phi[\xi/J/\tilde{\mathbf{x}}], \delta}$ where $(\xi, J) = \delta(\phi(\mathbf{a}), \tilde{\mathbf{x}} , 1)$	
$\llbracket \bar{\mathbf{a}}(\tilde{\mathbf{x}}) \rrbracket_\rho^{\phi, \delta} = (\xi, J) \ominus \bigcup_{j_k \in J} \mathfrak{F} \alpha_{\xi \cdot j_k, \phi(\mathbf{x}_k)}$ where $(\xi, J) = \delta(\phi(\mathbf{a}), \tilde{\mathbf{x}} , 1)$	
$\llbracket (\tilde{\mathbf{x}}) \prod_{i=1}^n P_i \rrbracket_\rho^{\phi, \delta} = \llbracket P_1 \rrbracket_\rho^{\phi, \delta} \circ \dots \circ \llbracket P_n \rrbracket_\rho^{\phi, \delta}$	
$\llbracket \sum_{i=1}^n \mathbf{a} \text{in}_i(\tilde{\mathbf{x}}_i).P_i \rrbracket_\rho^{\phi, \delta} = \sum_{i=1}^n (\xi, J_i) \ominus \llbracket P_i \rrbracket_\rho^{\phi[\xi/J_i/\tilde{\mathbf{x}}_i], \delta}$ where $(\xi, J_i) = \delta(\phi(\mathbf{a}), \tilde{\mathbf{x}}_i , i)$	
$\llbracket \bar{\mathbf{a}} \text{in}_i(\tilde{\mathbf{x}}) \rrbracket_\rho^{\phi, \delta} = (\xi, J) \ominus \bigcup_{j_k \in J} \mathfrak{F} \alpha_{\xi \cdot j_k, \phi(\mathbf{x}_k)}$ where $(\xi, J) = \delta(\phi(\mathbf{a}), \tilde{\mathbf{x}} , i)$	
$\llbracket F(\tilde{\mathbf{y}}) \rrbracket_\rho^{\phi, \delta} = \theta(\rho(F))$ where θ is a delocation such that $\theta(\phi(\mathbf{x}_i)) = \phi(\mathbf{y}_i)$ and $\Gamma(F) = \lambda \tilde{\mathbf{x}}. \mathcal{I}$	
$\llbracket \text{let } F(\tilde{\mathbf{x}}) = P \text{ in } [Q] \rrbracket_\rho^{\phi, \delta} = \llbracket Q \rrbracket_{\rho[\mathfrak{D}/F]}^{\phi, \delta}$ where $\Gamma(F) = \lambda \tilde{\mathbf{x}}. \mathcal{J}$ and $\mathfrak{D} = \text{fix}(\lambda \mathfrak{C} : \llbracket \mathcal{J} \rrbracket^\phi . \llbracket P \rrbracket_{\rho[\mathfrak{C}/F]}^{\phi, \delta})$	

The interpretation deserves some explanation. Interpretation of termination, input, asynchronous output and restriction of parallel composition (first, second, fourth and seventh row of the table) are given by the straightforward adaptation of the finitary case. Interpretation of the branching input construct $\sum_{i=1}^n \mathbf{a} \text{in}_i(\tilde{\mathbf{x}}_i).P_i$ (third row of the table) is given by the superposition of different multidesigns $(\xi, J_i). \llbracket P_i \rrbracket_\rho^{\phi[\xi/J_i/\tilde{\mathbf{x}}_i], \delta}$ corresponding to the different branches of the choice. Notice that all the actions (ξ, J_i) have negative polarity by construction. Interpretation of the asynchronous selective output construct $\bar{\mathbf{a}} \text{in}_i(\tilde{\mathbf{x}})$ is given by prefixing the action (ξ, J) given from $\delta(\phi(\mathbf{a}), |\tilde{\mathbf{x}}|, i)$ by the selection of the discriminator i , to the multidesign which performs the delocation from each locus $\phi(\mathbf{a}) \cdot j_k$ ($j_k \in J$) to the locus $\phi(\mathbf{x}_k)$. Notice that the polarity of (ξ, J) is positive, by construction. This allow us to interpret correctly the selection plus renaming induced by the contraction of the redex $\sum_{i=1}^n \mathbf{a} \text{in}_i(\tilde{\mathbf{x}}_i).P_i \llbracket \bar{\mathbf{a}} \text{in}_j(\tilde{\mathbf{y}}) \rrbracket$ when $j \in [1, n]$. The interpretation of a definition invocation $F(\tilde{\mathbf{y}})$ is obtained by delocating the multidesign $\rho(F)$ from the loci corresponding to the formal parameters $\tilde{\mathbf{x}}$ to the loci corresponding to the actual parameters $\tilde{\mathbf{y}}$. Finally the interpretation of $\text{let } F(\tilde{\mathbf{x}}) = P \text{ in } [Q]$ is the interpretation of Q where we associate to F the multidesign obtained as least fix point of the function $\lambda \mathfrak{C} : \llbracket \mathcal{J} \rrbracket^\phi . \llbracket P \rrbracket_{\rho[\mathfrak{C}/F]}^{\phi, \delta}$. This function takes in input a multidesign \mathfrak{C} on interface $\llbracket \mathcal{J} \rrbracket^\phi$ (where $\Gamma(F) = \lambda \tilde{\mathbf{x}}. \mathcal{J}$) and gives back the multidesign obtained from the interpretation of P where the design \mathfrak{C} is associated to F . The least fix point of this function exists since all the operators over multidesigns we used are continuous.

Proposition 5.6.1 (Soundness). *Let P be a process such that $\Gamma \vdash P \triangleright \mathcal{I}$ with \mathcal{I} neutral free. Then*

1. $\llbracket P \rrbracket_\rho^{\phi, \delta}$ is a multidesign on base $\llbracket \mathcal{I} \rrbracket^\phi$
2. if $P \longrightarrow Q$ then $\llbracket P \rrbracket = \llbracket Q \rrbracket$

Proof. (1.) can be proved by induction on the derivation of $\Gamma \vdash P \triangleright \mathcal{I}$. To prove (2.), the case $P \longrightarrow Q$ because $P = \sum_{i=1}^n \mathbf{a} \text{in}_i(\tilde{\mathbf{x}}_i).P_i \llbracket \bar{\mathbf{a}} \text{in}_j(\tilde{\mathbf{y}}) \rrbracket$ and $Q = P_j[\tilde{\mathbf{y}}/\tilde{\mathbf{x}}_i]$ follows since Lemmas 5.2.3 and 5.4.1, using a similar argument developed for the proof of Theorem 5.4.2. The case $P = \text{let } F(\tilde{\mathbf{x}}) = P_1 \text{ in } [F(\tilde{\mathbf{y}})] \llbracket P_2 \rrbracket$ and $Q = \text{let } F(\tilde{\mathbf{x}}) = P_1 \text{ in } [P_1[\tilde{\mathbf{y}}/\tilde{\mathbf{x}}]] \llbracket P_2 \rrbracket$ where $\Gamma(F) = \lambda \tilde{\mathbf{x}}. \mathcal{I}$ and $\Psi = \lambda \mathfrak{C} : \llbracket \mathcal{I} \rrbracket^\phi . \llbracket P_1 \rrbracket_{\rho[\mathfrak{C}/F]}^{\phi, \delta}$ and θ is a delocation from $\phi(\mathbf{x}_i)$ to

$\phi(y_i)$

$$\begin{aligned}
\llbracket P \rrbracket_\rho^{\phi, \delta} &= \theta(\text{fix } \Psi) \circ \llbracket P_2 \rrbracket_{\rho[\text{fix } \Psi / F]}^{\phi, \delta} \\
&= \theta(\Psi(\text{fix } \Psi)) \circ \llbracket P_2 \rrbracket_{\rho[\text{fix } \Psi / F]}^{\phi, \delta} \\
&= \theta(\llbracket P_1 \rrbracket_{\rho[\text{fix } \Psi / F]}^{\phi, \delta}) \circ \llbracket P_2 \rrbracket_{\rho[\text{fix } \Psi / F]}^{\phi, \delta} \\
&= \llbracket Q \rrbracket_\rho^{\phi, \delta}
\end{aligned}$$

□

Proposition 5.6.2. *Let $\Psi = \lambda \mathfrak{C} : \llbracket \mathcal{J} \rrbracket^\phi \cdot \llbracket P \rrbracket_{\rho[\mathfrak{C}/F]}^{\phi, \delta}$ be the function obtained from the interpretation $\llbracket \text{let } F(\bar{x}) = P \text{ in } [Q] \rrbracket_\rho^{\phi, \delta}$. Then its least fix point \mathfrak{D} is given by:*

$$\mathfrak{D} = \begin{cases} \bigcup_n \Psi^n(\mathfrak{F}\text{id}) & \text{if } \forall \mathbf{x} \in \text{FN}(\mathcal{J}). \mathcal{J}(\mathbf{x}) = - \\ \bigcup_n \Psi^n(\mathfrak{C}\text{funf}) & \text{otherwise} \end{cases}$$

Proof. It follows from the fact that all the operations used to define the semantics of a process are continuous. □

5.7 Conclusion

Summing up, we show that an extension of Ludics provides a fully abstract model for the finitary linear π -calculus. Moreover, we show that the model of Ludics is expressive enough to allow a sound interpretation of a Turing complete calculus. In particular, we prove that the simplest extension of the model for a Turing complete calculus is again sound with respect to equational theory induced by reduction rules.

We ask ourselves whether it is possible to prove that the adequacy and full abstraction of the considered model with respect to the testing equivalence, we defined in Definition 5.3.4. The answer to this question is negative. Observe in fact that the notion of observable considered by the testing equivalence is not interpreted into a finite multidesign. This invalidates all known techniques to prove that the recursive definition enjoys the computability predicate through the notion of approximant.

It is also not possible to obtain adequacy, simply by modifying the notion of observational equivalence by taking only processes under the form of $\sqrt{}$ as observables. Consider in fact the process $P = \sqrt{\text{a}} \parallel (\text{a})(\text{b})(\text{a}.\bar{\text{b}} \parallel \text{b}.\bar{\text{a}})$; it is easy to see that $\llbracket P \rrbracket = \llbracket \sqrt{} \rrbracket$ but it is not the case that $P \Rightarrow \sqrt{}$.

Moreover, even with adequacy, full abstraction cannot be obtained with a definability argument, since separation does not hold for the multidesigns which are interpretation of term of linear π -calculus, since they are not finite (so Theorem 5.2.6 does not apply).

As future work, we propose to extend the model we defined with *behaviour*, which are the semantic counterpart of types. Our aim is to prove soundness and completeness for the Multiplicative Additive fragment of Linear Logic with Mix. Then we would like to use this result in order to impose a typing discipline to our calculus. The goal is to prove soundness and completeness of the type assignment with respect to our model.

Part II

Behind Linearity using Intersection Types

Intersection types were introduced in [Coppo and Dezani-Ciancaglini, 1980] to increase the typability power of Curry's type discipline. Intersection types allow us to describe various properties of λ -terms, in particular they provide a complete characterisation of the strong normalisation property in λ -calculus. Moreover, intersection type assignment systems can be viewed as finitary logical descriptions of the denotational interpretation of λ -terms, giving rise to the notion of filter-models. Namely, a typing judgement can be interpreted as saying that a finite element of a model (described by a type) belongs to the interpretation of a given term.

In this part of the thesis, I will study non-linear languages and I will use intersection types in order to study normalisation properties of these language as well as to reason in a finitary way over the denotational interpretation of untyped languages. Namely this part of the thesis will consist in one chapter titled **Logical Semantics For Stability**. There, I will use intersection types to reason in a finitary way on stable models of λ -calculus based on Coherence Spaces.

6 Logical Semantics for Stability

ABSTRACT.

Type assignment systems for λ -calculus based on intersection types are a general framework for building models of λ -calculus (known as filter-models) which are useful tools for reasoning in a finitary way about the denotational interpretation of terms. Indeed the denotation of a term is the set of types derivable for it and a type is a “finite piece” of information on such a denotation. This approach to the λ -calculus semantics is called in the literature *logical semantics*, and it has been intensively studied in relation with λ -models in the Scott’s domain setting. In this chapter we define two intersection type assignment systems for λ -calculus, parametric with respect to a coherence relation between types. We prove that, when the instantiation of the parameter satisfies a given condition, our two type systems induce models of λ -calculus, that we call *clique-models*. Lastly we show that such systems give a logical characterisation of two classes of models built on the category of Girard’s coherence spaces and stable functions.

Introduction

In the general framework of denotational semantics for programming languages, logical semantics is a tool for building models of various kinds of λ -calculus, and for reasoning in finitary way about the interpretation of terms. Logical semantics is based on intersection type assignment systems [Coppo and Dezani-Ciancaglini, 1980]. Namely, an intersection type assignment system assigns types to terms of λ -calculus, and typing rules are closed under a pre-order relation. If the pre-order satisfies some constraints then the system gives rise to a λ -model, where the interpretation of a term is the set of types derivable for it. Such constraints are designed for assuring that the interpretation of a term enjoys the good properties we expect, like the closure under beta-equality, the closure under contexts, and so on. In all the known instances, the constraints can be expressed by few, very easy, syntactical rules.

Logical semantics has been intensively studied in connections with denotational model based on Scott's domains which are ω -algebraic complete lattices (Scott-models). Indeed, the constraints can be formalised in such a way that the induced model is isomorphic to a Scott-model. The models built in this way are called *filter λ -models*. The isomorphism between filter λ -models and Scott-models can be view as a particular instance of domain theories in logical form [Abramsky, 1991]. The isomorphism relates types with compact elements of the complete lattice on which the Scott-model is based, in particular arrow-types correspond to Scott-continuous step-functions. The pre-order between types reflects the partial order of the domain. The construction is made by using the intersection connective between types for mimicking the join operation in domains. Moreover, the type assignment system provides a logical description of the interpretation function, i.e. to assign a type (say σ) to a term M is the logical counterpart of the fact that the compact element corresponding to σ is less than or equal to the interpretation of M . Examples of filter models designed in order to study particular properties of λ -calculus are in [Barendregt et al., 1983, Coppo et al., 1987, Dezani-Ciancaglini et al., 2004, Ronchi Della Rocca and Paolini, 2004]. This approach has been applied also to the call-by-value λ -calculus in [Egidi et al., 1992]. For the study of the isomorphism between filter λ -models and Scott-models the reader can see [Coppo et al., 1984, Honsell and Ronchi Della Rocca, 1992, Plotkin, 1993], behind other. In [Ronchi Della Rocca and Paolini, 2004] a notion of *parametric filter model* has been defined, which can generate models of various kinds of λ -calculus, when the parameter has been specified in a suitable way (particular choices generate the classical and the call-by-value λ -calculus).

In this chapter we want to extend the above approach to the λ -models based on coherence spaces defined by Girard [Girard, 1986] (originally they was named binary qualitative domains, their renaming in coherence spaces has been given in [Girard, 1987]). Coherence spaces are based on Berry's stable functions [Berry, 1978]. We call such models *stable λ -models*. We consider two particular classes of stable λ -models, the linear and the lazy one. In order to describe such classes, let us recall that a stable λ -model is based on a coherence space X containing as retract the space $X \Rightarrow X$, where \Rightarrow denotes the stable functions constructor. A stable model is linear if both the functions realising the retraction are linear (and so they are strict, since they map the empty set into itself). Lazy stable models are particular cases of stable models where one of the functions realising the retraction is not strict. This difference in the space on which the models are based is reflected in the λ -theories that they induce. In fact, the λ -theory induced by a linear λ -model is always such that, if a term M is interpreted as the empty set, then also the interpretation of $\lambda x.M$ is the empty set. So the theory cannot be lazy, in the sense of [Abramsky and Ong, 1993]. While lazy stable models can generate lazy λ -theories. We want to stress the fact that these two classes are not too restricted: indeed, in our knowledge, all stable λ -models considered in literature belong to one of them [Bastonero et al., 1998, Girard, 1986, Honsell and Ronchi della Rocca, 1990].

We define two type assignment systems, parametric with respect to two relations between types, an equivalence and a strict coherence relation. We provide a *legality*

condition, and we prove that every choice of such relations satisfying this condition gives rise to a λ -model. We call respectively *clique models* and *lazy clique models* the λ -models obtained by the two type assignments. Then, we prove that the class of clique models is isomorphic to the class of linear stable models, while the class of lazy clique models is isomorphic to that one of lazy stable models. Such isomorphisms are based on the fact that types can be put in correspondence with tokens, in such a way that the equivalence and strict coherence relation between types reflect respectively the equality and the strict coherence relation in the space on which the (lazy) stable model is based, when the legality condition is satisfied. Lastly, the type assignment system is a logical description of the interpretation function. In fact, if a type σ is derivable for a term M , then it turns out that the token corresponding to the equivalence class of σ belongs to the clique which is the interpretation of M in the isomorphic stable model.

The main difference between our type assignments and the classical intersection type assignments reflects the difference between continuous and stable functions. Classical intersection types represent compact elements of complete lattices, where all elements are consistent. In fact the join is a total operation on Scott domains, and so the intersection is a total constructor on types. In coherence spaces the join is a partial operation, defined only between coherent elements, so we need to introduce in types a notion reflecting this fact, which is the strict coherence relation. Moreover, we choose to not use explicitly the intersection constructor on types, but our types are of the shape $[\sigma_1, \dots, \sigma_n] \rightarrow \sigma$ where the left-hand side of the arrow contains types which are pairwise strictly coherent (morally, in intersection). The fundamental difference between clique models and lazy clique models is the introduction of a particular type constant ν denoting the least functional behaviour. So, in order to type an application, we ask that the term in functional position can be assigned both the type ν and the type describing its correct functionality.

We want to stress the fact that the definition of (lazy)-clique models is given in a completely syntactical way, so they can be built without any acquaintance with stable functions and stable models.

In the literature some works has been done to connect intersection type assignment systems and stable λ -models, namely [Bastonero et al., 1998, Honsell and Ronchi della Rocca, 1990]. In Section 6.4 we will discuss the relation between the results of the present chapter and the previous ones.

The content of this section can be found in the article [Paolini et al., 2009], written with Luca Paolini and Simona Ronchi Della Rocca.

6.1 Models of pure untyped λ -calculus

In the next definition, we will give the properties that a structure must satisfy in order to be used as denotation space for λ -calculus, or, equivalently, to be a *model* for λ -calculus [Barendregt, 1984, Hindley and Longo, 1980, Koymans, 1982, Meyer, 1982]. In particular, we use the definition given in [Ronchi Della Rocca and Paolini, 2004]

which is a light equivalent variant of the one in [Hindley and Longo, 1980].

Definition 6.1.1. A λ -model is a triple $\mathcal{M} = \langle \mathbb{D}, \circ, \llbracket \cdot \rrbracket \rangle$, such that \mathbb{D} is a set and \circ is a map from \mathbb{D}^2 to \mathbb{D} . Moreover, if \mathbb{E} is the collection of functions (environments) from Var to \mathbb{D} , ranged over by ρ, ρ' , then the interpretation function $\llbracket \cdot \rrbracket : \Lambda \times \mathbb{E} \rightarrow \mathbb{D}$ satisfies the following conditions:

1. $\llbracket \mathbf{x} \rrbracket_\rho = \rho(\mathbf{x})$,
2. $\llbracket \mathbf{MN} \rrbracket_\rho = \llbracket \mathbf{M} \rrbracket_\rho \circ \llbracket \mathbf{N} \rrbracket_\rho$,
3. $\llbracket \lambda \mathbf{x}.\mathbf{M} \rrbracket_\rho \circ d = \llbracket \mathbf{M} \rrbracket_{\rho[d/\mathbf{x}]}$,
4. if $\llbracket \mathbf{M} \rrbracket_{\rho[d/\mathbf{x}]} = \llbracket \mathbf{M}' \rrbracket_{\rho'[d/\mathbf{y}]}$ for each $d \in \mathbb{D}$, then $\llbracket \lambda \mathbf{x}.\mathbf{M} \rrbracket_\rho = \llbracket \lambda \mathbf{y}.\mathbf{M}' \rrbracket_{\rho'}$,

where $\rho[d/\mathbf{x}](\mathbf{y}) =$ if $\mathbf{y} \equiv \mathbf{x}$ then d else $\rho(\mathbf{y})$.

This definition ensures that a λ -model respects some elementary key properties, namely the interpretation of a term depends only on the behaviour of the environment on the free variables of the term itself, the α -rule is respected, the syntactical substitution is modelled by the environment and the interpretation is contextually closed.

Proposition 6.1.1. Let $\langle \mathbb{D}, \circ, \llbracket \cdot \rrbracket \rangle$ be a λ -model.

1. If $\rho(\mathbf{x}) = \rho'(\mathbf{x})$, for all $\mathbf{x} \in \text{FV}(\mathbf{M})$, then $\llbracket \mathbf{M} \rrbracket_\rho = \llbracket \mathbf{M} \rrbracket_{\rho'}$;
2. If $\mathbf{y} \notin \text{FV}(\mathbf{M})$ then $\llbracket \mathbf{M} \rrbracket_{\rho[d/\mathbf{x}]} = \llbracket \mathbf{M}[\mathbf{y}/\mathbf{x}] \rrbracket_{\rho[d/\mathbf{y}]}$;
3. If $\mathbf{y} \notin \text{FV}(\mathbf{M})$ then $\llbracket \lambda \mathbf{x}.\mathbf{M} \rrbracket_\rho = \llbracket \lambda \mathbf{y}.\mathbf{M}[\mathbf{y}/\mathbf{x}] \rrbracket_\rho$;
4. $\llbracket \mathbf{M}[\mathbf{N}/\mathbf{x}] \rrbracket_\rho = \llbracket \mathbf{M} \rrbracket_{\rho[\llbracket \mathbf{N} \rrbracket_\rho/\mathbf{x}]}$;
5. If $\llbracket \mathbf{M} \rrbracket_\rho = \llbracket \mathbf{N} \rrbracket_\rho$ then, for every context $C[\cdot]$, $\llbracket C[\mathbf{M}] \rrbracket_\rho = \llbracket C[\mathbf{N}] \rrbracket_\rho$.

As consequence of the previous proposition, condition (iii) of Definition 6.1.1 is the semantics counterpart of the β -reduction rule, so the interpretation of a term is closed under $=_\beta$.

Corollary 6.1.1. Let $\langle \mathbb{D}, \circ, \llbracket \cdot \rrbracket \rangle$ be a λ -model. If $\mathbf{M} =_\beta \mathbf{N}$ then $\llbracket \mathbf{M} \rrbracket_\rho = \llbracket \mathbf{N} \rrbracket_\rho$, for all ρ .

Given a λ -model \mathcal{M} , the interpretation function $\llbracket \cdot \rrbracket^{\mathcal{M}}$ induces a denotational semantics on Λ . Namely, two terms \mathbf{M} and \mathbf{N} are denotationally equivalent in \mathcal{M} (and we write $\mathbf{M} \sim_{\mathcal{M}} \mathbf{N}$) if and only if:

$$\llbracket \mathbf{M} \rrbracket_\rho^{\mathcal{M}} = \llbracket \mathbf{N} \rrbracket_\rho^{\mathcal{M}}, \text{ for all environments } \rho.$$

Corollary 6.1.1 ensure us that $\sim_{\mathcal{M}}$ is a λ -theory, i.e., a congruence relation on terms closed under $=_\beta$.

6.2 Clique Models

In this section we will define two classes of type assignment systems, and we will prove that both give rise to λ -models, under particular conditions.

In order to present the systems in a clean way, we first define a super-set of types (row types) which enable us to formalise *some relations on them* needed to define well-formed types.

Definition 6.2.1. 1. Let C be a non-empty countable set of type constants, ranged over by p, q, r . The set $\text{Row}(C)$ of row types, ranged over by σ, τ, π, μ , is defined as follows:

$$\sigma ::= p \mid [\sigma_1, \dots, \sigma_n] \rightarrow \sigma \quad (n \geq 0)$$

where p belongs to C . The identity relation on $\text{Row}(C)$ is denoted by $=$.

2. A stable type theory on $\text{Row}(C)$ is a congruence \simeq (i.e. a reflexive, symmetric, transitive and contextual equivalence) on row types, satisfying

$$\frac{\forall i \exists j \sigma_i \simeq \tau_j \quad \forall i \exists j \tau_i \simeq \sigma_j \quad \sigma \simeq \tau \quad n, m \geq 0}{[\sigma_1, \dots, \sigma_n] \rightarrow \sigma \simeq [\tau_1, \dots, \tau_m] \rightarrow \tau}$$

We denote by $\text{Row}(C)/\simeq$ the set of equivalence classes induced by \simeq on $\text{Row}(C)$. By abuse of notation, a row type will denote also its class membership.

3. Let C be a set of type constants and \simeq a stable type theory on $\text{Row}(C)$. A typing relation is a binary endo-relation on $\text{Row}(C)/\simeq$. We define four typing relations, denoted by $\frown, \smile, \circ, \asymp$. \frown (strict coherence) is symmetric and irreflexive (equivalent types are not put in relation), and it is defined as follows:

$$\frac{\sigma \frown \tau}{[\sigma_1, \dots, \sigma_n] \rightarrow \sigma \frown [\tau_1, \dots, \tau_m] \rightarrow \tau} \quad (a) \quad \frac{\exists i \leq n, \exists j \leq m \text{ such that } \sigma_i \smile \tau_j}{[\sigma_1, \dots, \sigma_n] \rightarrow \sigma \frown [\tau_1, \dots, \tau_m] \rightarrow \tau} \quad (b)$$

$\sigma_i \smile \tau_j$ (strict incoherence) is defined as $\sigma_i \neq \tau_j$ and $\sigma_i \not\smile \tau_j$. \circ (coherence) is the reflexive closure (all equivalent types are put in relation) of \frown , finally \asymp (incoherence) is the reflexive closure of \smile .

If \textcircled{R} is either a typing relation or \simeq , then we write $\textcircled{R}\{\sigma_1, \dots, \sigma_n\}$ to denote that $\sigma_i \textcircled{R} \tau_j$, for $i \neq j$ ($1 \leq i, j \leq n$).

The previous definition deserves some comments.

We define row types only in order to provide a coarse syntax on which to define useful tools, as type theory and typing relations, that will be used in order to define types. A non constant row types has always the shape $[\sigma_1, \dots, \sigma_n] \rightarrow \sigma$, where $[\sigma_1, \dots, \sigma_n]$ is a sequence of row types, that can be empty and can contain repetitions. For instance, both $[\] \rightarrow \sigma$ and $[\sigma, \tau, \sigma] \rightarrow \sigma$ are row types. A stable type theory imposes an equivalence on row types, in particular sequences on the left-side of an arrow are managed up to set-theoretical equivalence. So, for instance $\tau' \simeq \tau$ implies that

$[\sigma, \tau, \sigma] \rightarrow \sigma \simeq [\tau', \sigma] \rightarrow \sigma$. We could have done a different choice, denoting directly the left hand of an arrow as a set, but we chose to use sequences (denoted by square brackets) in place of sets (usually denoted by curly bracket) in order to emphasise the difference between the syntax of row types and the metalanguage.

Strict coherence will be used in type formation, since it formalises what bits of type-information are consistent (in classical sense of domain theory¹). We remark that strict coherence is given up to equivalence between types, thus if $\sigma \simeq \tau$ and $\tau \wedge \pi$ then $\sigma \wedge \pi$. Let us observe that, in Definition 6.2.1.(iii), rules (a) and (b) together require that, given two not constant row types, either their right hand are strictly consistent or they have at least two inconsistent elements in the left hand side. So, for instance $[] \rightarrow \sigma$ is incoherent with all not constant row types having σ in the right hand side.

Definition 6.2.2. 1. A stable type system ∇ is a triple $\langle C_\nabla, \simeq_\nabla, \wedge_\nabla \rangle$, where C_∇ is a set of type constants, \simeq_∇ is a stable type theory on $\text{Row}(C_\nabla)$ and \wedge_∇ is a strict coherence relation on $\text{Row}(C_\nabla)/\simeq_\nabla$.

2. A lazy stable type system ∇ is a quadruple $\langle C_\nabla, \simeq_\nabla, \wedge_\nabla, \nu_\nabla \rangle$ which extends the stable type system $\langle C_\nabla, \simeq_\nabla, \wedge_\nabla \rangle$ by selecting a special type constant $\nu_\nabla \in C_\nabla$ such that $\nu_\nabla \wedge_\nabla \sigma$, for all $\sigma \in \text{Row}(C_\nabla) \setminus \{\nu_\nabla\}$. We will call ν_∇ the pivot of ∇ .

It should be clear that a stable type system can contain zero, one or many type constants that can be used as pivot. Hence, a stable type system can induce zero, one or many lazy type systems.

Given a stable type system (possibly lazy), the definition of row types can be refined in order to obtain *types*. Hence, a type assignment system can be designed by formalising rules assigning such types to terms of λ -calculus.

Definition 6.2.3. Let ∇ be a stable type system (possibly lazy).

1. The set of types T_∇ is the subset of $\text{Row}(C_\nabla)$ such that

$$\frac{p \in C_\nabla}{p \in T_\nabla} \quad \frac{\tau, \tau_1, \dots, \tau_n \in T_\nabla \quad \wedge_\nabla\{\tau_1, \dots, \tau_n\}}{[\tau_1, \dots, \tau_n] \rightarrow \tau \in T_\nabla}$$

2. A type assignment is a pair of the shape $\mathbf{x} : \sigma$, where \mathbf{x} is a variable and $\sigma \in T_\nabla$. A ∇ -basis B is a finite set of type assignments such that, if $\mathbf{x} : \sigma_1, \dots, \mathbf{x} : \sigma_n \in B$ then $\wedge_\nabla \{\sigma_1, \dots, \sigma_n\}$. Let $\text{dom}(B) = \{\mathbf{x} \mid \exists \sigma. \mathbf{x} : \sigma \in B\}$.

If $B = B' \cup \{\mathbf{x} : \sigma_1, \dots, \mathbf{x} : \sigma_n\}$ and $\mathbf{x} \notin \text{dom}(B')$ then $B(\mathbf{x}) = \{\sigma_1, \dots, \sigma_n\}$. Let B_i be a basis for $1 \leq i \leq n$; we write $\star(B_1, B_2, \dots, B_n)$ meaning that if $\mathbf{x} : \sigma \in B_i$ and $\mathbf{x} : \tau \in B_j$, for some $i \neq j$ ($1 \leq i, j \leq n$) then either $\sigma \wedge_\nabla \tau$ or $\sigma = \tau$.

3. A stable intersection type assignment system \vdash_∇ is a formal system proving statements of the shape:

$$B \vdash_\nabla \mathbb{M} : \sigma$$

where \mathbb{M} is a term, $\sigma \in T_\nabla$ and B is a ∇ -basis.

$$\begin{array}{c}
 \frac{}{\mathbf{x} : \sigma \vdash_{\nabla} \mathbf{x} : \sigma} \text{ (var)} \qquad \frac{B \vdash_{\nabla} \mathbf{M} : \tau \quad \sigma \simeq_{\nabla} \tau}{B \vdash_{\nabla} \mathbf{M} : \sigma} (\simeq) \\
 \\
 \frac{B \vdash_{\nabla} \mathbf{M} : [\sigma_1, \dots, \sigma_n] \rightarrow \sigma \quad (B_i \vdash_{\nabla} \mathbf{N} : \sigma_i)_{1 \leq i \leq n} \quad \star(B, B_1, \dots, B_n)}{\left(\bigcup_{1 \leq i \leq n} B_i \right) \cup B \vdash_{\nabla} \mathbf{M} \mathbf{N} : \sigma} (\rightarrow E) \\
 \\
 \frac{B \cup \{\mathbf{x} : \sigma_1, \dots, \mathbf{x} : \sigma_n\} \vdash_{\nabla} \mathbf{M} : \tau \quad \mathbf{x} \notin \text{dom}(B)}{B \vdash_{\nabla} \lambda \mathbf{x}. \mathbf{M} : [\sigma_1, \dots, \sigma_n] \rightarrow \tau} (\rightarrow I)
 \end{array}$$

 Figure 6.1: The Type Assignment Systems \vdash_{∇} .

$$\begin{array}{c}
 \frac{}{\mathbf{x} : \sigma \vdash_{\nabla}^{\ell} \mathbf{x} : \sigma} \text{ (var)} \qquad \frac{B \vdash_{\nabla}^{\ell} \mathbf{M} : \tau \quad \sigma \simeq_{\nabla} \tau}{B \vdash_{\nabla}^{\ell} \mathbf{M} : \sigma} (\simeq) \\
 \\
 \frac{B' \vdash_{\nabla}^{\ell} \mathbf{M} : \nu_{\nabla} \quad B \vdash_{\nabla}^{\ell} \mathbf{M} : [\sigma_1, \dots, \sigma_n] \rightarrow \sigma \quad (B_i \vdash_{\nabla}^{\ell} \mathbf{N} : \sigma_i)_{1 \leq i \leq n} \quad \star(B, B', B_1, \dots, B_n)}{\left(\bigcup_{1 \leq i \leq n} B_i \right) \cup B \cup B' \vdash_{\nabla}^{\ell} \mathbf{M} \mathbf{N} : \sigma} (\rightarrow E) \\
 \\
 \frac{B \cup \{\mathbf{x} : \sigma_1, \dots, \mathbf{x} : \sigma_n\} \vdash_{\nabla}^{\ell} \mathbf{M} : \tau \quad \mathbf{x} \notin \text{dom}(B)}{B \vdash_{\nabla}^{\ell} \lambda \mathbf{x}. \mathbf{M} : [\sigma_1, \dots, \sigma_n] \rightarrow \tau} (\rightarrow I) \qquad \frac{}{\emptyset \vdash_{\nabla}^{\ell} \lambda \mathbf{x}. \mathbf{M} : \nu_{\nabla}} \text{ (lazy)}
 \end{array}$$

 Figure 6.2: The Type Assignment Systems \vdash_{∇}^{ℓ} .

The rules of the system are in Figure 6.1.

4. Let ∇ be lazy and let ν_{∇} be its pivot. A lazy stable intersection type assignment system \vdash_{∇}^{ℓ} is a formal system proving statements of the shape:

$$B \vdash_{\nabla}^{\ell} \mathbf{M} : \sigma$$

where \mathbf{M} is a term, $\sigma \in \mathcal{T}_{\nabla}$ and B is a ∇ -basis.

The rules of the system are in Figure 6.2.

Differently from the syntax of row types, square brackets on the left-side of an arrow contain a sequence of types without multiple occurrences of equivalent elements. Note, for instance, that if $\sigma \simeq_{\nabla} \sigma'$ then $[\sigma, \sigma'] \rightarrow \tau \notin \mathcal{T}_{\nabla}$.

Note that a basis cannot assign equivalent types to the same variable. If B_i are bases, then it is easy to check that $\star(B_1, B_2, \dots, B_n)$ implies that $\bigcup_{1 \leq i \leq n} B_i$ is a well-

¹ Remember that for classical filter model a consistence relation is not necessary, since they live in the world of complete lattices [Ronchi Della Rocca and Paolini, 2004] where all elements are consistent.

defined basis. In particular, we note that this would be no longer true replacing $=$ by \simeq_{∇} in the definition of \star .

Both the type assignment systems are linear, in the sense that weakening does not hold. As we will see in a formal way, a basis contains the minimal information necessary for the typing.

In rule $(\rightarrow E)$ types in all bases need to be strictly coherent to each other, and this is essential for preserving the correct syntax of types. The lazy system \vdash_{∇}^{ℓ} uses all the rules of \vdash_{∇} , but a modified $(\rightarrow E)$ rule, and a new rule *(lazy)*, the latter assigning ν_{∇} to all abstractions and the former asking that only terms for which the type ν_{∇} is derivable can be used in functional position. So ν_{∇} characterises terms with a functional behaviour.

Notation 6.2.1. Let ∇ be a stable type system and let \mathbb{R} be a typing relation. $\{\sigma_1, \dots, \sigma_n\} \mathbb{R} \{\tau_1, \dots, \tau_m\}$ is an abbreviation for $\forall i \exists j \sigma_i \mathbb{R} \tau_j, \forall i \exists j \tau_i \mathbb{R} \sigma_j$. Moreover $B_1 \mathbb{R} B_2$ shortens $B_1(\mathbf{x}) \mathbb{R} B_2(\mathbf{x})$, for all \mathbf{x} .

Lastly, we use \vdash_{∇}^* to denote both type assignment systems.

Note that $B_1 \simeq_{\nabla} B_2$ implies that the cardinality of the set $B_1(\mathbf{x})$ is the same as the set $B_2(\mathbf{x})$, for all variables \mathbf{x} .

The legality condition given in the next definition characterises (lazy) stable type systems that give rise to models of λ -calculus.

Definition 6.2.4. A (lazy) stable type system ∇ is legal whenever, if $[\sigma_1, \dots, \sigma_n] \rightarrow \sigma \in T_{\nabla}$ and $[\tau_1, \dots, \tau_m] \rightarrow \tau \in T_{\nabla}$ then both:

1. $[\sigma_1, \dots, \sigma_n] \rightarrow \sigma \simeq_{\nabla} [\tau_1, \dots, \tau_m] \rightarrow \tau$ implies both $\sigma \simeq \tau$ and $\{\sigma_1, \dots, \sigma_n\} \simeq_{\nabla} \{\tau_1, \dots, \tau_m\}$,
2. $[\sigma_1, \dots, \sigma_n] \rightarrow \sigma \wedge_{\nabla} [\tau_1, \dots, \tau_m] \rightarrow \tau$ implies either $\sigma \wedge_{\nabla} \tau$ or $\exists i. 1 \leq i \leq n, \exists j. 1 \leq j \leq m$ such that $\sigma_i \vee_{\nabla} \tau_j$.

We remark that point (i) and (ii) of legality correspond to reversibility of the rules of Definition 6.2.1.(ii) and (iii), respectively. In particular, if ∇ is legal, then both $[\tau_1, \dots, \tau_n] \rightarrow \tau, [\sigma_1, \dots, \sigma_m] \rightarrow \sigma \in T_{\nabla}$ and $[\tau_1, \dots, \tau_n] \rightarrow \tau \simeq_{\nabla} [\sigma_1, \dots, \sigma_m] \rightarrow \sigma$ imply $n = m$.

Lemma 6.2.1 (Generation). Let ∇ be a legal stable type system.

1. $B \vdash_{\nabla}^* \mathbf{x} : \sigma$ implies $B = \{\mathbf{x} : \tau\}$ and $\sigma \simeq_{\nabla} \tau$.
2. $B \vdash_{\nabla}^* \mathbb{M} : \sigma$ implies $\text{dom}(B) \subseteq FV(\mathbb{M})$.
3.
 - $B \vdash_{\nabla} \lambda \mathbf{x}. \mathbb{M} : \sigma$ implies $\sigma \simeq [\sigma_1, \dots, \sigma_n] \rightarrow \tau$;
 - $B \vdash_{\nabla}^{\ell} \lambda \mathbf{x}. \mathbb{M} : \sigma$ implies either $\sigma = \nu_{\nabla}$ or $\sigma \simeq [\sigma_1, \dots, \sigma_n] \rightarrow \tau$.
4.
 - $B \vdash_{\nabla} \mathbb{M} \mathbb{N} : \sigma$ implies $B' \vdash_{\nabla} \mathbb{M} : [\sigma_1, \dots, \sigma_n] \rightarrow \sigma, B_i \vdash_{\nabla} \mathbb{N} : \sigma_i$, where $B = (\bigcup_{1 \leq i \leq n} B_i) \cup B'$;
 - $B \vdash_{\nabla}^{\ell} \mathbb{M} \mathbb{N} : \sigma$ implies $B' \vdash_{\nabla}^{\ell} \mathbb{M} : [\sigma_1, \dots, \sigma_n] \rightarrow \sigma, B'' \vdash_{\nabla}^{\ell} \mathbb{M} : \nu_{\nabla}$ and $B_i \vdash_{\nabla}^{\ell} \mathbb{N} : \sigma_i$, where $B = (\bigcup_{1 \leq i \leq n} B_i) \cup B' \cup B''$.

5. $B \vdash_{\nabla}^* \mathbb{M} : \sigma$ and $B \simeq_{\nabla} B'$ imply $B' \vdash_{\nabla}^* \mathbb{M} : \sigma$.
6. $B \vdash_{\nabla}^* \lambda \mathbf{x}. \mathbb{M} : [\sigma_1, \dots, \sigma_n] \rightarrow \tau$ if and only if $B \cup \{\mathbf{x} : \sigma_1, \dots, \mathbf{x} : \sigma_n\} \vdash_{\nabla}^* \mathbb{M} : \tau$ and $\mathbf{x} \notin \text{dom}(B)$.

Proof. The first five points follow easily by induction on derivation.

Note that in point (ii), the inclusion can be strict either in case of an application MN , where \mathbb{M} has type $[\] \rightarrow \sigma$, or in case rule *(lazy)* has been applied.

(vi) Let us consider first the system \vdash_{∇} . If the last applied rule is $(\rightarrow I)$, then the proof is immediate. Otherwise, the derivation proving $B \vdash_{\nabla} \lambda \mathbf{x}. \mathbb{M} : [\sigma_1, \dots, \sigma_n] \rightarrow \tau$ ends in the following way:

$$\frac{\frac{B \cup \{\mathbf{x} : \pi_1, \dots, \mathbf{x} : \pi_n\} \vdash_{\nabla} \mathbb{M} : \mu}{B \vdash_{\nabla} \lambda \mathbf{x}. \mathbb{M} : [\pi_1, \dots, \pi_n] \rightarrow \mu} (\rightarrow I)}{B \vdash_{\nabla} \lambda \mathbf{x}. \mathbb{M} : [\sigma_1, \dots, \sigma_n] \rightarrow \tau} (\simeq)$$

where $[\pi_1, \dots, \pi_n] \rightarrow \mu \simeq_{\nabla} [\sigma_1, \dots, \sigma_n] \rightarrow \tau$, by transitivity of \simeq_{∇} . Moreover, since the system is legal, $\{\pi_1, \dots, \pi_n\} \simeq_{\nabla} \{\sigma_1, \dots, \sigma_n\}$ and $\mu \simeq_{\nabla} \tau$. Then the proof follows by point (v) and by rule (\simeq) .

In case of system \vdash_{∇}^{ℓ} , if the derivation proving $B \vdash_{\nabla} \lambda \mathbf{x}. \mathbb{M} : [\sigma_1, \dots, \sigma_n] \rightarrow \tau$ is as before, then the same proof applies. The derivation cannot end by an application of rule *(lazy)*, since by definition $\nu_{\nabla} \frown_{\nabla} \sigma$ (and thus $\nu_{\nabla} \neq_{\nabla} \sigma$), for all $\sigma \in \text{Row}(C_{\nabla}) \setminus \{\nu_{\nabla}\}$, so $\nu_{\nabla} \neq [\sigma_1, \dots, \sigma_n] \rightarrow \tau$. \square

We remark that Lemma 6.2.1 holds only under the legality hypotheses. The property stated by following theorem, namely that in a derivation the basis collects the minimal information on the premises necessary for the typing, is crucial for building a model starting from a type assignment system.

Theorem 6.2.1. *Let ∇ be a legal stable type system.*

1. If $B_0 \vdash_{\nabla}^* \mathbb{M} : \sigma_0$, $B_1 \vdash_{\nabla}^* \mathbb{M} : \sigma_1$ and $B_0 \supset_{\nabla} B_1$ then $\sigma_0 \supset_{\nabla} \sigma_1$.
2. If $B_0 \vdash_{\nabla}^* \mathbb{M} : \sigma_0$, $B_1 \vdash_{\nabla}^* \mathbb{M} : \sigma_1$, $\sigma_0 \simeq_{\nabla} \sigma_1$ and $B_0 \supset_{\nabla} B_1$ then $B_0 \simeq_{\nabla} B_1$.
3. If $B_0 \vdash_{\nabla}^* \mathbb{M} : \sigma$, $B_1 \vdash_{\nabla}^* \mathbb{M} : \sigma$ and $B_0 \subseteq B_1$ then $B_0 = B_1$.

Proof. The proof of first two points is given by mutual induction on \mathbb{M} .

1. If $\mathbb{M} \equiv \mathbf{x}$ then the proof follows by Lemma 6.2.1.(i).
If $\mathbb{M} \equiv PQ$ then the proof follows by Lemma 6.2.1.(iv) and by induction, taking into account the following obvious property:

$$\star(B_1, \dots, B_n) \text{ and } \{p_1, \dots, p_k\} \subseteq \{1, \dots, n\} \text{ imply } \star(B_{p_1}, \dots, B_{p_k}) \quad (6.1)$$

Let $\mathbb{M} \equiv \lambda \mathbf{x}. \mathbb{N}$. If $\sigma_0 = \nu_{\nabla}$ or $\sigma_1 = \nu_{\nabla}$ then the proof follows definition of lazy stable type system, namely $\nu_{\nabla} \frown_{\nabla} \sigma$, for all $\sigma \in \text{Row}(C_{\nabla}) \setminus \{\nu_{\nabla}\}$. Let us assume $\sigma_i \equiv [\tau_1^i, \dots, \tau_{n_i}^i] \rightarrow \tau^i$ where $n_i \geq 0$ ($0 \leq i \leq 1$). In case there are $\mu_0 \sim_{\nabla} \mu_1$ such that $\mu_i \in \{\tau_1^i, \dots, \tau_{n_i}^i\}$ then by rule (b) of Definition 6.2.1.(iii) the proof follows. Otherwise, assume that $B'_i = B_i \cup \{\mathbf{x} : \tau_1^i, \dots, \mathbf{x} : \tau_{n_i}^i\}$ and $B'_0 \supset_{\nabla} B'_1$. Since, by Lemma

- 6.2.1.(vi), $B'_i \vdash_{\nabla}^* M : \tau^i$ and $\mathbf{x} \notin \text{dom}(B_i)$, $\tau^0 \circ_{\nabla} \tau^1$ by induction. If $\tau^0 \wedge_{\nabla} \tau^1$ then the proof follows by rule (a) of Definition 6.2.1.(iii). The case $\tau^0 \simeq_{\nabla} \tau^1$ follows by mutual induction, since $B'_0 \simeq_{\nabla} B'_1$ implies $\{\mathbf{x} : \tau_1^0, \dots, \mathbf{x} : \tau_{n_0}^0\} \simeq_{\nabla} \{\mathbf{x} : \tau_1^1, \dots, \mathbf{x} : \tau_{n_0}^1\}$.
2. If $M \equiv \mathbf{x}$ then the proof follows by Lemma 6.2.1.(i). If $M \equiv PQ$ then $B'_i \vdash_{\nabla}^* P : [\tau_1^i, \dots, \tau_{n_i}^i] \rightarrow \sigma_i$, $B'_j \vdash_{\nabla}^* Q : \tau_j^i$ (and $B''^i \vdash_{\nabla}^* P : \nu_{\nabla}$ if the system is lazy) where $B_i = (\bigcup_{1 \leq j \leq n} B_j^i) \cup B'_i \cup B''^i$, by Lemma 6.2.1.(iv). But $[\tau_1^0, \dots, \tau_{n_0}^0] \rightarrow \sigma_0 \circ_{\nabla} [\tau_1^1, \dots, \tau_{n_1}^1] \rightarrow \sigma_1$ by mutual induction and property stated in equation (6.1). By Definition 6.2.4.(ii) (legality), $[\tau_1^0, \dots, \tau_{n_0}^0] \rightarrow \sigma_0 \wedge_{\nabla} [\tau_1^1, \dots, \tau_{n_1}^1] \rightarrow \sigma_1$ is not possible, since the hypothesis $\sigma_0 \simeq_{\nabla} \sigma_1$ and since $\tau_j^i \circ_{\nabla} \tau_{j'}^{i'}$ where $0 \leq i, i' \leq 1$, $1 \leq j \leq n_i$ and $1 \leq j' \leq n_{i'}$, by mutual induction. Hence, $[\tau_1^0, \dots, \tau_{n_0}^0] \rightarrow \sigma_0 \simeq_{\nabla} [\tau_1^1, \dots, \tau_{n_1}^1] \rightarrow \sigma_1$. Thus the proof follows by legality and induction.
- Let $M \equiv \lambda x.N$. If $\sigma_0 = \nu_{\nabla}$ or $\sigma_1 = \nu_{\nabla}$ then $\sigma_0 = \sigma_1$ by definition of lazy stable type system. Hence $B_0 = B_1 = \emptyset$, since there is a unique rule assigning to an abstraction a non-arrow type. Otherwise, let us assume $\sigma_i \equiv [\tau_1^i, \dots, \tau_{n_i}^i] \rightarrow \tau^i$ for $0 \leq i \leq 1$. Thus $B_i \cup \{\mathbf{x} : \tau_1^i, \dots, \mathbf{x} : \tau_{n_i}^i\} \vdash_{\nabla}^* M : \tau^i$ where $n_i \geq 0$ by Lemma 6.2.1.(vi). The legality implies that $\tau^0 \simeq_{\nabla} \tau^1$ and $\{\tau_1^0, \dots, \tau_{n_0}^0\} \simeq_{\nabla} \{\tau_1^1, \dots, \tau_{n_1}^1\}$, so the proof follows by induction.
3. Easy, in fact $B_0 \simeq_{\nabla} B_1$ by applying the point (ii) of this Theorem. Since $B_0 \subseteq B_1$ the proof is done. □

Now we will prove that a legal stable type system induces a λ -model. We call this kind of model *clique model* for its relation with coherence spaces, which will be proved in Section 5. Indeed, Theorem 6.2.1 formalises a crucial ingredient of such a correspondence.

Definition 6.2.5. *Let ∇ be a legal stable type system.*

1. A typing clique on ∇ is a set of types, pairwise coherent, closed under \simeq_{∇} . Typing cliques are ranged over by s, t . Let $\mathcal{S}(\nabla)$ be the set of typing cliques on ∇ .
2. \circ_{∇} is a binary operation defined on $\mathcal{S}(\nabla)$ in the following way:
 $s_1 \circ_{\nabla} s_2 = \{\tau \mid [\sigma_1, \dots, \sigma_n] \rightarrow \tau \in s_1 \text{ and } \sigma_i \in s_2 (1 \leq i \leq n)\}$.
3. Let ∇ be lazy. \circ_{∇}^{ℓ} is a binary operation defined on $\mathcal{S}(\nabla)$ in the following way: $s_1 \circ_{\nabla}^{\ell} s_2 = \{\tau \mid \nu_{\nabla} \in s_1, [\sigma_1, \dots, \sigma_n] \rightarrow \tau \in s_1 \text{ and } \sigma_i \in s_2 (1 \leq i \leq n)\}$

We use \circ_{∇}^* to denote both compositions. Typing cliques play in this chapter a role similar to filters in filter models (see [Coppo et al., 1984, Honsell and Ronchi Della Rocca, 1992]).

Lemma 6.2.2. *Let ∇ be a legal stable type system.*

If $s_1, s_2 \in \mathcal{S}(\nabla)$ then both $s_1 \circ_{\nabla} s_2 \in \mathcal{S}(\nabla)$ and $s_1 \circ_{\nabla}^{\ell} s_2 \in \mathcal{S}(\nabla)$.

Proof. We check that the elements of $s_1 \circ_{\nabla} s_2$ are pairwise coherent. Let $\tau_1, \tau_2 \in s_1 \circ_{\nabla} s_2$. Thus, there are $[\sigma_1^0, \dots, \sigma_{n_0}^0] \rightarrow \tau_1, [\sigma_1^1, \dots, \sigma_{n_1}^1] \rightarrow \tau_2 \in s_1, \sigma_i^0 \in s_2 (1 \leq i \leq n_0)$ and $\sigma_i^1 \in s_2 (1 \leq i \leq n_1)$. Since s_1 and s_2 are typing cliques (so their elements are pairwise coherent) $\tau_1 \circ_{\nabla} \tau_2$. The closure under equivalence is immediate.

To prove $s_1 \circ_{\nabla}^{\ell} s_2 \in \mathcal{S}(\nabla)$, just note that if $\nu_{\nabla} \notin s_1$ then $s_1 \circ_{\nabla}^{\ell} s_2 = \emptyset$. Otherwise the composition is as the previous one. \square

The interpretation function associates to a term all types that can be assigned to it. If ∇ is a legal stable type system and $\rho : \text{Var} \rightarrow \mathcal{S}(\nabla)$, then we define

$$\llbracket \mathbf{M} \rrbracket_{\rho}^{\nabla} = \{ \sigma \in \mathcal{T}_{\nabla} \mid \exists B. \forall \mathbf{x}. B(\mathbf{x}) \subseteq \rho(\mathbf{x}) \wedge B \vdash_{\nabla} \mathbf{M} : \sigma \}.$$

Moreover, if ∇ is lazy then

$$\llbracket \mathbf{M} \rrbracket_{\rho}^{\nabla \ell} = \{ \sigma \in \mathcal{T}_{\nabla} \mid \exists B. \forall \mathbf{x}. B(\mathbf{x}) \subseteq \rho(\mathbf{x}) \wedge B \vdash_{\nabla}^{\ell} \mathbf{M} : \sigma \}.$$

Observe that, by Theorem 6.2.1, the two interpretations map λ -terms into typing cliques.

Theorem 6.2.2. *Let ∇ be a legal stable type system.*

1. *If $\nabla = \langle C, \simeq_{\nabla}, \circ_{\nabla} \rangle$ then $\mathcal{M}^{\nabla} = \langle \mathcal{S}(\nabla), \circ_{\nabla}, \llbracket \cdot \rrbracket^{\nabla} \rangle$ is a λ -model.*
2. *If $\nabla = \langle C, \simeq_{\nabla}, \circ_{\nabla}, \nu_{\nabla} \rangle$ then $\mathcal{M}^{\nabla \ell} = \langle \mathcal{S}(\nabla), \circ_{\nabla}, \llbracket \cdot \rrbracket^{\nabla \ell} \rangle$ is a λ -model.*

Proof. In both cases we need to check that the four conditions required to be a λ -model, given in Definition 6.1.1, are respected.

1. Easy, in fact $\{ \sigma \in \mathcal{T}_{\nabla} \mid \exists B. (\forall y. B(y) \subseteq \rho(y)) \wedge B \vdash_{\nabla}^* \mathbf{x} : \sigma \} = \rho(\mathbf{x})$, by Lemma 6.2.1.(i).
2. By Lemma 6.2.1.(iv), it is easy to check that

$$\begin{aligned} \llbracket \mathbf{MN} \rrbracket_{\rho}^{\nabla} &= \{ \sigma \in \mathcal{T}_{\nabla} \mid \exists B. (\forall y. B(y) \subseteq \rho(y)) \text{ such that } B \vdash_{\nabla} \mathbf{MN} : \sigma \} \\ &= \left\{ \sigma \in \mathcal{T}_{\nabla} \mid \exists B. (\forall y. B(y) \subseteq \rho(y)) \text{ such that } \begin{array}{l} B' \vdash_{\nabla} \mathbf{M} : [\sigma_1, \dots, \sigma_n] \rightarrow \sigma, \\ B_i \vdash_{\nabla} \mathbf{N} : \sigma_i, \\ B = (\bigcup_{1 \leq i \leq n} B_i) \cup B' \end{array} \right\} \\ &= \{ \sigma \in \mathcal{T}_{\nabla} \mid [\sigma_1, \dots, \sigma_n] \rightarrow \sigma \in \llbracket \mathbf{M} \rrbracket_{\rho}^{\nabla} \wedge \sigma_i \in \llbracket \mathbf{N} \rrbracket_{\rho}^{\nabla} \} = \llbracket \mathbf{M} \rrbracket_{\rho}^{\nabla} \circ_{\nabla} \llbracket \mathbf{N} \rrbracket_{\rho}^{\nabla} \end{aligned}$$

The lazy case is similar.

3. Let $s \in \mathcal{S}(\nabla)$. By Lemma 6.2.1.(vi), it is easy to check that

$$\begin{aligned} \llbracket \lambda \mathbf{x}. \mathbf{M} \rrbracket_{\rho}^{\nabla} \circ_{\nabla} s &= \{ \sigma \in \mathcal{T}_{\nabla} \mid \sigma_i \in s \text{ and } [\sigma_1, \dots, \sigma_n] \rightarrow \sigma \in \llbracket \lambda \mathbf{x}. \mathbf{M} \rrbracket_{\rho}^{\nabla} \} = \\ &= \left\{ \sigma \in \mathcal{T}_{\nabla} \mid \exists B. (\forall y. B(y) \subseteq \rho(y)) \text{ s.t. } \begin{array}{l} \sigma_i \in s \text{ for all } i \text{ and} \\ B \vdash_{\nabla} \lambda \mathbf{x}. \mathbf{M} : [\sigma_1, \dots, \sigma_n] \rightarrow \sigma \end{array} \right\} \\ &= \left\{ \sigma \in \mathcal{T}_{\nabla} \mid \exists B. (\forall y. B(y) \subseteq \rho(y)) \text{ s.t. } \begin{array}{l} \sigma_i \in s \text{ for all } i \text{ and } \mathbf{x} \notin \text{dom}(B) \\ \text{and } B \cup \{ \mathbf{x} : \sigma_1, \dots, \mathbf{x} : \sigma_n \} \vdash_{\nabla} \mathbf{M} : \sigma \end{array} \right\} \\ &= \{ \sigma \in \mathcal{T}_{\nabla} \mid \exists B'. (\forall y. B'(y) \subseteq \rho[s/\mathbf{x}](y)) \text{ s.t. } B' \vdash_{\nabla} \mathbf{M} : \sigma \} = \llbracket \mathbf{M} \rrbracket_{\rho[s/\mathbf{x}]}^{\nabla} \end{aligned}$$

The lazy case is similar, since $\nu_{\nabla} \in \llbracket \lambda \mathbf{x}. \mathbf{M} \rrbracket_{\rho}^{\nabla \ell}$.

4. Let $s \in \mathcal{S}(\nabla)$ and let $\llbracket \mathbb{M} \rrbracket_{\rho[s/\mathbf{x}]}^{\nabla} = \llbracket \mathbb{M}' \rrbracket_{\rho'[s/\mathbf{y}]}^{\nabla}$. Thus, for all $\sigma \in \mathcal{T}_{\nabla}$:
- $\exists B. (\forall \mathbf{y}. B(\mathbf{y}) \subseteq \rho[s/\mathbf{x}](\mathbf{y}))$ s.t. $B \vdash_{\nabla} \mathbb{M} : \sigma$ iff $\exists B'. (\forall \mathbf{y}. B'(\mathbf{y}) \subseteq \rho'[s/\mathbf{x}](\mathbf{y}))$ s.t. $B' \vdash_{\nabla} \mathbb{M}' : \sigma$. $B \vdash_{\nabla} \lambda \mathbf{x}. \mathbb{M} : [\sigma_1, \dots, \sigma_n] \rightarrow \tau$ if and only if $B \cup \{\mathbf{x} : \sigma_1, \dots, \mathbf{x} : \sigma_n\} \vdash_{\nabla} \mathbb{M} : \tau$ and $\mathbf{x} \notin \text{dom}(B)$ by Lemma 6.2.1.(vi). So $\exists B'. (\forall \mathbf{y}. B'(\mathbf{y}) \subseteq \rho'[B(\mathbf{x})/\mathbf{x}](\mathbf{y}))$ such that $B' \vdash_{\nabla} \mathbb{M}' : \sigma$. Likewise $\exists B^*. (\forall \mathbf{y}. B^*(\mathbf{y}) \subseteq \rho[B'(\mathbf{x})/\mathbf{x}](\mathbf{y}))$ such that $B^* \vdash_{\nabla} \mathbb{M} : \sigma$ therefore $B(\mathbf{x}) \subseteq B'(\mathbf{x}) \subseteq B^*(\mathbf{x})$. Hence $B(\mathbf{x}) = B'(\mathbf{x})$, since $B(\mathbf{x}) = B^*(\mathbf{x})$ by Theorem 6.2.1.(iii). The proof follows by Lemma 6.2.1.(vi). The lazy case is similar by Definition 6.2.5.(iii), since $\nu_{\nabla} \in \llbracket \lambda \mathbf{x}. \mathbb{M} \rrbracket_{\rho}^{\nabla \ell}$.

□

In such a way we defined two meta-models of λ -calculus, which can become models by choosing particular legal stable type systems. Some instances of such meta-models has been studied in [Honsell and Ronchi della Rocca, 1990] and in [Bastonerio et al., 1998].

Definition 6.2.6.

1. $\mathcal{M}^{\nabla} = \langle \mathcal{S}(\nabla), \circ_{\nabla}, \llbracket \cdot \rrbracket^{\nabla} \rangle$ is the clique model induced by the legal stable type system ∇ .
2. $\mathcal{M}^{\nabla \ell} = \langle \mathcal{S}(\nabla), \circ_{\nabla}, \llbracket \cdot \rrbracket^{\nabla \ell} \rangle$ is the lazy clique model induced by the lazy legal stable type system ∇ .

The two meta-models give a non empty interpretation to two interesting classes of terms.

Theorem 6.2.3. *Let ∇ be a legal stable type system.*

1. If \mathbb{M} has head normal form then there are B, σ such that $B \vdash_{\nabla} \mathbb{M} : \sigma$.
2. If \mathbb{M} has weak head normal form then there are B, σ such that $B \vdash_{\nabla}^{\ell} \mathbb{M} : \sigma$.

Proof. 1. The type assignment is closed under $=_{\beta}$, since \vdash_{∇} induces a λ -model. So we need to prove just that every term in head normal form can be typed. Let \mathbb{M} be $\lambda \mathbf{x}_1 \dots \lambda \mathbf{x}_m. \mathbf{x} \mathbb{M}_1 \dots \mathbb{M}_n$ and let $B = \{\mathbf{x} : \underbrace{[] \rightarrow [] \rightarrow \dots []}_{n} \rightarrow \sigma\}$, for a given σ . Then

$B \vdash_{\nabla} \mathbf{x} \mathbb{M}_1 \dots \mathbb{M}_n : \sigma$, by repeatedly applying $(\rightarrow E)$ rule.

If $\mathbf{x} \notin \{\mathbf{x}_1, \dots, \mathbf{x}_m\}$ then $B \vdash_{\nabla} \mathbb{M} : \underbrace{[] \rightarrow [] \rightarrow \dots []}_{m} \rightarrow \sigma$ by m applications of rule $(\rightarrow I)$. Otherwise, if $\mathbf{x} = \mathbf{x}_h$ then

$B \vdash_{\nabla} \mathbb{M} : \underbrace{[] \rightarrow [] \rightarrow \dots []}_{h-1} \rightarrow \underbrace{([] \rightarrow [] \rightarrow \dots [])}_{m} \rightarrow \sigma \rightarrow \underbrace{[] \rightarrow [] \rightarrow \dots []}_{m-h} \rightarrow \sigma$.

2. The case of an abstraction is immediate, by rule *(lazy)*. Otherwise, let \mathbb{M} be $\mathbf{x} \mathbb{M}_1 \dots \mathbb{M}_n$,

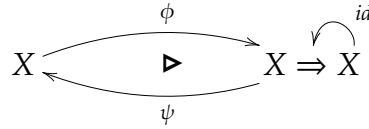
$B_i = \{\mathbf{x} : \underbrace{[] \rightarrow [] \rightarrow \dots []}_{i} \rightarrow \nu_{\nabla}\} (1 \leq i \leq n-1)$, and $B = \{\mathbf{x} : \underbrace{[] \rightarrow [] \rightarrow \dots []}_{n} \rightarrow \sigma\}$,

for a given σ . By rule (\rightarrow_E) , $\bigcup_{j < i} B_j \vdash_{\nabla}^{\ell} \mathbf{xM}_1 \dots \mathbf{M}_i : \nu_{\nabla}$ and $B \cup \bigcup_{j < i} B_j \vdash_{\nabla}^{\ell} \mathbf{xM}_1 \dots \mathbf{M}_i : \underbrace{[] \rightarrow [] \rightarrow \dots []}_{n-i} \rightarrow \sigma$. So $B \cup \bigcup_{j < n} B_j \vdash_{\nabla}^{\ell} \mathbf{xM}_1 \dots \mathbf{M}_n : \sigma$, since it is easy to check that $\star(B, B_1, \dots, B_{n-1})$. □

Theorem 6.2.3 is a necessary ingredient in order to prove that a clique model is adequate with respect to some well-studied operational semantics of λ -calculus (see Property 10.1.15 in page 117 of [Ronchi Della Rocca and Paolini, 2004]).

6.3 Stable λ -models

Let X be a reflexive object in the category **StabCoh**, namely a coherence space X such that $X \Rightarrow X$ is a retract of X (noted $X \triangleright X \Rightarrow X$) through a retraction pair (ϕ, ψ) , where ϕ and ψ are two stable functions such that $\phi : Cl(X) \rightarrow Cl(X \Rightarrow X)$, $\psi : Cl(X \Rightarrow X) \rightarrow Cl(X)$ and $\phi \cdot \psi = id_{X \Rightarrow X}$. We call ϕ an *immersion morphism* and ψ a *projection morphism*. The following categorical diagram summarises the previous notions.



A retraction through (ϕ, ψ) on a coherence space X gives rise to the model for λ -calculus $\mathcal{M} = \langle Cl(X), \circ, \llbracket \cdot \rrbracket \rangle$, where:

- $x \circ y = \mathcal{F}(\phi(x))(y)$
- $\llbracket \cdot \rrbracket : \Lambda \rightarrow Env \rightarrow Cl(X)$ is defined as:
 - $\llbracket \mathbf{x} \rrbracket_{\rho} = \rho(\mathbf{x})$;
 - $\llbracket \lambda \mathbf{x}. \mathbf{M} \rrbracket_{\rho} = \psi(\text{tr}(\lambda \mathbf{y} \in Cl(X). \llbracket \mathbf{M} \rrbracket_{\rho[y/x]}))$
 - $\llbracket \mathbf{MN} \rrbracket_{\rho} = \llbracket \mathbf{M} \rrbracket_{\rho} \circ \llbracket \mathbf{N} \rrbracket_{\rho}$;

where $Env = \{\rho \mid \rho : Var \rightarrow Cl(X)\}$ and λ denotes the meta-theoretic abstraction.

Here we will define two particular classes of such models, putting some restrictions on the retraction pair. If $X = \langle |X|, \circ_X \rangle$ and $Y = \langle |Y|, \circ_Y \rangle$ are two coherence spaces, then a function $i : |X| \rightarrow |Y|$ is *iso-coherent* when $a \circ_X b$ if and only if $i(a) \circ_Y i(b)$, for all $a, b \in |X|$. Note that an iso-coherent function is a map from tokens to tokens.

Definition 6.3.1. *Let X be a coherence space and let $i : |X \Rightarrow X| \rightarrow |X|$ be a total injective iso-coherent function. The pair $\langle X, i \rangle$ is a coherence lambda-structure.*

1. A linear λ -model is the λ -model \mathcal{M} induced by the retraction pair (ϕ_i, ψ_i) defined as

$$\phi_i(x) = \{a \mid i(a) \in x\} \quad \psi_i(y) = \{i(a) \mid a \in y\}$$

2. Let X be such that $|X|$ contains a distinguished token j such that it is coherent with all other tokens and $j \notin i(|X \Rightarrow X|)$. A lazy stable λ -model is the λ -model \mathcal{M} induced by the retraction pair (ϕ_i^j, ψ_i^j) defined as

$$\phi_i^j(x) = \begin{cases} \{a \mid i(a) \in x\} & \text{if } j \in x \\ \emptyset & \text{otherwise} \end{cases} \quad \psi_i^j(y) = \{i(a) \mid a \in y\} \cup \{j\}$$

It is routine to check that linear λ -models and lazy λ -models defined in the previous definition are well-given λ -model in the sense of Definition 6.1.1, since ϕ_i, ψ_i, ϕ_i^j and ψ_i^j are stable functions and both (ϕ_i, ψ_i) and (ϕ_i^j, ψ_i^j) are retraction pairs.

The class of linear λ -models (first defined in [Girard, 1986]) collects all models in the category **StabCoh** whose immersion-projection morphisms (ϕ, ψ) are linear functions, hence the adjective “linear”. The class of lazy λ -models collects instead a sub-class of those models whose projection morphism ψ is not strict, i.e. $\psi(\emptyset) \neq \emptyset$.

Let ∇ be a legal stable type system. In the following of this section, we denote respectively by $[\tau]_{\nabla}$ and $\mathbb{T}_{\nabla}/\simeq_{\nabla}$ the equivalence class $\{\sigma \in \mathbb{T}_{\nabla} \mid \sigma \simeq_{\nabla} \tau\}$ and the whole set of such classes, i.e., $\{[\tau]_{\nabla} \mid \tau \in \mathbb{T}_{\nabla}\}$. The next lemma proves that types are names for tokens. The type theory formalises the homonymy under which tokens are represented. Yet, an arrow is a special name denoting that the subjecting token is in the image of the projection of an opportune retraction.

Lemma 6.3.1. *If ∇ be a legal stable type system then $X_{\nabla} = \langle \mathbb{T}_{\nabla}/\simeq_{\nabla}, \circ_{\nabla} \rangle$ is a coherence space.*

Proof. Easy, since the relation \circ_{∇} is reflexive and symmetric by construction. \square

So, by Definition 2.3.18, $X_{\nabla} \Rightarrow X_{\nabla} = \langle |X_{\nabla} \Rightarrow X_{\nabla}|, \circ_{X_{\nabla} \Rightarrow X_{\nabla}} \rangle$ is the coherence space, where:

1. $|X_{\nabla} \Rightarrow X_{\nabla}| = \left\{ \left(\{[\sigma_1]_{\nabla}, \dots, [\sigma_n]_{\nabla}\}, [\tau]_{\nabla} \mid \circ_{\nabla} \{\sigma_1, \dots, \sigma_n\} \right) \right\}$,
2. $(\{[\sigma_1]_{\nabla}, \dots, [\sigma_n]_{\nabla}\}, [\sigma]_{\nabla}) \circ_{X_{\nabla} \Rightarrow X_{\nabla}} (\{[\tau_1]_{\nabla}, \dots, [\tau_m]_{\nabla}\}, [\tau]_{\nabla})$ whenever, if $\circ_{\nabla} \{[\sigma_1]_{\nabla}, \dots, [\sigma_n]_{\nabla}, [\tau_1]_{\nabla}, \dots, [\tau_m]_{\nabla}\}$ then $[\sigma]_{\nabla} \circ_{\nabla} [\tau]_{\nabla}$ and $[\sigma]_{\nabla} = [\tau]_{\nabla}$ implies $\{[\sigma_1]_{\nabla}, \dots, [\sigma_n]_{\nabla}\} = \{[\tau_1]_{\nabla}, \dots, [\tau_m]_{\nabla}\}$.

Definition 6.3.2. *Let ∇ be a legal stable type system. The legal-structure induced by ∇ is the pair $\langle X_{\nabla}, i_{\nabla} \rangle$ where $X_{\nabla} = \langle \mathbb{T}_{\nabla}/\simeq_{\nabla}, \circ_{\nabla} \rangle$ and $i_{\nabla} : |X_{\nabla} \Rightarrow X_{\nabla}| \rightarrow |X_{\nabla}|$ is the map such that:*

$$i_{\nabla} \left(\left(\{[\sigma_1]_{\nabla}, \dots, [\sigma_n]_{\nabla}\}, [\tau]_{\nabla} \right) \right) = [[\sigma_1, \dots, \sigma_n] \rightarrow \tau]_{\nabla}.$$

Note that we denote sets by sequences of their elements without repetitions.

We can show that legal-structures induce coherence lambda-structures.

Lemma 6.3.2. *Let $\langle X_{\nabla}, i_{\nabla} \rangle$ be a legal-structure.*

1. $\langle X_{\nabla}, i_{\nabla} \rangle$ is a coherence λ -structure, so it induces a linear λ -model.

2. If ∇ is lazy then $\langle X_\nabla, i_\nabla \rangle$ induces also a lazy λ -model by putting $j = [v_\nabla]_{\underline{v}}$.

Proof. 1. Let $x = \{[\sigma_1]_{\underline{v}}, \dots, [\sigma_n]_{\underline{v}}\}$, $y = \{[\tau_1]_{\underline{v}}, \dots, [\tau_m]_{\underline{v}}\}$, $a = [\sigma]_{\underline{v}}$ and $b = [\tau]_{\underline{v}}$. We want to check that i_∇ is injective. Suppose $i_\nabla((x, a)) = i_\nabla((y, b))$. By definition of i_∇ , this means that $[\sigma_1, \dots, \sigma_n] \rightarrow \sigma \simeq_\nabla [\tau_1, \dots, \tau_m] \rightarrow \tau$. Since the type system is legal then $\{\sigma_1, \dots, \sigma_m\} \simeq_\nabla \{\tau_1, \dots, \tau_n\}$, $m = n$ and $\sigma \simeq_\nabla \tau$. Thus $(x, a) = (y, b)$. Moreover, it is easy to check that $i_\nabla((x, a)) \frown_\nabla i_\nabla((y, b))$ if and only if $(a, x) \frown_{X_\nabla \Rightarrow X_\nabla} (y, b)$ by Definition 6.2.1.(iii).

2. The proof follows easily, since $j = [v_\nabla]_{\underline{v}}$ trivially enjoys the required coherence constraints. □

We are now able to state a soundness result proving that all our clique models correspond to linear and lazy models. Let us say that two λ -models $\mathcal{M} = \langle \mathbb{D}, \circ, \llbracket \cdot \rrbracket^{\mathcal{M}} \rangle$ and $\mathcal{M}' = \langle \mathbb{D}', \circ', \llbracket \cdot \rrbracket^{\mathcal{M}'} \rangle$ coincide if \mathbb{D} and \mathbb{D}' are isomorphic through a bijection $h : \mathbb{D}' \rightarrow \mathbb{D}$, and $\llbracket \mathbf{M} \rrbracket_\rho^{\mathcal{M}} = h(\llbracket \mathbf{M} \rrbracket_{\rho'}^{\mathcal{M}'})$, where $\rho(\mathbf{x}) = h(\rho'(\mathbf{x}))$.

Theorem 6.3.1 (Soundness). *Let ∇ be a legal stable type system.*

1. The clique model \mathcal{M}^∇ and the linear λ -model induced by $\langle X_\nabla, i_\nabla \rangle$ coincide.
2. If ∇ is lazy with pivot v_∇ then the lazy clique model $\mathcal{M}^{\nabla \ell}$ and the lazy λ -model induced by $\langle X_\nabla, i_\nabla \rangle$, with $j = [v_\nabla]_{\underline{v}}$, coincide.

Proof. 1. Let us recall that the clique model induced by ∇ is $\mathcal{M}^\nabla = \langle \mathcal{S}(\nabla), \circ_\nabla, \llbracket \cdot \rrbracket^\nabla \rangle$. The model induced by $\langle X_\nabla, i_\nabla \rangle$, is $\mathcal{X}^\nabla = \langle Cl(X_\nabla), \circ, \llbracket \cdot \rrbracket^{\mathcal{X}^\nabla} \rangle$, where $x \circ y = \mathcal{F}(\phi_{i_\nabla}(x))(y)$, and the interpretation function is defined as at the beginning of this section. The isomorphism between $\mathcal{S}(\nabla)$ and $Cl(X_\nabla)$ follows from the definition of X_∇ , namely, for each $s \in \mathcal{S}(\nabla)$ the corresponding element of $Cl(X_\nabla)$ is $\{[\tau]_{\underline{v}} \mid \tau \in s\}$. Then it is boring but easy to check, by cases, that $\llbracket \mathbf{M} \rrbracket_\rho^{\mathcal{X}^\nabla} = \llbracket \mathbf{M} \rrbracket_{\rho^*}^{\nabla} / \simeq_\nabla$ where $\rho^*(\mathbf{x}) = \bigcup_{[\tau]_{\underline{v}} \in \rho(\mathbf{x})} [\tau]_{\underline{v}}$, for all \mathbf{M} .

2. The proof is similar to the previous point. □

In order to prove completeness we need to recall some basic property of coherence spaces.

Proposition 6.3.1. *Let X and Y be object of **StabCoh**, \mathcal{I} be the set of all (possible) isomorphisms between X and Y and \mathcal{T} be the set of all (possible) iso-coherent bijective maps between $|X|$ and $|Y|$. A biunivocal correspondence can be established between \mathcal{I} and \mathcal{T} .*

Proof. Isomorphisms of **StabCoh** between X and Y are all and only bijective linear functions [Girard, 1987] whose trace univocally determines an iso-coherent bijective maps between $|X|$ and $|Y|$. For the other direction, an iso-coherent bijective map

between $|X|$ and $|Y|$ defines univocally a trace of a bijective linear function, i.e. an isomorphism (the hypothesis of iso-coherence is crucial in order to have a well-defined linear function). So there exists a biunivocal correspondence between \mathcal{I} and \mathcal{T} . \square

Now we will prove that, given a (lazy) linear λ -model induced by a coherence space we can define a type system inducing a (lazy) clique model coinciding with it. Then the completeness follows.

Lemma 6.3.3. *Let $\langle X, i \rangle$ be a coherence lambda-structure.*

1. *There is a legal stable type system ∇ inducing the legal-structure $\langle X_\nabla, i_\nabla \rangle$ and there are two iso-coherent bijections $(\cdot)^b : |X| \rightarrow |X_\nabla|$ and $(\cdot)^\# : |X \Rightarrow X| \rightarrow |X_\nabla \Rightarrow X_\nabla|$ such that the following diagram commutes*

$$\begin{array}{ccc} |X \Rightarrow X| & \xrightarrow{i} & |X| \\ (\cdot)^\# \downarrow & & \downarrow (\cdot)^b \\ |X_\nabla \Rightarrow X_\nabla| & \xrightarrow{i_\nabla} & |X_\nabla| \end{array}$$

2. *Let X be such that $|X|$ contains a distinguished token j coherent with all its other tokens and $j \notin i(|X \Rightarrow X|)$. Then there is a legal lazy stable type system ∇ such that the previous diagram commutes.*

Proof. Let C_∇ be a set in bijection with the set $|X|$ through the map $(\cdot)^b : |X| \rightarrow C_\nabla$. Let \simeq_∇ be the least stable type theory on $\text{Row}(C_\nabla)$ such that, if $(x, a) \in |X \Rightarrow X|$, $x = \{a_1, \dots, a_n\}$ and $i((x, a)) = b \in |X|$ then $[a_1^b, \dots, a_n^b] \rightarrow a^b \simeq_\nabla (b)^b$. Since i is injective, $(\cdot)^b$ induces an injection from $|X|$ to $\text{Row}(C_\nabla)/\simeq_\nabla$, i.e. if $a, b \in |X|$ and $a^b \simeq_\nabla b^b$ then $a = b$. Let \wedge_∇ be the least binary relation on $\text{Row}(C_\nabla)/\simeq_\nabla$ satisfying Definition 6.2.1.(iii) and such that, if $a \wedge_X b$ then $[(a)^b]_{\simeq_\nabla} \wedge_\nabla [(b)^b]_{\simeq_\nabla}$, for all $a, b \in |X|$.

We will prove that $\nabla = \langle C_\nabla, \wedge_\nabla, \simeq_\nabla \rangle$ is the desired stable type system.

We already proved that $(\cdot)^b$ induces a bijection between $|X|$ and $\text{T}_\nabla/\simeq_\nabla$. Injectivity follows since $\text{T}_\nabla/\simeq_\nabla \subseteq \text{Row}(C_\nabla)/\simeq_\nabla$. We can easily prove by induction that if $\sigma \in \text{T}_\nabla$ then $\sigma \simeq_\nabla a^b$ for a $a \in |X|$. Thus surjectivity follows.

We show that the type system so obtained is legal. We begin by the point (i) of Definition 6.2.4. Suppose that $\{\sigma_1, \dots, \sigma_n\} \not\equiv_\nabla \{\tau_1, \dots, \tau_m\}$ or $\sigma \not\equiv_\nabla \tau$. Let x, y, a, b such that $x = \{a_1, \dots, a_n\}$ with $(a_j)^b \simeq_\nabla \sigma_j$ for all $j \in [1, n]$, $y = \{b_1, \dots, b_m\}$ with $(b_k)^b \simeq_\nabla \tau_k$ for all $k \in [1, m]$, $(a)^b \simeq_\nabla \sigma$, $(b)^b \simeq_\nabla \tau$. Since $(\cdot)^b$ induces a bijection between $|X|$ and $\text{T}_\nabla/\simeq_\nabla$, either $\sigma \not\equiv_\nabla \tau$ implies $a \neq b$ or $\{\sigma_1, \dots, \sigma_n\} \not\equiv_\nabla \{\tau_1, \dots, \tau_m\}$ implies that there is $a_j \neq b_k$. Then (x, a) and (y, b) are two different tokens of $|X \Rightarrow X|$. Since $i : |X \Rightarrow X| \rightarrow |X|$ is an injective function, we conclude $[\sigma_1, \dots, \sigma_n] \rightarrow \sigma \not\equiv_\nabla [\tau_1, \dots, \tau_m] \rightarrow \tau$ as required. To prove the remaining point, we just observe that $(x, a) \wedge_{X \Rightarrow X} (y, b)$ if and only if either $a \wedge_X b$ or there exist $c_1 \in x, c_2 \in y$ such that $c_1 \smile_X c_2$. Thus point (ii) can be proved by using a reasoning similar to that for point (i).

Legality implies that $X_\nabla = \langle \mathbb{T}_\nabla / \approx_\nabla, \circ_\nabla \rangle$ is a coherence space. Hence, the map $(\cdot)^\sharp : |X \Rightarrow X| \longrightarrow |X_\nabla \Rightarrow X_\nabla|$ is induced by the bijection between $|X|$ and $|X_\nabla|$. If $(\{a_1, \dots, a_n\}, b) \in |X \Rightarrow X|$, then $\wedge_\nabla \{[(a_1)^b]_{\underline{v}}, \dots, [(a_n)^b]_{\underline{v}}\}$ and we define $(\{a_1, \dots, a_n\}, b)^\sharp = (([(a_1)^b]_{\underline{v}}, \dots, [(a_n)^b]_{\underline{v}}), [(a)^b]_{\underline{v}})$. It is easy to see that such a map is bijective, since $(\cdot)^\flat$ is. Last, we choose as i_∇ exactly that of Definition 6.3.2. It is easy to check that $(i((x, a)))^\flat = i_\nabla((x, a)^\sharp)$, for all $(x, a) \in |X \Rightarrow X|$. So, the proof is done.

The proof of point (ii) is similar, taking $(j)^\sharp = [v_\nabla]_{\underline{v}}$. □

Corollary 6.3.1 (Completeness).

Let \mathcal{M} and \mathcal{M}' be respectively a linear λ -model and a lazy λ -model.

1. *There is a legal type system ∇ inducing a clique model that coincides with \mathcal{M} .*
2. *There is a legal lazy type system ∇ inducing a lazy clique model that coincides with \mathcal{M}' .*

Proof. By Lemma 6.3.3 we can build a legal type system ∇ inducing a linear model coinciding with \mathcal{M} . Thus, the proof follows by Theorem 6.3.1. The proof of point (ii) is similar. □

6.4 Conclusions and comparison with related works

Logical semantics in the setting of coherence spaces and stable functions has been previously studied in [Bastonero et al., 1998] and [Honsell and Ronchi della Rocca, 1990]. In [Honsell and Ronchi della Rocca, 1990] a class of stable λ -models is considered, containing as proper subclass the linear stable models. More precisely, in [Honsell and Ronchi della Rocca, 1990] stable models are considered, based on a space X containing as retract any subset of $X \Rightarrow X$, where the retract is realised by linear functions. Thus our linear models are a proper subset of them. Starting from a λ -model \mathcal{M} of this kind, a formal system is designed, assigning to λ -terms tokens of the coherence space on which \mathcal{M} is based. This formal system provides a logical description of the interpretation function of the model \mathcal{M} : the fact that the token a belongs to the interpretation of a term M is logically equivalent to the fact that the formal system is able to assign a to M . From this formal system, three type assignment systems are designed, in order to study three particular stable λ -models. All these models are built as an inverse limit solution of a domain equation, which is $X \approx X \Rightarrow X$ for two of them and $X \approx \mathbb{N} \otimes (X \Rightarrow X)$ for the remaining one, where \mathbb{N} is the coherence space of natural numbers.

The rules of our type assignment system \vdash are in some sense inspired by this work. But in [Honsell and Ronchi della Rocca, 1990] the formal system is guided by the semantics, since it manipulates directly tokens of a coherence space, while we define the type assignment system in a completely syntactical manner, and then the legality condition assure us that the resulting clique model is isomorphic to a stable model. So we prove an isomorphism between clique models and stable models,

while in [Honsell and Ronchi della Rocca, 1990] an isomorphism is proved between denotational interpretation and “token” assignment.

In [Bastone et al., 1998] a general definition of a model of the lazy λ -calculus (in the sense of [Abramsky, 1990, Abramsky and Ong, 1993, Ong, 1988]) is given, and two particular stable models for it are built, through two type assignment systems. Both these models are built as an inverse limit solution of the domain equation $X \approx \mathbf{1} \& (X \Rightarrow X)$, where $\mathbf{1}$ denotes the coherence space with just one token. One of the systems defined in [Bastone et al., 1998] is an instance of our system \vdash_{∇}^{ℓ} .

In conclusion we want to point out that (lazy) clique models have a very easy syntactical definition, and they can be used for studying different denotational semantics of λ -calculus, without any acquaintance with coherence spaces and stable functions. The results in Section 6.3 assure that reasoning on a legal clique model corresponds to reasoning in a well established mathematical category.

Bibliography

- [Abelson et al., 1985] Abelson, H., Sussman, G. J., and Sussman, J. (1985). *Structure and Interpretation of Computer Programs*. The MIT Press, Cambridge, MA. <http://mitpress.mit.edu/sicp/full-text/book/book.html>.
- [Abramsky, 1990] Abramsky, S. (1990). The lazy lambda-calculus. In Turner, D. N., editor, *Research Topics in Functional Programming*, pages 65–117. Addison-Wesley Publ. Co.
- [Abramsky, 1991] Abramsky, S. (1991). Domain theory in logical form. *Annals of Pure and Applied Logic*, 51(1-2):1–77.
- [Abramsky et al., 2000] Abramsky, S., Malacaria, P., and Jagadeesan, R. (2000). Full abstraction for PCF. *Information and Computation*, 163(2):409–470. An extended abstract can be found in *Theoretical Aspects of Computer Software (Sendai, 1994)*, Lecture Notes in Computer Science, 789:1-15, Springer-Verlag, Berlin, 1994.
- [Abramsky and Ong, 1993] Abramsky, S. and Ong, L. (1993). Full abstraction in the lazy lambda calculus. *Information and Computation*, 105(2):159–267.
- [Alves et al., 2006] Alves, S., Fernández, M., Florido, M., and Mackie, I. (2006). The power of linear functions. In Ésik, Z., editor, *Proceedings of the 20th International Workshop on Computer Science Logic, 15th Annual Conference of the EACSL, Szeged, Hungary, September 25-29, 2006*, volume 4207 of *Lecture Notes in Computer Science*, pages 119–134. Springer-Verlag.
- [Alves et al., 2008] Alves, S., Florido, M., Mackie, I., and Sinot, F.-R. (2008). Minimality in a linear calculus with iteration. *Electronic Notes in Theoretical Computer Science*, 204:163–179.
- [Asperti and Guerrini, 1998] Asperti, A. and Guerrini, S. (1998). *The optimal implementation of functional programming languages*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge, UK.
- [Asperti and Longo, 1991] Asperti, A. and Longo, G. (1991). *Categories, Types, and Structures: An Introduction to Category Theory for the Working Computer Scientist*. Foundations of Computing Series. The MIT Press, Cambridge, MA.
- [Asperti and Roversi, 2002] Asperti, A. and Roversi, L. (2002). Intuitionistic light affine logic. *ACM Transactions on Computational Logic*, 3(1):1–39.

- [Barendregt, 1984] Barendregt, H. (1984). *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in logic and the foundation of mathematics*. North-Holland, Amsterdam, revised edition. (First edition, 1981).
- [Barendregt et al., 1983] Barendregt, H., Coppo, M., and Dezani-Ciancaglini, M. (1983). A filter lambda model and the completeness of type assignment. *The Journal of Symbolic Logic*, 48(4):931–940.
- [Bastonerio et al., 1998] Bastonerio, O., Pravato, A., and Ronchi Della Rocca, S. (1998). Structures for lazy semantics. In de Roeper, G., editor, *Programming Concepts and Methods*, pages 30–48. Chaptman & Hall. International Conference on Programming Concepts and Methods - PROCOMET'98, 8-12 June 1998, Shelter Island, New York, USA.
- [Beffara, 2008] Beffara, E. (2008). An algebraic process calculus. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008, 24-27 June 2008, Pittsburgh, PA, USA*, pages 130–141. IEEE Computer Society.
- [Bellantoni et al., 2000] Bellantoni, S., Niggl, K. H., and Schwichtenberg, H. (2000). Higher type recursion, ramification and polynomial time. *Annals of Pure and Applied Logic*, 104:17–30.
- [Benton et al., 1992] Benton, N., Bierman, G. M., Hyland, J. M. E., and de Paiva, V. (1992). Linear λ -calculus and categorical models revisited. In Börger, E., editor, *Proceedings of the Sixth Workshop on Computer Science Logic - CSL*, volume 702 of *Lecture Notes in Computer Science*, pages 61–84. Springer-Verlag.
- [Berger et al., 2001] Berger, M., Honda, K., and Yoshida, N. (2001). Sequentiality and the pi-calculus. In Abramsky, S., editor, *Proceedings of Typed Lambda Calculi and Applications, 5th International Conference, TLCA 2001, Krakow, Poland, May 2-5, 2001*, volume 2044 of *Lecture Notes in Computer Science*, pages 29–45. Springer-Verlag.
- [Berger et al., 2003] Berger, M., Honda, K., and Yoshida, N. (2003). Genericity and the pi-calculus. In Gordon, A. D., editor, *Proceedings of Foundations of Software Science and Computational Structures, 6th International Conference, FOSSACS 2003 Held as Part of the Joint European Conference on Theory and Practice of Software, ETAPS 2003, Warsaw, Poland, April 7-11, 2003*, volume 2620 of *Lecture Notes in Computer Science*, pages 103–119. Springer-Verlag.
- [Berry, 1978] Berry, G. (1978). Stable models of typed λ -calculi. In Ausiello, G. and Böhm, C., editors, *Fifth International Colloquium on Automata, Languages and Programming - - ICALP'78, Udine, Italy, July 17-21, 1978*, volume 62 of *Lecture Notes in Computer Science*, pages 72–89. Springer-Verlag.
- [Berry, 1979] Berry, G. (1979). *Modèles complètement adéquats et stable du lambda-calcul typé*. PhD thesis, Thèse de Doctorat d'État, Université de Paris VII, France.

- [Berry et al., 1985] Berry, G., Curien, P.-L., and Lévy, J.-J. (1985). Full abstraction for sequential languages: the state of the art. In Nivat, M. and Reynolds, J., editors, *Algebraic Semantics*, pages 89–132. Cambridge University Press.
- [Blass, 1992] Blass, A. (1992). A game semantics for linear logic. *Annals of Pure and Applied Logic*, 56:183–220. Special Volume dedicated to the memory of John Myhill.
- [Blute and Scott, 2004] Blute, R. and Scott, P. (2004). Category theory for linear logician. In Ruet, P., Ehrhard, T., Girard, J., and Scott, P., editors, *Proceedings Linear Logic Summer School*. Cambridge University Press.
- [Boreale and Sangiorgi, 1998] Boreale, M. and Sangiorgi, D. (1998). A fully abstract semantics for causality in the π -calculus. *Acta Informatica*, 35(5):353–400.
- [Bucciarelli and Ehrhard, 1991] Bucciarelli, A. and Ehrhard, T. (1991). Sequentiality and strong stability. In *Proceedings of the Symposium on Logic in Computer Science - LICS'91*, pages 138–145.
- [Bucciarelli and Ehrhard, 1994] Bucciarelli, A. and Ehrhard, T. (1994). Sequentiality in an extensional framework. *Information and Computation*, 110(2):265–296.
- [Cartwright et al., 1994] Cartwright, R., Curien, P.-L., and Felleisen, M. (1994). Fully abstract semantics for observably sequential languages. *Information and Computation*, 111(2):297–401.
- [Cockett and Seely, 1997] Cockett, J. R. B. and Seely, R. A. G. (1997). Proof theory for full intuitionistic logic, bilinear logic and mixcategories. *Theory Appl. Categories* 3, pages 85–131.
- [Coppo and Dezani-Ciancaglini, 1980] Coppo, M. and Dezani-Ciancaglini, M. (1980). An extension of the basic functionality theory for the λ -calculus. *Notre Dame Journal of Formal Logic*, 21(4):685–693.
- [Coppo et al., 1984] Coppo, M., Dezani-Ciancaglini, M., Honsell, F., and Longo, G. (1984). Extended type structure and filter lambda models. In Lolli, G., Longo, G., and Marcja, A., editors, *Logic Colloquim'82*, pages 241–262. Elsevier Science Publishers, Amsterdam.
- [Coppo et al., 1987] Coppo, M., Dezani-Ciancaglini, M., and Zacchi, M. (1987). Type theories, normal forms, and D_∞ -lambda-models. *Information and Computation*, 72(2):85–116.
- [Crafa et al., 2007] Crafa, S., Varacca, D., and Yoshida, N. (2007). Compositional event structure semantics for the internal π -calculus. In Caires, L. and Vasconcelos, V. T., editors, *CONCUR 2007 - Concurrency Theory, 18th International Conference, CONCUR 2007, Lisbon, Portugal, September 3-8, 2007, Proceedings*, volume 4703 of *Lecture Notes in Computer Science*, pages 317–332. Springer.

- [Crole, 1993] Crole, R. L. (1993). *Categories for Types*. Cambridge University Press, Cambridge.
- [Curien, 1986] Curien, P. L. (1986). *Categorical Combinators, Sequential Algorithms, and Functional Programming*. Research Notes in Theoretical Computer Science. Pitman, London. Available in U.S. from John Wiley and Sons.
- [Curien, 2007] Curien, P.-L. (2007). Notes on game semantics. Universit de Paris 7 Course Notes. www.pps.jussieu.fr/~curien.
- [Curien and Faggian, 2005] Curien, P.-L. and Faggian, C. (2005). L-nets, strategies and proof-nets. In Ong, C.-H. L., editor, *Proceedings of Computer Science Logic, 19th International Workshop, CSL 2005, 14th Annual Conference of the EACSL, Oxford, UK, August 22-25, 2005, Proceedings*, volume 3634 of *Lecture Notes in Computer Science*, pages 167–183. Springer.
- [Cutland, 1980] Cutland, N. (1980). *Computability: an Introduction to Recursive Function Theory*. Cambridge University Press.
- [Dal Lago, 2005] Dal Lago, U. (2005). The geometry of linear higher-order recursion. In *Proceedings of 20th IEEE Symposium on Logic in Computer Science (LICS 2005), 26-29 June 2005, Chicago, IL, USA*, pages 366–375. IEEE Computer Society Press.
- [Dal Lago et al., 2004] Dal Lago, U., Martini, S., and Roversi, L. (2004). Higher-order linear ramified recurrence. In Berardi, S., Coppo, M., and Damiani, F., editors, *International Workshop Types for Proofs and Programs, TYPES 2003, Torino, Italy, April 30 - May 4, 2003, Revised Selected Papers*, volume 3085 of *Lecture Notes in Computer Science*, pages 178–193. Springer-Verlag.
- [Davis and Weyuker, 1983] Davis, M. and Weyuker, E. J. (1983). *Computability, Complexity and Languages*. Computer Science and Applied Mathematics. Academic Press.
- [de Paiva, 2006] de Paiva, V. (2006). Categorical semantics of linear logic for all. Manuscript.
- [Degano and Priami, 1995] Degano, P. and Priami, C. (1995). Causality for mobile processes. In Fülöp, Z. and Gécseg, F., editors, *Proceedings of Automata, Languages and Programming, 22nd International Colloquium, ICALP95, Szeged, Hungary, July 10-14, 1995*, volume 944 of *Lecture Notes in Computer Science*, pages 660–671. Springer-Verlag.
- [Dezani-Ciancaglini et al., 2004] Dezani-Ciancaglini, M., Ghilezan, S., and Likavec, S. (2004). Behavioural inverse limit lambda-models. *Theoretical Computer Science*. To appear.

- [Dezani-Ciancaglini et al., 1986] Dezani-Ciancaglini, M., Honsell, F., and Ronchi Della Rocca, S. (1986). Models for theories of functions strictly depending on all their arguments. *The Journal of Symbolic Logic*, 51(3):845–846. (Abstract).
- [Egidi et al., 1992] Egidi, L., Honsell, F., and Ronchi Della Rocca, S. (1992). Operational, denotational and logical descriptions: a case study. *Fundamenta Informaticæ*, 16(2):149–170.
- [Ehrhard, 1995] Ehrhard, T. (1995). Hypercoherence: A strongly stable model of linear logic. In Girard, J.-Y., Lafont, Y., and Regnier, L., editors, *Proceedings of the Workshop on Advances in Linear Logic*, number 222 in London Mathematical Society Lecture Note Series, pages 83–108. Cambridge University Press, Ithaca, New York.
- [Faggian and Maurel, 2005] Faggian, C. and Maurel, F. (2005). Ludics nets, a game model of concurrent interaction. In *20th IEEE Symposium on Logic in Computer Science (LICS 2005), 26-29 June 2005, Chicago, IL, USA, Proceedings*, pages 376–385. IEEE Computer Society.
- [Faggian and Piccolo, 2007a] Faggian, C. and Piccolo, M. (2007a). A graph abstract machine describing event structure composition. *Electronic Notes in Theoretical Computer Science*, 175(4):21–36.
- [Faggian and Piccolo, 2007b] Faggian, C. and Piccolo, M. (2007b). Ludics is a model for the finitary linear pi-calculus. In Rocca, S. R. D., editor, *Proceedings of Typed Lambda Calculi and Applications, 8th International Conference, TLCA 2007, Paris, France, June 26-28, 2007*, volume 4583 of *Lecture Notes in Computer Science*, pages 148–162. Springer.
- [Faggian and Piccolo, 2009] Faggian, C. and Piccolo, M. (2009). Partial orders, event structures and linear strategies. In Curien, P.-L., editor, *Typed Lambda Calculi and Applications, 9th International Conference, TLCA 2009, Brasilia, Brazil, July 1-3, 2009. Proceedings*, volume 5608 of *Lecture Notes in Computer Science*, pages 95–111. Springer.
- [Gaboardi, 2007] Gaboardi, M. (2007). *Linearity: an Analytic Tool in the study of Complexity and Semantics of Programming Languages*. PhD thesis, Università degli Studi di Torino - Institut National Polytechnique de Lorraine.
- [Gaboardi and Paolini, 2007] Gaboardi, M. and Paolini, L. (2007). Syntactical, operational and denotational linearity. In *Workshop on Linear Logic, Ludics, Implicit Complexity and Operator Algebras. Dedicated to Jean-Yves Girard on his 60th birthday, Certosa di Pontignano, Siena*.
- [Gaboardi and Piccolo, 2009] Gaboardi, M. and Piccolo, M. (2009). Categorical models for a semantically linear λ -calculus. In *Proceedings of LINEARITY'09*. To appear.
- [Girard, 1986] Girard, J.-Y. (1986). The system F of variable types, fifteen years later. *Theoretical Computer Science*, 45(2):159–192.

- [Girard, 1987] Girard, J.-Y. (1987). Linear logic. *Theoretical Computer Science*, 50:1–102.
- [Girard, 2001] Girard, J.-Y. (2001). Locus solum: from the rules of logics to the logics of rules. *Mathematical Structures in Computer Science*, 11:301–506.
- [Girard et al., 1989] Girard, J.-Y., Lafont, Y., and Taylor, P. (1989). *Proofs and Types*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge.
- [Gunter, 1992] Gunter, C. A. (1992). *Semantics of Programming Languages: Structures and Techniques*. Foundations of Computing Series. The MIT Press, Cambridge, MA.
- [Hennessy, 1991] Hennessy, M. (1991). A model for the π -calculus. Technical report, University of Sussex.
- [Hindley, 1989] Hindley, J. R. (1989). BCK-combinators and linear λ -terms have types. *Theoretical Computer Science*, 64:97–105.
- [Hindley and Longo, 1980] Hindley, J. R. and Longo, G. (1980). Lambda calculus models and extensionality. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 26:289–310.
- [Hoare, 1985] Hoare, C. (1985). *Communicating Sequential Processes*. Prentice Hall.
- [Honda et al., 2000] Honda, K., Vasconcelos, V. T., and Yoshida, N. (2000). Secure information flow as typed process behaviour. In Smolka, G., editor, *Programming Languages and Systems, 9th European Symposium on Programming, ESOP 2000, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS 2000, Berlin, Germany, March 25 - April 2, 2000, Proceedings*, volume 1782 of *Lecture Notes in Computer Science*, pages 180–199. Springer.
- [Honda and Yoshida, 1995] Honda, K. and Yoshida, N. (1995). On reduction-based process semantics. *Theoretical Computer Science*, 151(2):437–486.
- [Honda et al., 2004] Honda, K., Yoshida, N., and Berger, M. (2004). Control in the pi-calculus. submitted.
- [Honsell and Ronchi della Rocca, 1990] Honsell, F. and Ronchi della Rocca, S. (1990). Reasoning about interpretation in qualitative lambda-models. In Broy, M. and Jones, C., editors, *Proceedings of Working Conference on Programming Concepts and Methods - IFIP 2.2*, pages 505–521, Sea of Galilee, Israel. North-Holland.
- [Honsell and Ronchi Della Rocca, 1992] Honsell, F. and Ronchi Della Rocca, S. (1992). An approximation theorem for topological lambda models and the topological incompleteness of lambda calculus. *Journal of Computer and System Sciences*, 45(1):49–75.

- [Hyland and Ong, 1995] Hyland, J. M. E. and Ong, L. C.-H. (1995). Pi-calculus, dialogue games and PCF. In *Proceedings of Conference on Functional Programming Languages and Computer Architecture FPCA95, La Jolla, CA, USA*, pages 96–107. ACM Press.
- [Hyland and Ong, 2000] Hyland, J. M. E. and Ong, L. C.-H. (2000). On full abstraction for PCF: I, II, and III. *Information and Computation*, 163(2):285–408.
- [Klop, 2007] Klop, J. W. (2007). New fix point combinators from old. In Barendsen, E., Capretta, V., Geuvers, H., and Niqui, M., editors, *Reflections on Type Theory*. Radboud University Nijmegen.
- [Kobayashi et al., 1996] Kobayashi, N., Pierce, B. C., and Turner, D. N. (1996). Linear types and the Pi-Calculus. In *23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages - POPL*, pages 358–371. ACM Press.
- [Kobayashi et al., 1999] Kobayashi, N., Pierce, B. C., and Turner, D. N. (1999). Linearity and the Pi-Calculus. *ACM Transactions on Computational Logic*, 21(5):914–947.
- [Koymans, 1982] Koymans, C. P. J. (1982). Models of the lambda calculus. *Information and Computation*, 52(3):306–323.
- [Lambek and Scott, 1986] Lambek, J. and Scott, P. J. (1986). *Introduction to higher order categorical logic*. Cambridge Studies in Advanced Mathematics. Cambridge University Press.
- [Laneve and Victor, 2003] Laneve, C. and Victor, B. (2003). Solos in concert. *Mathematical Structures in Computer Science*, 13(5):657–683.
- [Longley, 2002] Longley, J. R. (2002). The sequentially realizable functionals. *Annals of Pure and Applied Logic*, 117:1–93.
- [Mackie, 1994] Mackie, I. (1994). Lilac: A functional programming language based on linear logic. *Journal of Functional Programming*, 4(4):1 – 39.
- [Mackie et al., 1993] Mackie, I., Román, L., and Abramsky, S. (1993). An internal language for autonomous categories. *Applied Categorical Structures*, 1(3):311–343.
- [MacLane, 1998] MacLane, S. (1998). *Categories for the Working Mathematician*. Graduate Texts in Mathematics. Springer - Verlag. Second Edition.
- [Melliès, 2003] Melliès, P.-A. (2003). Categorical models of linear logic revisited. Prépublication de l'équipe PPS.
- [Meyer, 1982] Meyer, A. (1982). What is a model of the lambda calculus? *Information and Computation*, 52(1):87–122.

- [Meyer, 1988] Meyer, A. (1988). Semantical paradigms. In *Proceedings of the Third Annual Symposium on Logic in Computer Science - LICS*, pages 236–253. Notes for an invited lecture with two appendices by Stavros Cosmadakis.
- [Milner, 1889] Milner, R. (1889). *Communication and Concurrency*. Prentice Hall.
- [Milner, 1977] Milner, R. (1977). Fully abstract models of typed lambda-calculus. *Theoretical Computer Science*, 4:1–22.
- [Milner, 1990] Milner, R. (1990). Functions as processes. In Paterson, M., editor, *Automata, Languages and Programming, 17th International Colloquium, ICALP90, Warwick University, England, July 16-20, 1990, Proceedings*, volume 443 of *Lecture Notes in Computer Science*, pages 167–180. Springer.
- [Milner, 1992] Milner, R. (1992). Functions as processes. *Mathematical Structures in Computer Science*, 2(2):119–141. Previous version as Rapport de Recherche 1154, INRIA Sophia-Antipolis, 1990, and in *Proceedings of ICALP '91*, LNCS 443.
- [Milner, 1993] Milner, R. (1993). The polyadic π -calculus: A tutorial. In Bauer, F. L., Brauer, W., and Schwichtenberg, H., editors, *Logic and Algebra of Specification, Proceedings of International NATO Summer School (Marktobendorf, Germany, 1991)*, volume 94. Springer-Verlag. Available as Technical Report ECS-LFCS-91-180, University of Edinburgh, October 1991.
- [Nielsen et al., 1981] Nielsen, M., Plotkin, G., and Winskel, G. (1981). Petri nets, event structures and domains. *Theoretical Computer Science*, 13(1):85–108.
- [Ong, 1988] Ong, C.-H. L. (1988). Fully abstract models of the lazy lambda calculus. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science - FOCS'88*, pages 368–376, White Plains, New York. IEEE Computer Society Press.
- [Paolini, 2006] Paolini, L. (2006). A stable programming language. *Information and Computation*, 204(3):339–375.
- [Paolini and Piccolo, 2008] Paolini, L. and Piccolo, M. (2008). Semantically linear programming languages. In Antoy, S. and Albert, E., editors, *Proceedings of the 10th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, July 15-17, 2008, Valencia, Spain*, pages 97–107. ACM.
- [Paolini and Piccolo, 2009] Paolini, L. and Piccolo, M. (2009). A process-model for linear programs. In Berardi, S., Damiani, F., and de'Liguoro, U., editors, *Types for Proofs and Programs, International Conference, TYPES 2008, Torino, Italy, March 26-29, 2008, Revised Selected Papers*, volume 5497 of *Lecture Notes in Computer Science*, pages 289–305. Springer.
- [Paolini et al., 2009] Paolini, L., Piccolo, M., and Ronchi Della Rocca, S. (2009). Logical semantics for stability. In Abramsky, S., Mislove, M. W., and Palamidessi, C.,

- editors, *Proceedings of the 25th Conference on Mathematical Foundations of Programming Semantics (MFPS 2009)*, volume 249 of *Electronic Notes in Theoretical Computer Science*, pages 429–449. Elsevier.
- [Paré and Román, 1988] Paré, R. and Román, L. (1988). Monoidal categories with natural numbers object. *Studia Logica*, 48(3):361–376.
- [Parrow and Victor, 1998] Parrow, J. and Victor, B. (1998). The fusion calculus: Expressiveness and symmetry in mobile processes. In *Thirteenth Annual Symposium on Logic in Computer Science (LICS) (Indiana)*, pages 176–185. IEEE Computer Society Press, Computer Society Press.
- [Petersen et al., 2003] Petersen, L., Harper, R., Crary, K., and Pfenning, F. (2003). A type theory for memory allocation and data layout. In *The 30th SIGPLAN-SIGACT Symposium on Principles of Programming Languages, New Orleans, Louisiana, January 15-17, 2003*. *ACM SIGPLAN Notices* 38(1), January, pages 172–184.
- [Piccolo, 2006] Piccolo, M. (2006). Strutture ad eventi e strategie: un ponte fra teoria della concorrenza e semantica dei giochi. Tesi di laurea in informatica, Università degli Studi di Padova.
- [Platek, 1966] Platek, R. A. (1966). *Foundations of Recursion Theory*. PhD thesis, Stanford University, Stanford, USA.
- [Plotkin, 1975] Plotkin, G. D. (1975). Call-by-name, call-by-value and the λ -calculus. *Theoretical Computer Science*, 1:125–159.
- [Plotkin, 1977] Plotkin, G. D. (1977). LCF considered as a programming language. *Theoretical Computer Science*, 5:225–255.
- [Plotkin, 1993] Plotkin, G. D. (1993). Set-theoretical and other elementary models of the λ -calculus. In Dezani-Ciancaglini, M., Ronchi Della Rocca, S., and Venturini-Zilli, M., editors, *Theoretical Computer Science*, volume 121(1-2), pages 351–409. Logic, Semantics and Theory of Programming. *A Collection of Contributions in Honour of Corrado Bhm on the Occasion of his 70th Birthday*.
- [Pravato et al., 1999] Pravato, A., Ronchi Della Rocca, S., and Roversi, L. (1999). The call by value λ -calculus: a semantic investigation. *Mathematical Structures in Computer Science*, 9(5):617–650.
- [Román, 1989] Román, L. (1989). Cartesian categories with natural numbers object. *Journal of Pure and Applied Algebra*, 58(3):267–278.
- [Ronchi Della Rocca and Paolini, 2004] Ronchi Della Rocca, S. and Paolini, L. (2004). *The Parametric λ -Calculus: a Metamodel for Computation*. Texts in Theoretical Computer Science: An EATCS Series. Springer-Verlag, Berlin.

- [Sangiorgi, 1994] Sangiorgi, D. (1994). An investigation into functions as processes. In Brookes, S. D., Main, M. G., Melton, A., Mislove, M. W., and Schmidt, D. A., editors, *Mathematical Foundations of Programming Semantics, 9th International Conference, New Orleans, LA, USA, April 7-10, 1993, Proceedings*, volume 802 of *Lecture Notes in Computer Science*, pages 143–159. Springer.
- [Sangiorgi, 1995] Sangiorgi, D. (1995). Internal mobility and agent-passing calculi. In Fülöp, Z. and Gécseg, F., editors, *Proceedings of Automata, Languages and Programming, 22nd International Colloquium, ICALP95, Szeged, Hungary, July 10-14, 1995*, volume 944 of *Lecture Notes in Computer Science*, pages 672–683. Springer-Verlag.
- [Sangiorgi and Walker, 2001] Sangiorgi, D. and Walker, D. (2001). *The π -calculus: a theory of mobile processes*. Cambridge University Press, Cambridge.
- [Scott, 1972] Scott, D. S. (1972). Continuous lattices. In Lawvere, F. W., editor, *Toposes, Algebraic Geometry, and Logic*, volume 274 of *Lecture Notes in Mathematics*, pages 97–136. Springer-Verlag, Berlin.
- [Scott, 1993] Scott, D. S. (1993). A type-theoretical alternative to ISWIM, CUCH, OWHY. *Theoretical Computer Science*, 121(1–2):411–440. A Collection of Contributions in Honour of Corrado Böhm on the Occasion of his 70th Birthday. This paper widely circulated in unpublished form since 1969.
- [Turner and Wadler, 1999] Turner, D. N. and Wadler, P. (1999). Operational interpretations of linear logic. *Theoretical Computer Science*, 227(1–2):231–248.
- [Varacca and Yoshida, 2006] Varacca, D. and Yoshida, N. (2006). Typed event structures and the π -calculus: Extended abstract. In *Proceedings of the 22nd Annual Conference on Mathematical Foundations of Programming Semantics*, volume 158 of *Electronic Notes in Theoretical Computer Science*, pages 373–397.
- [Winskel, 1982] Winskel, G. (1982). Event structure semantics for ccs and related languages. In Nielsen, M. and Schmidt, E. M., editors, *Automata, Languages and Programming, 9th Colloquium, Aarhus, Denmark, July 12-16, 1982, Proceedings*, volume 140 of *Lecture Notes in Computer Science*, pages 561–576. Springer.
- [Winskel, 1986] Winskel, G. (1986). Event structures. In Brauer, W., Reisig, W., and Rozenberg, G., editors, *Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986, Part II, Proceedings of an Advanced Course, Bad Honnef, 8.-19. September 1986*, volume 255 of *Lecture Notes in Computer Science*, pages 325–392. Springer.
- [Winskel and Nielsen, 1995] Winskel, G. and Nielsen, M. (1995). Models for concurrency. In *Handbook of Logic in Computer Science*, pages 1–148. Oxford University Press.
- [Yoshida et al., 2004] Yoshida, N., Berger, M., and Honda, K. (2004). Strong normalisation in the PI-calculus. *Information and Computation*, 191(2):145–202.

Index

- T*-coalgebra, 24
 - free, 24
- \exists , 43
- GLin*, 90
- gor*, 46
- StPCF*, 65
 - Operational equivalence, 68
- Sℓλ* category, 57
- Sℓλ*-calculus, 51, 52
 - categorical model, 58
 - reduction rules, 53
- $\tilde{\gamma}\alpha x$, 128
- $G\ddot{or}$, 81
- λ -calculus, 31
 - Categorical Model, 35
- λ -model, 163
 - lazy stable, 173
 - linear, 172
- linProc*, 98
 - Observational equivalence, 106
 - Reduction rules, 104
 - Structural congruence, 103
- Bcdom**, 41
- LinCoh**, 44, 73
- Lud**, 129
- StabCoh**, 45
- StrictBcdom**, 69
- ∇ -basis, 165
- pif*, 43
- strict?*, 46
- which?*, 86
- *-Autonomous Category, 29

- Action, 120
 - daimon, 120
 - proper, 120

- Action modality, 100, 141
- Action types, 101
- Action-environment, 143
- Adjonction, 22
 - co-unit, 22
 - unit, 22
- Arena, 120

- Base, 120
 - negative, 120
 - positive, 120

- C-fix category, 37
- Cartesian Category, 24
- Cartesian Closed Category, 25
 - Enough Points, 26
 - Well-pointed, 26
- Categorical Co-product, 25
- Categorical model, 35
- Categorical Product, 25
- Category, 17
 - Functor, 21
 - Large, 18
 - Locally small, 18
 - small, 18
- Channel type, 100
- Clique model, 171
 - lazy, 171
- Co-cartesian Category, 25
- Co-monad, 23
 - Co-multiplication, 23
 - Co-unit, 23
 - Monoidal, 29
- Co-Monoid, 27
 - Commutative, 28
- Coherence Space, 43

- Compact Closed Category, 30
- Delocation, 128
- Dual Category, 18
- Eilenberg-Moore Category, 24
- Event structure, 116
 - arborescent, 116
 - cell, 119
 - co-product, 117
 - configuration, 116
 - conflict free, 119
 - confusion free, 119
 - disjoint union, 118
 - immediate conflict, 116
 - labelled, 116
 - morphism, 117
 - prefix order, 118
 - prefixing, 118
 - pull-back, 133
 - relabelling, 118
 - restriction, 118
- Exponential object of numerals, 57
- Faithful translation, 97, 107
- Focus, 120
- Functor, 19
 - bi, 20
 - Contravariant, 20
 - Covariant, 19
 - Faithful, 20
 - Full, 20
 - Hom, 20
 - Inclusion, 20
 - Monoidal, 28
 - Strong, 29
 - Symmetric, 29
- Ground-driven sum, 99
- Ground-input, 99
- Ground-output, 99
- High order interface, 150
- Interface, 141
- Iso-coherent map, 172
- Isomorphism, 19
 - Natural, 21
- Kleisli Category, 23
- Lawvere-Lambek Isomorphism, 37
- Lazy stable intersection type assignment system, 166
- Left/Right Adjoint, 21
 - Universal Property, 22
- Legality, 167
- Linear π -calculus, 148
 - finitary, 139
 - reduction, 140
 - structural congruence, 140
 - recursive definition, 149
 - Testing equivalence, 143
- Linear category, 54
- Loci-environment, 143
- Locus, 120
 - disjoint, 120
 - sub, 120
- Materiality, 132
- Monad, 22
 - Multiplication, 22
 - Unit, 22
- Monoidal Category, 27
 - Symmetric, 28
- Monoidal object of numerals, 54
- Morphism, 17
 - Co-Monoid, 28
 - coalgebra, 24
 - Epi, 19
 - Identity, 17
 - immersion, 172
 - Iso, *see* Isomorphism
 - Mono, 19
 - projection, 172
- Multidesign, 121
 - prefixing, 122
 - stable order, 122
 - sum, 122
- Natural number object, 55

- Natural Transformation, 21
 - Monoidal, 29
- Normalisation, 127
- Object, 17
 - Exponential, 26
 - Initial, 25
 - reflexive, 172
 - Terminal, 25
- Parallel composition, 123, 124
- Parallel or, 42
- Partial type structure, 94
 - extensional, 94
 - effective, 94
 - simulation, 94
- PCF, 33
 - Operational Equivalence, 34
- Pivot, 165
- Process environment, 101, 150
- Product Category, 18
- Pull-back, 26

- Retraction pair, 19, 172

- Scott domain, 40
- Second Order Gustave Or,
 - see Gor^2_{81}
- Separation, 130
- Sorting, 100
- Stable intersection type assignment system, 165
- Stable type system, 165
 - lazy, 165
- Stable type theory, 164
- Strong stability, 82
- Subcategory, 20
 - Full, 20
- Symmetric Monoidal Closed Category, 29
- Synchronisation morphism, 123

- Tensor Product, 27
 - Associativity law, 27
 - Left identity law, 27
 - Right identity law, 27
- Types, 31, 165
 - row, 164
- Typing clique, 169
- Typing relation, 164
 - coherence, 164
 - incoherence, 164
 - strict coherence, 164
 - strict incoherence, 164