BRICS DS-03-13 M. Nygaard: Domain Theory for Concurrency

BRICS

Basic Research in Computer Science

Domain Theory for Concurrency

Mikkel Nygaard

BRICS Dissertation Series **DS-03-13**

ISSN 1396-7002 **November 2003**

See back inner page for a list of recent BRICS Dissertation Series publications. Copies may be obtained by contacting:

BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK–8000 Aarhus C
Denmark
Telephone: +45 8942 3360
Telefax: +45 8942 3255
Internet: BRICS@brics.dk

BRICS publications are in general accessible through the World Wide Web and anonymous FTP through these URLs:

http://www.brics.dk
ftp://ftp.brics.dk
This document in subdirectory DS/03/13/

Domain Theory for Concurrency

Mikkel Nygaard

PhD Dissertation

Department of Computer Science
University of Aarhus
Denmark

Domain Theory for Concurrency

A Dissertation
Presented to the Faculty of Science
of the University of Aarhus
in Partial Fulfilment of the Requirements for the
PhD Degree

by
Mikkel Nygaard
July 31st, 2003
(Revised on December 4th, 2003)

Abstract

Concurrent computation can be given an abstract mathematical treatment very similar to that provided for sequential computation by domain theory and denotational semantics of Scott and Strachey.

A simple domain theory for concurrency is presented. Based on a categorical model of linear logic and associated comonads, it highlights the role of linearity in concurrent computation. Two choices of comonad yield two expressive metalanguages for higher-order processes, both arising from canonical constructions in the model. Their denotational semantics are fully abstract with respect to contextual equivalence.

One language, called HOPLA for Higher-Order Process LAnguage, derives from an exponential of linear logic. It can be viewed as an extension of the simply-typed lambda calculus with CCS-like nondeterministic sum and prefix operations, in which types express the form of computation path of which a process is capable. HOPLA can directly encode calculi like CCS, CCS with process passing, and mobile ambients with public names, and it can be given a straightforward operational semantics supporting a standard bisimulation congruence. The denotational and operational semantics are related with simple proofs of soundness and adequacy. Full abstraction implies that contextual equivalence coincides with logical equivalence for a fragment of Hennessy-Milner logic, linking up with simulation equivalence.

The other language is called Affine HOPLA and is based on a weakening comonad that yields a model of affine-linear logic. This language adds to HOPLA an interesting tensor operation at the price of linearity constraints on the occurrences of variables. The tensor can be understood as a juxtaposition of independent processes, and allows Affine HOPLA to encode processes of the kind found in treatments of nondeterministic dataflow.

The domain theory can be generalised to presheaf models, providing a more refined treatment of nondeterministic branching and supporting notions of bisimulation. The operational semantics for HOPLA is guided by the idea that derivations of transitions in the operational semantics should correspond to elements of the presheaf denotations. Similar guidelines lead to an operational semantics for the first-order fragment of Affine HOPLA. An extension of the operational semantics to the full language is based on a stable denotational semantics which associates to each computation the minimal input necessary for it. Such a semantics is provided, based on event structures; it agrees with the presheaf semantics at first order and exposes the tensor operation as a simple parallel composition of event structures.

The categorical model obtained from presheaves is very rich in structure and points towards more expressive languages than HOPLA and Affine HOPLA—in particular concerning extensions to cover independence models. The thesis concludes with a discussion of related work towards a fully fledged domain theory for concurrency.

Acknowledgments

Thanks to

> my supervisor, Glynn Winskel, for four years of joint work under his expert leadership—and for introducing me to some of the good things in life besides research, like Jalfrezi and Habit Ale;
>
> my committee members Pierre-Louis Curien and Guy McCusker for their thoughtful and detailed comments, corrections and suggestions for improving the thesis;
>
> Pino Rosolini and the people at DISI, University of Genoa, for their hospitality during my stay there and for helpful comments on my work—and especially to Matías Menni for his friendship;
>
> Marcelo Fiore for his insightful suggestions at my Part A exam;
>
> Erik Meineche Schmidt and Mogens Nielsen for their early encouragement;
>
> the staff and students at Daimi/BRICS for creating a stimulating working environment;
>
> "Læsegruppen" and "Frokostklubben" for forcing me to have a life besides my studies (or, at least for trying);
>
> my family for boldly asking questions about my research even though my answers were often incomprehensible.

Last, but certainly not least, I wish to thank my wife Mette for her support, her patience, and her unwavering faith in me.

Mikkel Nygaard
Århus, Denmark
December 2003

Contents

Chapter 1

Introduction

Theories of *sequential* computation concern transformations of input to output following a well-defined sequence of basic computational steps. Sequential programs implement mathematical functions mapping from the set of possible inputs to the set of possible outputs. The class of functions arising in this way is very well understood. Even before the first computer was built, the work of Church, Kleene, and Turing had established the notion of computable function, captured by the untyped lambda calculus, recursive functions, and Turing machines in unison [21, 45, 89].

While these basic models give little guidance on how to design and reason about programming languages, domain theory and denotational semantics of Scott and Strachey provide a global mathematical setting for sequential computation, building on top of the foundational theories [82]. It places programming languages in connection with each other; connects with the mathematical worlds of algebra, topology, and logic; and inspires programming languages, type disciplines, and methods of reasoning.

However, the majority of real computer systems are not merely computing output from input. Rather, their main purpose is to be in ongoing interaction with the environment, and the central aspect of their behaviour is therefore their changing patterns of interaction over time. So the input/output paradigm of sequential computation is not immediately applicable.

Theories of *concurrent* computation concern the behaviour of systems of communicating, autonomous processes. In contrast to sequential computation, there is no class of "computable processes" with universal status, and the global mathematical guidance provided by a domain theory is missing. As a consequence, theories of concurrency form a rather fragmented picture.

The point of the thesis is to show that this state of affairs need not persist. We develop a simple domain theory for processes. It inspires process programming languages whose denotational semantics will map programs to mathematical functions *on processes*, thus combining the input/output paradigm with the notion of a process interacting with its environment.

Section 1.1 recalls the prominent features of Scott and Strachey's approach, focusing on the structure it provides to theories of sequential computation. This is contrasted with the situation in concurrent computation in Section 1.2 where seemingly incompatible theories are abundant. The work described in the thesis is part of a broader programme of research towards a unifying theory. This effort is outlined in Section 1.3 which also gives an overview of the thesis.

1.1 Sequential Computation

The purpose of this section is to establish some notation and give an overview of those concepts from sequential computation that we wish to advance to a concurrent setting. We'll take for granted a basic understanding of the untyped lambda calculus including concepts like free and bound variables, substitution, α, β, η-equivalence, and β-reduction (see [71] for an elementary account or [6] for more information).

1.1.1 Denotational Semantics

Following Strachey [85] the meaning, or *denotation*, of a program construct is a representation of its contribution to overall behaviour, specified in the untyped lambda calculus. For instance, commands in imperative programming languages may be represented as partial mappings from states to states, a state being a function s from variable names (or locations) to values. Names and simple values may be represented using Church numerals or similar encodings which allow equality tests, but it is customary to leave out such detail and write e.g.

$$[\![\mathit{skip}]\!] = \lambda s.s \quad \text{and} \quad [\![i := 7]\!] = \lambda s.\lambda x.[x = i \Rightarrow 7, s\ x] \ . \tag{1.1}$$

The notation $[b \Rightarrow t_1, t_2]$ stands for the encoding of a conditional which equals (reduces to) t_1 if b is true and t_2 otherwise. We'll use similar notation for similar kinds of tests throughout the thesis.

A hallmark of denotational semantics is *compositionality*, meaning that the denotation of a compound program construct is given inductively in terms of the denotations of its constituents. As an example, the meaning of a command sequence $c\,;c'$ can be given as the functional composition of the meanings of c and c', i.e.

$$[\![c\,; c']\!] = [\![c']\!] \circ [\![c]\!] = \lambda s.[\![c']\!]([\![c]\!] s) \ . \tag{1.2}$$

Consider now a loop *while* b *do* c with b denoting a map $[\![b]\!]$ from states to booleans. The loop should have the same semantics as its unfolding,

$$[\![\mathit{while}\ b\ \mathit{do}\ c]\!] = [\![\mathit{if}\ b\ \mathit{then}\ (c\,; \mathit{while}\ b\ \mathit{do}\ c)\ \mathit{else}\ \mathit{skip}]\!] \ . \tag{1.3}$$

Therefore, the denotation w of the loop should satisfy

$$w = fw \quad \text{where } f = \lambda c'.\lambda s.[[\![b]\!]s \Rightarrow c'([\![c]\!]s), s]. \tag{1.4}$$

This clearly does not work as a definition because w recurs on the right-hand side of the equation $w = fw$. But the untyped lambda calculus has a fixed-point operator

$$Y \equiv_{\mathrm{def}} \lambda f.(\lambda x.f\ x\ x)\ (\lambda x.f\ x\ x)\ \ , \tag{1.5}$$

satisfying $Yf = f(Yf)$ for all lambda terms f. We can then define $w =_{\mathrm{def}} Yf$, and so $w = Yf = f(Yf) = fw$ as wanted. Denotations of other kinds of infinite computational behaviour, like recursive procedures, can be given compositionally in a similar way.

1.1.2 Domains

Of course, the above is really just a translation of one programming language, a command language, into another, the untyped lambda calculus. Mathematical foundations are provided by Scott's famous construction of a model for the untyped lambda calculus [80] using certain ordered structures, called *domains*. Domains come in many variants, see [3]. Here, a domain $\mathbb{D}$ will be a complete partial order (cpo) with a least element, written $\perp$. Completeness of $\mathbb{D}$ means that any ω-*chain*, i.e. any ordered chain

$$d_0 \leq_{\mathbb{D}} d_1 \leq_{\mathbb{D}} \cdots \leq_{\mathbb{D}} d_n \leq_{\mathbb{D}} \cdots \tag{1.6}$$

in $\mathbb{D}$ indexed by the natural numbers $\omega = \{0, 1, 2, \ldots\}$, has a least upper bound in $\mathbb{D}$, written $\bigsqcup_{n\in\omega} d_n$. A monotone map $f : \mathbb{D} \to \mathbb{E}$ between domains is said to be *continuous* if it preserves the completeness structure

$$\textstyle\bigsqcup_{n\in\omega} fd_n = f(\bigsqcup_{n\in\omega} d_n) \quad \text{for all } \omega\text{-chains } (d_n)_{n\in\omega} \text{ in } \mathbb{D}. \tag{1.7}$$

Continuous endofunctions $f : \mathbb{D} \to \mathbb{D}$ on domains $\mathbb{D}$ are of special interest because they admit least fixed-points, obtained as $\mathit{fix}_{\mathbb{D}} f = \bigsqcup_{n\in\omega} (f^n \perp)$. This gives an alternative to the Y combinator of the untyped lambda calculus, in the form of a typed fixed-point operator $\mathit{fix}_{\mathbb{D}} : (\mathbb{D} \to \mathbb{D}) \to \mathbb{D}$ for each domain $\mathbb{D}$. Here, $\mathbb{D} \to \mathbb{D}$ is the domain of continuous maps $f : \mathbb{D} \to \mathbb{D}$ under pointwise ordering, i.e.

$$f \leq_{\mathbb{D}\to\mathbb{D}} g \iff \forall d \in \mathbb{D}.\ fd \leq_{\mathbb{D}} gd\ \ . \tag{1.8}$$

Using domains in denotational semantics, program constructs denote elements of domains. The construction of a model for the untyped lambda calculus then involves finding a domain $\mathbb{D}$ in which to interpret lambda terms. Considering the self-application xx as in the Y-combinator, note that if the second occurrence of x has type $\mathbb{D}$, and the whole term xx has type $\mathbb{D}$, then the first occurrence of x must be assigned both the type $\mathbb{D}$ and the type $\mathbb{D} \to \mathbb{D}$. Thus, we need at least an isomorphism $\mathbb{D} \cong (\mathbb{D} \to \mathbb{D})$, and Scott's fundamental contribution was to construct a nontrivial solution to this recursive domain equation.

Some intuition about domains can be obtained by considering the commands of the previous section. They may be mapped to elements of the domain $\mathbb{C} =_{\mathrm{def}} \mathbb{S} \to \mathbb{S}$ where $\mathbb{S}$ is a domain representing states. It is well-known that some commands do not terminate when run in certain states, and their denotation will then map such states to the least element $\perp \in \mathbb{S}$, thought of as an "undefined" state, or rather, no knowledge about a state. The ordering of $\mathbb{C}$ can then be understood intuitively as one of *information*: a map $\mathbb{S} \to \mathbb{S}$ can be considered more informative than another if it yields more informative output on each input. The denotation of the while loop considered in the previous section is obtained as the least fixed-point $\mathit{fix}_{\mathbb{C}}\, f$ of the continuous map $f : \mathbb{C} \to \mathbb{C}$ defined in (1.4). The approximation $f^n\perp$ of the fixed-point is the denotation of the n'th unfolding of the loop with zero unfoldings represented by $\perp \in \mathbb{C}$, the everywhere $\perp$ map. Thus, the ω-chain $(f^n\perp)_{n\in\omega}$ is intuitively a chain of increasing information about the denotation of the while loop.

1.1.3 Universal Constructions

Scott's construction gave birth to domain theory which in fact does a lot more than provide a model for the untyped lambda calculus. Indeed, it forms an informative mathematical world in which one can give denotational semantics to programming languages by mapping program constructs to elements of domains, while the domains themselves function as "types" of constructs. It is therefore important to study the possible constructions on domains, and it is customary to employ *category theory* for this. Category theory is an effective tool for investigating structure, guided by the notion of *universal construction*. Such constructions are characterised by their "behaviour" rather than by their "implementation" in a way similar to abstract data types. Introductions to category theory are [70, 9] while the classic reference remains Mac Lane's book [51].

For simplicity we'll consider just cpos and continuous maps between them. Continuous functions between cpos compose as functions and the identity function $1_{\mathbb{C}} : \mathbb{C} \to \mathbb{C}$ is continuous for any cpo $\mathbb{C}$. Thus, cpos and continuous functions form a category, which we'll call **Cpo**.

This category has *products* given at objects $\mathbb{A}$ and $\mathbb{B}$ by the cartesian product $\mathbb{A}\times\mathbb{B}$ of posets with associated projections $\mathit{fst} : \mathbb{A}\times\mathbb{B} \to \mathbb{A}$ and $\mathit{snd} : \mathbb{A} \times \mathbb{B} \to \mathbb{B}$. This "implements" the following "behaviour", characterising the implementation up to isomorphism: any pair of maps $f : \mathbb{C} \to \mathbb{A}$ and $g : \mathbb{C} \to \mathbb{B}$ from some object $\mathbb{C}$ in **Cpo** can be tupled together to form a unique map $\langle f, g\rangle : \mathbb{C} \to \mathbb{A} \times \mathbb{B}$ with the property that $\mathit{fst} \circ \langle f, g\rangle = f$ and

$\mathit{snd} \circ \langle f, g \rangle = g$. Given $h : \mathbb{A} \to \mathbb{A}'$ and $k : \mathbb{B} \to \mathbb{B}'$, we'll write $h \times k$ for the unique map $\langle h \circ \mathit{fst}, k \circ \mathit{snd} \rangle : \mathbb{A} \times \mathbb{B} \to \mathbb{A}' \times \mathbb{B}'$. The empty product is given by the singleton cpo $\mathbb{1} = \{\bot\}$ and, as the *terminal object*, is associated with unique maps $\bot_{\mathbb{C}} : \mathbb{C} \to \mathbb{1}$, constantly $\bot$, for any cpo $\mathbb{C}$.

Exponentials are obtained as the poset of continuous maps $\mathbb{A} \to \mathbb{B}$, ordered pointwise as defined by (1.8). Associated to this object is a map $\mathit{app} : (\mathbb{A} \to \mathbb{B}) \times \mathbb{A} \to \mathbb{B}$, given as $\mathit{app}(f, a) =_{\mathrm{def}} f a$. The construction satisfies the universal property that given any map $g : \mathbb{C} \times \mathbb{A} \to \mathbb{B}$ in **Cpo**, there is a unique map $\mathit{curry}\, g : \mathbb{C} \to (\mathbb{A} \to \mathbb{B})$ such that $\mathit{app} \circ (\mathit{curry}\, g \times 1_{\mathbb{A}}) = g$. Since **Cpo** has finite products and exponentials, it is *cartesian closed* and thereby a model of intuitionistic propositional logic (see [46] or [86] for more information on the correspondence between intuitionistic logic and categorical structure).

The category **Cpo** also has *coproducts*, given at objects $\mathbb{A}$ and $\mathbb{B}$ by the disjoint juxtaposition of cpos, $\mathbb{A} + \mathbb{B}$. There are obvious injection maps $\mathit{inl} : \mathbb{A} \to \mathbb{A} + \mathbb{B}$ and $\mathit{inr} : \mathbb{B} \to \mathbb{A} + \mathbb{B}$. Given maps $f : \mathbb{A} \to \mathbb{C}$ and $g : \mathbb{B} \to \mathbb{C}$ in **Cpo** there is a unique map $[f, g] : \mathbb{A} + \mathbb{B} \to \mathbb{C}$ such that $[f, g] \circ \mathit{inl} = f$ and $[f, g] \circ \mathit{inr} = g$. The empty coproduct, i.e. *initial object*, is given by the empty cpo $\mathbb{O}$. The empty map $\varnothing_{\mathbb{C}} : \mathbb{O} \to \mathbb{C}$ is the unique map from $\mathbb{O}$ to any cpo $\mathbb{C}$.

1.1.4 Simply-Typed Lambda Calculus

We now illustrate how one can use universal constructions to define the denotational semantics of a programming language. The *simply-typed lambda calculus* is directly suggested by the constructions available in **Cpo**. Types are given by the grammar

$$\mathbb{A}, \mathbb{B} ::= \mathbb{A} \to \mathbb{B} \mid \mathbb{1} \mid \mathbb{A} \times \mathbb{B} \mid \mathbb{A} + \mathbb{B} \ . \tag{1.9}$$

—and are interpreted as objects of **Cpo** in the obvious way. The raw syntax of terms is given by

$$\begin{array}{rll} t, u ::= & x, y, z, \ldots & \text{(variables)} \\ & \mid \lambda x.t \mid t\ u & \text{(abstraction and application)} \\ & \mid \bot \mid (t, u) \mid \mathit{fst}\, t \mid \mathit{snd}\, t & \text{(unit, pairing and projections)} \\ & \mid \mathit{inl}\, t \mid \mathit{inr}\, t \mid & \text{(injections)} \\ & \mid [u > \mathit{inl}\, x \Rightarrow t_1, \mathit{inr}\, x \Rightarrow t_2] & \text{(match for sums)} \end{array} \tag{1.10}$$

The variables x in a match term $[u > \mathit{inl}\, x \Rightarrow t_1, \mathit{inr}\, x \Rightarrow t_2]$ are binding occurrences and so bind later occurrences of the variable in the bodies t_1 and t_2, respectively.

Assume that the variables $x_1, \ldots, x_k$ are distinct. A syntactic judgement $x_1 : \mathbb{A}_1, \ldots, x_k : \mathbb{A}_k \vdash t : \mathbb{B}$ stands for a map

$$[\![x_1 : \mathbb{A}_1, \ldots, x_k : \mathbb{A}_k \vdash t : \mathbb{B}]\!] : \mathbb{A}_1 \times \cdots \times \mathbb{A}_k \to \mathbb{B} \tag{1.11}$$

in **Cpo**. We'll write Γ, or Λ, for an environment list $x_1 : \mathbb{A}_1, \ldots, x_k : \mathbb{A}_k$ and often abbreviate the denotation to $\mathbb{A}_1 \times \cdots \times \mathbb{A}_k \xrightarrow{t} \mathbb{B}$, or even $\Gamma \xrightarrow{t} \mathbb{B}$. When Γ is empty, the corresponding product is the singleton cpo $\mathbb{1}$.

The term-formation rules are displayed below alongside their interpretations as constructors on maps of **Cpo**, taking the maps denoted by the premises to that denoted by the conclusion (cf. [12]). We assume that the variables in any environment list are distinct.

Structural rules. The rules handling environment lists are given as follows:

$$\frac{}{x : \mathbb{A} \vdash x : \mathbb{A}} \qquad \frac{}{\mathbb{A} \xrightarrow{1_{\mathbb{A}}} \mathbb{A}} \tag{1.12}$$

$$\frac{\Gamma \vdash t : \mathbb{B}}{\Gamma, x : \mathbb{A} \vdash t : \mathbb{B}} \qquad \frac{\Gamma \xrightarrow{t} \mathbb{B}}{\Gamma \times \mathbb{A} \xrightarrow{1_\Gamma \times \perp_{\mathbb{A}}} \Gamma \times \mathbb{1} \xrightarrow{r_{\mathbb{B}}} \Gamma \xrightarrow{t} \mathbb{B}} \tag{1.13}$$

$$\frac{\Gamma, y : \mathbb{B}, x : \mathbb{A}, \Lambda \vdash t : \mathbb{C}}{\Gamma, x : \mathbb{A}, y : \mathbb{B}, \Lambda \vdash t : \mathbb{C}} \qquad \frac{\Gamma \times \mathbb{B} \times \mathbb{A} \times \Lambda \xrightarrow{t} \mathbb{C}}{\Gamma \times \mathbb{A} \times \mathbb{B} \times \Lambda \xrightarrow{t \circ (1_\Gamma \times s_{\mathbb{A},\mathbb{B}} \times 1_\Lambda)} \mathbb{C}} \tag{1.14}$$

$$\frac{\Gamma, x : \mathbb{A}, y : \mathbb{A} \vdash t : \mathbb{B}}{\Gamma, z : \mathbb{A} \vdash t[z/x, z/y] : \mathbb{B}} \qquad \frac{\Gamma \times \mathbb{A} \times \mathbb{A} \xrightarrow{t} \mathbb{B}}{\Gamma \times \mathbb{A} \xrightarrow{1_\Gamma \times \Delta_{\mathbb{A}}} \Gamma \times \mathbb{A} \times \mathbb{A} \xrightarrow{t} \mathbb{B}} \tag{1.15}$$

The rules for weakening (1.13) and exchange (1.14) make use of the obvious isomorphisms $r_{\mathbb{B}} : \mathbb{B} \times \mathbb{1} \cong \mathbb{B}$ (right unit) and $s_{\mathbb{A},\mathbb{B}} : \mathbb{A} \times \mathbb{B} \cong \mathbb{B} \times \mathbb{A}$ (symmetry). In the rule for contraction (1.15), the variable z must be fresh; the map $\Delta_{\mathbb{A}}$ is the usual diagonal, given as $\langle 1_{\mathbb{A}}, 1_{\mathbb{A}} \rangle$.

Function space. Interpreted using exponentials of **Cpo**:

$$\frac{\Gamma, x : \mathbb{A} \vdash t : \mathbb{B}}{\Gamma \vdash \lambda x.t : \mathbb{A} \to \mathbb{B}} \qquad \frac{\Gamma \times \mathbb{A} \xrightarrow{t} \mathbb{B}}{\Gamma \xrightarrow{\mathit{curry}\, t} \mathbb{A} \to \mathbb{B}} \tag{1.16}$$

$$\frac{\Gamma \vdash t : \mathbb{A} \to \mathbb{B} \quad \Lambda \vdash u : \mathbb{A}}{\Gamma, \Lambda \vdash t\ u : \mathbb{B}} \qquad \frac{\Gamma \xrightarrow{t} \mathbb{A} \to \mathbb{B} \quad \Lambda \xrightarrow{u} \mathbb{A}}{\Gamma \times \Lambda \xrightarrow{t \times u} (\mathbb{A} \to \mathbb{B}) \times \mathbb{A} \xrightarrow{\mathit{app}} \mathbb{B}} \tag{1.17}$$

Products. Interpreted using products of **Cpo**:

$$\frac{}{\Gamma \vdash \perp : \mathbb{1}} \qquad \frac{}{\Gamma \xrightarrow{\perp_\Gamma} \mathbb{1}} \tag{1.18}$$

$$\frac{\Gamma \vdash t : \mathbb{A} \quad \Lambda \vdash u : \mathbb{B}}{\Gamma, \Lambda \vdash (t, u) : \mathbb{A} \times \mathbb{B}} \qquad \frac{\Gamma \xrightarrow{t} \mathbb{A} \quad \Lambda \xrightarrow{u} \mathbb{B}}{\Gamma \times \Lambda \xrightarrow{t \times u} \mathbb{A} \times \mathbb{B}} \tag{1.19}$$

$$\frac{\Gamma \vdash t : \mathbb{A} \times \mathbb{B}}{\Gamma \vdash \mathit{fst}\, t : \mathbb{A}} \qquad \frac{\Gamma \xrightarrow{t} \mathbb{A} \times \mathbb{B}}{\Gamma \xrightarrow{t} \mathbb{A} \times \mathbb{B} \xrightarrow{\mathit{fst}} \mathbb{A}} \tag{1.20}$$

$$\frac{\Gamma \vdash t : \mathbb{A} \times \mathbb{B}}{\Gamma \vdash \mathit{snd}\, t : \mathbb{B}} \qquad \frac{\Gamma \xrightarrow{t} \mathbb{A} \times \mathbb{B}}{\Gamma \xrightarrow{t} \mathbb{A} \times \mathbb{B} \xrightarrow{\mathit{snd}} \mathbb{B}} \tag{1.21}$$

Sums. Interpreted using coproducts of **Cpo**:

$$\frac{\Gamma \vdash t : \mathbb{A}}{\Gamma \vdash \mathit{inl}\, t : \mathbb{A} + \mathbb{B}} \quad \frac{\Gamma \xrightarrow{t} \mathbb{A}}{\Gamma \xrightarrow{t} \mathbb{A} \xrightarrow{\mathit{inl}} \mathbb{A} + \mathbb{B}} \tag{1.22}$$

$$\frac{\Gamma \vdash t : \mathbb{B}}{\Gamma \vdash \mathit{inr}\, t : \mathbb{A} + \mathbb{B}} \quad \frac{\Gamma \xrightarrow{t} \mathbb{B}}{\Gamma \xrightarrow{t} \mathbb{B} \xrightarrow{\mathit{inr}} \mathbb{A} + \mathbb{B}} \tag{1.23}$$

$$\frac{\Gamma, x : \mathbb{A} \vdash t_1 : \mathbb{C} \quad \Gamma, x : \mathbb{B} \vdash t_2 : \mathbb{C} \quad \Lambda \vdash u : \mathbb{A} + \mathbb{B}}{\Gamma, \Lambda \vdash [u > \mathit{inl}\, x \Rightarrow t_1, \mathit{inr}\, x \Rightarrow t_2] : \mathbb{C}}$$

$$\frac{\Gamma \times \mathbb{A} \xrightarrow{t_1} \mathbb{C} \quad \Gamma \times \mathbb{B} \xrightarrow{t_2} \mathbb{C} \quad \Lambda \xrightarrow{u} \mathbb{A} + \mathbb{B}}{\Gamma \times \Lambda \xrightarrow{1_\Gamma \times u} \Gamma \times (\mathbb{A} + \mathbb{B}) \xrightarrow{\mathit{dist}_{\Gamma,\mathbb{A},\mathbb{B}}} \Gamma \times \mathbb{A} + \Gamma \times \mathbb{B} \xrightarrow{[t_1, t_2]} \mathbb{C}} \tag{1.24}$$

In the last rule, the map $\mathit{dist}_{\Gamma,\mathbb{A},\mathbb{B}}$ is the isomorphism witnessing that products distribute over sums in **Cpo**.

There is analogy between type constructors $\rightarrow, \times, +$ and logical connectives $\rightarrow, \wedge, \vee$: typing derivations $\Gamma \vdash t : \mathbb{A}$ correspond to proofs of $\mathbb{A}$ from assumptions Γ in intuitionistic logic. This is the so-called *Curry-Howard isomorphism* (see [86], Ch. 2.4, or [46] for a full account).

By induction on the rules above, one can prove a *substitution lemma*, essentially saying that β-reduction is valid:

Lemma 1.1 (Substitution) Suppose $\Gamma, x : \mathbb{A} \vdash t : \mathbb{B}$ and $\Lambda \vdash u : \mathbb{A}$ with Γ and Λ disjoint. Then $\Gamma, \Lambda \vdash t[u/x] : \mathbb{B}$ with denotation given by the composition

$$\Gamma \times \Lambda \xrightarrow{1_\Gamma \times u} \Gamma \times \mathbb{A} \xrightarrow{t} \mathbb{B} \ . \tag{1.25}$$

Proposition 1.2 By the equational properties of the universal constructions and the substitution lemma we get the following "β-equivalences":

$$\begin{aligned}
[\![\Gamma \vdash (\lambda x.t)\ u : \mathbb{B}]\!] &= [\![\Gamma \vdash t[u/x] : \mathbb{B}]\!] \\
[\![\Gamma \vdash \mathit{fst}(t,u) : \mathbb{A}]\!] &= [\![\Gamma \vdash t : \mathbb{A}]\!] \\
[\![\Gamma \vdash \mathit{snd}(t,u) : \mathbb{B}]\!] &= [\![\Gamma \vdash u : \mathbb{B}]\!] \\
[\![\Gamma \vdash [\mathit{inl}\, u > \mathit{inl}\, x \Rightarrow t_1, \mathit{inr}\, x \Rightarrow t_2] : \mathbb{C}]\!] &= [\![\Gamma \vdash t_1[u/x] : \mathbb{C}]\!] \\
[\![\Gamma \vdash [\mathit{inr}\, u > \mathit{inl}\, x \Rightarrow t_1, \mathit{inr}\, x \Rightarrow t_2] : \mathbb{C}]\!] &= [\![\Gamma \vdash t_2[u/x] : \mathbb{C}]\!]
\end{aligned} \tag{1.26}$$

Proposition 1.3 By the uniqueness properties of the universal constructions we get the following "η-equivalences":

$$\begin{aligned}
[\![\Gamma \vdash \lambda x.(t\ x) : \mathbb{A} \rightarrow \mathbb{B}]\!] &= [\![\Gamma \vdash t : \mathbb{A} \rightarrow \mathbb{B}]\!] \\
[\![\Gamma \vdash \bot : \mathbb{1}]\!] &= [\![\Gamma \vdash t : \mathbb{1}]\!] \\
[\![\Gamma \vdash (\mathit{fst}\, t, \mathit{snd}\, t) : \mathbb{A} \times \mathbb{B}]\!] &= [\![\Gamma \vdash t : \mathbb{A} \times \mathbb{B}]\!] \\
[\![\Gamma \vdash [u > \mathit{inl}\, x \Rightarrow \mathit{inl}\, x, \mathit{inr}\, x \Rightarrow \mathit{inr}\, x] : \mathbb{A} + \mathbb{B}]\!] &= [\![\Gamma \vdash u : \mathbb{A} + \mathbb{B}]\!]
\end{aligned} \tag{1.27}$$

1.1.5 A Metalanguage

The denotational semantics of a given programming language can be specified along the lines above by assigning to every kind of program construct a suitable domain and a compositional mapping $[\![-]\!]$ sending individual constructs of that kind to elements of the chosen domain. Since most such domains tend to be function spaces, it quickly becomes cumbersome to check that $[\![-]\!]$ always maps to continuous functions. This problem can be addressed by providing a *metalanguage* for sequential programming. We show once and for all that the terms of the metalanguage denote continuous functions and so, by specifying the denotational semantics of each programming language of our interest in the metalanguage, we can rest assured that $[\![-]\!]$ maps every construct of function space type to a continuous map.

Based directly on canonical constructions, the simply typed lambda calculus would be a good candidate for a metalanguage, if not for the fact that it lacks fixed-point operators because types are interpreted as cpos rather than domains. However, any cpo $\mathbb{C}$ can be turned into domain $\mathbb{C}_\perp$ by adding a new least element $\perp$ below a copy of $\mathbb{C}$. Each element $c \in \mathbb{C}$ gives rise to an element $\lfloor c \rfloor$ of $\mathbb{C}_\perp$, distinct from $\perp$. The ordering of $\mathbb{C}_\perp$ is given by $\perp \leq_{\mathbb{C}_\perp} d$ for all $d \in \mathbb{C}_\perp$ and $\lfloor c \rfloor \leq_{\mathbb{C}_\perp} \lfloor c' \rfloor$ iff $c \leq_{\mathbb{C}} c'$. We write **Dom** for the full subcategory of **Cpo** given by those objects that are domains. **Dom** inherits cartesian closed structure from **Cpo**, and using a combination of the universal constructions above and lifting one can obtain a typed lambda calculus with a fixed-point operator at every type so that we can interpret the typing rule

$$\frac{\Gamma, x : \mathbb{A} \vdash t : \mathbb{A}}{\Gamma \vdash rec\, x.t : \mathbb{A}} \tag{1.28}$$

We'll not go into the details here. Chapter 11 of Winskel's book [95] gives more information on how different uses of lifting give rise to different calculi with different evaluation strategies like "call-by-value" or "call-by-name". Our metalanguage will employ a "lazy" strategy where subterms are only evaluated as needed.

In addition to recursion on terms we need to add recursive types to deal with infinite data types like the natural numbers. In the full metalanguage, types are given by the grammar

$$\mathbb{T} ::= \mathbb{T}_1 \to \mathbb{T}_2 \mid \mathbb{1} \mid \mathbb{T}_1 \times \mathbb{T}_2 \mid \mathbb{T}_1 + \mathbb{T}_2 \mid T \mid \mu T.\mathbb{T} \ . \tag{1.29}$$

The symbol T is drawn from a set of type variables used in defining recursive types; closed type expressions are interpreted as domains. The expression $\mu T.\mathbb{T}$ is interpreted as "the least" solution to the defining equation $T = \mathbb{T}$ in which the expression $\mathbb{T}$ may contain T. We shall confuse a closed expression for a domain with the domain itself. Recursive domain equations of this form can be solved using information systems [81, 48], characterising solutions (at least) up to an isomorphism $abs : \mathbb{T}[\mu T.\mathbb{T}/T] \cong \mu T.\mathbb{T}$ whose inverse we call *rep*. The associated typing rules are given by

$$\frac{\Gamma \vdash t : \mathbb{T}[\mu T.\mathbb{T}/T]}{\Gamma \vdash abs\, t : \mu T.\mathbb{T}} \qquad \frac{\Gamma \vdash t : \mu T.\mathbb{T}}{\Gamma \vdash rep\, t : \mathbb{T}[\mu T.\mathbb{T}/T]} \tag{1.30}$$

One can prove an analogue of the substitution lemma for the extended language and add equations

$$\begin{aligned} [\![\Gamma \vdash rec\, x.t : \mathbb{A}]\!] &= [\![\Gamma \vdash t[rec\, x.t/x] : \mathbb{A}]\!] \\ [\![\Gamma \vdash rep(abs\, t) : \mathbb{T}[\mu T.\mathbb{T}/T]]\!] &= [\![\Gamma \vdash t : \mathbb{T}[\mu T.\mathbb{T}/T]]\!] \end{aligned} \tag{1.31}$$

to Proposition 1.2. (On the other hand, items 1 and 3 of Proposition 1.3 will fail because of our lazy evaluation strategy.)

Example 1.4 The natural numbers can be defined by $\mathbb{N} \equiv_{\text{def}} \mu N.\mathbb{1} + N$. The constant zero and operations of successor and addition (curried) can be defined by

$$\begin{aligned} Zero &\equiv_{\text{def}} abs(inl\, \bot) \\ Succ &\equiv_{\text{def}} \lambda n.\, abs(inr\, n) \\ Add &\equiv_{\text{def}} rec\, f.\lambda n.\lambda m.[rep\, n > inl\, x \Rightarrow m, inr\, x \Rightarrow Succ(f\ x\ m)] \end{aligned} \tag{1.32}$$

We have $\vdash Zero : \mathbb{N}$ and $\vdash Succ : \mathbb{N} \to \mathbb{N}$, and $\vdash Add : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$.

In programming languages like ML or Haskell, data type constructors and pattern matching are used to eliminate the need for *abs* and *rep*. In ML one would write

```
datatype N = Zero | Succ of N

fun Add(Zero) m = m
  | Add(Succ x) m = Succ(Add x m)
```
(1.33)

The ability to mirror functional programming in this way illustrates the expressiveness of the metalanguage. □

1.1.6 Operational Semantics

As it stands, using the metalanguage is inferior to Strachey's original approach in one respect: β-reduction in the untyped lambda calculus provides an *executable* specification of a language, against which an implementation can be contrasted. Fortunately, the metalanguage also admits such an operational interpretation, building on Proposition 1.2 and (1.31).

The operational semantics defines an evaluation relation $\mathbb{A} : t \Downarrow v$ where $\vdash t : \mathbb{A}$ and v is a *value*, i.e. a closed, well-formed term generated by the grammar

$$v ::= \lambda x.t \mid (t, u) \mid inl\, t \mid inr\, t \mid abs\, v\ . \tag{1.34}$$

The denotations of values satisfy $[\![\vdash v : \mathbb{A}]\!] >_{\mathbb{A}} \bot$; intuitively, values represent positive information. The evaluation relation is defined inductively using a *structural operational semantics* [75]; we leave out obvious symmetric rules:

$$
\begin{gathered}
\frac{}{\mathbb{A} : v \Downarrow v} \qquad \frac{\mathbb{A} : t[rec\,x.t/x] \Downarrow v}{\mathbb{A} : rec\,x.t \Downarrow v} \\
\frac{\mathbb{A} \to \mathbb{B} : t \Downarrow \lambda x.t' \quad \mathbb{A} : t'[u/x] \Downarrow v}{\mathbb{B} : t\,u \Downarrow v} \\
\frac{\mathbb{A} \times \mathbb{B} : t \Downarrow (t', t'') \quad \mathbb{A} : t' \Downarrow v}{\mathbb{A} : fst\,t \Downarrow v} \quad \frac{\mathbb{A} + \mathbb{B} : u \Downarrow inl\,u' \quad \mathbb{C} : t_1[u'/x] \Downarrow v}{\mathbb{C} : [u > inl\,x \Rightarrow t_1, inr\,x \Rightarrow t_2] \Downarrow v} \\
\frac{\mathbb{T}[\mu T.\mathbb{T}/T] : t \Downarrow v}{\mu T.\mathbb{T} : abs\,t \Downarrow abs\,v} \quad \frac{\mu T.\mathbb{T} : t \Downarrow abs\,v}{\mathbb{T}[\mu T.\mathbb{T}/T] : rep\,t \Downarrow v}
\end{gathered}
\tag{1.35}
$$

Proposition 1.5 The operational semantics satisfies:

- Determinacy: suppose $\mathbb{A} : t \Downarrow v$ and $\mathbb{A} : t \Downarrow v'$. Then $v \equiv v'$.
- Type-correctness: suppose $\vdash t : \mathbb{A}$ and $\mathbb{A} : t \Downarrow v$. Then $\vdash v : \mathbb{A}$.

Both properties are proved by induction on the derivation rules.

1.1.7 Relating Operational and Denotational Semantics

Often, small programming languages considered for research purposes are defined directly using an operational semantics without an underlying denotational semantics.[1] One reason for this is that operational semantics is very flexible and so ideal for a first exploration of programming concepts. But flexibility is a two-edged sword, leading also to ad hoc constructions. This becomes apparent when one tries to relate and formally compare different languages. Without a link to the global mathematical setting provided by domain theory, one is left with syntactic translations which can be hard to find, justify, and understand.

Another problem with a purely syntactic approach is the lack of good reasoning methods. Consider two open terms $\Gamma \vdash t_1 : \mathbb{A}$ and $\Gamma \vdash t_2 : \mathbb{A}$ and suppose that we would prefer one to the other for performance reasons. To verify that such an optimisation is sound, we need to show that the two terms are equivalent in *all* contexts, so that no amount of programming can tell them apart. It is not hard to see that this follows iff the two terms are *contextually equivalent* [61], as defined below.

We define a *program* to be a closed term of type $\mathbb{N}$. If C is a term with holes into which a term $\Gamma \vdash t : \mathbb{A}$ may be put to form a program $\vdash C(t) : \mathbb{N}$,

[1] The (operational) semantics of full-scale languages tend to be defined only informally, using prose. Standard ML is a noteworthy exception to this [59].

we call C a $(\Gamma, \mathbb{A})$-*program context*. Suppose $\Gamma \vdash t_1 : \mathbb{A}$ and $\Gamma \vdash t_2 : \mathbb{A}$. We write $\Gamma \vdash t_1 \precsim t_2 : \mathbb{A}$ iff for all $(\Gamma, \mathbb{A})$-program contexts C we have

$$\mathbb{N} : C(t_1) \Downarrow \mathit{Zero} \implies \mathbb{N} : C(t_2) \Downarrow \mathit{Zero} \tag{1.36}$$

The terms t_1 and t_2 are said to be *contextually equivalent* if we have both $\Gamma \vdash t_1 \precsim t_2 : \mathbb{A}$ and $\Gamma \vdash t_2 \precsim t_1 : \mathbb{A}$.

Unfortunately, it can be quite hard to prove contextual equivalence because of the quantification over all program contexts. It is often much easier to show that two terms have the same denotation, and so a suitable link to the denotational semantics becomes very useful. Two standard results can be shown:

Proposition 1.6 (Soundness) Suppose $\vdash t : \mathbb{A}$. Then

$$\mathbb{A} : t \Downarrow v \implies [\![\vdash t : \mathbb{A}]\!] = [\![\vdash v : \mathbb{A}]\!] \ . \tag{1.37}$$

Proposition 1.7 (Adequacy) Suppose $\vdash t : \mathbb{A}$. Then

$$[\![\vdash t : \mathbb{A}]\!] \neq \bot \iff \exists v.\ \mathbb{A} : t \Downarrow v \ . \tag{1.38}$$

Soundness is proved by induction on the derivation rules using Proposition 1.2 and (1.31). Adequacy is quite a bit harder, involving logical relations and a trick to deal with recursive types which exploits the representation using information systems and also makes clever use of the *abs* and *rep* constructors, see [95], Ch. 13.

It follows from soundness and adequacy that if two terms have the same denotation, then they are contextually equivalent. This result is sometimes called *computational adequacy* [72]:

Theorem 1.8 (Computational Adequacy)

$$[\![\Gamma \vdash t_1 : \mathbb{A}]\!] \leq_{\Gamma \to \mathbb{A}} [\![\Gamma \vdash t_2 : \mathbb{A}]\!] \implies \Gamma \vdash t_1 \precsim t_2 : \mathbb{A} \tag{1.39}$$

Depending on the exact nature of the denotational semantics, one may also be able to show the converse of the implication (1.39). This is called *full abstraction* and is a kind of holy grail of semantics, because it is normally very difficult to obtain [39]. Indeed, the denotational semantics of the metalanguage is not fully abstract. It contains elements like "parallel or" [74], which are not definable in the metalanguage, but which can nevertheless distinguish between terms that the operational semantics cannot. Berry's "stable" domain theory [8] rules out elements like parallel or by demanding that the output of every function is associated with a unique, minimal input. But other problematic elements remain and there is no known domain-based fully abstract semantics for a language such as the metalanguage. Curiously enough, the problem seems to be to capture within domain theory the sequentiality of sequential computation.

1.2 Concurrent Computation

Broadly speaking, approaches to concurrency either start from the syntax of a process calculus or are based on a specific mathematical model of processes. We indicate the diversity of both approaches below.

1.2.1 Process Calculi

In 1972, Milner tried to apply standard semantic techniques like those we have seen to a concurrent programming language—and failed [57]. In particular, Milner found that viewing a program as a function over memory states, and thus identifying the programs

$$p_1 \equiv_{\mathrm{def}} x := 1 \quad \text{and} \quad p_2 \equiv_{\mathrm{def}} x := 0 \,;\, x := x + 1 \ , \tag{1.40}$$

only makes sense if the program has full control over memory. Interference from another program, say $x := 1$, will not change the behaviour of program p_1, but will make program p_2 behave nondeterministically, changing x to either 1 or 2.

Here, the problem is the interaction between storage and a number of programs. The same kind of problem arises with interactions between programs, between machines, and indeed, between a machine and its user. This led Milner to search for something more fundamental—a new calculus with interaction, or communication, as the central idea. A primitive notion of *indivisible interaction* was put forward independently by Milner and Hoare [56, 36]; in Milner's case embodied in his "Calculus of Communicating Systems", or CCS. Like the basic lambda calculus, it is untyped.

Let N be a set of names and $\bar{N} =_{\mathrm{def}} \{\bar{n} : n \in N\}$ the set of "complemented" names. Let l range over labels $L =_{\mathrm{def}} N \cup \bar{N}$, with complementation extended to L by taking $\bar{\bar{n}} =_{\mathrm{def}} n$. We let α range over the set of *actions* $A =_{\mathrm{def}} L \cup \{\tau\}$ where τ does not occur in L. The syntax of CCS processes is given by

$$\begin{array}{rll} t, u ::= & x, y, z, \ldots & \text{(variables)} \\ & \mid \mathit{rec}\, x.t & \text{(recursion)} \\ & \mid \Sigma_{i \in I} t_i & \text{(nondeterministic sum)} \\ & \mid \alpha.t & \text{(prefixing by atomic action } \alpha\text{)} \\ & \mid t|u & \text{(parallel composition)} \\ & \mid t \setminus S & \text{(restriction from names in } S \subseteq N\text{)} \\ & \mid t[r] & \text{(relabelling using } r : N \to N\text{)} \end{array} \tag{1.41}$$

In a nondeterministic sum the indexing set I may be an arbitrary set; we write $\varnothing$ when I is empty and $t_1 + \cdots + t_k$ for a typical finite sum.

Relabelling functions $r : N \to N$ are extended to all of A by taking $r\bar{l} =_{\mathrm{def}} \overline{(rl)}$ and $r\tau =_{\mathrm{def}} \tau$. An operational semantics defines a *transition* *relation* $t \xrightarrow{\alpha} t'$ between closed CCS terms, using actions as labels.

$$
\begin{array}{c}
\dfrac{t[rec\,x.t/x] \xrightarrow{\alpha} t'}{rec\,x.t \xrightarrow{\alpha} t'} \quad \dfrac{t_j \xrightarrow{\alpha} t'}{\Sigma_{i\in I} t_i \xrightarrow{\alpha} t'}\ j \in I \quad \dfrac{}{\alpha.t \xrightarrow{\alpha} t} \\
\dfrac{t \xrightarrow{\alpha} t'}{t|u \xrightarrow{\alpha} t'|u} \quad \dfrac{t \xrightarrow{l} t' \quad u \xrightarrow{\bar{l}} u'}{t|u \xrightarrow{\tau} t'|u'} \quad \dfrac{u \xrightarrow{\alpha} u'}{t|u \xrightarrow{\alpha} t|u'} \\
\dfrac{t \xrightarrow{\alpha} t'}{t \setminus S \xrightarrow{\alpha} t' \setminus S}\ \alpha \notin S \cup \bar{S} \quad \dfrac{t \xrightarrow{\alpha} t'}{t[r] \xrightarrow{r\alpha} t'[r]}
\end{array}
\tag{1.42}
$$

Intuitively, $t \xrightarrow{n} t'$ means that process t is willing to offer a communication on a channel named $n \in N$, after which it will continue as the process t'. “Channel” should be understood in a very broad sense, cf. the example below or Milner’s book [57]. If in addition $u \xrightarrow{\bar{n}} u'$ so that process u is willing to offer a complementary communication on the same channel, then these processes can communicate, signalling to the environment with the special action τ that a communication has taken place. As τ is not in L, no further communication involving this transition can take place.

Example 1.9 We model two beer/coffee vending machines and a customer in CCS. Names are given by the set $N = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$, standing for “accept money”, “beer”, and “coffee”. The first vending machine, V_1 below, can either sell beer or coffee, the choice made by the machine as the money is accepted. For machine V_2 both possibilities are still available after the first action of accepting money. The customer just wants a beer.

$$
\begin{aligned}
V_1 &\equiv_{\mathrm{def}} \mathsf{a}.\mathsf{b}.\varnothing + \mathsf{a}.\mathsf{c}.\varnothing \\
V_2 &\equiv_{\mathrm{def}} \mathsf{a}.(\mathsf{b}.\varnothing + \mathsf{c}.\varnothing) \\
C &\equiv_{\mathrm{def}} \bar{\mathsf{a}}.\bar{\mathsf{b}}.\varnothing
\end{aligned}
\tag{1.43}
$$

The vending machines can serve only a single customer after which they both become inactive. Longer-lasting machines can be specified using recursion, e.g. $rec\,x.\mathsf{a}.(\mathsf{b}.x + \mathsf{c}.x)$. Following the operational semantics, we obtain the following transitions from process $V_1|C$:

$$V_1|C \xrightarrow{\tau} \mathsf{b}.\varnothing|\bar{\mathsf{b}}.\varnothing \quad \text{and} \quad V_1|C \xrightarrow{\tau} \mathsf{c}.\varnothing|\bar{\mathsf{b}}.\varnothing \tag{1.44}$$

Note that the system encounters a deadlock in the state $\mathsf{c}.\varnothing|\bar{\mathsf{b}}.\varnothing$ where the machine insists on delivering coffee which the customer will not accept. By contrast,

$$V_2|C \xrightarrow{\tau} (\mathsf{b}.\varnothing + \mathsf{c}.\varnothing)|\bar{\mathsf{b}}.\varnothing\ , \tag{1.45}$$

and so no such deadlock can occur starting from $V_2|C$. □

CCS is accompanied by a notion of process equivalence, based on *simulation*, that is, the ability of one process to mirror the behaviour of another. The equivalence distinguishes between the processes V_1 and V_2 above intuitively because V_1 is unable to simulate V_2. Formally, a relation R between processes is a *bisimulation* [68, 57] if the following holds. If $t_1 \mathrel{R} t_2$, then

1. if $t_1 \xrightarrow{\alpha} t_1'$, then $t_2 \xrightarrow{\alpha} t_2'$ for some t_2' such that $t_1' \mathrel{R} t_2'$;
2. if $t_2 \xrightarrow{\alpha} t_2'$, then $t_1 \xrightarrow{\alpha} t_1'$ for some t_1' such that $t_1' \mathrel{R} t_2'$.

Bisimilarity, written $\sim$, is the largest bisimulation. If we only demand that the second item above is satisfied, then R is called a *simulation* and t_1 is said to simulate t_2. If t_2 also simulates t_1, then t_1 and t_2 are said to be *simulation equivalent*. Note that simulation equivalence is strictly weaker than bisimilarity because the simulations used need not be reverse images.

Example 1.10 The two vending machines V_1 and V_2 of Example 1.9 are not bisimilar, because if R relates V_1 and V_2, then by requirement 1. it should also relate $\mathsf{b}.\varnothing$ and $\mathsf{b}.\varnothing + \mathsf{c}.\varnothing$. But then requirement 2. fails because the latter process can perform a c-action which cannot be matched by the former. The machine V_2 simulates V_1, but the converse is not true. □

Bisimilarity is a *congruence* for CCS so that whenever $t_1 \sim t_2$, then $C(t_1) \sim C(t_2)$ for any context C. This allows the same kind of local reasoning possible with denotational semantics, cf. Section 1.1.7: there is no need to quantify over all contexts to show that we may replace t_1 by t_2 in any program.

Bisimilarity has a logical characterisation based on *Hennessy-Milner logic* [31], a modal logic with formulae given by

$$\phi ::= \langle\alpha\rangle\phi \mid \textstyle\bigwedge_{i\in I} \phi_i \mid \neg\phi \ . \tag{1.46}$$

The notion of *satisfaction*, written $t \vDash \phi$, is defined by

$$\begin{aligned} t \vDash \langle\alpha\rangle\phi &\iff_{\text{def}} \exists t'.\ t \xrightarrow{\alpha} t' \text{ and } t' \vDash \phi \\ t \vDash \textstyle\bigwedge_{i\in I} \phi_i &\iff_{\text{def}} t \vDash \phi_i \text{ for each } i \in I \\ t \vDash \neg\phi &\iff_{\text{def}} t \vDash \phi \text{ does not hold.} \end{aligned} \tag{1.47}$$

We write $\top$ and $\phi_1 \wedge \cdots \wedge \phi_n$ for the empty and finite conjunctions, respectively. The formula $\top$ is satisfied by any process.

Two processes are bisimilar iff they satisfy exactly the same formulae. The fragment of Hennessy-Milner logic obtained by leaving out negation is characteristic for simulation equivalence, while restricting the logic to finite conjunctions restricts its characterising power to *image-finite* processes, so processes t where the set $\{t' : t \xrightarrow{\alpha} t'\}$ is finite for each α [31, 90].

Example 1.11 The formula $\langle\mathsf{a}\rangle(\langle\mathsf{b}\rangle\top \wedge \langle\mathsf{c}\rangle\top)$ is satisfied by V_2, but not by V_1, witnessing that the two machines are neither bisimilar, nor simulation equivalent. □

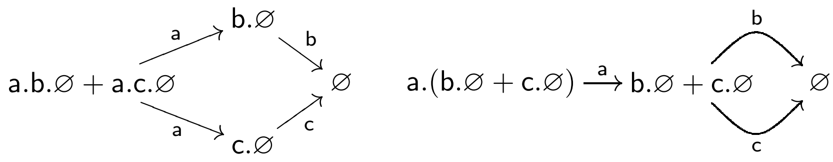

Figure 1.1: The two vending machines as transition systems

Milner's seminal work has led to the study of a variety of process calculi and suitable notions of bisimulation, following the pattern above (see [23] for an annotated bibliography). In many cases the calculi are much more complicated than CCS, e.g. allowing communication of processes as in Thomsen's CHOCS [88] or new-name generation as in the π-calculus [60, 79], and it quickly becomes difficult to give a reasonable definition of bisimulation which can be shown to yield a congruence (this is particularly so when higher-order features are combined with name-generation [77, 78]). A good case in point is Cardelli and Gordon's Ambient Calculus [14] where the calculus and the notion of bisimulation need to be adjusted to each other, in what seems to be an ad hoc fashion [55]. A lot of energy is used on these "local optimisations" to specific process calculi, optimisations that may obscure connections and the global picture—being based on operational semantics alone, it is already hard to relate and compare different calculi. So the lessons learnt often remain isolated for lack of the commonality a global framework would provide.

1.2.2 Process Models

One could hope that a suitable mathematical model of processes would provide that global framework. But over the years researchers in concurrency theory have produced a large number of different process models which has just made the picture even more fragmented. We list a few:

A *transition system* is a quadruple (S, i, A, tran) where S is a set of states with $i \in S$ the initial state, A is an alphabet of labels, and $\mathit{tran} \subseteq S \times A \times S$ the transition relation. It is common to write $s \xrightarrow{\alpha} s'$ for $(s, \alpha, s') \in \mathit{tran}$. CCS gives rise to a transition system for each closed process term t: states are all closed terms with t the initial state, labels are actions, and transitions are derived by the operational rules. The two vending machine processes of Example 1.9 give rise to the transition systems in Figure 1.1. It is customary to leave out unreachable states in such pictures. Note that the definition of bisimulation in the previous section makes sense for any transition system: two transition systems are bisimilar if there is a bisimulation relating their initial states.

$$\begin{tikzcd}[column sep=small]
 & \mathsf{b}.\varnothing \arrow[r, "\mathsf{b}"] & \varnothing \\
\mathsf{a}.\mathsf{b}.\varnothing + \mathsf{a}.\mathsf{c}.\varnothing \arrow[ur, "\mathsf{a}"] \arrow[dr, "\mathsf{a}"'] & & \\
 & \mathsf{c}.\varnothing \arrow[r, "\mathsf{c}"'] & \varnothing
\end{tikzcd}
\qquad
\begin{tikzcd}[column sep=small]
 & & \varnothing \\
\mathsf{a}.(\mathsf{b}.\varnothing + \mathsf{c}.\varnothing) \arrow[r, "\mathsf{a}"] & \mathsf{b}.\varnothing + \mathsf{c}.\varnothing \arrow[ur, "\mathsf{b}"] \arrow[dr, "\mathsf{c}"'] & \\
 & & \varnothing
\end{tikzcd}$$

Figure 1.2: The two vending machines as synchronisation trees

A *synchronisation tree* [56] is a transition system whose underlying graph is a tree with the initial state as root. Any transition system can be unfolded into a synchronisation tree by unwinding loops, removing unreachable states, and duplicating states which are reachable in more than one way. The two transition systems of Figure 1.1 unfold into the trees of Figure 1.2.

A *trace set* [37] is simply a prefix-closed set X of finite words over some alphabet. Any synchronisation tree over the label set A gives rise to a trace set over A, containing all words obtained by concatenating the labels of a finite path from the root to some node in the tree. Conversely, a trace set X over A gives rise to a synchronisation tree over A with states being the elements of X, the root given by the empty string ϵ, and the transition relation given by $s \xrightarrow{\alpha} s'$ iff $s\alpha = s'$ for any $s, s' \in X$ and $\alpha \in A$. Trace sets abstract away from the nondeterministic branching behaviour of processes. Indeed, our two vending machine processes give rise to the same trace set, $\{\epsilon, \mathsf{a}, \mathsf{ab}, \mathsf{ac}\}$, so they are *trace equivalent* even though they are not simulation equivalent.

The above three models are *interleaving models*, which means that they reduce parallelism to nondeterminism. In our vending-machine terminology, the models do not distinguish between two vending machines running in parallel, one ready to serve beer the other ready to serve coffee, and a single vending machine which can serve both in any order,

$$\mathsf{b}.\varnothing | \mathsf{c}.\varnothing \quad \text{vs.} \quad \mathsf{b}.\mathsf{c}.\varnothing + \mathsf{c}.\mathsf{b}.\varnothing \ . \tag{1.48}$$

But among the variety of models for concurrency, one can discern an increasing use of *independence models* (or causal, or partial-order models) according to which these two processes are different. Independence models thread through partial-order model checking [69], security protocols [87], nondeterministic dataflow [24], self-timed circuits [26], term-rewriting, game semantics [4], and the analysis of distributed algorithms [47]. While interleaving models have been given a domain-theoretic treatment using powerdomains and domains of resumptions [73, 30], this approach will fall short of accommodating independence models, because it insists on a nondeterministic choice of actions one at a time.

Key independence models include: *Petri nets* [10, 11] and *asynchronous transition systems* [5, 84] which are closely related and can be understood as transition systems with an independence structure added; *event structures* [94] which can be obtained by unfolding such transition systems as with synchronisation trees above; and *Mazurkiewicz trace sets* [52] which are trace sets with an independence structure on the underlying alphabet.

To give a flavour of independence models (and because we'll need it later) we give the definition of a particular kind of event structures, namely "prime event structures with binary conflict", hereafter just called event structures.

An *event structure* [94] is a triple $\mathbb{E} = (E, \leq, \#)$ where E is a set of events upon which a partial order $\leq$ of causality and a binary, symmetric, irreflexive relation $\#$ of conflict are defined. This data must satisfy

(i) $\forall e \in E.\ \lceil e \rceil =_{\mathrm{def}} \{e' : e' \leq e\}$ is finite.

(ii) $\forall e, e', e'' \in E.\ e \# e' \leq e'' \implies e \# e''$.

A *configuration* of $\mathbb{E}$ is a subset $x \subseteq E$ which is

(i) down-closed: $\forall e \in x.\ \lceil e \rceil \subseteq x$.

(ii) consistent: $\forall e, e' \in x.\ \neg(e \# e')$.

An event structure models a process as follows: the set E contains the set of events that can possibly occur during the execution of the process. The causality relation expresses that some events depend on each other, while the conflict relation expresses that some events exclude each other (here in a binary fashion). A configuration represents a point in the execution of the process, collecting the events that have occurred so far.

Example 1.12 The CCS process $\mathsf{b}.\varnothing|\mathsf{c}.\varnothing$ can be modelled as an event structure with two events, serving beer (b) and coffee (c). These events neither depend on nor exclude each other, and so the causality relation is the identity while the conflict relation is empty. The configurations are $\varnothing$, $\{b\}$, $\{c\}$, and $\{b, c\}$.

On the other hand, the process $\mathsf{b}.\mathsf{c}.\varnothing + \mathsf{c}.\mathsf{b}.\varnothing$ will be modelled using four events, b_1, c_1, b_2, and c_2, with $b_1 \leq c_1$ and $c_2 \leq b_2$ and events indexed by 1 in conflict with events indexed by 2. The configurations are $\varnothing$, $\{b_1\}$, $\{b_1, c_1\}$, $\{c_2\}$, and $\{c_2, b_2\}$. $\square$

Interestingly, event structures are also representations of domains and match well with Berry's stable domain theory [91].

Nondeterministic dataflow [24, 43] is a rather different model, describing reactive systems with input and output ports, interacting with the environment by reading values, e.g. letters of an alphabet, on the input ports and producing values on the output ports, see the top part of Figure 1.3. An essential idea of the dataflow paradigm is that a collection of networks can be

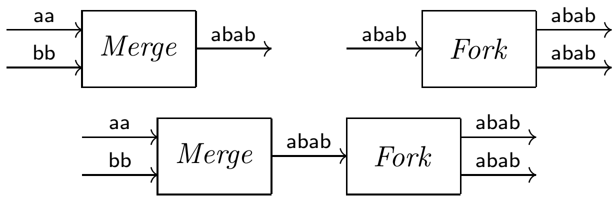

Figure 1.3: Nondeterministic dataflow

combined into a single, larger network, possibly connecting some of the free input and output ports as in the bottom part of the figure. The wires can be understood as unbounded first-in-first-out buffers. It is usually assumed that two wires cannot be connected to the same port. Connecting two ports of the same network yields a feedback loop, an operation called *trace* [33]. Kahn [43] observed that one could give a semantics to *deterministic* dataflow using an interleaving model, whereas it has been recognised [13, 33] that an independence model is useful in the nondeterministic case.

1.3 Towards a Domain Theory for Concurrency

With a plethora of models around, it is hardly surprising that relations between different approaches are often unclear, hampering technology transfer. Indeed, ideas may be rediscovered; for example, special event structures reappear as "strand spaces" in reasoning about security protocols [87, 22]. We discuss recent work towards providing more structure to the field.

1.3.1 Categorical Structure

The work presented in the handbook chapter [96] by Winskel and Nielsen concentrates on understanding the structure of different models and how they relate. Again, category theory has been the natural tool for this task. With the exception of dataflow, each model above is turned into a category whose morphisms stand for "simulations". For example, individual transition systems form the objects of the category $\mathbf{T}$. Assuming a common alphabet A for brevity, if $T_1 = (S_1, i_1, A, \mathit{tran}_1)$ and $T_2 = (S_2, i_2, A, \mathit{tran}_2)$ are transition systems, a morphism $f : T_1 \to T_2$ in $\mathbf{T}$ is a function $f : S_1 \to S_2$ such that $fi_1 = i_2$ and $s_1 \xrightarrow{\alpha} s_1' \in \mathit{tran}_1$ implies $fs_1 \xrightarrow{\alpha} fs_1' \in \mathit{tran}_2$. Hence, f expresses how T_1 may be simulated by T_2. Synchronisation trees form a subcategory $\mathbf{S}$ of $\mathbf{T}$ and the operation of unfolding results in a right adjoint to the inclusion functor $\mathbf{S} \hookrightarrow \mathbf{T}$. The other models are related in a similar way. In particular, the interleaving models can be seen as independence models with trivial independence structure; also here, the associated functors are part of adjunctions.

The presentation of models as categories not only makes precise how they relate. Operations like nondeterministic sum and parallel composition found in process calculi can be exposed as universal constructions in the categorical models. This can be used to give uniform semantics across different calculi and, exploiting preservation properties of adjoints, to relate semantics of a calculus across different models [96].

1.3.2 Bisimulation from Open Maps

Joyal, Nielsen, and Winskel gave further impetus to the approach by a sweeping definition of bisimulation across the categorical models using *open maps* [42]. The idea is to single out those maps of the categories that not only preserve behaviour, but also reflect it. In the case of a map $f : T_1 \to T_2$ between the transition systems above, this would mean that if $fs_1 \xrightarrow{\alpha} s_2' \in tran_2$, then there exists $s_1' \in S_1$ such that $s_1 \xrightarrow{\alpha} s_1' \in tran_1$ and $fs_1' = s_2'$. It should be clear that this is equivalent to demanding that (the graph of) f is *functional bisimulation* between the two transition systems, i.e. a bisimulation which is also a function.

To see what this requirement amounts to in categorical terms, consider first a finite sequence of transitions in T_1:

$$p_1 : i_1 \xrightarrow{\alpha_1} r_1 \xrightarrow{\alpha_2} r_2 \xrightarrow{\alpha_3} \cdots \xrightarrow{\alpha_n} s_1 \ . \tag{1.49}$$

Since f is a morphism $T_1 \to T_2$, this induces a simulating sequence in T_2:

$$p_2 : i_2 \xrightarrow{\alpha_1} fr_1 \xrightarrow{\alpha_2} fr_2 \xrightarrow{\alpha_3} \cdots \xrightarrow{\alpha_n} fs_1 \ . \tag{1.50}$$

Now, if f also reflects behaviour, then any extension of p_2, say

$$q_2 : i_2 \xrightarrow{\alpha_1} fr_1 \xrightarrow{\alpha_2} fr_2 \xrightarrow{\alpha_3} \cdots \xrightarrow{\alpha_n} fs_1 \xrightarrow{\alpha} s_2' \ , \tag{1.51}$$

can be reflected by an extension of p_1,

$$q_1 : i_1 \xrightarrow{\alpha_1} r_1 \xrightarrow{\alpha_2} r_2 \xrightarrow{\alpha_3} \cdots \xrightarrow{\alpha_n} s_1 \xrightarrow{\alpha} s_1' \ . \tag{1.52}$$

Moreover, s_1' can be chosen to be compatible with q_2 under f i.e. $fs_1' = s_2'$. Now, the four sequences above are themselves transition systems, and so are objects of $\mathbf{T}$. Clearly, we have $p_1 \cong p_2$ and $q_1 \cong q_2$ in $\mathbf{T}$, and we'll refer to them as just p and q below. That q extends p is witnessed by an obvious morphism $e : p \to q$ in $\mathbf{T}$, and because p is a sequence of T_1 and q a sequence of T_2 we have morphisms $x : p \to T_1$ and $y : q \to T_2$, so that the square on the left below commutes. Commutativity just rephrases the fact that the simulation by T_2 of p can be extended to the simulation of q.

$$\begin{tikzcd} p \arrow[r, "x"] \arrow[d, "e"'] & T_1 \arrow[d, "f"] \\ q \arrow[r, "y"'] & T_2 \end{tikzcd} \qquad\qquad \begin{tikzcd} p \arrow[r, "x"] \arrow[d, "e"'] & T_1 \arrow[d, "f"] \\ q \arrow[r, "y"'] \arrow[ur, dotted, "z"] & T_2 \end{tikzcd} \tag{1.53}$$

The reflection requirement can then be stated by saying that any such commutative square can be split into two commutative triangles as on the right above. Commutativity of the upper triangle expresses that the sequence in T_1 that simulates q is an extension of the sequence simulating p. The lower triangle amounts to saying that this extension is compatible under f with the sequence simulating q in T_2.

It is reasonable to think of objects like p and q as *computation paths* with respect to the model of transition systems, because they correspond to sequences of transitions. In general let $\mathbf{M}$ be any category of models and let $\mathbb{P}$ be any subcategory $\mathbb{P} \hookrightarrow \mathbf{M}$ of $\mathbf{M}$ whose objects and morphisms are thought of as computation paths and ways that they extend each other. We define $\mathbb{P}$-*open maps* to be those morphisms f in $\mathbf{M}$ such that for all morphisms $e : p \to q$ in $\mathbb{P}$, any commuting square like that on the left of (1.53) can be split into two commuting triangles as on the right.

Open maps account only for functional bisimulations, and to get all possible bisimulations, two models X and Y of $\mathbf{M}$ are said to be *open map bisimilar* iff there exists a span $X \xleftarrow{f} Z \xrightarrow{g} Y$ of open maps between them. In the case of transition systems and the subcategory of finite sequences, open-map bisimilarity coincides with Park and Milner's notion as defined in Section 1.2.1. Reasonable equivalences are obtained for other models as well, given suitable choices of subcategories of paths, see [42].

1.3.3 Presheaf Models

Of course, the freedom in the choice of the path category $\mathbb{P}$ can be problematic as there is no reason to expect all models to furnish a "natural" choice as in the case above. Once more, category theory provides a helpful suggestion. There is a well-studied class of categories, called *presheaf categories*,[2] with a canonical choice of path category.

The objects of a presheaf category, *presheaves*, can be understood as the collection of processes constructed from the underlying path category by freely adding coproducts (nondeterministic sums of paths) and coequalisers (gluing paths together by identifying subpaths), thereby creating "sheaves" of computation paths—or processes with branching behaviour. An alternative view is that a presheaf, seen as a process, assigns to each computation path the set of ways it is simulated by, or "realised by", the process.

Taking the path category to be finite sequences of transitions as above, the presheaf category is the category of synchronisation forests, that is, objects are *sets* of synchronisations trees, because a presheaf may assign more than one way of simulating the empty transition sequence. Restricting

[2] A note on the terminology "open map": in presheaf categories, open maps as defined above satisfy Joyal and Moerdijk's axioms [41] for open maps in toposes, of which presheaf categories are a prime example.

to "rooted" presheaves, so presheaves that correspond to synchronisation trees, open-map bisimilarity coincides again with Park and Milner's notion.

Guided by these intuitions and the categorical situation, Joyal, Nielsen, and Winskel suggested presheaves as a general model of processes. Presheaf models were subsequently studied by Winskel and his PhD students Cattani [19] and Hildebrandt [35] together with a number of coworkers. This work has shown that presheaf models are expressive enough to encompass, and generalise, a wide range of existing models and notions from concurrent computation, including CCS-like languages, their models, and equivalences [97, 16, 25, 34], higher-order features [98], nondeterministic dataflow [33], independence models [100], and name-generation [17].

1.3.4 Domain Theory for Concurrency

A domain theory which handled higher-order processes, independence models, name-generation, and possessed an operational interpretation would provide a global mathematical framework for most theories of concurrency. In light of the work above it would seem that presheaf models make a good candidate for such a framework.

Cattani and Winskel have drawn attention to a 2-categorical model of linear logic and associated pseudo-comonads, based on presheaves [20]. Linear maps preserve open-map bisimilarity, but are too restricted to model many important process operations, including prefixing. From a weakening comonad one derives a model of affine-linear logic (in the sense of Jacobs [40]) whose morphisms, affine maps, allow prefixing and still preserve open-map bisimilarity.

This led Winskel to the discovery of an expressive metalanguage for concurrency [98]. As terms of this language are interpreted as affine maps, open-map bisimilarity is automatically a congruence. Affine-linearity restricts copying and while it allows a process to ignore its input, it does not allow it to investigate the behaviour of the input in more than one context. This matches the general situation in a distributed setting where processes interact as peers with very little control over one another.

The starting point for the work presented below was the hope that an operational semantics for the affine language would provide a general operational understanding of presheaf models. A first result [63, 64] was an operational semantics for the first-order fragment of the affine language guided by the idea that derivations of transitions in the operational semantics should correspond to realisers in the presheaf denotations [19]. The operational semantics indicated that the tensor of affine-linear logic should be understood as a parallel composition of independent processes. This intuition was backed up by an event-structure representation of definable presheaves, in which the tensor operation corresponds to the simple parallel composition of event structures got by juxtaposition [63].

It proved difficult to extend the operational semantics for the affine language beyond first order, and we were led to consider another pseudo-comonad, an exponential of linear logic, giving rise to a model of intuitionistic logic and another expressive metalanguage [65]. This language, called HOPLA for Higher-Order Process LAnguage, can be viewed as a simply-typed lambda calculus extended with CCS-like nondeterministic sum and prefix operations. In contrast to the affine metalanguage, which we call Affine HOPLA below, HOPLA allows a process to copy its input and investigate the behaviour of it in different contexts. Intuitively, a HOPLA process receives the *code* of an input process and so is in full control of its execution.

Affine HOPLA can be viewed as a variation of HOPLA obtained by adding the tensor at the cost of linearity constraints on variables.

HOPLA supports a straightforward operational semantics, again guided by the presheaf model. However, as HOPLA and its operational semantics are very simple, it was felt that the presentation of its denotational semantics suffered from a significant overhead in terms of the category theory needed. As it turned out, both HOPLA and Affine HOPLA can be obtained working within a much simpler domain theory, avoiding the 2-categorical structure.

Presheaf models are directly based on computation paths, a line which has been followed in what seemed originally to be a different direction. In Hennessy's semantics for CCS with process passing [32], a process denotes the set of its computation paths. We'll call this kind of semantics a *path semantics* because of its similarity to trace semantics; in both cases, processes denote downwards-closed sets of computation paths and the corresponding notion of process equivalence, called *path equivalence*, is given by equality of such sets. Computation paths, however, may have more structure than traditional traces, e.g. allowing path semantics to take nondeterministic branching into account in a limited way. In fact, path equivalence can be linked to simulation equivalence for image-finite processes [67].

Path semantics and presheaf semantics can be seen as variations on a common idea: that a process denotes a form of characteristic function in which the truth values are sets of realisers. A path set may be viewed as a special presheaf that yields at most one realiser for each path. The extra structure of presheaves, saying *how* each computation path may be realised, allows the incorporation of complete branching information.

Though path sets are considerably simpler than presheaves they furnish models which are sufficiently rich in structure to show how both the above languages arise from canonical constructions on path sets [66, 67]. The path semantics admits simple proofs of full abstraction, showing that path equivalence coincides with contextual equivalence. Still, the path semantics cannot stand alone. It provides little guidance for operational semantics; it does not capture enough branching information to characterise bisimulation; and there are extensions to deal with independence models which cannot be catered for.

1.3.5 Overview

The thesis is based on the author's progress report [63] and four papers coauthored with Glynn Winskel:

[64] Linearity in process languages. Presented at LICS'02.

[65] HOPLA—a higher-order process language. Presented at CONCUR'02.

[66] Full abstraction for HOPLA. Presented at CONCUR'03.

[67] Domain theory for concurrency. To appear in *Theoretical Computer Science* special issue on domain theory.

Part I concentrates on the simple domain theory for concurrency based on path sets while Part II discusses the guidance provided by the extra structure of presheaves.

Chapter 2 shows how path sets give rise to a simpler version of the abovementioned categorical model of linear logic with associated comonads, avoiding the 2-categorical structure [67].

Chapter 3 presents the metalanguage HOPLA [65, 66] and shows how its operations arise from canonical constructions in the underlying path set model, using an exponential of linear logic. The path semantics is shown fully abstract such that contextual equivalence coincides with path equivalence. The language is given a straightforward operational semantics which is endowed with a standard bisimulation congruence. The denotational and operational semantics are related with pleasingly simple proofs of soundness and adequacy. Contextual equivalence is shown to coincide with logical equivalence for the fragment of Hennessy-Milner logic that characterises simulation equivalence. The expressive power of HOPLA is indicated by encodings of calculi like CCS, CCS with process passing, and mobile ambients with public names. Work is in progress on extending HOPLA with name-generation [101].

Chapter 4 presents the affine metalanguage, which is here called Affine HOPLA [98, 64, 67]. It adds to HOPLA an interesting tensor operation at the price of linearity constraints on the occurrences of variables. Again its denotational semantics, directly based on canonical constructions on path sets using a weakening comonad, is shown fully abstract. An interleaving operational semantics for the first-order fragment of the language is provided along with proofs of soundness and adequacy. It is shown how the tensor operation allows Affine HOPLA to encode nondeterministic dataflow processes without feedback loops.

Chapter 5 shows how the extra structure of presheaves is used to capture the branching behaviour of processes, and provides a brief overview of the 2-categorical models considered by Cattani and Winskel.

Chapter 6 proves correspondence results relating derivations in the operational semantics of the two languages above to realisers in their presheaf denotations [63, 64, 65].

Chapter 7 is based on unpublished work from [63]. It studies the independence structure of the tensor operation of Affine HOPLA by defining an event-structure semantics of the language which at first order provides a representation of the presheaf denotations. Here, the tensor operation denotes the simple parallel composition of event structures got by juxtaposition. The event-structure semantics is analogous to Berry's stable denotational semantics, and diverges from the presheaf semantics at higher order. Based on the representation, an alternative, "stable" operational semantics is given to the first-order fragment of Affine HOPLA. Derivations correspond to configurations in event-structures and so to elements of presheaves. The stable semantics agrees with the operational semantics from Chapter 4 and shows how to refine it with independence structure. Further, it can be extended to higher-order processes although there is at present no proof of correspondence with the denotational semantics for the extension.

Chapter 8 concludes with a summary of the thesis and a discussion of related work towards a fully fledged domain theory for concurrency, in particular concerning name-generation, independence models, and bisimilarity.

Part I

Path Semantics

Chapter 2

Domain Theory from Path Sets

This chapter presents a simple domain theory based on path sets. Section 2.1 introduces path semantics of processes and highlights the difference between path semantics and traditional trace semantics. In the path semantics, processes are represented as elements of "nondeterministic domains", the properties of which are investigated in Section 2.2. The observation that nondeterministic domains are free completions highlights a natural notion of map between domains and leads to a categorical model of linear logic in Section 2.3. Maps of this category, "linear maps", are rather restricted—important operations like prefixing are not linear—and following the discipline of linear logic we therefore consider linear maps whose domain is under a comonad. One choice of comonad, an exponential of linear logic, gives rise to Scott-continuous maps and a model of intuitionistic logic; another choice yields a model of affine-linear logic. Constructions in both these models are induced by those of the linear category via adjunctions.

2.1 Processes as Path Sets

In the path semantics, processes are represented as sets of computation paths. Paths are elements of preorders $\mathbb{P}$ called *path orders* which function as process types, each describing the set of possible paths for processes of that type together with their sub-path ordering. A process of type $\mathbb{P}$ is then represented as a downwards-closed subset $X \subseteq \mathbb{P}$, called a *path set*.

To illustrate, consider the path order A^+ given by the poset of nonempty strings over some set $A = \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \ldots\}$ of actions, ordered by the prefix ordering. If the elements of A are atomic actions of processes like those of CCS, a path set over A^+ represents such a process by the set of finite sequences of actions that the process can perform. This is just the trace semantics of [37], except that trace sets include the empty string whereas our computation paths will be non-empty. Nondeterministic sum is interpreted by union of path sets with the inactive process $\varnothing$ represented as the empty set of paths. Prefixing by the action $\alpha \in A$ maps a path set $X \subseteq A^+$ to the path set $\{\alpha\} \cup \{\alpha s : s \in X\}$. Like trace semantics, this gives an abstract representation of processes.

Example 2.1 Consider again the CCS processes from Example 1.9:

$$\mathsf{a.b.}\varnothing + \mathsf{a.c.}\varnothing \quad \text{and} \quad \mathsf{a.}(\mathsf{b.}\varnothing + \mathsf{c.}\varnothing) \ . \tag{2.1}$$

By the above, both interpreted as the path set $\{\mathsf{a}, \mathsf{ab}, \mathsf{ac}\}$. □

The two processes of the example can be distinguished by refining the notion of path. Rather than strings of atomic actions, we may consider paths defined inductively by the rule below, using a judgement $p : \mathbb{P}$ to mean that path p is an element of the path order $\mathbb{P}$.

$$\frac{p_1 : \mathbb{P} \cdots p_k : \mathbb{P} \quad \alpha \in A}{\alpha\{p_1, \ldots, p_k\} : \mathbb{P}} \tag{2.2}$$

Intuitively, a process can perform the path $\alpha\{p_1, \ldots, p_k\}$ if it can perform the action α and, following that, is able to perform each of the paths in the set $\{p_1, \ldots, p_k\}$. The ordering $\leq_\mathbb{P}$ on $\mathbb{P}$ can be defined inductively based on the following preorder defined for arbitrary subsets X, X' of $\mathbb{P}$:

$$X \preceq_\mathbb{P} X' \iff \forall p \in X. \exists p' \in X'.\ p \leq_\mathbb{P} p' \ . \tag{2.3}$$

So, with P, P' ranging over finite subset of $\mathbb{P}$, we define $\alpha P \leq_\mathbb{P} \beta P'$ iff $\alpha = \beta$ and $P \preceq_\mathbb{P} P'$.

Using $\mathbb{P}$ rather than A^+ we change the interpretation of prefixing so that prefixing by the action $\alpha \in A$ now maps a path set $X \subseteq \mathbb{P}$ to the set $\{\alpha P : P \preceq_\mathbb{P} X\}$. The interpretation of nondeterministic sum is unchanged.

Example 2.2 Our example processes are now represented by the path sets

$$\{\mathsf{a}\varnothing, \mathsf{a}\{\mathsf{b}\varnothing\}, \mathsf{a}\{\mathsf{c}\varnothing\}\} \quad \text{and} \quad \{\mathsf{a}\varnothing, \mathsf{a}\{\mathsf{b}\varnothing\}, \mathsf{a}\{\mathsf{c}\varnothing\}, \mathsf{a}\{\mathsf{b}\varnothing, \mathsf{c}\varnothing\}\} \tag{2.4}$$

respectively. The extra element in the denotation of $\mathsf{a.}(\mathsf{b.}\varnothing + \mathsf{c.}\varnothing)$ distinguishes between them. □

The path order $\mathbb{P}$ and the associated notion of prefixing may seem a bit contrived at this point, but the definitions are actually based on canonical constructions on path sets, see Section 2.3.1 below. Moreover, we show in Section 3.5.1 that path equivalence with respect to path orders like $\mathbb{P}$ can be linked to simulation equivalence. Indeed, two image-finite CCS processes over the alphabet A may simulate each other iff their representations as path sets over $\mathbb{P}$ are identical.

2.2 Nondeterministic Domains

Path sets over a path order $\mathbb{P}$ may be ordered by inclusion to form a poset $\widehat{\mathbb{P}}$ which we'll think of as a domain of meanings of processes of type $\mathbb{P}$. The poset $\widehat{\mathbb{P}}$ has many interesting properties. First of all, it is a complete lattice with joins given by union. In the sense of Hennessy and Plotkin [30], $\widehat{\mathbb{P}}$ is a "nondeterministic domain", with joins used to interpret nondeterministic sums of processes. Accordingly, given a family $(X_i)_{i\in I}$ of elements of $\widehat{\mathbb{P}}$, we sometimes write $\Sigma_{i\in I}X_i$ for their join. A typical finite join is written $X_1 + \cdots + X_k$ while the empty join is the empty path set, the inactive process, written $\varnothing$.

A second important property of $\widehat{\mathbb{P}}$ is that any $X \in \widehat{\mathbb{P}}$ is the join of certain "prime" elements below it; $\widehat{\mathbb{P}}$ is a *prime algebraic complete lattice* [62]. Primes are down-closures

$$\mathrm{y}_{\mathbb{P}}p = \{p' : p' \leq_{\mathbb{P}} p\} \tag{2.5}$$

of individual elements $p \in \mathbb{P}$, representing a process that may perform the computation path p. The map $\mathrm{y}_{\mathbb{P}}$ reflects as well as preserves order, so that $p \leq_{\mathbb{P}} p'$ iff $\mathrm{y}_{\mathbb{P}}p \subseteq \mathrm{y}_{\mathbb{P}}p'$, and $\mathrm{y}_{\mathbb{P}}$ thus "embeds" $\mathbb{P}$ in $\widehat{\mathbb{P}}$. We clearly have $\mathrm{y}_{\mathbb{P}}p \subseteq X$ iff $p \in X$ and prime algebraicity of $\widehat{\mathbb{P}}$ amounts to saying that any $X \in \widehat{\mathbb{P}}$ is the union of its elements:

$$X = \bigcup_{p\in X} \mathrm{y}_{\mathbb{P}}p \ . \tag{2.6}$$

Finally, $\widehat{\mathbb{P}}$ is characterised abstractly as the *free join-completion* of $\mathbb{P}$, meaning (i) it is join-complete and (ii) given any join-complete poset C and a monotone map $f : \mathbb{P} \to C$, there is a unique join-preserving map $f^\dagger : \widehat{\mathbb{P}} \to C$ such that the diagram on the left below commutes.

$$\begin{tikzcd} \mathbb{P} \arrow[r, "\mathrm{y}_{\mathbb{P}}"] \arrow[dr, "f"'] & \widehat{\mathbb{P}} \arrow[d, dotted, "f^\dagger"] \\ & C \end{tikzcd} \qquad f^\dagger X = \bigcup_{p\in X} fp \ . \tag{2.7}$$

We call $f^\dagger$ *the extension of f along* $\mathrm{y}_{\mathbb{P}}$. Uniqueness of $f^\dagger$ follows from (2.6).

Notice that we may instantiate C to any poset of the form $\widehat{\mathbb{Q}}$, drawing our attention to join-preserving maps $\widehat{\mathbb{P}} \to \widehat{\mathbb{Q}}$. By the freeness property (2.7), join-preserving maps $\widehat{\mathbb{P}} \to \widehat{\mathbb{Q}}$ are in bijective correspondence with monotone maps $\mathbb{P} \to \widehat{\mathbb{Q}}$. Each element Y of $\widehat{\mathbb{Q}}$ can be represented using its "characteristic function", a monotone map $f_Y : \mathbb{Q}^{\mathrm{op}} \to \mathbf{2}$ from the opposite order to the simple poset $0 < 1$ such that $Y = \{q : f_Y q = 1\}$ and $\widehat{\mathbb{Q}} \cong [\mathbb{Q}^{\mathrm{op}}, \mathbf{2}]$. Uncurrying then yields the following chain:

$$[\mathbb{P}, \widehat{\mathbb{Q}}] \cong [\mathbb{P}, [\mathbb{Q}^{\mathrm{op}}, \mathbf{2}]] \cong [\mathbb{P} \times \mathbb{Q}^{\mathrm{op}}, \mathbf{2}] = [(\mathbb{P}^{\mathrm{op}} \times \mathbb{Q})^{\mathrm{op}}, \mathbf{2}] \cong \widehat{\mathbb{P}^{\mathrm{op}} \times \mathbb{Q}} \ . \tag{2.8}$$

So the order $\mathbb{P}^{\mathrm{op}} \times \mathbb{Q}$ provides a function space type. We'll now investigate what additional type structure is at hand.

2.3 Linear and Nonlinear Maps

Write **Lin** for the category with path orders $\mathbb{P}, \mathbb{Q}, \ldots$ as objects and join-preserving maps $\widehat{\mathbb{P}} \to \widehat{\mathbb{Q}}$ as arrows. It turns out **Lin** has enough structure to be understood as a categorical model of Girard's linear logic [27, 83]. Accordingly, we'll call arrows of **Lin** *linear* maps.

Linear maps are represented by elements of $\widehat{\mathbb{P}^{\mathrm{op}} \times \mathbb{Q}}$ and so by downwards-closed subsets of the order $\mathbb{P}^{\mathrm{op}} \times \mathbb{Q}$. This relational presentation exposes an involution central in understanding **Lin** as a categorical model of classical linear logic. The involution of linear logic, yielding $\mathbb{P}^{\perp}$ on an object $\mathbb{P}$, is given by $\mathbb{P}^{\mathrm{op}}$; clearly, downwards-closed subsets of $\mathbb{P}^{\mathrm{op}} \times \mathbb{Q}$ correspond to downwards-closed subsets of $(\mathbb{Q}^{\mathrm{op}})^{\mathrm{op}} \times \mathbb{P}^{\mathrm{op}}$, showing how maps $\mathbb{P} \to \mathbb{Q}$ correspond to maps $\mathbb{Q}^{\perp} \to \mathbb{P}^{\perp}$ in **Lin**. The tensor product of $\mathbb{P}$ and $\mathbb{Q}$ is given by the product of preorders $\mathbb{P} \times \mathbb{Q}$; the singleton order $\mathbb{1}$ is a unit for tensor. Linear function space $\mathbb{P} \multimap \mathbb{Q}$ is then obtained as $\mathbb{P}^{\mathrm{op}} \times \mathbb{Q}$. Products $\mathbb{P} \,\&\, \mathbb{Q}$ are given by $\mathbb{P} + \mathbb{Q}$, the disjoint juxtaposition of preorders. An element of $\widehat{\mathbb{P} \,\&\, \mathbb{Q}}$ can be identified with a pair (X, Y) with $X \in \widehat{\mathbb{P}}$ and $Y \in \widehat{\mathbb{Q}}$, which provides the projections $\pi_1 : \mathbb{P} \,\&\, \mathbb{Q} \to \mathbb{P}$ and $\pi_2 : \mathbb{P} \,\&\, \mathbb{Q} \to \mathbb{Q}$ in **Lin**. More general, not just binary, products $\mathbin{\&}_{i \in I} \mathbb{P}_i$ with projections π_j, for $j \in I$, are defined similarly. From the universal property of products, a collection of maps $f_i : \mathbb{P} \to \mathbb{P}_i$, for $i \in I$, can be tupled together to form a unique map $\langle f_i \rangle_{i \in I} : \mathbb{P} \to \mathbin{\&}_{i \in I} \mathbb{P}_i$ with the property that $\pi_j \circ \langle f_i \rangle_{i \in I} = f_j$ for all $j \in I$. The empty product is given by the empty order $\mathbb{O}$ and, as the terminal object, is associated with unique maps $\varnothing_{\mathbb{P}} : \mathbb{P} \to \mathbb{O}$, constantly $\varnothing$, for any path order $\mathbb{P}$. All told, **Lin** is a $*$-autonomous category, so a symmetric monoidal closed category with a dualising object, and has finite products (indeed, all products) as required by Seely's definition of a model of linear logic [83].

In fact, **Lin** also has all coproducts, also given on objects $\mathbb{P}$ and $\mathbb{Q}$ by the juxtaposition $\mathbb{P} + \mathbb{Q}$ and so coinciding with products. Injection maps $in_1 : \mathbb{P} \to \mathbb{P} + \mathbb{Q}$ and $in_2 : \mathbb{Q} \to \mathbb{P} + \mathbb{Q}$ in **Lin** derive from the obvious injections into the disjoint sum of preorders. The empty coproduct is the empty order $\mathbb{O}$ which is then a zero object. This collapse of products and coproducts highlights that **Lin** has arbitrary *biproducts*. Via the isomorphism $\mathbf{Lin}(\mathbb{P}, \mathbb{Q}) \cong \widehat{\mathbb{P}^{\mathrm{op}} \times \mathbb{Q}}$, each homset of **Lin** can be seen as a commutative monoid with neutral element the always $\varnothing$ map, itself written $\varnothing : \mathbb{P} \to \mathbb{Q}$, and multiplication given by union, written $+$. Composition in **Lin** is bilinear in that, given $f, f' : \mathbb{P} \to \mathbb{Q}$ and $g, g' : \mathbb{Q} \to \mathbb{R}$, we have $(g + g') \circ (f + f') = g \circ f + g \circ f' + g' \circ f + g' \circ f'$. Further, given a family of objects $(\mathbb{P}_\alpha)_{\alpha \in A}$, we have for each $\beta \in A$ a diagram

$$\mathbb{P}_\beta \underset{in_\beta}{\overset{\pi_\beta}{\rightleftarrows}} \Sigma_{\alpha \in A} \mathbb{P}_\alpha \quad \text{such that} \quad \begin{aligned} \pi_\beta \circ in_\beta &= 1_{\mathbb{P}_\beta} \; , \\ \pi_\beta \circ in_\alpha &= \varnothing \text{ if } \alpha \neq \beta, \text{ and} \\ \Sigma_{\alpha \in A} (in_\alpha \circ \pi_\alpha) &= 1_{\Sigma_{\alpha \in A} \mathbb{P}_\alpha} \; . \end{aligned} \tag{2.9}$$

Processes of type $\Sigma_{\alpha \in A} \mathbb{P}_\alpha$ may intuitively perform computation paths in any of the component path orders $\mathbb{P}_\alpha$.

We see that **Lin** is rich in structure. But linear maps alone are too restrictive. Being join-preserving, they in particular preserve the empty join. So, unlike e.g. prefixing, linear maps always send the inactive process $\varnothing$ to itself.

2.3.1 Continuous Maps

Looking for a broader notion of maps between nondeterministic domains we follow the discipline of linear logic and consider *non-linear* maps, i.e. maps whose domain is under an exponential, !. One choice of a suitable exponential for **Lin** is got by taking $!\mathbb{P}$ to be the preorder obtained as the free finite-join completion of $\mathbb{P}$. Concretely, $!\mathbb{P}$ can be defined to have finite subsets of $\mathbb{P}$ as elements with ordering given by (2.3).

Example 2.3 The path order $\mathbb{P}$ of Section 2.1 is built using this exponential and a particular choice of biproduct given by tagging with actions $\alpha \in A$. Because of its inductive definition, we may think of it as the least solution to the equation $\mathbb{P} = \Sigma_{\alpha \in A} !\mathbb{P}$. This will be made precise in Section 3.1. $\square$

When $!\mathbb{P}$ is quotiented by the equivalence induced by the preorder we obtain a poset which is the free finite-join completion of $\mathbb{P}$. By further using the obvious inclusion of this completion into $\widehat{\mathbb{P}}$, we get a map $i_\mathbb{P} : !\mathbb{P} \to \widehat{\mathbb{P}}$ with

$$i_\mathbb{P}\{p_1, \ldots, p_n\} = \mathrm{y}_\mathbb{P} p_1 + \cdots + \mathrm{y}_\mathbb{P} p_n \ . \tag{2.10}$$

Such finite sums of primes are the finite (isolated, compact) elements of $\widehat{\mathbb{P}}$. The map $i_\mathbb{P}$ assumes the role of $\mathrm{y}_\mathbb{P}$ above. For any $X \in \widehat{\mathbb{P}}$ and $P \in !\mathbb{P}$, we have $i_\mathbb{P} P \subseteq X$ iff $P \preceq_\mathbb{P} X$, and X is the directed join of the finite elements below it:

$$X = \textstyle\bigcup_{P \preceq_\mathbb{P} X} i_\mathbb{P} P \ . \tag{2.11}$$

Further, $\widehat{\mathbb{P}}$ is the *free directed-join completion* of $!\mathbb{P}$.[1] This means that given any monotone map $f : !\mathbb{P} \to C$ for some directed-join complete poset C, there is a unique directed-join preserving (i.e. Scott continuous) map $f^\ddagger : \widehat{\mathbb{P}} \to C$ such that the diagram below commutes.

$$\begin{tikzcd} !\mathbb{P} \arrow[r, "i_\mathbb{P}"] \arrow[dr, "f"'] & \widehat{\mathbb{P}} \arrow[d, dotted, "f^\ddagger"] \\ & C \end{tikzcd} \qquad f^\ddagger X = \textstyle\bigcup_{P \preceq_\mathbb{P} X} fP \ . \tag{2.12}$$

[1]This is also known as the *ideal completion* of $!\mathbb{P}$. We note that $\widehat{\mathbb{P}}$ is obtained by applying the "Hoare powerdomain" to the ideal completion of $\mathbb{P}$ [92]. No other powerdomains are considered here, but it may be possible e.g. to use the Smyth powerdomain to characterise the "must" rather than the "may" behaviour of processes.

Uniqueness of $f^\ddagger$, called the *extension of f along $i_\mathbb{P}$*, follows from (2.11). As before, we can replace C by a nondeterministic domain $\widehat{\mathbb{Q}}$ and by the freeness properties (2.7) and (2.12), there is a bijective correspondence between linear maps $!\mathbb{P} \to \mathbb{Q}$ and continuous maps $\widehat{\mathbb{P}} \to \widehat{\mathbb{Q}}$.

We define the category **Cts** to have path orders $\mathbb{P}, \mathbb{Q}, \ldots$ as objects and continuous maps $\widehat{\mathbb{P}} \to \widehat{\mathbb{Q}}$ as arrows. These arrows allow more process operations, including prefixing, to be expressed.

The structure of **Cts** is induced by that of **Lin** via an adjunction between the two categories. As linear maps are continuous, **Cts** has **Lin** as a subcategory, one which shares the same objects. We saw above that there is a bijection

$$\mathbf{Lin}(!\mathbb{P}, \mathbb{Q}) \cong \mathbf{Cts}(\mathbb{P}, \mathbb{Q}) \ . \tag{2.13}$$

This is in fact natural in $\mathbb{P}$ and $\mathbb{Q}$ so an adjunction with the inclusion $\mathbf{Lin} \hookrightarrow \mathbf{Cts}$ as right adjoint. Via (2.12) the map $\mathrm{y}_{!\mathbb{P}} : !\mathbb{P} \to \widehat{!\mathbb{P}}$ extends to a map $\eta_\mathbb{P} = \mathrm{y}_{!\mathbb{P}}^\ddagger : \mathbb{P} \to !\mathbb{P}$ in **Cts**. Conversely, $i_\mathbb{P} : !\mathbb{P} \to \widehat{\mathbb{P}}$ extends to a map $\varepsilon_\mathbb{P} = i_\mathbb{P}^\dagger : !\mathbb{P} \to \mathbb{P}$ in **Lin** using (2.7). These maps are the unit and counit, respectively, of the adjunction:

$$\begin{aligned} \eta_\mathbb{P} X &= \textstyle\bigcup_{P \preceq_\mathbb{P} X} \mathrm{y}_{!\mathbb{P}} P & \varepsilon_\mathbb{P} X &= \textstyle\bigcup_{P \in X} i_\mathbb{P} P \\ &= \{P \in !\mathbb{P} : P \preceq_\mathbb{P} X\} & &= \{p \in \mathbb{P} : \exists P \in X .\ p \in P\} \end{aligned} \tag{2.14}$$

Example 2.4 The prefix operation associated with the path order $\mathbb{P}$ of Section 2.1 is defined using the unit together with injections into the biproduct. In Chapter 3 we'll use the unit to interpret an anonymous action prefix operation and the counit to interpret a corresponding destructor. □

The left adjoint is the functor $! : \mathbf{Cts} \to \mathbf{Lin}$ given on arrows $f : \mathbb{P} \to \mathbb{Q}$ by $(\eta_\mathbb{Q} \circ f \circ i_\mathbb{P})^\dagger : !\mathbb{P} \to !\mathbb{Q}$. The bijection (2.13) then maps $g : !\mathbb{P} \to \mathbb{Q}$ in **Lin** to $\bar{g} = g \circ \eta_\mathbb{P} : \mathbb{P} \to \mathbb{Q}$ in **Cts** while its inverse maps $f : \mathbb{P} \to \mathbb{Q}$ in **Cts** to $\bar{f} = \varepsilon_\mathbb{Q} \circ !f$ in **Lin**. We call $\bar{g}$ and $\bar{f}$ the *transpose* of g and f, respectively; of course, transposing twice yields back the original map. As **Lin** is a subcategory of **Cts**, the counit is also a map in **Cts** and we have $\varepsilon_\mathbb{P} \circ \eta_\mathbb{P} = 1_\mathbb{P}$ while $X \subseteq \eta_\mathbb{P}(\varepsilon_\mathbb{P} X)$ for $X \in \widehat{!\mathbb{P}}$.

Right adjoints preserve products, and so **Cts** has finite products given as in **Lin**. Hence, **Cts** is a symmetric monoidal category like **Lin**, and in fact, our adjunction is symmetric monoidal. In detail, there are isomorphisms of path orders,

$$k : \mathbb{1} \cong !\mathbb{O} \quad \text{and} \quad m_{\mathbb{P},\mathbb{Q}} : !\mathbb{P} \times !\mathbb{Q} \cong !(\mathbb{P} \mathbin{\&} \mathbb{Q}) \ , \tag{2.15}$$

with $m_{\mathbb{P},\mathbb{Q}}$ mapping a pair $(P, Q) \in !\mathbb{P} \times !\mathbb{Q}$ to the union $in_1\, P \cup in_2\, Q$; any element of $!(\mathbb{P} \mathbin{\&} \mathbb{Q})$ can be written on this form. These isomorphisms induce isomorphisms with the same names in **Lin** with m natural. Moreover, k and m commute with the associativity, symmetry and unit maps of **Lin** and **Cts**, such as $s^{\mathbf{Lin}}_{\mathbb{P},\mathbb{Q}} : \mathbb{P} \times \mathbb{Q} \cong \mathbb{Q} \times \mathbb{P}$ and $r^{\mathbf{Cts}}_{\mathbb{Q}} : \mathbb{Q} \,\&\, \mathbb{O} \cong \mathbb{Q}$, making ! symmetric monoidal. It then follows [54] that the inclusion $\mathbf{Lin} \hookrightarrow \mathbf{Cts}$ is symmetric monoidal as well, and that the unit and counit are monoidal transformations. Thus, there are maps

$$l : \mathbb{O} \to \mathbb{1} \quad \text{and} \quad n_{\mathbb{P},\mathbb{Q}} : \mathbb{P} \,\&\, \mathbb{Q} \to \mathbb{P} \times \mathbb{Q} \tag{2.16}$$

in **Cts**, with n natural, corresponding to k and m above; l maps $\varnothing$ to $\{*\}$ while $n_{\mathbb{P},\mathbb{Q}}$ is the extension $h^\ddagger$ of the map $h(in_1\, P \cup in_2\, Q) = i_{\mathbb{P}}P \times i_{\mathbb{Q}}Q$. The unit also makes the diagrams below commute and the counit satisfies similar properties.

$$\begin{tikzcd}
 & \mathbb{P} \,\&\, \mathbb{Q} \arrow[dl, "\eta_{\mathbb{P}} \& \eta_{\mathbb{Q}}"'] \arrow[dr, "\eta_{\mathbb{P}\&\mathbb{Q}}"] & \\
!\mathbb{P} \,\&\, !\mathbb{Q} \arrow[r, "n_{!\mathbb{P},!\mathbb{Q}}"'] & !\mathbb{P} \times !\mathbb{Q} \arrow[r, "m_{\mathbb{P},\mathbb{Q}}"'] & !(\mathbb{P} \,\&\, \mathbb{Q})
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathbb{O} \arrow[r, "l"] \arrow[dr, "\eta_{\mathbb{O}}"'] & \mathbb{1} \arrow[d, "k"] \\
 & !\mathbb{O}
\end{tikzcd} \tag{2.17}$$

The diagram on the left can be written as $str_{\mathbb{P},\mathbb{Q}} \circ (1_{\mathbb{P}} \,\&\, \eta_{\mathbb{Q}}) = \eta_{\mathbb{P}\&\mathbb{Q}}$ where str, the *strength* of ! viewed as a monad on **Cts**, is the natural transformation

$$\mathbb{P} \,\&\, !\mathbb{Q} \xrightarrow{\eta_{\mathbb{P}} \& 1_{!\mathbb{Q}}} !\mathbb{P} \,\&\, !\mathbb{Q} \xrightarrow{n_{!\mathbb{P},!\mathbb{Q}}} !\mathbb{P} \times !\mathbb{Q} \xrightarrow{m_{\mathbb{P},\mathbb{Q}}} !(\mathbb{P} \,\&\, \mathbb{Q}) \quad . \tag{2.18}$$

Each map $str_{\mathbb{P},\mathbb{Q}}$ is linear in its second argument. Assuming $f : \mathbb{P}' \to \mathbb{P}$ and $g_i : \mathbb{Q}' \to !\mathbb{Q}$ for each $i \in I$, we have:

$$\begin{aligned}
& str_{\mathbb{P},\mathbb{Q}} \circ (f \,\&\, \Sigma_{i\in I} g_i) \\
&= m_{\mathbb{P},\mathbb{Q}} \circ n_{!\mathbb{P},!\mathbb{Q}} \circ (\eta_{\mathbb{P}} \,\&\, 1_{!\mathbb{Q}}) \circ (f \,\&\, \Sigma_{i\in I} g_i) \\
&= m_{\mathbb{P},\mathbb{Q}} \circ n_{!\mathbb{P},!\mathbb{Q}} \circ (\eta_{\mathbb{P}} f \,\&\, \Sigma_{i\in I} g_i) \\
&= m_{\mathbb{P},\mathbb{Q}} \circ (\eta_{\mathbb{P}} f \times \Sigma_{i\in I} g_i) \circ n_{\mathbb{P},\mathbb{Q}} && \text{(nat. of } n) \\
&= m_{\mathbb{P},\mathbb{Q}} \circ \Sigma_{i\in I} (\eta_{\mathbb{P}} f \times g_i) \circ n_{\mathbb{P},\mathbb{Q}} && \text{(bilinearity of } \times) \\
&= \Sigma_{i\in I} m_{\mathbb{P},\mathbb{Q}} \circ (\eta_{\mathbb{P}} f \times g_i) \circ n_{\mathbb{P},\mathbb{Q}} && \text{(linearity of } m_{\mathbb{P},\mathbb{Q}}) \\
&= \Sigma_{i\in I} m_{\mathbb{P},\mathbb{Q}} \circ n_{!\mathbb{P},!\mathbb{Q}} \circ (\eta_{\mathbb{P}} f \,\&\, g_i) && \text{(nat. of } n) \\
&= \Sigma_{i\in I} m_{\mathbb{P},\mathbb{Q}} \circ n_{!\mathbb{P},!\mathbb{Q}} \circ (\eta_{\mathbb{P}} \,\&\, 1_{!\mathbb{Q}}) \circ (f \,\&\, g_i) \\
&= \Sigma_{i\in I} str_{\mathbb{P},\mathbb{Q}} \circ (f \,\&\, g_i)
\end{aligned}$$

Finally, recall that the category **Lin** is symmetric monoidal closed so that the functor $(\mathbb{Q} \multimap -)$ is right adjoint to $(- \times \mathbb{Q})$ for any object $\mathbb{Q}$. Together with the natural isomorphism m this provides a right adjoint $(\mathbb{Q} \to -)$, defined by $(!\mathbb{Q} \multimap -)$, to the functor $(- \,\&\, \mathbb{Q})$ in **Cts** via the chain

$$\begin{aligned}
\mathbf{Cts}(\mathbb{P} \,\&\, \mathbb{Q}, \mathbb{R}) &\cong \mathbf{Lin}(!(\mathbb{P} \,\&\, \mathbb{Q}), \mathbb{R}) \cong \mathbf{Lin}(!\mathbb{P} \times !\mathbb{Q}, \mathbb{R}) \\
&\cong \mathbf{Lin}(!\mathbb{P}, !\mathbb{Q} \multimap \mathbb{R}) \cong \mathbf{Cts}(\mathbb{P}, !\mathbb{Q} \multimap \mathbb{R}) = \mathbf{Cts}(\mathbb{P}, \mathbb{Q} \to \mathbb{R})
\end{aligned} \tag{2.19}$$

—natural in $\mathbb{P}$ and $\mathbb{R}$. This demonstrates that **Cts** is cartesian closed, as is well known. The adjunction between **Lin** and **Cts** now satisfies the conditions put forward by Benton for a categorical model of intuitionistic linear logic, strengthening those of Seely [7, 83]; see also [54] for a recent survey of such models.

2.3.2 Affine Maps

The move from **Lin** to **Cts** has allowed us to interpret prefixing. In fact, we can do much the same more cheaply.

The category **Cts** is obtained from **Lin** using an exponential which allows arbitrary copying in linear logic. An element $P \in !\mathbb{P}$ consists of several, possibly no, computation paths of $\mathbb{P}$. An element of the path order $!\mathbb{P}$ can therefore be understood intuitively as describing a compound computation path associated with running several copies of a process of type $\mathbb{P}$. Maps $\mathbb{P} \to \mathbb{Q}$ of **Cts**, corresponding to maps $!\mathbb{P} \to \mathbb{Q}$ of **Lin**, allow their input to be copied.

However, copying is generally restricted in a distributed computation. A communication received is most often the result of a single run of the process communicated with. Of course, process code can be sent and copied. But generally the receiver has no possibility of rewinding or copying the state of an ongoing computation. On the other hand, ignoring another process is often easy. For this reason, many operations of distributed computation have the following property [64]:

> *Affine linearity:* a computation path of the process arising from the application of an operation to an input process has resulted from at most one computation path of the input process.

Note in particular that prefix operations are affine in this sense: if we wish to observe just the initial action of a CCS process $\mathsf{a}.t$, no computation path of t is needed, though observing any longer path will involve a (single) computation path of t.

Recall the diagram (2.7) which says that linear maps $\mathbb{P} \to \mathbb{Q}$ are determined by their values on single paths, elements of $\mathbb{P}$. Via the adjunction between **Lin** and **Cts**, continuous maps $\mathbb{P} \to \mathbb{Q}$ are determined by their values on compound paths in $!\mathbb{P}$ (diagram (2.12)). To summarise:

- linear operations use *a single* path of the input;
- affine operations use *at most one* path of the input;
- continuous operations use *any number of* paths of the input.

Affine maps are defined by their values on singleton copies of paths together with the empty path. Accordingly, affine maps derive from the lifting operation $(-)_\perp$ adding a new element $\perp$, to be thought of as the empty computation path, below a copy of a path order $\mathbb{P}$ to produce a path order $\mathbb{P}_\perp$. Abstractly, $\mathbb{P}_\perp$ is the empty-join completion of $\mathbb{P}$; concretely, we can take $\mathbb{P}_\perp$ to contain the empty set, written $\perp$, together with singletons $\{p\}$ for $p \in \mathbb{P}$, ordered by $\preceq_{\mathbb{P}}$.

Example 2.5 The path order A^+ of non-empty strings over A with prefix ordering discussed in Section 2.1 is obtained, to within isomorphism, as the least solution to the equation $\mathbb{P} = \Sigma_{\alpha \in A} \mathbb{P}_\perp$. □

There is an obvious inclusion of the poset obtained as the empty-join completion of $\mathbb{P}$ into $\widehat{\mathbb{P}}$, and so we obtain a map $j_\mathbb{P} : \mathbb{P}_\perp \to \widehat{\mathbb{P}}$ given by

$$j_\mathbb{P} \perp = \varnothing \quad \text{and} \quad j_\mathbb{P}\{p\} = \mathrm{y}_\mathbb{P} p \ . \tag{2.20}$$

We'll use P to range over $\mathbb{P}_\perp$ in the remainder of this chapter. The map $j_\mathbb{P}$ assumes the role of $i_\mathbb{P}$; for any $X \in \widehat{\mathbb{P}}$ and $P \in \mathbb{P}_\perp$ we have $j_\mathbb{P} P \subseteq X$ iff $P \preceq_\mathbb{P} X$, and from (2.6) we get

$$X = \bigcup_{p \in X} \mathrm{y}_\mathbb{P} p = \varnothing \cup \bigcup_{p \in X} \mathrm{y}_\mathbb{P} p = \bigcup_{P \preceq_\mathbb{P} X} j_\mathbb{P} P \ . \tag{2.21}$$

This join is manifestly nonempty and in fact, $\widehat{\mathbb{P}}$ is the free closure of $\mathbb{P}_\perp$ under nonempty joins. This means that given any monotone map $f : \mathbb{P}_\perp \to C$ for some nonempty-join complete poset C, there is a unique nonempty-join preserving (i.e. *affine*) map $f^\S : \widehat{\mathbb{P}} \to C$ such that the diagram below commutes:

$$\begin{tikzcd} \mathbb{P}_\perp \arrow[r, "j_\mathbb{P}"] \arrow[dr, "f"'] & \widehat{\mathbb{P}} \arrow[d, dotted, "f^\S"] \\ & C \end{tikzcd} \qquad f^\S X = \bigcup_{P \preceq_\mathbb{P} X} f P \ . \tag{2.22}$$

Uniqueness of $f^\S$, called the *extension of f along $j_\mathbb{P}$*, follows from (2.21). As before, we can replace C by a nondeterministic domain $\widehat{\mathbb{Q}}$ and by the freeness properties (2.7) and (2.22), there is a bijective correspondence between linear maps $\mathbb{P}_\perp \to \mathbb{Q}$ and affine maps $\widehat{\mathbb{P}} \to \widehat{\mathbb{Q}}$.

We define the category **Aff** to have path orders $\mathbb{P}, \mathbb{Q}, \ldots$ as objects and affine maps $\widehat{\mathbb{P}} \to \widehat{\mathbb{Q}}$ as arrows. Again, the structure of **Aff** is induced by that of **Lin** via an adjunction between the two categories with the inclusion **Lin** $\hookrightarrow$ **Aff** (linear maps are affine) as right adjoint:

$$\mathbf{Lin}(\mathbb{P}_\perp, \mathbb{Q}) \cong \mathbf{Aff}(\mathbb{P}, \mathbb{Q}) \ . \tag{2.23}$$

The unit $\eta_\mathbb{P} : \mathbb{P} \to \mathbb{P}_\perp$ in **Aff**, the counit $\varepsilon_\mathbb{P} : \mathbb{P}_\perp \to \mathbb{P}$ in **Lin**, and the left adjoint $(-)_\perp : \mathbf{Aff} \to \mathbf{Lin}$ are obtained precisely as in Section 2.3.1:

$$\begin{aligned} \eta_\mathbb{P} X &= \bigcup_{P \preceq_\mathbb{P} X} \mathrm{y}_{\mathbb{P}_\perp} P & \varepsilon_\mathbb{P} X &= \bigcup_{P \in X} j_\mathbb{P} P \\ &= \{P \in \mathbb{P}_\perp : P \preceq_\mathbb{P} X\} & &= \{p \in \mathbb{P} : \exists P \in X.\ p \in P\} \\ &= \{\varnothing\} \cup \{\{p\} : p \in X\} & &= \{p \in \mathbb{P} : \{p\} \in X\} \end{aligned} \tag{2.24}$$

Example 2.6 This time, the unit together with tagging can be used to interpret the usual prefix operations associated with the path order A^+. Again, we'll base an anonymous action prefix operation directly on the unit in Chapter 4. □

The category **Aff** inherits products $\Sigma_{\alpha\in A}\mathbb{P}_\alpha$ from **Lin** in the same way as **Cts** does. However, unlike **Cts**, the category **Aff** is not cartesian closed because $\mathbb{P}_\perp \times \mathbb{Q}_\perp$ and $(\mathbb{P} \mathbin{\&} \mathbb{Q})_\perp$ are not isomorphic in **Lin**. On the other hand we can easily define a tensor operation $\otimes$ on **Aff** such that the path orders $\mathbb{P}_\perp \times \mathbb{Q}_\perp$ and $(\mathbb{P}\otimes\mathbb{Q})_\perp$ become isomorphic: simply take $\mathbb{P}\otimes\mathbb{Q}$ to be $(\mathbb{P}_\perp \times \mathbb{Q}_\perp) \setminus \{(\perp,\perp)\}$. Paths of $\mathbb{P}\otimes\mathbb{Q}$ then consist of a (possibly empty) path of $\mathbb{P}$ and a (possibly empty) path of $\mathbb{Q}$, and so a path set $X \in \widehat{\mathbb{P}\otimes\mathbb{Q}}$ can be thought of as a process performing two parallel computation paths, one of type $\mathbb{P}$ and one of type $\mathbb{Q}$. On arrows $f : \mathbb{P} \to \mathbb{P}'$ and $g : \mathbb{Q} \to \mathbb{Q}'$ in **Aff**, we define $f \otimes g : \mathbb{P}\otimes\mathbb{Q} \to \mathbb{P}'\otimes\mathbb{Q}'$ as the extension $h^\S$ of the map $h : \mathbb{P}_\perp \times \mathbb{Q}_\perp \cong (\mathbb{P}\otimes\mathbb{Q})_\perp \to \widehat{\mathbb{P}'\otimes\mathbb{Q}'}$ defined by

$$(P', Q') \in h(P, Q) \iff P' \in f_\perp(\mathrm{y}_{\mathbb{P}_\perp} P) \text{ and } Q' \in g_\perp(\mathrm{y}_{\mathbb{Q}_\perp} Q)\ . \tag{2.25}$$

The unit of tensor is the empty path order $\mathbb{O}$. Elements $X \in \widehat{\mathbb{P}}$ correspond to maps $\bar{X} : \mathbb{O} \to \mathbb{P}$ in **Aff** and with $Y \in \widehat{\mathbb{Q}}$, we'll write $X \otimes Y$ for the element of $\widehat{\mathbb{P}\otimes\mathbb{Q}}$ pointed to by the map $\bar{X} \otimes \bar{Y}$.

The tensor makes **Aff** a symmetric monoidal category, and again, the adjunction (2.23) is symmetric monoidal. The obvious isomorphisms of path orders,

$$\mathbb{1} \cong \mathbb{O}_\perp \quad\text{and}\quad \mathbb{P}_\perp \times \mathbb{Q}_\perp \cong (\mathbb{P}\otimes\mathbb{Q})_\perp\ , \tag{2.26}$$

induce natural isomorphisms in **Lin** and we obtain a monoidal strength $\mathbb{P}\otimes\mathbb{Q}_\perp \to (\mathbb{P}\otimes\mathbb{Q})_\perp$ precisely as for **Cts**.

Finally, the monoidal closed structure of **Lin** together with the natural isomorphism $\mathbb{P}_\perp \times \mathbb{Q}_\perp \cong (\mathbb{P}\otimes\mathbb{Q})_\perp$ provide a right adjoint $(\mathbb{Q} \multimap -)$, defined by $(\mathbb{Q}_\perp \multimap -)$, to the functor $(- \otimes \mathbb{Q})$ in **Aff** via the chain

$$\begin{aligned}\mathbf{Aff}(\mathbb{P}\otimes\mathbb{Q}, \mathbb{R}) &\cong \mathbf{Lin}((\mathbb{P}\otimes\mathbb{Q})_\perp, \mathbb{R}) \cong \mathbf{Lin}(\mathbb{P}_\perp \times \mathbb{Q}_\perp, \mathbb{R}) \\ &\cong \mathbf{Lin}(\mathbb{P}_\perp, \mathbb{Q}_\perp \multimap \mathbb{R}) \cong \mathbf{Aff}(\mathbb{P}, \mathbb{Q}_\perp \multimap \mathbb{R}) = \mathbf{Aff}(\mathbb{P}, \mathbb{Q} \multimap \mathbb{R})\end{aligned} \tag{2.27}$$

—natural in $\mathbb{P}$ and $\mathbb{R}$. This demonstrates that **Aff** is symmetric monoidal closed and since the unit of the tensor is terminal, a model of affine linear logic, as already observed by Jacobs [40].

Chapter 3

HOPLA—A Higher-Order Process Language

HOPLA (Higher-Order Process LAnguage [65, 66, 67]) is an economic yet expressive language for higher-order nondeterministic processes. The language is typed. The type of a process describes the possible computation paths the process can perform. Computation paths may be of the kind considered in Example 2.2, but they may also represent the input-output behaviour of a process as in Hennessy's work [32]. A typing judgement

$$x_1 : \mathbb{P}_1, \dots, x_k : \mathbb{P}_k \vdash t : \mathbb{Q} \tag{3.1}$$

means that a process t yields computation paths in $\mathbb{Q}$ once processes with computation paths in $\mathbb{P}_1, \dots, \mathbb{P}_k$ are assigned to the variables $x_1, \dots, x_k$ respectively. The types $\mathbb{P}$ are built using function space, sum, an anonymous action prefix type, and recursive definition. The language can be viewed as an extension of the simply-typed lambda calculus. Indeed, it will be interpreted in the cartesian closed category **Cts** with types being interpreted as path orders and judgements like (3.1) interpreted as arrows

$$\mathbb{P}_1 \mathbin{\&} \cdots \mathbin{\&} \mathbb{P}_k \to \mathbb{Q} \tag{3.2}$$

of **Cts**. In general, the language allows processes to be copied and discarded arbitrarily. In particular, the language will allow us to write terms which take a process as argument, copy it, and then set those copies in parallel composition with each other. However some operations are by nature linear in certain arguments. Linearity, detected in the denotational semantics as being interpreted in the subcategory **Lin** of **Cts**, amounts to the property of preserving nondeterministic sums.

The denotational semantics, given in Section 3.1, is canonical in the sense that the operations of the language arise from canonical constructions in the underlying model. Universal properties of these constructions induce a number of useful results about the semantics which are collected in Section 3.2. Section 3.3 shows that the denotational semantics is fully abstract, characterising contextual equivalence as path equivalence. HOPLA supports a straightforward operational semantics, given in Section 3.4 together with simple proofs of soundness and adequacy. A standard notion of bisimulation is shown to be a congruence in Section 3.5 while full abstraction links contextual equivalence to simulation equivalence via a fragment of Hennessy-Milner logic. It is notable that although we can express many kinds of concurrent processes in the language, the language itself does not have many features typical of process calculi built-in, beyond that of a nondeterministic sum and prefix operations. In particular, parallel composition of processes a la CCS can be *defined* in HOPLA, and is not a primitive construct. Section 3.6 contains encodings of CCS, CCS with process passing and mobile ambients with public names.

Before formally defining the language, we discuss informally the constructs associated with the sum and prefix types—the remainder of the language is just the simply-typed lambda calculus plus nondeterministic sums, and so the sum and prefix types are central to expressiveness.

The sum type constructed from a family of types $(\mathbb{P}_\alpha)_{\alpha\in A}$ is written $\Sigma_{\alpha\in A}\mathbb{P}_\alpha$. Interpreted using the biproduct of **Lin**, the sum is both a product and a coproduct. It is associated with *injection* ("tagging") term constructors, producing a term βt of sum type from a term t of type $\mathbb{P}_\beta$, with $\beta \in A$. Conversely, *projection* term constructors produce a term $\pi_\beta t$ of type $\mathbb{P}_\beta$ from a term t of the sum type above. Injections and projections are interpreted as the structural maps of the biproduct of **Lin**, cf. (2.9). Accordingly, the semantics will identify the process $\pi_\beta(\beta t)$ with the process t and $\pi_\alpha(\beta t)$ with the inactive process $\varnothing$, if $\alpha \neq \beta$. Moreover, injections and projections distribute over nondeterministic sum by linearity, and so the processes $\beta(\Sigma_{i\in I}t_i)$ and $\Sigma_{i\in I}(\beta t_i)$ are identified, as are $\pi_\beta(\Sigma_{i\in I}t_i)$ and $\Sigma_{i\in I}(\pi_\beta t_i)$.

A prefix type has the form $!\mathbb{P}$; it describes computation paths in which first an anonymous action, which we call "!", is performed before resuming as a computation path in $\mathbb{P}$. The prefix type is associated with a *prefix operation* taking a process t of type $\mathbb{P}$ to $!t$ of type $!\mathbb{P}$, as well as a *prefix match* $[u > !x \Rightarrow t]$, where u has prefix type $!\mathbb{P}$, the variable x has type $\mathbb{P}$, and t generally involves x. The term $[u > !x \Rightarrow t]$ matches u against the pattern $!x$ and passes the results of successful matches for x on to t. In particular, first prefixing and then matching yields a single successful match in that the processes $[!u > !x \Rightarrow t]$ and $t[u/x]$ are identified. Prefix match is linear in the argument u so that the possibly multiple results of successful matches are nondeterministically summed together; the interpretations of $[\Sigma_{i\in I}u_i > !x \Rightarrow t]$ and $\Sigma_{i\in I}[u_i > !x \Rightarrow t]$ are identical.

Using the sum type to give names to anonymous actions, we can encode CCS-like processes. Recall from Example 2.3 the recursive type of CCS processes given by the equation $\mathbb{P} = \Sigma_{\alpha\in A}!\mathbb{P}$. From a CCS process, a term t of type $\mathbb{P}$, we obtain a term $!t$ of type $!\mathbb{P}$ by anonymous prefixing, and then $\beta!t$ of type $\Sigma_{\alpha\in A}!\mathbb{P}$ by tagging with any $\beta \in A$. Identifying the recursive type $\mathbb{P}$ with its unfolding $\Sigma_{\alpha\in A}!\mathbb{P}$, we then have $\beta!t$ of type $\mathbb{P}$, corresponding to the CCS process obtained by prefixing t by the action β.

Example 3.1 Our CCS processes $\mathsf{a.b.}\varnothing + \mathsf{a.c.}\varnothing$ and $\mathsf{a.(b.}\varnothing + \mathsf{c.}\varnothing)$ can be translated to the HOPLA terms

$$u_1 \equiv_{\text{def}} \mathsf{a!b!}\varnothing + \mathsf{a!c!}\varnothing \quad \text{and} \quad u_2 \equiv_{\text{def}} \mathsf{a!(b!}\varnothing + \mathsf{c!}\varnothing) \ , \tag{3.3}$$

both of type $\mathbb{P}$. The following process identities are easily deduced from the properties of projection and prefix match:

$$\begin{aligned} &[\pi_{\mathsf{a}} u_1 > !x \Rightarrow t] = t[\mathsf{b!}\varnothing/x] + t[\mathsf{c!}\varnothing/x] \\ &[\pi_{\mathsf{a}} u_2 > !x \Rightarrow t] = t[(\mathsf{b!}\varnothing + \mathsf{c!}\varnothing)/x] \\ &[\pi_{\mathsf{b}} u_i > !x \Rightarrow t] = [\pi_{\mathsf{c}} u_i > !x \Rightarrow t] = \varnothing \quad \text{for } i = 1, 2 \end{aligned} \tag{3.4}$$

Projection and prefix match in combination can thus be used to test whether a process of type $\mathbb{P}$ can perform a given CCS action. By linearity, each possible successor gives rise to a different component of a nondeterministic sum; if there are no successors, we get the inactive process.

A *nonlinear* term can be used to test for the ability to perform two different actions. If $t \equiv [\pi_{\mathsf{b}} x > !x' \Rightarrow [\pi_{\mathsf{c}} x > !x'' \Rightarrow !\varnothing]]$ we get

$$\begin{aligned} t[\mathsf{b!}\varnothing/x] + t[\mathsf{c!}\varnothing/x] &= \varnothing \\ t[(\mathsf{b!}\varnothing + \mathsf{c!}\varnothing)/x] &= !\varnothing \end{aligned} \tag{3.5}$$

Note how the two occurrences of x in t allow the process $t(-)$ to copy its input and then force each copy to take a different computation path. Intuitively, $t(-)$ uses two computation paths of its input and is thus not linear (cf. the discussion in Section 2.3.2); semantically, $t(-)$ does not preserve nondeterministic sums, as (3.5) shows, and so its interpretation lies outside **Lin**.

So HOPLA has contexts that are strong enough to distinguish between our two vending machine processes. The full abstraction result of Section 3.3 shows that this follows from the fact that the path set interpretations of these processes are different, cf. Example 2.2. □

The example illustrates a way of combining the sum and prefix types which yields the "prefix-sum" type of the original treatment of HOPLA [65]. The decomposition embodied in HOPLA as defined below increases the expressiveness of the language (for example, to include the type used by Winskel for CCS with late value-passing [97]), while still ensuring that every operation in the language has a canonical semantics.

3.1 Denotational Semantics

Types are given by the grammar

$$\mathbb{T} ::= \mathbb{T}_1 \to \mathbb{T}_2 \mid \Sigma_{\alpha\in A}\mathbb{T}_\alpha \mid !\mathbb{T} \mid T \mid \mu_j\vec{T}.\vec{\mathbb{T}} \ . \tag{3.6}$$

The symbol T is drawn from a set of type variables used in defining recursive types; closed type expressions are interpreted as path orders. Using vector notation, $\mu_j\vec{T}.\vec{\mathbb{T}}$ abbreviates $\mu_j T_1,\dots,T_k.(\mathbb{T}_1,\dots,\mathbb{T}_k)$ and is interpreted as the j-component, for $1 \le j \le k$, of "the least" solution to the defining equations $T_1 = \mathbb{T}_1,\dots,T_k = \mathbb{T}_k$, in which the expressions $\mathbb{T}_1,\dots,\mathbb{T}_k$ may contain the T_j's. We shall write $\mu\vec{T}.\vec{\mathbb{T}}$ as an abbreviation for the k-tuple with j-component $\mu_j\vec{T}.\vec{\mathbb{T}}$, and confuse a closed expression for a path order with the path order itself. Simultaneous recursive equations for path orders can be solved using information systems [81, 48]. Here, it will be convenient to give a concrete, inductive characterisation based on a language of *paths*:

$$p, q ::= P \mapsto q \mid \beta p \mid P \mid \mathit{abs}\, p \ . \tag{3.7}$$

Above, P ranges over finite sets of paths. We use $P \mapsto q$ as notation for pairs in the function space $(!\mathbb{P})^{\mathrm{op}} \times \mathbb{Q}$. The language is complemented by formation rules using judgements $p : \mathbb{P}$, meaning that p belongs to $\mathbb{P}$, displayed below alongside rules defining the ordering on $\mathbb{P}$ using judgements $p \le_{\mathbb{P}} p'$. Recall that $P \preceq_{\mathbb{P}} P'$ means $\forall p \in P.\exists p' \in P'.\ p \le_{\mathbb{P}} p'$.

$$\begin{array}{ll}
\dfrac{P : !\mathbb{P} \quad q : \mathbb{Q}}{P \mapsto q : \mathbb{P} \to \mathbb{Q}} & \dfrac{P' \le_{!\mathbb{P}} P \quad q \le_{\mathbb{Q}} q'}{P \mapsto q \le_{\mathbb{P}\to\mathbb{Q}} P' \mapsto q'} \\[2ex]
\dfrac{p : \mathbb{P}_\beta \quad \beta \in A}{\beta p : \Sigma_{\alpha\in A}\mathbb{P}_\alpha} & \dfrac{p \le_{\mathbb{P}_\beta} p'}{\beta p \le_{\Sigma_{\alpha\in A}\mathbb{P}_\alpha} \beta p'} \\[2ex]
\dfrac{p_1 : \mathbb{P} \cdots p_n : \mathbb{P}}{\{p_1,\dots,p_n\} : !\mathbb{P}} & \dfrac{P \preceq_{\mathbb{P}} P'}{P \le_{!\mathbb{P}} P'} \\[2ex]
\dfrac{p : \mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}]}{\mathit{abs}\, p : \mu_j\vec{T}.\vec{\mathbb{T}}} & \dfrac{p \le_{\mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}]} p'}{\mathit{abs}\, p \le_{\mu_j\vec{T}.\vec{\mathbb{T}}} \mathit{abs}\, p'}
\end{array} \tag{3.8}$$

Using information systems as in [48] yields the same representation, except for the tagging with *abs* in recursive types, done to help in the proof of adequacy in Section 3.4.1. So rather than the straight equality between a recursive type and its unfolding which we are used to from [48], we get an isomorphism $\mathit{abs} : \mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}] \cong \mu_j\vec{T}.\vec{\mathbb{T}}$ whose inverse we call *rep*.

Example 3.2 The least solution to the equation $\mathbb{P} = \Sigma_{\alpha\in A}!\mathbb{P}$ of Example 2.3 is formally written $\mu T.\Sigma_{\alpha\in A}!T$. By composing the rules for the sum and prefix types above we obtain a rule for deriving its elements which is identical to (2.2) except for the *abs* tag:

$$\frac{p_1 : \mathbb{P} \cdots p_k : \mathbb{P} \quad \alpha \in A}{\mathit{abs}\, \alpha\{p_1,\dots,p_k\} : \mathbb{P}} \tag{3.9}$$

□

The raw syntax of HOPLA terms is given by

$$\begin{array}{lll} t, u ::= x, y, z, \dots & \text{(variables)} \\ \quad \mid \mathit{rec}\, x.t & \text{(recursive definition)} \\ \quad \mid \Sigma_{i\in I} t_i & \text{(nondeterministic sum)} \\ \quad \mid \lambda x.t \mid t\ u & \text{(abstraction and application)} \\ \quad \mid \beta t \mid \pi_\beta t & \text{(injection and projection)} \\ \quad \mid !t \mid [u > !x \Rightarrow t] & \text{(prefix operation and match)} \\ \quad \mid \mathit{abs}\, t \mid \mathit{rep}\, t & \text{(folding and unfolding)} \end{array} \tag{3.10}$$

The variable x in a match term $[u > !x \Rightarrow t]$ is a binding occurrence with scope t.

Let $\mathbb{P}_1, \dots, \mathbb{P}_k, \mathbb{Q}$ be closed type expressions and assume that the variables $x_1, \dots, x_k$ are distinct. A syntactic judgement

$$x_1 : \mathbb{P}_1, \dots, x_k : \mathbb{P}_k \vdash t : \mathbb{Q} \tag{3.11}$$

stands for a map

$$[\![x_1 : \mathbb{P}_1, \dots, x_k : \mathbb{P}_k \vdash t : \mathbb{Q}]\!] : \mathbb{P}_1 \,\&\, \cdots \,\&\, \mathbb{P}_k \to \mathbb{Q} \tag{3.12}$$

in **Cts**. We'll write Γ, or Λ, for an environment list $x_1 : \mathbb{P}_1, \dots, x_k : \mathbb{P}_k$ and most often abbreviate the denotation to $\mathbb{P}_1 \,\&\, \cdots \,\&\, \mathbb{P}_k \xrightarrow{t} \mathbb{Q}$, or $\Gamma \xrightarrow{t} \mathbb{Q}$, or even $[\![t]\!]$, suppressing the typing information. When the environment list is empty, the corresponding product is the empty path order $\mathbb{O}$.

The term-formation rules are displayed below alongside their interpretations as constructors on maps of **Cts**, taking the maps denoted by the premises to that denoted by the conclusion (cf. [12]). We assume that the variables in any environment list which appears are distinct.

Structural rules. The rules handling environment lists are given as follows:

$$\frac{}{x : \mathbb{P} \vdash x : \mathbb{P}} \qquad \frac{}{\mathbb{P} \xrightarrow{1_\mathbb{P}} \mathbb{P}} \tag{3.13}$$

$$\frac{\Gamma \vdash t : \mathbb{Q}}{\Gamma, x : \mathbb{P} \vdash t : \mathbb{Q}} \qquad \frac{\Gamma \xrightarrow{t} \mathbb{Q}}{\Gamma \,\&\, \mathbb{P} \xrightarrow{1_\Gamma \& \varnothing_\mathbb{P}} \Gamma \,\&\, \mathbb{O} \xrightarrow{r^{\mathbf{Cts}}_\mathbb{Q}} \Gamma \xrightarrow{t} \mathbb{Q}} \tag{3.14}$$

$$\frac{\Gamma, y : \mathbb{Q}, x : \mathbb{P}, \Lambda \vdash t : \mathbb{R}}{\Gamma, x : \mathbb{P}, y : \mathbb{Q}, \Lambda \vdash t : \mathbb{R}} \qquad \frac{\Gamma \,\&\, \mathbb{Q} \,\&\, \mathbb{P} \,\&\, \Lambda \xrightarrow{t} \mathbb{R}}{\Gamma \,\&\, \mathbb{P} \,\&\, \mathbb{Q} \,\&\, \Lambda \xrightarrow{t \circ (1_\Gamma \& s^{\mathbf{Cts}}_{\mathbb{P},\mathbb{Q}} \& 1_\Lambda)} \mathbb{R}} \tag{3.15}$$

$$\frac{\Gamma, x : \mathbb{P}, y : \mathbb{P} \vdash t : \mathbb{Q}}{\Gamma, z : \mathbb{P} \vdash t[z/x, z/y] : \mathbb{Q}} \qquad \frac{\Gamma \,\&\, \mathbb{P} \,\&\, \mathbb{P} \xrightarrow{t} \mathbb{Q}}{\Gamma \,\&\, \mathbb{P} \xrightarrow{1_\Gamma \& \Delta_\mathbb{P}} \Gamma \,\&\, \mathbb{P} \,\&\, \mathbb{P} \xrightarrow{t} \mathbb{Q}} \tag{3.16}$$

In the formation rule for contraction (3.16), the variable z must be fresh; the map $\Delta_\mathbb{P}$ is the usual diagonal, given as $\langle 1_\mathbb{P}, 1_\mathbb{P} \rangle$. We'll write $\Delta^k_\mathbb{P} : \mathbb{P} \to \mathbb{P}^k$ for the obvious extension to k components, $\mathbb{P}^k = \mathbb{P} \,\&\, \cdots \,\&\, \mathbb{P}$. Note that $\Delta^0_\mathbb{P} = \varnothing_\mathbb{P} : \mathbb{P} \to \mathbb{O}$ and $\Delta^1_\mathbb{P} = 1_\mathbb{P} : \mathbb{P} \to \mathbb{P}$.

Recursive definition. Since each $\widehat{\mathbb{P}}$ is a complete lattice, it admits least fixed-points of continuous maps. If $f : \widehat{\mathbb{P}} \to \widehat{\mathbb{P}}$ is continuous, it has a least fixed-point, $\mathit{fix}\, f \in \widehat{\mathbb{P}}$, obtained as $\bigcup_{n\in\omega} f^n(\varnothing)$.

$$\frac{\Gamma, x : \mathbb{P} \vdash t : \mathbb{P}}{\Gamma \vdash \mathit{rec}\, x.t : \mathbb{P}} \quad \frac{\Gamma \,\&\, \mathbb{P} \xrightarrow{t} \mathbb{P}}{\Gamma \xrightarrow{\mathit{fix}\, f} \mathbb{P}} \tag{3.17}$$

Here, $\mathit{fix}\, f$ is the fixed-point in $\mathbf{Cts}(\Gamma, \mathbb{P}) \cong \widehat{\Gamma \to \mathbb{P}}$ of the continuous operation f mapping $g : \Gamma \to \mathbb{P}$ in $\mathbf{Cts}$ to the composition

$$\Gamma \xrightarrow{\Delta_\Gamma} \Gamma \,\&\, \Gamma \xrightarrow{1_\Gamma \& g} \Gamma \,\&\, \mathbb{P} \xrightarrow{t} \mathbb{P} \ . \tag{3.18}$$

Nondeterministic sum. Each path order $\mathbb{P}$ is associated with a join operation, $\Sigma : \&_{i\in I} \mathbb{P} \to \mathbb{P}$ in $\mathbf{Cts}$ taking a tuple $\langle t_i \rangle_{i\in I}$ to the join $\Sigma_{i\in I} t_i$ in $\widehat{\mathbb{P}}$. We'll write $\varnothing$ and $t_1 + \cdots + t_k$ for finite sums.

$$\frac{\Gamma \vdash t_j : \mathbb{P} \quad \text{all } j \in I}{\Gamma \vdash \Sigma_{i\in I} t_i : \mathbb{P}} \quad \frac{\Gamma \xrightarrow{t_j} \mathbb{P} \quad \text{all } j \in I}{\Gamma \xrightarrow{\langle t_i \rangle_{i\in I}} \&_{i\in I} \mathbb{P} \xrightarrow{\Sigma} \mathbb{P}} \tag{3.19}$$

Function space. As noted at the end of Section 2.3.1, the category $\mathbf{Cts}$ is cartesian closed with function space $\mathbb{P} \to \mathbb{Q}$. Thus, there is a 1-1 correspondence *curry* from maps $\mathbb{P} \,\&\, \mathbb{Q} \to \mathbb{R}$ to maps $\mathbb{P} \to (\mathbb{Q} \to \mathbb{R})$ in $\mathbf{Cts}$; its inverse is called *uncurry*. We obtain application, $\mathit{app} : (\mathbb{P} \to \mathbb{Q}) \,\&\, \mathbb{P} \to \mathbb{Q}$ as $\mathit{uncurry}(1_{\mathbb{P}\to\mathbb{Q}})$.

$$\frac{\Gamma, x : \mathbb{P} \vdash t : \mathbb{Q}}{\Gamma \vdash \lambda x.t : \mathbb{P} \to \mathbb{Q}} \quad \frac{\Gamma \,\&\, \mathbb{P} \xrightarrow{t} \mathbb{Q}}{\Gamma \xrightarrow{\mathit{curry}\, t} \mathbb{P} \to \mathbb{Q}} \tag{3.20}$$

$$\frac{\Gamma \vdash t : \mathbb{P} \to \mathbb{Q} \quad \Lambda \vdash u : \mathbb{P}}{\Gamma, \Lambda \vdash t\ u : \mathbb{Q}} \quad \frac{\Gamma \xrightarrow{t} \mathbb{P} \to \mathbb{Q} \quad \Lambda \xrightarrow{u} \mathbb{P}}{\Gamma \,\&\, \Lambda \xrightarrow{t\&u} (\mathbb{P} \to \mathbb{Q}) \,\&\, \mathbb{P} \xrightarrow{\mathit{app}} \mathbb{Q}} \tag{3.21}$$

Sum type. The category $\mathbf{Cts}$ does not have coproducts, but we can build a useful sum type out of the biproduct of $\mathbf{Lin}$. The properties of (2.9) are obviously also satisfied in $\mathbf{Cts}$, even though the construction is universal only in the subcategory of linear maps because composition is generally not bilinear in $\mathbf{Cts}$. We'll write $\mathbb{O}$ and $\mathbb{P}_1 + \cdots + \mathbb{P}_k$ for the empty and finite sum types. The product $\mathbb{P}_1 \,\&\, \mathbb{P}_2$ of [65] with pairing (t, u) and projection terms $\mathit{fst}\, t$, $\mathit{snd}\, t$ can be encoded, respectively, as the type $\mathbb{P}_1 + \mathbb{P}_2$, and the terms $1t + 2u$ and $\pi_1 t$, $\pi_2 t$.

$$\frac{\Gamma \vdash t : \mathbb{P}_\beta \quad \beta \in A}{\Gamma \vdash \beta t : \Sigma_{\alpha\in A} \mathbb{P}_\alpha} \quad \frac{\Gamma \xrightarrow{t} \mathbb{P}_\beta \quad \beta \in A}{\Gamma \xrightarrow{t} \mathbb{P}_\beta \xrightarrow{\mathit{in}_\beta} \Sigma_{\alpha\in A} \mathbb{P}_\alpha} \tag{3.22}$$

$$\frac{\Gamma \vdash t : \Sigma_{\alpha\in A} \mathbb{P}_\alpha \quad \beta \in A}{\Gamma \vdash \pi_\beta t : \mathbb{P}_\beta} \quad \frac{\Gamma \xrightarrow{t} \Sigma_{\alpha\in A} \mathbb{P}_\alpha \quad \beta \in A}{\Gamma \xrightarrow{t} \Sigma_{\alpha\in A} \mathbb{P}_\alpha \xrightarrow{\pi_\beta} \mathbb{P}_\beta} \tag{3.23}$$

Prefixing. The adjunction between **Lin** and **Cts** provides a type constructor, $!(-)$, for which the unit $\eta_{\mathbb{P}} : \mathbb{P} \to !\mathbb{P}$ and counit $\varepsilon_{\mathbb{P}} : !\mathbb{P} \to \mathbb{P}$ may interpret term constructors and deconstructors, respectively. The behaviour of $\eta_{\mathbb{P}}$ with respect to maps of **Cts** fits that of an anonymous prefix operation.

$$\frac{\Gamma \vdash u : \mathbb{P}}{\Gamma \vdash !u : !\mathbb{P}} \qquad \frac{\Gamma \xrightarrow{u} \mathbb{P}}{\Gamma \xrightarrow{u} \mathbb{P} \xrightarrow{\eta_{\mathbb{P}}} !\mathbb{P}} \tag{3.24}$$

By the universal property of $\eta_{\mathbb{P}}$, if t of type $\mathbb{Q}$ has a free variable of type $\mathbb{P}$, and so is interpreted as a map $t : \mathbb{P} \to \mathbb{Q}$ in **Cts**, then the transpose $\bar{t} = \varepsilon_{\mathbb{Q}} \circ !t$ is the unique map $!\mathbb{P} \to \mathbb{Q}$ in **Lin** such that $t = \bar{t} \circ \eta_{\mathbb{P}}$. With u of type $!\mathbb{P}$, we'll interpret the prefix match $[u > !x \Rightarrow t]$ as $\bar{t}u$. Then, if $u \equiv !u'$ for some u' of type $\mathbb{P}$, our interpretation yields $\bar{t}(\eta_{\mathbb{P}} u') = tu'$ and so by the substitution lemma (Lemma 3.4 below), the match is identified with $t[u'/x]$ as wanted. Moreover, by linearity of $\bar{t}$, the match will distribute over nondeterministic sum.

The above clearly generalises to the case where u is an open term, but if t has free variables other than x, we need to make use of the strength map (2.18) to distribute the exponential over the corresponding product. Proposition 3.8 below shows that the general definition also satisfies the required properties.

$$\frac{\Gamma, x : \mathbb{P} \vdash t : \mathbb{Q} \quad \Lambda \vdash u : !\mathbb{P}}{\Gamma, \Lambda \vdash [u > !x \Rightarrow t] : \mathbb{Q}}$$
$$\frac{\Gamma \,\&\, \mathbb{P} \xrightarrow{t} \mathbb{Q} \quad \Lambda \xrightarrow{u} !\mathbb{P}}{\Gamma \,\&\, \Lambda \xrightarrow{1_\Gamma \& u} \Gamma \,\&\, !\mathbb{P} \xrightarrow{str_{\Gamma,\mathbb{P}}} !(\Gamma \,\&\, \mathbb{P}) \xrightarrow{\bar{t}} \mathbb{Q}} \tag{3.25}$$

Recursive types. Folding and unfolding recursive types is accompanied by term constructors *abs* and *rep*:

$$\frac{\Gamma \vdash t : \mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}]}{\Gamma \vdash abs\, t : \mu_j\vec{T}.\vec{\mathbb{T}}} \qquad \frac{\Gamma \xrightarrow{t} \mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}]}{\Gamma \xrightarrow{t} \mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}] \xrightarrow{abs} \mu_j\vec{T}.\vec{\mathbb{T}}} \tag{3.26}$$

$$\frac{\Gamma \vdash t : \mu_j\vec{T}.\vec{\mathbb{T}}}{\Gamma \vdash rep\, t : \mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}]} \qquad \frac{\Gamma \xrightarrow{t} \mu_j\vec{T}.\vec{\mathbb{T}}}{\Gamma \xrightarrow{t} \mu_j\vec{T}.\vec{\mathbb{T}} \xrightarrow{rep} \mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}]} \tag{3.27}$$

Example 3.3 The translation in Example 3.1 of the CCS processes $\mathsf{a}.\mathsf{b}.\varnothing + \mathsf{a}.\mathsf{c}.\varnothing$ and $\mathsf{a}.(\mathsf{b}.\varnothing + \mathsf{c}.\varnothing)$ did not use the abstract syntax *abs* for folding the recursive type $\mathbb{P} = \Sigma_{\alpha \in A} !\mathbb{P}$. Formally, the translations and their interpretations are as follows (cf. Example 2.2):

$$\begin{aligned}
&[\![abs(\mathsf{a}!\, abs(\mathsf{b}!\varnothing)) + abs(\mathsf{a}!\, abs(\mathsf{c}!\varnothing))]\!] \\
&\quad = \{abs\,\mathsf{a}\varnothing, abs\,\mathsf{a}\{abs\,\mathsf{b}\varnothing\}, abs\,\mathsf{a}\{abs\,\mathsf{c}\varnothing\}\} \\
&[\![abs(\mathsf{a}!(abs(\mathsf{b}!\varnothing) + abs(\mathsf{c}!\varnothing)))]\!] \\
&\quad = \{abs\,\mathsf{a}\varnothing, abs\,\mathsf{a}\{abs\,\mathsf{b}\varnothing\}, abs\,\mathsf{a}\{abs\,\mathsf{c}\varnothing\}, abs\,\mathsf{a}\{abs\,\mathsf{b}\varnothing, abs\,\mathsf{c}\varnothing\}\}
\end{aligned} \tag{3.28}$$

We'll continue to dispense with both *abs* and *rep* in examples for clarity. □

3.2 Useful Identities

We provide some technical results about the path semantics which are used in later proofs. They are also useful for reasoning about encodings of process calculi, see Section 3.6 below.

Lemma 3.4 (Substitution) Suppose $\Gamma, x : \mathbb{P} \vdash t : \mathbb{Q}$ and $\Lambda \vdash u : \mathbb{P}$ with Γ and Λ disjoint. Then $\Gamma, \Lambda \vdash t[u/x] : \mathbb{Q}$ with denotation given by the composition

$$\Gamma \,\&\, \Lambda \xrightarrow{1_\Gamma \& u} \Gamma \,\&\, \mathbb{P} \xrightarrow{t} \mathbb{Q} \ . \tag{3.29}$$

Proof. By rule-induction on the term formation rules above using the induction hypothesis

> Suppose $\Gamma' \vdash t : \mathbb{Q}$ with Γ' a permutation of $\Gamma, x_1 : \mathbb{P}, \ldots, x_k : \mathbb{P}$. Let $p : \Gamma \,\&\, \mathbb{P}^k \cong \Gamma'$ be the associated isomorphism. If $\Lambda \vdash u : \mathbb{P}$ with Γ and Λ disjoint, then $\Gamma, \Lambda \vdash t[u/x_1, \ldots, u/x_k] : \mathbb{Q}$ with denotation given by
>
> $$\Gamma \,\&\, \Lambda \xrightarrow{1_\Gamma \& (\Delta^k_\mathbb{P} \circ u)} \Gamma \,\&\, \mathbb{P}^k \xrightarrow{p} \Gamma' \xrightarrow{t} \mathbb{Q} \ . \tag{3.30}$$

The substitution lemma itself follows from the statement above by taking $\Gamma' \equiv \Gamma, x : \mathbb{P}$. We'll abbreviate the generic substitution $[u/x_1, \ldots, u/x_k]$ to $[u]$ and $\Delta^k_\mathbb{P} \circ [\![u]\!]$ to h^k.

Identity (case 1). Suppose $x : \mathbb{P} \vdash x : \mathbb{P}$ is derived using the rule for identity. Let $\Lambda \vdash u : \mathbb{P}$. As $k = 1$ in this case, we want to show that $\Lambda \vdash x[u/x] : \mathbb{P}$. But $x[u/x] \equiv u$ and so we are done.

Semantically we have $[\![u]\!] = 1_\mathbb{P} \circ \Delta^1_\mathbb{P} \circ [\![u]\!]$ as wanted.

Identity (case 2). Suppose $y : \mathbb{Q} \vdash y : \mathbb{Q}$ is derived using the rule for identity. Let $\Lambda \vdash u : \mathbb{P}$ with y not occurring in Λ. As $k = 0$ in this case we want to show that $y : \mathbb{Q}, \Lambda \vdash y : \mathbb{Q}$. This can be derived from $y : \mathbb{Q} \vdash y : \mathbb{Q}$ by repeated use of weakening.

The semantic result follows because

$$1_\mathbb{Q} \circ r^{\mathbf{Cts}}_\mathbb{Q} \circ (1_\mathbb{Q} \,\&\, \varnothing_\Lambda) = 1_\mathbb{Q} \circ r^{\mathbf{Cts}}_\mathbb{Q} \circ (1_\mathbb{Q} \,\&\, h^0) \tag{3.31}$$

by naturality of $\varnothing = \Delta^0_\mathbb{P}$.

Weakening (case 1). Suppose $\Gamma', x : \mathbb{P} \vdash t : \mathbb{Q}$ is obtained by weakening from $\Gamma' \vdash t : \mathbb{Q}$ and that Γ' is a permutation of $\Gamma, x_1 : \mathbb{P}, \ldots, x_k : \mathbb{P}$. Let $\Lambda \vdash u : \mathbb{P}$ with Γ and Λ disjoint. We want to show that $\Gamma, \Lambda \vdash t[u/x_1, \ldots, u/x_k, u/x] : \mathbb{Q}$. As x does not occur freely in t, we have $t[u/x_1, \ldots, u/x_k, u/x] \equiv t[u]$, and by the induction hypothesis, $\Gamma, \Lambda \vdash t[u] : \mathbb{Q}$ as wanted.

Up to exchange we must show that $[\![t[u]]\!]$ equals $[\![t]\!] \circ r^{\mathbf{Cts}}_{\Gamma\&\mathbb{P}^k} \circ (1_{\Gamma\&\mathbb{P}^k} \& \varnothing_{\mathbb{P}}) \circ (1_\Gamma \& h^{k+1})$. We have

$$\begin{aligned}
&[\![t]\!] \circ r^{\mathbf{Cts}}_{\Gamma\&\mathbb{P}^k} \circ (1_{\Gamma\&\mathbb{P}^k} \& \varnothing_{\mathbb{P}}) \circ (1_\Gamma \& h^{k+1}) \\
&= [\![t]\!] \circ (1_\Gamma \& (r^{\mathbf{Cts}}_{\mathbb{P}^k} \circ (1_{\mathbb{P}^k} \& \varnothing_{\mathbb{P}})) \circ (1_\Gamma \& h^{k+1})) \\
&= [\![t]\!] \circ (1_\Gamma \& (r^{\mathbf{Cts}}_{\mathbb{P}^k} \circ (1_{\mathbb{P}^k} \& \varnothing_{\mathbb{P}}) \circ h^{k+1})) \\
&= [\![t]\!] \circ (1_\Gamma \& (r^{\mathbf{Cts}}_{\mathbb{P}^k} \circ (1_{\mathbb{P}^k} \& \varnothing_{\mathbb{P}}) \circ \Delta^{k+1}_{\mathbb{P}} \circ [\![u]\!])) \\
&= [\![t]\!] \circ (1_\Gamma \& \Delta^k_{\mathbb{P}} \circ [\![u]\!]) \\
&= [\![t]\!] \circ (1_\Gamma \& h^k) \\
&= [\![t[u]]\!] && \text{(ind. hyp.)}
\end{aligned}$$

—as wanted.

Weakening (case 2). Suppose $\Gamma', y : \mathbb{Q} \vdash t : \mathbb{R}$ is obtained by weakening from $\Gamma' \vdash t : \mathbb{R}$ and that $\Gamma', y : \mathbb{Q}$ is a permutation of $\Gamma, x_1 : \mathbb{P}, \ldots, x_k : \mathbb{P}$. Here, $\Gamma \equiv \Gamma_1, y : \mathbb{Q}, \Gamma_2$. Let $\Lambda \vdash u : \mathbb{P}$ with Γ and Λ disjoint. We want to show that $\Gamma, \Lambda \vdash t[u] : \mathbb{R}$. By the induction hypothesis, we have $\Gamma_1, \Gamma_2, \Lambda \vdash t[u] : \mathbb{R}$ and so we obtain $\Gamma_1, \Gamma_2, \Lambda, y : \mathbb{Q} \vdash t[u] : \mathbb{R}$ by weakening. Repeated use of exchange then yields the wanted derivation.

Up to use of exchange we need to show that $[\![t[u]]\!] \circ r^{\mathbf{Cts}}_{\Gamma\&\Lambda} \circ (1_{\Gamma\&\Lambda} \& \varnothing_{\mathbb{Q}})$ equals $[\![t]\!] \circ r^{\mathbf{Cts}}_{\Gamma\&\mathbb{P}^k} \circ (1_{\Gamma\&\mathbb{P}^k} \& \varnothing_{\mathbb{Q}}) \circ (1_\Gamma \& h^k \& 1_{\mathbb{Q}})$. We have

$$\begin{aligned}
&[\![t]\!] \circ r^{\mathbf{Cts}}_{\Gamma\&\mathbb{P}^k} \circ (1_{\Gamma\&\mathbb{P}^k} \& \varnothing_{\mathbb{Q}}) \circ (1_\Gamma \& h^k \& 1_{\mathbb{Q}}) \\
&= [\![t]\!] \circ r^{\mathbf{Cts}}_{\Gamma\&\mathbb{P}^k} \circ (1_\Gamma \& h^k \& 1_{\mathbb{O}}) \circ (1_{\Gamma\&\Lambda} \& \varnothing_{\mathbb{Q}}) \\
&= [\![t]\!] \circ (1_\Gamma \& h^k) \circ r^{\mathbf{Cts}}_{\Gamma\&\Lambda} \circ (1_{\Gamma\&\Lambda} \& \varnothing_{\mathbb{Q}}) && \text{(nat. of } r^{\mathbf{Cts}}\text{)} \\
&= [\![t[u]]\!] \circ r^{\mathbf{Cts}}_{\Gamma\&\Lambda} \circ (1_{\Gamma\&\Lambda} \& \varnothing_{\mathbb{Q}}) && \text{(ind. hyp.)}
\end{aligned}$$

—as wanted.

Exchange. Directly from the induction hypothesis.

Contraction (case 1). Suppose that $\Gamma', z : \mathbb{P} \vdash t[z/x, z/y] : \mathbb{Q}$ is obtained by contraction from $\Gamma', x : \mathbb{P}, y : \mathbb{P} \vdash t : \mathbb{Q}$ and that Γ' is a permutation of $\Gamma, x_1 : \mathbb{P}, \ldots, x_k : \mathbb{P}$. Let $\Lambda \vdash u : \mathbb{P}$ with Γ and Λ disjoint. We want to show that $\Gamma, \Lambda \vdash t[z/x, z/y][u/x_1, \ldots, u/x_k, u/z] : \mathbb{Q}$. By the induction hypothesis we have $\Gamma, \Lambda \vdash t[u/x_1, \ldots, u/x_k, u/x, u/y] : \mathbb{Q}$ and as $t[z/x, z/y][u/x_1, \ldots, u/x_k, u/z] \equiv t[u/x_1, \ldots, u/x_k, u/x, u/y]$ we are done.

Assuming $\Gamma' \equiv \Gamma, x_1 : \mathbb{P}, \ldots, x_k : \mathbb{P}$ for simplicity, we must show that $[\![t[u/x_1, \ldots, u/x_k, u/x, u/y]]\!]$ equals $[\![t]\!] \circ (1_{\Gamma\&\mathbb{P}^k} \& \Delta_{\mathbb{P}}) \circ (1_\Gamma \& h^{k+1})$. We have

$$\begin{aligned}
&[\![t]\!] \circ (1_{\Gamma\&\mathbb{P}^k} \& \Delta_{\mathbb{P}}) \circ (1_\Gamma \& h^{k+1}) \\
&= [\![t]\!] \circ (1_\Gamma \& ((1_{\mathbb{P}^k} \& \Delta_{\mathbb{P}}) \circ h^{k+1})) \\
&= [\![t]\!] \circ (1_\Gamma \& h^{k+2}) \\
&= [\![t[u/x_1, \ldots, u/x_k, u/x, u/y]]\!] && \text{(ind. hyp.)}
\end{aligned}$$

Contraction (case 2). Suppose that $\Gamma', z : \mathbb{Q} \vdash t[z/x, z/y] : \mathbb{R}$ is obtained by contraction from $\Gamma', x : \mathbb{Q}, y : \mathbb{Q} \vdash t : \mathbb{R}$ and that $\Gamma', z : \mathbb{Q}$ is a permutation of

$\Gamma, x_1 : \mathbb{P}, \ldots, x_k : \mathbb{P}$. Here, $\Gamma \equiv \Gamma_1, z : \mathbb{Q}, \Gamma_2$. Let $\Lambda \vdash u : \mathbb{P}$ with Γ and Λ disjoint. By renaming x and y if necessary, we may assume that these variables do not occur freely in u. We want to show that $\Gamma, \Lambda \vdash t[z/x, z/y][u] : \mathbb{Q}$. By the induction hypothesis we have $\Gamma_1, \Gamma_2, x : \mathbb{Q}, y : \mathbb{Q}, \Lambda \vdash t[u] : \mathbb{R}$. Repeated use of exchange yields $\Gamma_1, \Gamma_2, \Lambda, x : \mathbb{Q}, y : \mathbb{Q} \vdash t[u] : \mathbb{R}$ and thus, by contraction, $\Gamma_1, \Gamma_2, \Lambda, z : \mathbb{Q} \vdash t[u][z/x, z/y]$. As x, y do not occur freely in u, we have $t[z/x, z/y][u] \equiv t[u][z/x, z/y]$ and exchange then yields the wanted derivation.

Up to exchange we must show that $[\![t[u]]\!] \circ (1_\Gamma \mathbin{\&} \Delta_\mathbb{Q} \mathbin{\&} 1_\Lambda)$ equals $[\![t]\!] \circ (1_\Gamma \mathbin{\&} \Delta_\mathbb{Q} \mathbin{\&} 1_{\mathbb{P}^k}) \circ (1_\Gamma \mathbin{\&} h^k)$. This is immediate from the induction hypothesis.

Recursive definition. Suppose that $\Gamma' \vdash \mathit{rec}\, x.t : \mathbb{Q}$ is obtained from $\Gamma', x : \mathbb{Q} \vdash t : \mathbb{Q}$ and that Γ' is a permutation of $\Gamma, x_1 : \mathbb{P}, \ldots, x_k : \mathbb{P}$. Let $\Lambda \vdash u : \mathbb{P}$ with Γ and Λ disjoint. By renaming x if necessary, we may assume that x does not occur freely in u. We want to show that $\Gamma, \Lambda \vdash (\mathit{rec}\, x.t)[u] : \mathbb{Q}$. By the induction hypothesis, $\Gamma, x : \mathbb{Q}, \Lambda \vdash t[u] : \mathbb{Q}$ and so by exchange, $\Gamma, \Lambda, x : \mathbb{Q} \vdash t[u] : \mathbb{Q}$. This yields $\Gamma, \Lambda \vdash \mathit{rec}\, x.(t[u]) : \mathbb{Q}$. Since x does not occur freely in u, $(\mathit{rec}\, x.t)[u] \equiv \mathit{rec}\, x.(t[u])$ and we are done.

Up to exchange we need to show that $[\![\mathit{rec}\, x.(t[u])]\!] = [\![\mathit{rec}\, x.t]\!] \circ (1_\Gamma \mathbin{\&} h^k)$. Here, $[\![\mathit{rec}\, x.(t[u])]\!] = \mathit{fix}\, f_u$ where f_u maps $g : \Gamma \mathbin{\&} \Lambda \mathbin{\&} \mathbb{Q} \to \mathbb{Q}$ to $[\![t[u]]\!] \circ (1_{\Gamma\&\Lambda} \mathbin{\&} g) \circ \Delta_{\Gamma\&\Lambda}$ whereas $[\![\mathit{rec}\, x.t]\!] = \mathit{fix}\, f$ with f mapping $g : \Gamma \& \mathbb{P}^k \& \mathbb{Q} \to \mathbb{Q}$ to $[\![t]\!] \circ (1_{\Gamma\&\mathbb{P}^k} \mathbin{\&} g) \circ \Delta_{\Gamma\&\mathbb{P}^k}$. We show by mathematical induction that for each $n \in \omega$, we have $f_u^n \varnothing = f^n \varnothing \circ (1_\Gamma \mathbin{\&} h^k)$:

Basis. By naturality of $\varnothing$ as $f_u^0 \varnothing = \varnothing_{\Gamma\&\Lambda}$ and $f^0 \varnothing = \varnothing_{\Gamma\&\mathbb{P}^k}$.

Step. We have

$$
\begin{array}{ll}
f^{n+1}\varnothing \circ (1_\Gamma \mathbin{\&} h^k) & \\
\quad = [\![t]\!] \circ (1_{\Gamma\&\mathbb{P}^k} \mathbin{\&} f^n\varnothing) \circ \Delta_{\Gamma\&\mathbb{P}^k} \circ (1_\Gamma \mathbin{\&} h^k) & \\
\quad = [\![t]\!] \circ (1_{\Gamma\&\mathbb{P}^k} \mathbin{\&} f^n\varnothing) \circ ((1_\Gamma \mathbin{\&} h^k) \mathbin{\&} (1_\Gamma \mathbin{\&} h^k)) \circ \Delta_{\Gamma\&\Lambda} & (\text{nat. of } \Delta) \\
\quad = [\![t]\!] \circ ((1_\Gamma \mathbin{\&} h^k) \mathbin{\&} (f^n\varnothing \circ (1_\Gamma \mathbin{\&} h^k))) \circ \Delta_{\Gamma\&\Lambda} & \\
\quad = [\![t]\!] \circ ((1_\Gamma \mathbin{\&} h^k) \mathbin{\&} f_u^n\varnothing) \circ \Delta_{\Gamma\&\Lambda} & (\text{ind. hyp. for } n) \\
\quad = [\![t]\!] \circ (1_\Gamma \mathbin{\&} h^k \mathbin{\&} 1_\mathbb{Q}) \circ (1_{\Gamma\&\Lambda} \mathbin{\&} f_u^n\varnothing) \circ \Delta_{\Gamma\&\Lambda} & \\
\quad = [\![t[u]]\!] \circ (1_{\Gamma\&\Lambda} \mathbin{\&} f_u^n\varnothing) \circ \Delta_{\Gamma\&\Lambda} & (\text{ind. hyp.}) \\
\quad = f_u^{n+1}\varnothing &
\end{array}
$$

—as wanted.

By mathematical induction, we have the wanted equality for all $n \in \omega$ from which the equality for the fixed-points follow.

Nondeterministic sum. Suppose that $\Gamma' \vdash \Sigma_{i\in I} t_i : \mathbb{Q}$ is obtained from $\Gamma' \vdash t_j : \mathbb{Q}$ for all $j \in I$ and that Γ' is a permutation of $\Gamma, x_1 : \mathbb{P}, \ldots, x_k : \mathbb{P}$. Let $\Lambda \vdash u : \mathbb{P}$ with Γ and Λ disjoint. We want to show that $\Gamma, \Lambda \vdash (\Sigma_{i\in I} t_i)[u] : \mathbb{Q}$. By the induction hypotheses, we have $\Gamma, \Lambda \vdash t_j[u] : \mathbb{Q}$ for all $j \in I$ and so $\Gamma, \Lambda \vdash \Sigma_{i\in I}(t_i[u]) : \mathbb{Q}$. Since $(\Sigma_{i\in I} t_i)[u] \equiv \Sigma_{i\in I}(t_i[u])$ we are done.

Up to exchange we need to show that $\Sigma \circ \langle t_i[u] \rangle_{i \in I} = \Sigma \circ \langle [\![t_i]\!] \rangle_{i \in I} \circ (1_\Gamma \,\&\, h^k)$. We have

$$\begin{aligned}
&\Sigma \circ \langle [\![t_i]\!] \rangle_{i \in I} \circ (1_\Gamma \,\&\, h^k) \\
&= \Sigma \circ \langle [\![t_i]\!] \circ (1_\Gamma \,\&\, h^k) \rangle_{i \in I} \\
&= \Sigma \circ \langle [\![t_i[u]]\!] \rangle_{i \in I} && \text{(ind. hyp.)}
\end{aligned}$$

—as wanted.

Abstraction. Suppose that $\Gamma' \vdash \lambda x.t : \mathbb{Q} \to \mathbb{R}$ is obtained from $\Gamma', x : \mathbb{Q} \vdash t : \mathbb{R}$ and that Γ' is a permutation of $\Gamma, x_1 : \mathbb{P}, \ldots, x_k : \mathbb{P}$. Let $\Lambda \vdash u : \mathbb{P}$ with Γ and Λ disjoint. By renaming x if necessary, we may assume that x does not occur freely in u. We want to show that $\Gamma, \Lambda \vdash (\lambda x.t)[u] : \mathbb{Q} \to \mathbb{R}$. By the induction hypothesis, $\Gamma, x : \mathbb{Q}, \Lambda \vdash t[u] : \mathbb{R}$ and so by exchange, $\Gamma, \Lambda, x : \mathbb{Q} \vdash t[u] : \mathbb{R}$. This yields $\Gamma, \Lambda \vdash \lambda x.(t[u]) : \mathbb{Q} \to \mathbb{R}$. Since x does not occur freely in u, $(\lambda x.t)[u] \equiv \lambda x.(t[u])$ and we are done.

Up to exchange we must show that $\mathit{curry}([\![t[u]]\!]) = \mathit{curry}[\![t]\!] \circ (1_\Gamma \,\&\, h^k)$. We have

$$\begin{aligned}
&\mathit{app} \circ ((\mathit{curry}[\![t]\!] \circ (1_\Gamma \,\&\, h^k)) \,\&\, 1_\mathbb{Q}) \\
&= \mathit{app} \circ (\mathit{curry}[\![t]\!] \,\&\, 1_\mathbb{Q}) \circ (1_\Gamma \,\&\, h^k \,\&\, 1_\mathbb{Q}) \\
&= [\![t]\!] \circ (1_\Gamma \,\&\, h^k \,\&\, 1_\mathbb{Q}) && \text{(prop. of } \mathit{curry}\text{)} \\
&= [\![t[u]]\!] && \text{(ind. hyp.)}
\end{aligned}$$

The wanted equality then follows from the universal property of *curry*.

Application. Suppose that $\Gamma'_1, \Gamma'_2 \vdash t_1\ t_2 : \mathbb{R}$ is obtained from $\Gamma'_1 \vdash t_1 : \mathbb{Q} \to \mathbb{R}$ and $\Gamma'_2 \vdash t_2 : \mathbb{Q}$ and that Γ'_i is a permutation of $\Gamma_i, x_1 : \mathbb{P}, \ldots, x_{k_i} : \mathbb{P}$ for $i = 1, 2$. Let $\Lambda \vdash u : \mathbb{P}$ with Γ_1, Γ_2 and Λ disjoint. We want to show that $\Gamma_1, \Gamma_2, \Lambda \vdash (t_1\ t_2)[u] : \mathbb{R}$. By the induction hypotheses, we have $\Gamma_1, \Lambda \vdash t_1[u] : \mathbb{Q} \to \mathbb{R}$ and $\Gamma_2, \Lambda \vdash t_2[u] : \mathbb{Q}$. By renaming the variables of Λ and u to fresh names in the latter derivation we get $\Gamma_2, \Lambda' \vdash t_2[u'] : \mathbb{Q}$ and so $\Gamma_1, \Lambda, \Gamma_2, \Lambda' \vdash (t_1[u])\ (t_2[u']) : \mathbb{R}$ by the typing rule for application. By repeated use of exchange and contraction we get $\Gamma_1, \Gamma_2, \Lambda \vdash (t_1[u])\ (t_1[u]) : \mathbb{R}$ and as $(t_1\ t_2)[u] \equiv (t_1[u])\ (t_2[u])$ we are done.

Assume for simplicity that Γ_1 and Γ_2 are empty. We must then show that $\mathit{app} \circ ([\![t_1[u]]\!] \,\&\, [\![t_2[u]]\!]) \circ \Delta_\Lambda$ equals $\mathit{app} \circ ([\![t_1]\!] \,\&\, [\![t_2]\!]) \circ h^{k_1+k_2}$. We have

$$\begin{aligned}
&\mathit{app} \circ ([\![t_1]\!] \,\&\, [\![t_2]\!]) \circ h^{k_1+k_2} \\
&= \mathit{app} \circ ([\![t_1]\!] \,\&\, [\![t_2]\!]) \circ \Delta_\mathbb{P}^{k_1+k_2} \circ [\![u]\!] \\
&= \mathit{app} \circ ([\![t_1]\!] \,\&\, [\![t_2]\!]) \circ (\Delta_\mathbb{P}^{k_1} \,\&\, \Delta_\mathbb{P}^{k_2}) \circ \Delta_\mathbb{P} \circ [\![u]\!] \\
&= \mathit{app} \circ ([\![t_1]\!] \,\&\, [\![t_2]\!]) \circ (\Delta_\mathbb{P}^{k_1} \,\&\, \Delta_\mathbb{P}^{k_2}) \circ ([\![u]\!] \,\&\, [\![u]\!]) \circ \Delta_\Lambda && \text{(nat. of } \Delta\text{)} \\
&= \mathit{app} \circ (([\![t_1]\!] \circ h^{k_1}) \,\&\, ([\![t_2]\!] \circ h^{k_2})) \circ \Delta_\Lambda \\
&= \mathit{app} \circ ([\![t_1[u]]\!] \,\&\, [\![t_2[u]]\!]) \circ \Delta_\Lambda && \text{(ind. hyp.)}
\end{aligned}$$

—as wanted.

Injection. Suppose that $\Gamma' \vdash \beta t : \Sigma_{\alpha \in A} \mathbb{P}_\alpha$ is obtained from $\Gamma' \vdash t : \mathbb{P}_\beta$ and that Γ' is a permutation of $\Gamma, x_1 : \mathbb{P}, \ldots, x_k : \mathbb{P}$. Let $\Lambda \vdash u : \mathbb{P}$ with Γ and

Λ disjoint. We want to show that $\Gamma, \Lambda \vdash (\beta t)[u] : \Sigma_{\alpha\in A}\mathbb{P}_\alpha$. By the induction hypothesis, we have $\Gamma, \Lambda \vdash t[u] : \mathbb{P}_\beta$ and so $\Gamma, \Lambda \vdash \beta(t[u]) : \Sigma_{\alpha\in A}\mathbb{P}_\alpha$. Since $(\beta t)[u] \equiv \beta(t[u])$ we are done.

Up to exchange we need to show that $in_\beta \circ [\![t[u]]\!] = in_\beta \circ [\![t]\!] \circ (1_\Gamma \,\&\, h^k)$. This is immediate using the induction hypothesis.

Prefix match. Suppose that $\Gamma'_1, \Gamma'_2 \vdash [t_2 > !x \Rightarrow t_1] : \mathbb{R}$ is obtained from $\Gamma'_1, x : \mathbb{Q} \vdash t_1 : \mathbb{R}$ and $\Gamma'_2 \vdash t_2 : !\mathbb{Q}$ and that Γ'_i is a permutation of $\Gamma_i, x_1 : \mathbb{P}, \ldots, x_{k_i} : \mathbb{P}$ for $i = 1, 2$. Let $\Lambda \vdash u : \mathbb{P}$ with Γ_1, Γ_2 and Λ disjoint. By renaming x if necessary, we may assume that x does not occur freely in u. We want to show that $\Gamma_1, \Gamma_2, \Lambda \vdash [t_2 > !x \Rightarrow t_1][u] : \mathbb{R}$. By the induction hypotheses, we have $\Gamma_1, x : \mathbb{Q}, \Lambda \vdash t_1[u] : \mathbb{R}$ and $\Gamma_2, \Lambda \vdash t_2[u] : !\mathbb{Q}$. By repeated use of exchange on the former derivation and by renaming the variables of Λ and u to fresh names in the latter derivation we get $\Gamma_1, \Lambda, \Gamma_2, \Lambda' \vdash [t_2[u'] > !x \Rightarrow t_1[u]] : \mathbb{R}$ by the typing rule for prefix match. Using exchange and contraction repeatedly we get $\Gamma_1, \Gamma_2, \Lambda \vdash [t_2[u] > !x \Rightarrow t_1[u]] : \mathbb{R}$ and as $[t_2 > !x \Rightarrow t_1][u] \equiv [t_2[u] > !x \Rightarrow t_1[u]]$ by the assumption on x, we are done.

Assume for simplicity that Γ_1 and Γ_2 are empty. We must then show that $\overline{[\![t_1[u]]\!]} \circ str_{\Lambda,\mathbb{Q}} \circ (1_\Lambda \,\&\, [\![t_2[u]]\!]) \circ \Delta_\Lambda$ equals $\overline{[\![t_1]\!]} \circ str_{\mathbb{P}^{k_1},\mathbb{Q}} \circ (1_{\mathbb{P}^k} \,\&\, [\![t_2]\!]) \circ h^{k_1+k_2}$. We have

$$
\begin{aligned}
&\overline{[\![t_1]\!]} \circ str_{\mathbb{P}^{k_1},\mathbb{Q}} \circ (1_{\mathbb{P}^{k_1}} \,\&\, [\![t_2]\!]) \circ h^{k_1+k_2} \\
&= \overline{[\![t_1]\!]} \circ str_{\mathbb{P}^{k_1},\mathbb{Q}} \circ (1_{\mathbb{P}^{k_1}} \,\&\, [\![t_2]\!]) \circ (h^{k_1} \,\&\, h^{k_2}) \circ \Delta_\Lambda && \text{(nat. of } \Delta) \\
&= \overline{[\![t_1]\!]} \circ str_{\mathbb{P}^{k_1},\mathbb{Q}} \circ (h^{k_1} \,\&\, ([\![t_2]\!] \circ h^{k_2})) \circ \Delta_\Lambda \\
&= \overline{[\![t_1]\!]} \circ str_{\mathbb{P}^{k_1},\mathbb{Q}} \circ (h^{k_1} \,\&\, [\![t_2[u]]\!]) \circ \Delta_\Lambda && \text{(ind. hyp.)} \\
&= \overline{[\![t_1]\!]} \circ str_{\mathbb{P}^{k_1},\mathbb{Q}} \circ (h^{k_1} \,\&\, 1_{!\mathbb{Q}}) \circ (1_\Lambda \,\&\, [\![t_2[u]]\!]) \circ \Delta_\Lambda \\
&= \overline{[\![t_1]\!]} \circ !(h^{k_1} \,\&\, 1_\mathbb{Q}) \circ str_{\Lambda,\mathbb{Q}} \circ (1_\Lambda \,\&\, [\![t_2[u]]\!]) \circ \Delta_\Lambda && \text{(nat. of } str) \\
&= \varepsilon_\mathbb{R} \circ ![\![t_1]\!] \circ !(h^{k_1} \,\&\, 1_\mathbb{Q}) \circ str_{\Lambda,\mathbb{Q}} \circ (1_\Lambda \,\&\, [\![t_2[u]]\!]) \circ \Delta_\Lambda \\
&= \varepsilon_\mathbb{R} \circ !([\![t_1]\!] \circ (h^{k_1} \,\&\, 1_\mathbb{Q})) \circ str_{\Lambda,\mathbb{Q}} \circ (1_\Lambda \,\&\, [\![t_2[u]]\!]) \circ \Delta_\Lambda && \text{(! a functor)} \\
&= \varepsilon_\mathbb{R} \circ !([\![t_1[u]]\!]) \circ str_{\Lambda,\mathbb{Q}} \circ (1_\Lambda \,\&\, [\![t_2[u]]\!]) \circ \Delta_\Lambda && \text{(ind. hyp.)} \\
&= \overline{[\![t_1[u]]\!]} \circ str_{\Lambda,\mathbb{Q}} \circ (1_\Lambda \,\&\, [\![t_2[u]]\!]) \circ \Delta_\Lambda
\end{aligned}
$$

—as wanted.

The remaining cases (projection, prefixing, folding, and unfolding) are handled similarly to injection. By rule-induction, the proof is complete. □

Proposition 3.5 Suppose $\Gamma, x : \mathbb{P} \vdash t : \mathbb{P}$. Then

$$[\![rec\, x.t]\!] = [\![t[rec\, x.t/x]]\!] \tag{3.32}$$

Proof. By renaming variables y of Γ to y' and y'' we get $\Gamma', x : \mathbb{P} \vdash t' : \mathbb{P}$ and $\Gamma'', x : \mathbb{P} \vdash t'' : \mathbb{P}$ with Γ' and Γ'' disjoint. Then by the substitution lemma, $\Gamma', \Gamma'' \vdash t'[rec\, x.t''/x] : \mathbb{P}$ with denotation

$$\Gamma' \,\&\, \Gamma'' \xrightarrow{1_{\Gamma'} \& rec\, x.t''} \Gamma' \,\&\, \mathbb{P} \xrightarrow{t'} \mathbb{P} \;. \tag{3.33}$$

By suitable use of exchange and contraction, substituting y for y' and y'', we get $\Gamma \vdash t[rec\, x.t/x] : \mathbb{P}$ with denotation

$$\Gamma \xrightarrow{\Delta_\Gamma} \Gamma \,\&\, \Gamma \xrightarrow{1_\Gamma \& rec\, x.t} \Gamma \,\&\, \mathbb{P} \xrightarrow{t} \mathbb{P} \ . \tag{3.34}$$

This is the same as $f(\mathit{fix}\, f)$ where $\mathit{fix}\, f$ is the denotation of $rec\, x.t$, and by property of the fixed-point, $f(\mathit{fix}\, f) = \mathit{fix}\, f$ as wanted. □

Proposition 3.6 (i) Suppose $\Gamma, x : \mathbb{P} \vdash t : \mathbb{Q}$ and $\Lambda \vdash u : \mathbb{P}$ with Γ and Λ disjoint. Then

$$[\![(\lambda x.t)\ u]\!] = [\![t[u/x]]\!] \ . \tag{3.35}$$

(ii) Suppose $\Gamma \vdash t : \mathbb{P} \to \mathbb{Q}$. Then

$$[\![\lambda x.(t\ x)]\!] = [\![t]\!] \ . \tag{3.36}$$

(iii) Suppose $\Gamma, x : \mathbb{P} \vdash t_i : \mathbb{Q}$ for all $i \in I$. Then

$$[\![\lambda x.(\Sigma_{i\in I} t_i)]\!] = [\![\Sigma_{i\in I}(\lambda x.t_i)]\!] \ . \tag{3.37}$$

(iv) Suppose $\Gamma \vdash t_i : \mathbb{P} \to \mathbb{Q}$ for all $i \in I$. Then for any suitable u we have

$$[\![(\Sigma_{i\in I} t_i)\ u]\!] = [\![\Sigma_{i\in I}(t_i\ u)]\!] \ . \tag{3.38}$$

Proof. (i) We calculate as follows:

$$\begin{aligned}
&[\![(\lambda x.t)\ u]\!] \\
&= app \circ (curry[\![t]\!] \,\&\, [\![u]\!]) \\
&= app \circ (curry[\![t]\!] \,\&\, 1_{\mathbb{P}}) \circ (1_\Gamma \,\&\, [\![u]\!]) \\
&= [\![t]\!] \circ (1_\Gamma \,\&\, [\![u]\!]) && \text{(univ. prop. of } curry\text{)} \\
&= [\![t[u/x]]\!] && \text{(substitution lemma)}
\end{aligned}$$

(ii) We calculate as follows:

$$\begin{aligned}
&[\![\lambda x.(t\ x)]\!] \\
&= curry(app \circ ([\![t]\!] \,\&\, 1_{\mathbb{P}})) \\
&= [\![t]\!] && \text{(univ. prop. of } curry\text{)}
\end{aligned}$$

(iii) and (iv) are obtained using linearity of *curry* and *uncurry*. □

Proposition 3.7 (i) Suppose $\Gamma \vdash t : \mathbb{P}_\beta$ for some $\beta \in A$. Then

$$[\![\pi_\beta(\beta t)]\!] = [\![t]\!] \quad \text{and} \quad [\![\pi_\alpha(\beta t)]\!] = \varnothing \quad \text{if } \alpha \neq \beta. \tag{3.39}$$

(ii) Suppose $\Gamma \vdash t : \Sigma_{\alpha\in A}\mathbb{P}_\alpha$. Then

$$[\![\Sigma_{\alpha\in A}\alpha(\pi_\alpha(t))]\!] = [\![t]\!] \ . \tag{3.40}$$

(iii) Suppose $\Gamma \vdash t_i : \mathbb{P}_\beta$ for all $i \in I$ and some $\beta \in A$. Then

$$[\![\beta(\Sigma_{i\in I} t_i)]\!] = [\![\Sigma_{i\in I}(\beta t_i)]\!] \ . \tag{3.41}$$

(iv) Suppose $\Gamma \vdash t_i : \Sigma_{\alpha\in A}\mathbb{P}_\alpha$ for all $i \in I$. Then for each $\beta \in A$,

$$[\![\pi_\beta(\Sigma_{i\in I} t_i)]\!] = [\![\Sigma_{i\in I}(\pi_\beta t_i)]\!] \ . \tag{3.42}$$

Proof. (i) and (ii) follow from the properties of the biproduct while (iii) and (iv) are obtained using linearity of in_β and π_β. $\square$

Proposition 3.8 (i) Suppose $\Gamma, x : \mathbb{P} \vdash t : \mathbb{Q}$ and $\Lambda \vdash u : \mathbb{P}$ with Γ and Λ disjoint. Then

$$\llbracket [!u > !x \Rightarrow t] \rrbracket = \llbracket t[u/x] \rrbracket \ . \tag{3.43}$$

(ii) Suppose $\Gamma, x : \mathbb{P} \vdash t : \mathbb{Q}$ and $\Lambda \vdash u_i : !\mathbb{P}$ for all $i \in I$ with Γ and Λ disjoint. Then

$$\llbracket [\Sigma_{i \in I} u_i > !x \Rightarrow t] \rrbracket = \llbracket \Sigma_{i \in I} [u_i > !x \Rightarrow t] \rrbracket \ . \tag{3.44}$$

Proof. (i) We calculate as follows:

$$\begin{array}{ll}
\llbracket [!u > !x \Rightarrow t] \rrbracket & \\
= \overline{\llbracket t \rrbracket} \circ str_{\Gamma,\mathbb{P}} \circ (1_\Gamma \mathbin{\&} (\eta_\mathbb{P} \circ \llbracket u \rrbracket)) & \\
= \overline{\llbracket t \rrbracket} \circ str_{\Gamma,\mathbb{P}} \circ (1_\Gamma \mathbin{\&} \eta_\mathbb{P}) \circ (1_\Gamma \mathbin{\&} \llbracket u \rrbracket) & \\
= \overline{\llbracket t \rrbracket} \circ \eta_{\Gamma \& \mathbb{P}} \circ (1_\Gamma \mathbin{\&} \llbracket u \rrbracket) & \text{(by (2.17))} \\
= \llbracket t \rrbracket \circ (1_\Gamma \mathbin{\&} \llbracket u \rrbracket) & \text{(univ. prop. of } \eta\text{)} \\
= \llbracket t[u/x] \rrbracket & \text{(substitution lemma)}
\end{array}$$

(ii) We calculate as follows:

$$\begin{array}{ll}
\llbracket [\Sigma_{i \in I} u_i > !x \Rightarrow t] \rrbracket & \\
= \overline{\llbracket t \rrbracket} \circ str_{\Gamma,\mathbb{P}} \circ (1_\Gamma \mathbin{\&} \Sigma_{i \in I} \llbracket u_i \rrbracket) & \\
= \overline{\llbracket t \rrbracket} \circ \Sigma_{i \in I} str_{\Gamma,\mathbb{P}} \circ (1_\Gamma \mathbin{\&} \llbracket u_i \rrbracket) & \text{(linearity of 2nd arg. of } str\text{)} \\
= \Sigma_{i \in I} (\overline{\llbracket t \rrbracket} \circ str_{\Gamma,\mathbb{P}} \circ (1_\Gamma \mathbin{\&} \llbracket u_i \rrbracket)) & \text{(linearity of } \overline{\llbracket t \rrbracket}\text{)} \\
= \llbracket \Sigma_{i \in I} [u_i > !x \Rightarrow t] \rrbracket &
\end{array}$$

—as wanted. $\square$

Proposition 3.9 (i) Suppose $\Gamma \vdash t : \mathbb{T}_j[\mu \vec{T}.\vec{\mathbb{T}}/\vec{T}]$. Then

$$\llbracket rep(abs\, t) \rrbracket = \llbracket t \rrbracket \ . \tag{3.45}$$

(ii) Suppose $\Gamma \vdash t : \mu_j \vec{T}.\vec{\mathbb{T}}$. Then

$$\llbracket abs(rep\, t) \rrbracket = \llbracket t \rrbracket \ . \tag{3.46}$$

(iii) Suppose $\Gamma \vdash t_i : \mathbb{T}_j[\mu \vec{T}.\vec{\mathbb{T}}/\vec{T}]$ for all $i \in I$. Then

$$\llbracket abs(\Sigma_{i \in I} t_i) \rrbracket = \llbracket \Sigma_{i \in I} (abs\, t_i) \rrbracket \ . \tag{3.47}$$

(iv) Suppose $\Gamma \vdash t_i : \mu_j \vec{T}.\vec{\mathbb{T}}$ for all $i \in I$. Then

$$\llbracket rep(\Sigma_{i \in I} t_i) \rrbracket = \llbracket \Sigma_{i \in I} (rep\, t_i) \rrbracket \ . \tag{3.48}$$

Proof. The maps *abs* and *rep* are inverses and both linear. $\square$

3.3 Full Abstraction

We'll show that path equivalence captures a notion of contextual equivalence, defined using the denotational semantics. An operational formulation is given after we prove adequacy in Section 3.4.1. Contextual equivalence is formally a type-respecting relation between typing judgements. Whenever such a relation R relates

$$\Gamma_1 \vdash t_1 : \mathbb{P}_1 \quad \text{and} \quad \Gamma_2 \vdash t_2 : \mathbb{P}_2 \ , \tag{3.49}$$

we have syntactic identities $\Gamma_1 \equiv \Gamma_2 \equiv \Gamma$ and $\mathbb{P}_1 \equiv \mathbb{P}_2 \equiv \mathbb{P}$. We'll then use the notation $\Gamma \vdash t_1 \ R \ t_2 : \mathbb{P}$. Below, all our relations will respect types and we'll sometimes write just $t_1 \ R \ t_2$ when the typing information is irrelevant or clear from context.

We define a *program* to be a closed term t of type $!\mathbb{O}$. This type has two values, $\varnothing$ and $\{\varnothing\}$, and so is the simplest type which allows the denotational semantics to distinguish between processes, sometimes called to "make observations". Here, making observations of a program t amounts to saying whether $[\![t]\!] = \varnothing$ or not.

A $(\Gamma, \mathbb{P})$-*program context* C is a term with holes into which a term t with $\Gamma \vdash t : \mathbb{P}$ may be put to form a program $\vdash C(t) : !\mathbb{O}$. The observations made by the denotational semantics give rise to a type-respecting contextual preorder. Suppose $\Gamma \vdash t_1 : \mathbb{P}$ and $\Gamma \vdash t_2 : \mathbb{P}$. We say that t_1 and t_2 are related by *contextual preorder*, written $t_1 \precsim t_2$, iff for all $(\Gamma, \mathbb{P})$-program contexts C, we have $[\![C(t_1)]\!] \neq \varnothing \implies [\![C(t_2)]\!] \neq \varnothing$. If both $t_1 \precsim t_2$ and $t_2 \precsim t_1$, we say that t_1 and t_2 are *contextually equivalent*.

The full abstraction result below implies that contextual equivalence coincides with path equivalence. We formulate it using the associated preorders:

Theorem 3.10 (Full Abstraction) Suppose $\Gamma \vdash t_1 : \mathbb{P}$ and $\Gamma \vdash t_2 : \mathbb{P}$. Then

$$[\![t_1]\!] \subseteq [\![t_2]\!] \iff t_1 \precsim t_2 \ . \tag{3.50}$$

Proof. Suppose $[\![t_1]\!] \subseteq [\![t_2]\!]$ and let C be a $(\Gamma, \mathbb{P})$-program context with $[\![C(t_1)]\!] \neq \varnothing$. As $[\![t_1]\!] \subseteq [\![t_2]\!]$ we have $[\![C(t_2)]\!] \neq \varnothing$ by compositionality and monotonicity, and so $t_1 \precsim t_2$ as wanted.

To prove the converse we define for each path $p : \mathbb{P}$ a closed term t_p of type $\mathbb{P}$ and a $(\mathbb{O}, \mathbb{P})$-program context C_p that respectively "realise" and "consume" the path p, by induction on the structure of p.[1] We'll also need

[1] We have recently become aware that this technique has been applied by Guy McCusker to prove full abstraction for a version of Idealized Algol [53].

realisers t'_P and consumers C'_P of finite sets of paths:

$$\begin{aligned} t_{P\mapsto q} &\equiv_{\mathrm{def}} \lambda x.[C'_P(x) > !x' \Rightarrow t_q] & C_{P\mapsto q} &\equiv_{\mathrm{def}} C_q(-\ t'_P) \\ t_{\beta p} &\equiv_{\mathrm{def}} \beta t_p & C_{\beta p} &\equiv_{\mathrm{def}} C_p(\pi_\beta -) \\ t_P &\equiv_{\mathrm{def}} !t'_P & C_P &\equiv_{\mathrm{def}} [- > !x \Rightarrow C'_P(x)] \\ t_{abs\, p} &\equiv_{\mathrm{def}} abs\, t_p & C_{abs\, p} &\equiv_{\mathrm{def}} C_p(rep\ -) \\ t'_{\{p_1,\dots,p_n\}} &\equiv_{\mathrm{def}} t_{p_1} + \cdots + t_{p_n} \\ C'_{\{p_1,\dots,p_n\}} &\equiv_{\mathrm{def}} [C_{p_1} > !x_1 \Rightarrow \cdots \Rightarrow [C_{p_n} > !x_n \Rightarrow !\varnothing]\cdots] \end{aligned} \tag{3.51}$$

Note that $t'_\varnothing \equiv \varnothing$ and $C'_\varnothing \equiv !\varnothing$. Although the syntax of t'_P and C'_P depends on a choice of permutation of the elements of P, the semantics obtained for different permutations is the same. Indeed, we have (z being a fresh variable):

$$\begin{aligned} [\![t_p]\!] &= \mathrm{y}_{\mathbb{P}}p & [\![\lambda z.C_p(z)]\!] &= \mathrm{y}_{\mathbb{P}\to!\mathbb{O}}(\{p\}\mapsto\varnothing) \\ [\![t'_P]\!] &= i_{\mathbb{P}}P & [\![\lambda z.C'_P(z)]\!] &= \mathrm{y}_{\mathbb{P}\to!\mathbb{O}}(P\mapsto\varnothing) \end{aligned} \tag{3.52}$$

It then follows from the substitution lemma that for any $p : \mathbb{P}$ and $\vdash t : \mathbb{P}$,

$$p \in [\![t]\!] \iff [\![C_p(t)]\!] \neq \varnothing\ . \tag{3.53}$$

Suppose $t_1 \lesssim t_2$ with t_1 and t_2 closed. Given any $p \in [\![t_1]\!]$ we have $[\![C_p(t_1)]\!] \neq \varnothing$ and so using $t_1 \lesssim t_2$, we get $[\![C_p(t_2)]\!] \neq \varnothing$, so that $p \in [\![t_2]\!]$. It follows that $[\![t_1]\!] \subseteq [\![t_2]\!]$.

As for open terms, suppose $\Gamma \equiv x_1 : \mathbb{P}_1, \dots, x_k : \mathbb{P}_k$. Writing $\lambda\vec{x}.t_1$ for the closed term $\lambda x_1.\cdots\lambda x_k.t_1$ and likewise for t_2, we get

$$\begin{aligned} t_1 \lesssim t_2 &\implies \lambda\vec{x}.t_1 \lesssim \lambda\vec{x}.t_2 \\ &\implies [\![\lambda\vec{x}.t_1]\!] \subseteq [\![\lambda\vec{x}.t_2]\!] \\ &\implies [\![t_1]\!] \subseteq [\![t_2]\!]\ . \end{aligned} \tag{3.54}$$

The proof is complete. □

3.4 Operational Semantics

HOPLA can be given an operational semantics using *actions* defined by

$$a ::= u \mapsto a \mid \beta a \mid\ ! \mid abs\, a\ . \tag{3.55}$$

Here, u is a closed term and β ranges over labels associated with sum types. Actions are assigned types using judgements of the form $\mathbb{P} : a : \mathbb{P}'$. Intuitively, performing the action a turns a process of type $\mathbb{P}$ into a process of type $\mathbb{P}'$.

$$\begin{gathered} \frac{\vdash u : \mathbb{P} \quad \mathbb{Q} : a : \mathbb{P}'}{\mathbb{P}\to\mathbb{Q} : u \mapsto a : \mathbb{P}'} \qquad \frac{\mathbb{P}_\beta : a : \mathbb{P}' \quad \beta \in A}{\Sigma_{\alpha\in A}\mathbb{P}_\alpha : \beta a : \mathbb{P}'} \\ \frac{}{!\mathbb{P} : ! : \mathbb{P}} \qquad \frac{\mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}] : a : \mathbb{P}'}{\mu_j\vec{T}.\vec{\mathbb{T}} : abs\, a : \mathbb{P}'} \end{gathered} \tag{3.56}$$

Notice that in $\mathbb{P} : a : \mathbb{P}'$, the type $\mathbb{P}'$ is unique given $\mathbb{P}$ and a. The operational rules define a relation $\mathbb{P} : t \xrightarrow{a} t'$ where $\vdash t : \mathbb{P}$ and $\mathbb{P} : a : \mathbb{P}'$.[2]

$$
\begin{gathered}
\frac{\mathbb{P} : t[rec\,x.t/x] \xrightarrow{a} t'}{\mathbb{P} : rec\,x.t \xrightarrow{a} t'} \qquad \frac{\mathbb{P} : t_j \xrightarrow{a} t'}{\mathbb{P} : \Sigma_{i \in I} t_i \xrightarrow{a} t'} j \in I \\
\frac{\mathbb{Q} : t[u/x] \xrightarrow{a} t'}{\mathbb{P} \to \mathbb{Q} : \lambda x.t \xrightarrow{u \mapsto a} t'} \qquad \frac{\mathbb{P} \to \mathbb{Q} : t \xrightarrow{u \mapsto a} t'}{\mathbb{Q} : t\,u \xrightarrow{a} t'} \\
\frac{\mathbb{P}_\beta : t \xrightarrow{a} t'}{\Sigma_{\alpha \in A} \mathbb{P}_\alpha : \beta t \xrightarrow{\beta a} t'} \qquad \frac{\Sigma_{\alpha \in A} \mathbb{P}_\alpha : t \xrightarrow{\beta a} t'}{\mathbb{P}_\beta : \pi_\beta t \xrightarrow{a} t'} \\
\frac{}{!\mathbb{P} : !t \xrightarrow{!} t} \qquad \frac{!\mathbb{P} : u \xrightarrow{!} u' \quad \mathbb{Q} : t[u'/x] \xrightarrow{a} t'}{\mathbb{Q} : [u > !x \Rightarrow t] \xrightarrow{a} t'} \\
\frac{\mathbb{T}_j[\mu \vec{T}.\vec{\mathbb{T}}/\vec{T}] : t \xrightarrow{a} t'}{\mu_j \vec{T}.\vec{\mathbb{T}} : abs\,t \xrightarrow{abs\,a} t'} \qquad \frac{\mu_j \vec{T}.\vec{\mathbb{T}} : t \xrightarrow{abs\,a} t'}{\mathbb{T}_j[\mu \vec{T}.\vec{\mathbb{T}}/\vec{T}] : rep\,t \xrightarrow{a} t'}
\end{gathered} \tag{3.57}
$$

Example 3.11 The four derivation fragments below show how the operational semantics validates "β-equivalence":

$$
\begin{gathered}
\dfrac{\dfrac{\mathbb{Q} : t[u/x] \xrightarrow{a} t'}{\mathbb{P} \to \mathbb{Q} : \lambda x.t \xrightarrow{u \mapsto a} t'}}{\mathbb{Q} : (\lambda x.t)\, u \xrightarrow{a} t'} \qquad \dfrac{\dfrac{\mathbb{P}_\beta : t \xrightarrow{a} t'}{\Sigma_{\alpha \in A} \mathbb{P}_\alpha : \beta t \xrightarrow{\beta a} t'}}{\mathbb{P}_\beta : \pi_\beta(\beta t) \xrightarrow{a} t'} \\
\dfrac{\dfrac{}{!\mathbb{P} : !u \xrightarrow{!} u} \quad \mathbb{Q} : t[u/x] \xrightarrow{a} t'}{\mathbb{Q} : [!u > !x \Rightarrow t] \xrightarrow{a} t'} \qquad \dfrac{\dfrac{\mathbb{T}_j[\mu \vec{T}.\vec{\mathbb{T}}/\vec{T}] : t \xrightarrow{a} t'}{\mu_j \vec{T}.\vec{\mathbb{T}} : abs\,t \xrightarrow{abs\,a} t'}}{\mathbb{T}_j[\mu \vec{T}.\vec{\mathbb{T}}/\vec{T}] : rep(abs\,t) \xrightarrow{a} t'}
\end{gathered} \tag{3.58}
$$

The actions $u \mapsto a$, βa, and $abs\,a$ carry information about deconstructor contexts up the derivation tree. Similar use of labels appears e.g. in [28]. □

The operational rules are type-correct:

Proposition 3.12 If $\mathbb{P} : t \xrightarrow{a} t'$ with $\mathbb{P} : a : \mathbb{P}'$, then $\vdash t' : \mathbb{P}'$.

Proof. By rule-induction on the operational rules.

Abstraction. Suppose $\mathbb{P} \to \mathbb{Q} : \lambda x.t \xrightarrow{u \mapsto a} t'$ is derived from $\mathbb{Q} : t[u/x] \xrightarrow{a} t'$ with $\mathbb{P} \mapsto \mathbb{Q} : u \mapsto a : \mathbb{P}'$. By typing of actions, we have $\vdash u : \mathbb{P}$ and $\mathbb{Q} : a : \mathbb{P}'$. The induction hypothesis then yields $\vdash t' : \mathbb{P}'$ as wanted. Note also that the substitution $t[u/x]$ is well-formed as $x : \mathbb{P} \vdash t : \mathbb{Q}$ follows from $\vdash \lambda x.t : \mathbb{P} \to \mathbb{Q}$ by the typing rules.

[2]The explicit types in the operational rules were missing in the rules given in [65]. They are needed to ensure that the types of t and a agree in transitions.

Application. Suppose $\mathbb{Q} : t\ u \xrightarrow{a} t'$ is derived from $\mathbb{P} \to \mathbb{Q} : t \xrightarrow{u \mapsto a} t'$ with $\mathbb{Q} : a : \mathbb{P}'$. By the premise and typing rules, we have $\vdash t : \mathbb{P} \to \mathbb{Q}$ and $\vdash u : \mathbb{P}$, such that $\mathbb{P} \to \mathbb{Q} : u \mapsto a : \mathbb{P}'$. The induction hypothesis then yields $\vdash t' : \mathbb{P}'$ as wanted.

Prefixing. Suppose $!\mathbb{P} : !t \xrightarrow{!} t$ with $!\mathbb{P} : ! : \mathbb{P}$. Then $\vdash !t : !\mathbb{P}$ and so by the typing rules, $\vdash t : \mathbb{P}$ as wanted.

Prefix match. Suppose $\mathbb{Q} : [u > !x \Rightarrow t] \xrightarrow{a} t'$ is derived from $!\mathbb{P} : u \xrightarrow{!} u'$ and $\mathbb{Q} : t[u'/x] \xrightarrow{a} t'$ with $\mathbb{Q} : a : \mathbb{P}'$. By the induction hypothesis applied to the right premise, we get $\vdash t' : \mathbb{P}'$ as wanted. Note also that we have $\vdash u : !\mathbb{P}$ and therefore $\vdash u' : \mathbb{P}$ by the induction hypothesis for the left premise. Thus, as $x : \mathbb{P} \vdash t : \mathbb{Q}$, the substitution $t[u'/x]$ is well-formed.

The remaining cases are handled similarly. $\square$

In accordance with the above, we'll write $\mathbb{P} : t \xrightarrow{a} t' : \mathbb{P}'$ when $\mathbb{P} : t \xrightarrow{a} t'$ and $\mathbb{P} : a : \mathbb{P}'$.

3.4.1 Soundness and Adequacy

The operational semantics gives rise to another notion of observation that can be made of a process: we may observe the action $\mathbb{P} : a : \mathbb{P}'$ when deriving $\mathbb{P} : t \xrightarrow{a} t' : \mathbb{P}'$. We'll start by reducing such general observations to observations of !-transitions (Lemma 3.13 below) in order to simplify stating and proving soundness and adequacy results.

Intuitively the reduction comes about by applying the deconstructor contexts carried by the action a to the process t, cf. the remark of Example 3.11. Formally, we define a syntactic operator a^* by structural induction on a in the left column below. For convenience, the right column defines corresponding linear maps $a^* : \mathbb{P} \to !\mathbb{P}'$ with the property that $a^*[\![t]\!] = [\![a^* t]\!]$ whenever $\mathbb{P} : a : \mathbb{P}'$ and $\vdash t : \mathbb{P}$.

$$\begin{aligned} (u \mapsto a)^* t &\equiv_{\text{def}} a^*(t\ u) & \qquad (u \mapsto a)^* &=_{\text{def}} a^* \circ \mathit{app} \circ (- \,\&\, [\![u]\!]) \\ (\beta a)^* t &\equiv_{\text{def}} a^*(\pi_\beta t) & (\beta a)^* &=_{\text{def}} a^* \circ \pi_\beta \\ !^* t &\equiv_{\text{def}} t & !^* &=_{\text{def}} 1_{!\mathbb{P}} \\ (\mathit{abs}\, a)^* t &\equiv_{\text{def}} a^*(\mathit{rep}\, t) & (\mathit{abs}\, a)^* &=_{\text{def}} a^* \circ \mathit{rep} \end{aligned} \tag{3.59}$$

Lemma 3.13 $\mathbb{P} : t \xrightarrow{a} t' : \mathbb{P}' \iff !\mathbb{P}' : a^* t \xrightarrow{!} t' : \mathbb{P}'$.

Proof. By structural induction on a exploiting the fact that there is only one operational rule deriving transitions from each of the constructs application $t\ u$, injection βt, and folding $\mathit{abs}\, t$ so that

$$\begin{aligned} \mathbb{P} \to \mathbb{Q} : t \xrightarrow{u \mapsto a} t' : \mathbb{P}' &\iff \mathbb{Q} : t\ u \xrightarrow{a} t' : \mathbb{P}' \\ \Sigma_{\alpha \in A} \mathbb{P}_\alpha : t \xrightarrow{\beta a} t' : \mathbb{P}' &\iff \mathbb{P}_\beta : \pi_\beta t \xrightarrow{a} t' : \mathbb{P}' \\ \mu_j \vec{T}.\vec{\mathbb{T}} : t \xrightarrow{\mathit{abs}\, a} t' : \mathbb{P}' &\iff \mathbb{T}_j[\mu \vec{T}.\vec{\mathbb{T}}/\vec{T}] : \mathit{rep}\, t \xrightarrow{a} t' : \mathbb{P}' \end{aligned} \tag{3.60}$$

Function space. We argue as follows:

$$\begin{array}{lll} \mathbb{P} \to \mathbb{Q} : t \xrightarrow{u \mapsto a} t' : \mathbb{P}' & & \\ \iff \mathbb{Q} : t\ u \xrightarrow{a} t' : \mathbb{P}' & & \text{(by (3.60))} \\ \iff !\mathbb{P}' : a^*(t\ u) \xrightarrow{!} t' : \mathbb{P}' & & \text{(ind. hyp.)} \\ \iff !\mathbb{P}' : (u \mapsto a)^* t \xrightarrow{!} t' : \mathbb{P}' & & \text{(def. of } (u \mapsto a)^* t) \end{array}$$

Sum. We argue as follows:

$$\begin{array}{ll} \Sigma_{\alpha \in A} \mathbb{P}_\alpha : t \xrightarrow{\beta a} t' : \mathbb{P}' & \\ \iff \mathbb{P}_\beta : \pi_\beta t \xrightarrow{a} t' : \mathbb{P}' & \text{(by (3.60))} \\ \iff !\mathbb{P}' : a^*(\pi_\beta t) \xrightarrow{!} t' : \mathbb{P}' & \text{(ind. hyp.)} \\ \iff !\mathbb{P}' : (\beta a)^* t \xrightarrow{!} t' : \mathbb{P}' & \text{(def. of } (\beta a)^* t) \end{array}$$

Prefix. We argue as follows:

$$\begin{array}{ll} !\mathbb{P} : t \xrightarrow{!} t' : \mathbb{P} & \\ \iff !\mathbb{P} : !^* t \xrightarrow{!} t' : \mathbb{P} & \text{(def. of } !^* t) \end{array}$$

Recursion. We argue as follows:

$$\begin{array}{ll} \mu_j \vec{T}.\vec{\mathbb{T}} : t \xrightarrow{abs\ a} t' : \mathbb{P}' & \\ \iff \mathbb{T}_j[\mu \vec{T}.\vec{\mathbb{T}}/\vec{T}] : rep\ t \xrightarrow{a} t' : \mathbb{P}' & \text{(by (3.60))} \\ \iff !\mathbb{P}' : a^*(rep\ t) \xrightarrow{!} t' : \mathbb{P}' & \text{(ind. hyp.)} \\ \iff !\mathbb{P}' : (abs\ a)^* t \xrightarrow{!} t' : \mathbb{P}' & \text{(def. of } (abs\ a)^* t) \end{array}$$

The structural induction is complete. □

Proposition 3.14 (Soundness) If $\mathbb{P} : t \xrightarrow{a} t' : \mathbb{P}'$, then $[\![!t']\!] \subseteq a^*[\![t]\!]$.

Proof. By rule-induction on the transition rules. We'll dispense with the typing information in transitions for clarity.

Recursive definition. Suppose $rec\,x.t \xrightarrow{a} t'$ is derived from $t[rec\,x.t/x] \xrightarrow{a} t'$. By the induction hypothesis and Proposition 3.5,

$$[\![!t']\!] \subseteq a^*[\![t[rec\,x.t/x]]\!] = a^*[\![rec\,x.t]\!] \ . \tag{3.61}$$

Nondeterministic sum. Suppose $\Sigma_{i \in I} t_i \xrightarrow{a} t'$ is derived from $t_j \xrightarrow{a} t'$ for some $j \in I$. By the induction hypothesis and linearity of a^*,

$$[\![!t']\!] \subseteq a^*[\![t_j]\!] = [\![a^* t_j]\!] \subseteq [\![\Sigma_{i \in I} a^* t_i]\!] = a^*[\![\Sigma_{i \in I} t_i]\!] \ . \tag{3.62}$$

Abstraction. Suppose $\lambda x.t \xrightarrow{u \mapsto a} t'$ is derived from $t[u/x] \xrightarrow{a} t'$. By the induction hypothesis and Proposition 3.6,

$$[\![!t']\!] \subseteq a^*[\![t[u/x]]\!] = a^*[\![(\lambda x.t)\ u]\!] = (u \mapsto a)^*[\![\lambda x.t]\!] \ . \tag{3.63}$$

Application. Suppose $t\ u \xrightarrow{a} t'$ is derived from $t \xrightarrow{u \mapsto a} t'$. By the induction hypothesis,

$$[\![!t']\!] \subseteq (u \mapsto a)^* [\![t]\!] = a^* [\![t\ u]\!] \ . \tag{3.64}$$

Injection. Suppose $\beta t \xrightarrow{\beta a} t'$ is derived from $t \xrightarrow{a} t'$. By the induction hypothesis and Proposition 3.7,

$$[\![!t']\!] \subseteq a^* [\![t]\!] = a^* [\![\pi_\beta(\beta t)]\!] = (\beta a)^* [\![\beta t]\!] \ . \tag{3.65}$$

Projection. Suppose $\pi_\beta t \xrightarrow{a} t'$ is derived from $t \xrightarrow{\beta a} t'$. By the induction hypothesis,

$$[\![!t']\!] \subseteq (\beta a)^* [\![t]\!] = a^* [\![\pi_\beta t]\!] \ . \tag{3.66}$$

Prefixing. Consider the transition $!t \xrightarrow{!} t$. By definition, $[\![!t]\!] = !^* [\![!t]\!]$.

Prefix match. Suppose $[u > !x \Rightarrow t] \xrightarrow{a} t'$ is derived from $u \xrightarrow{!} u'$ and $t[u'/x] \xrightarrow{a} t'$. By the induction hypothesis for u, we have $[\![!u']\!] \subseteq !^* [\![u]\!] = [\![u]\!]$, and so by the induction hypothesis for t, Proposition 3.8, and monotonicity,

$$[\![!t']\!] \subseteq a^* [\![t[u'/x]]\!] = a^* [\![[!u' > !x \Rightarrow t]]\!] \subseteq a^* [\![[u > !x \Rightarrow t]]\!] \ . \tag{3.67}$$

Fold. Suppose $abs\, t \xrightarrow{abs\, a} t'$ is derived from $t \xrightarrow{a} t'$. By the induction hypothesis and Proposition 3.9,

$$[\![!t']\!] \subseteq a^* [\![t]\!] = a^* [\![rep(abs\, t)]\!] = (abs\, a)^* [\![abs\, t]\!] \ . \tag{3.68}$$

Unfold. Suppose $rep\, t \xrightarrow{a} t'$ is derived from $t \xrightarrow{abs\, a} t'$. By the induction hypothesis,

$$[\![!t']\!] \subseteq (abs\, a)^* [\![t]\!] = a^* [\![rep\, t]\!] \ . \tag{3.69}$$

The rule-induction is complete. □

We prove adequacy by using logical relations $X \subseteq_{\mathbb{P}} t$ between subsets $X \subseteq \mathbb{P}$ and closed terms of type $\mathbb{P}$. Intuitively, $X \subseteq_{\mathbb{P}} t$ means that all paths in X can be "operationally realised" by t. Because of recursive types, these relations cannot be defined by structural induction on the type $\mathbb{P}$ and we therefore employ a trick essentially due to Martin-Löf (see [95], Ch. 13). We define auxiliary relations $p \in_{\mathbb{P}} t$ between paths $p : \mathbb{P}$ and closed terms t of type $\mathbb{P}$, by induction on the structure of p:

$$X \subseteq_{\mathbb{P}} t \iff_{\text{def}} \forall p \in X.\ p \in_{\mathbb{P}} t$$

$$\begin{aligned}
P \mapsto q \in_{\mathbb{P} \to \mathbb{Q}} t &\iff_{\text{def}} \forall u.\ (P \subseteq_{\mathbb{P}} u \implies q \in_{\mathbb{Q}} t\ u) \\
\beta p \in_{\Sigma_{\alpha \in A} \mathbb{P}_\alpha} t &\iff_{\text{def}} p \in_{\mathbb{P}_\beta} \pi_\beta t \\
P \in_{!\mathbb{P}} t &\iff_{\text{def}} \exists t'.\ !\mathbb{P} : t \xrightarrow{!} t' : \mathbb{P} \text{ and } P \subseteq_{\mathbb{P}} t' \\
abs\, p \in_{\mu_j \vec{T}.\vec{\mathbb{T}}} t &\iff_{\text{def}} p \in_{\mathbb{T}_j[\mu \vec{T}.\vec{\mathbb{T}}/\vec{T}]} rep\, t
\end{aligned} \tag{3.70}$$

Lemma 3.15 (Main Lemma) Suppose $\vdash t : \mathbb{P}$. Then $[\![t]\!] \subseteq_{\mathbb{P}} t$.

Proof. We need two technical results, which can both be proved by induction on the structure of paths. One says that $\in_{\mathbb{P}}$ is closed on the left by $\leq_{\mathbb{P}}$, the other that $\in_{\mathbb{P}}$ is closed on the right by the relation $\precsim_1$, defined by $t_1 \precsim_1 t_2$ iff $\mathbb{P} : t_1 \xrightarrow{a} t' : \mathbb{P}'$ implies $\mathbb{P} : t_2 \xrightarrow{a} t' : \mathbb{P}'$.

Lemma 3.16 If $p \leq_{\mathbb{P}} p'$ and $p' \in_{\mathbb{P}} t$, then $p \in_{\mathbb{P}} t$.

Lemma 3.17 If $p \in_{\mathbb{P}} t_1$ and $t_1 \precsim_1 t_2$, then $p \in_{\mathbb{P}} t_2$.

It follows from Lemma 3.16 that for any subset X of $\mathbb{P}$ we have $X \subseteq_{\mathbb{P}} t$ iff the down-closure of X, written $\bar{X}$, satisfies $\bar{X} \subseteq_{\mathbb{P}} t$. Lemma 3.17 will be used freely below.

The proof of the main lemma proceeds by structural induction on terms using the induction hypothesis

> Suppose $x_1 : \mathbb{P}_1, \ldots, x_k : \mathbb{P}_k \vdash t : \mathbb{P}$ and let $\vdash s_j : \mathbb{P}_j$ with $X_j \subseteq_{\mathbb{P}_j} s_j$ for $1 \leq j \leq k$. Then $[\![t]\!](\bar{X}_1, \ldots, \bar{X}_k) \subseteq_{\mathbb{P}} t[s_1/x_1, \ldots, s_k/x_k]$.

We'll abbreviate $x_1 : \mathbb{P}_1, \ldots, x_k : \mathbb{P}_k$ to Γ, $(\bar{X}_1, \ldots, \bar{X}_k)$ to X, and the substitution $[s_1/x_1, \ldots, s_k/x_k]$ to $[s]$.

Variable. Let $\Gamma \vdash x_j : \mathbb{P}_j$, with j between 1 and k, and $\vdash s_j : \mathbb{P}_j$ with $X_j \subseteq_{\mathbb{P}_j} s_j$ for $1 \leq j \leq k$. We must show that $[\![x_j]\!]X \subseteq_{\mathbb{P}_j} x_j[s]$. Now, $[\![x_j]\!]X = \bar{X}_j$ and $x_j[s] \equiv s_j$ so this amounts to $\bar{X}_j \subseteq_{\mathbb{P}_j} s_j$ which by the remarks above is equivalent to $X_j \subseteq_{\mathbb{P}_j} s_j$.

Recursive definition. Let $\Gamma \vdash \mathit{rec}\, x.t : \mathbb{P}$ and $\vdash s_j : \mathbb{P}_j$ with $X_j \subseteq_{\mathbb{P}_j} s_j$ for $1 \leq j \leq k$. We must show that $[\![\mathit{rec}\, x.t]\!]X \subseteq_{\mathbb{P}} \mathit{rec}\, x.t[s]$. Now, $[\![\mathit{rec}\, x.t]\!]X = (\mathit{fix}\, f)X$ where f maps $g : \Gamma \to \mathbb{P}$ to the composition

$$\Gamma \xrightarrow{\Delta_\Gamma} \Gamma \,\&\, \Gamma \xrightarrow{1_\Gamma \& g} \Gamma \,\&\, \mathbb{P} \xrightarrow{t} \mathbb{P} \ . \tag{3.71}$$

We'll show by induction on n that $f^n(\varnothing)X \subseteq_{\mathbb{P}} \mathit{rec}\, x.t[s]$ for all $n \in \omega$. Having done so we may argue as follows: Since

$$[\![\mathit{rec}\, x.t]\!]X = (\mathit{fix}\, f)X = (\textstyle\bigcup_{n\in\omega} f^n\varnothing)X = \bigcup_{n\in\omega} ((f^n\varnothing)X) \ , \tag{3.72}$$

we have that $p \in [\![\mathit{rec}\, x.t]\!]X$ implies the existence of an $n \in \omega$ such that $p \in (f^n\varnothing)X$. Therefore $[\![\mathit{rec}\, x.t]\!]X \subseteq_{\mathbb{P}} \mathit{rec}\, x.t[s]$ as wanted.

Basis. Here, $(f^0\varnothing)X = \varnothing$. By definition of $\subseteq_{\mathbb{P}}$ we get $\varnothing \subseteq_{\mathbb{P}} t$ for any type $\mathbb{P}$ and term $\vdash t : \mathbb{P}$.

Step. Suppose $(f^n\varnothing)X \subseteq_{\mathbb{P}} \mathit{rec}\, x.t[s]$. By the assumption of the lemma, $X_j \subseteq_{\mathbb{P}} s_j$ for each $1 \leq j \leq k$, and so by the induction hypothesis of the structural induction,

$$[\![t]\!](X, (f^n\varnothing)X) \subseteq_{\mathbb{P}} t[s][\mathit{rec}\, x.t[s]/x] \ . \tag{3.73}$$

So if $p \in (f^{n+1}\varnothing)X$, then since $(f^{n+1}\varnothing)X = [\![t]\!](X,(f^n\varnothing)X)$ we have $p \in_{\mathbb{P}} t[s][rec\,x.t[s]/x]$. By the transition rules we have $t[s][rec\,x.t[s]/x] \precsim_1 rec\,x.t[s]$, and so $p \in_{\mathbb{P}} rec\,x.t[s]$. We conclude $(f^{n+1}\varnothing)X \subseteq_{\mathbb{P}} rec\,x.t[s]$ and the mathematical induction is complete.

Nondeterministic sum. Let $\Gamma \vdash \Sigma_{i\in I}t_i : \mathbb{P}$ and $\vdash s_j : \mathbb{P}_j$ with $X_j \subseteq_{\mathbb{P}_j} s_j$ for $1 \leq j \leq k$. We must show that $[\![\Sigma_{i\in I}t_i]\!]X \subseteq_{\mathbb{P}} \Sigma_{i\in I}t_i[s]$. Now, $[\![\Sigma_{i\in I}t_i]\!]X = \Sigma_{i\in I}[\![t_i]\!]X$. So if $p \in [\![\Sigma_{i\in I}t_i]\!]X$, there exists $j \in I$ with $p \in [\![t_j]\!]X$. Using the induction hypothesis for t_j we have $p \in_{\mathbb{P}} t_j[s]$. By the transition rules, $t_j[s] \precsim_1 \Sigma_{i\in I}t_i[s]$ and so $p \in_{\mathbb{P}} \Sigma_{i\in I}t_i[s]$ as wanted.

Abstraction. Let $\Gamma \vdash \lambda x.t : \mathbb{P} \to \mathbb{Q}$ and $\vdash s_j : \mathbb{P}_j$ with $X_j \subseteq_{\mathbb{P}_j} s_j$ for $1 \leq j \leq k$. We must show that $[\![\lambda x.t]\!]X \subseteq_{\mathbb{P}\to\mathbb{Q}} (\lambda x.t)[s]$. So let $P \mapsto q \in [\![\lambda x.t]\!]X$. By the denotational semantics, we then have $q \in [\![t]\!](X, i_{\mathbb{P}}P)$. We must show that $P \mapsto q \in_{\mathbb{P}\to\mathbb{Q}} (\lambda x.t)[s]$. So suppose $\vdash u : \mathbb{P}$ with $P \subseteq_{\mathbb{P}} u$. We must then show $q \in_{\mathbb{Q}} (\lambda x.t)[s]\ u$. By the transition rules, $t[s][u/x] \precsim_1 (\lambda x.t)[s]\ u$ and so it is sufficient to show $q \in_{\mathbb{Q}} t[s][u/x]$. Now, by the induction hypothesis, we know that $[\![t]\!](X, i_{\mathbb{P}}P) \subseteq_{\mathbb{Q}} t[s][u/x]$ and so, with $q \in [\![t]\!](X, i_{\mathbb{P}}P)$, we are done.

Application. Let $\Gamma \vdash t\ u : \mathbb{Q}$ and $\vdash s_j : \mathbb{P}_j$ with $X_j \subseteq_{\mathbb{P}_j} s_j$ for $1 \leq j \leq k$. We must show that $[\![t\ u]\!]X \subseteq_{\mathbb{Q}} (t\ u)[s]$. So suppose $q \in [\![t\ u]\!]X$. By the denotational semantics, there exists $P \in {!\mathbb{P}}$ such that $P \mapsto q \in [\![t]\!]X$ and $P \subseteq [\![u]\!]X$. By the induction hypothesis for t, we have $[\![t]\!]X \subseteq_{\mathbb{P}\to\mathbb{Q}} t[s]$ and so $P \mapsto q \in_{\mathbb{P}\to\mathbb{Q}} t[s]$. This means that given any $\vdash u' : \mathbb{P}$ with $P \subseteq_{\mathbb{P}} u'$, we have $q \in_{\mathbb{Q}} t[s]\ u'$. Now using the induction hypothesis for u we get that $[\![u]\!]X \subseteq_{\mathbb{P}} u[s]$ and so, since $P \subseteq [\![u]\!]X$, we have $P \subseteq_{\mathbb{P}} u[s]$ so that $q \in_{\mathbb{Q}} t[s]\ u[s] \equiv (t\ u)[s]$ as wanted.

Injection. Let $\Gamma \vdash \beta t : \Sigma_{\alpha\in A}\mathbb{P}_\alpha$ and $\vdash s_j : \mathbb{P}_j$ with $X_j \subseteq_{\mathbb{P}_j} s_j$ for $1 \leq j \leq k$. We must show that $[\![\beta t]\!]X \subseteq_{\Sigma_{\alpha\in A}\mathbb{P}_\alpha} (\beta t)[s]$. So suppose $\beta p \in [\![\beta t]\!]X$; by the denotational semantics, $p \in [\![t]\!]X$. We must then show that $\beta p \in_{\Sigma_{\alpha\in A}\mathbb{P}_\alpha} (\beta t)[s]$ which means that $p \in_{\mathbb{P}_\beta} \pi_\beta(\beta t[s])$. By the transition rules, we have $t[s] \precsim_1 \pi_\beta(\beta t[s])$ so it is sufficient to show that $p \in_{\mathbb{P}_\beta} t[s]$. By the induction hypothesis, $[\![t]\!]X \subseteq_{\mathbb{P}_\beta} t[s]$ and so, since $p \in [\![t]\!]X$ we have $p \in_{\mathbb{P}_\beta} t[s]$ as wanted.

Projection. Let $\Gamma \vdash \pi_\beta t : \mathbb{P}_\beta$ with $\Gamma \vdash t : \Sigma_{\alpha\in A}\mathbb{P}_\alpha$ and $\beta \in A$, and $\vdash s_j : \mathbb{P}_j$ with $X_j \subseteq_{\mathbb{P}_j} s_j$ for $1 \leq j \leq k$. We must show that $[\![\pi_\beta t]\!]X \subseteq_{\mathbb{P}_\beta} \pi_\beta t[s]$. So suppose $p \in [\![\pi_\beta t]\!]X$; by the denotational semantics, $\beta p \in [\![t]\!]X$. By the induction hypothesis, $[\![t]\!]X \subseteq_{\Sigma_{\alpha\in A}\mathbb{P}_\alpha} t[s]$ and so $\beta p \in_{\Sigma_{\alpha\in A}\mathbb{P}_\alpha} t[s]$ which means that $p \in_{\mathbb{P}_\beta} \pi_\beta t[s]$ as wanted.

Prefixing. Let $\Gamma \vdash {!t} : {!\mathbb{P}}$ and $\vdash s_j : \mathbb{P}_j$ with $X_j \subseteq_{\mathbb{P}_j} s_j$ for $1 \leq j \leq k$. We must show that $[\![!t]\!]X \subseteq_{!\mathbb{P}} {!t[s]}$. So suppose $P \in [\![!t]\!]X$; by the denotational semantics, $P \subseteq [\![t]\!]X$. We must then show that $P \in_{!\mathbb{P}} {!t[s]}$, and so since the transition rules provide a derivation ${!\mathbb{P}} : {!t[s]} \xrightarrow{!} t[s] : \mathbb{P}$, it is enough to show that $P \subseteq_{\mathbb{P}} t[s]$. Now, by the induction hypothesis, $[\![t]\!]X \subseteq_{\mathbb{P}} t[s]$ and so, since $P \subseteq [\![t]\!]X$ we have $P \subseteq_{\mathbb{P}} t[s]$ as wanted.

Prefix match. Let $\Gamma \vdash [u > !x \Rightarrow t] : \mathbb{Q}$ and $\vdash s_j : \mathbb{P}_j$ with $X_j \subseteq_{\mathbb{P}_j} s_j$ for $1 \leq j \leq k$. By renaming x if necessary, we may assume that x is not one of the x_j. We must show that $[\![[u > !x \Rightarrow t]]\!]X \subseteq_{\mathbb{Q}} [u > !x \Rightarrow t][s]$. So suppose $q \in [\![[u > !x \Rightarrow t]]\!]X$; by the denotational semantics, there exists $P \in !\mathbb{P}$ such that $q \in [\![t]\!](X, i_{\mathbb{P}}P)$ and $P \in [\![u]\!]X$. By the induction hypothesis for u we have $[\![u]\!]X \subseteq_{!\mathbb{P}} u[s]$ and so since $P \in [\![u]\!]X$, there exists u' such that $!\mathbb{P} : u[s] \xrightarrow{!} u' : \mathbb{P}$ and $P \subseteq_{\mathbb{P}} u'$. Hence, by the induction hypothesis for t we have $[\![t]\!](X, i_{\mathbb{P}}P) \subseteq_{\mathbb{Q}} t[s][u'/x]$ and so since $q \in [\![t]\!](X, i_{\mathbb{P}}P)$ we have $q \in_{\mathbb{Q}} t[s][u'/x]$. Now, by the transition rules, $t[s][u'/x] \precsim_1 [u > !x \Rightarrow t][s]$ and so $q \in_{\mathbb{Q}} [u > !x \Rightarrow t][s]$ as wanted.

Fold. Let $\Gamma \vdash abs\, t : \mu_j \vec{T}.\vec{\mathbb{T}}$ and $\vdash s_j : \mathbb{P}_j$ with $X_j \subseteq_{\mathbb{P}_j} s_j$ for $1 \leq j \leq k$. We must show that $[\![abs\, t]\!]X \subseteq_{\mu_j \vec{P}.\vec{\mathbb{T}}} abs\, t[s]$. So suppose $abs\, q \in [\![abs\, t]\!]X$ such that $q \in [\![t]\!]X$. By the induction hypothesis, $q \in_{\mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}]} t[s]$ and since $t[s] \precsim_1 rep\, abs\, t[s]$, we have $q \in_{\mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}]} rep\, abs\, t[s]$ which means that $abs\, q \in_{\mu_j \vec{P}.\vec{\mathbb{T}}} abs\, t[s]$ as wanted.

Unfold. Let $\Gamma \vdash rep\, t : \mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}]$ and $\vdash s_j : \mathbb{P}_j$ with $X_j \subseteq_{\mathbb{P}_j} s_j$ for $1 \leq j \leq k$. We must show that $[\![rep\, t]\!]X \subseteq_{\mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}]} rep\, t[s]$. So suppose $q \in [\![rep\, t]\!]X$ such that $abs\, q \in [\![t]\!]X$. By the induction hypothesis, $abs\, q \in_{\mu_j \vec{T}.\vec{\mathbb{T}}} t[s]$ and so $q \in_{\mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}]} rep\, t[s]$ as wanted.

The structural induction is complete. □

Proposition 3.18 (Adequacy) Suppose $\vdash t : \mathbb{P}$ and $\mathbb{P} : a : \mathbb{P}'$. Then

$$a^*[\![t]\!] \neq \varnothing \iff \exists t'.\ \mathbb{P} : t \xrightarrow{a} t' : \mathbb{P}' \ . \tag{3.74}$$

Proof. The "$\Leftarrow$" direction follows from soundness. For the converse, assume $a^*[\![t]\!] = [\![a^* t]\!] \neq \varnothing$. Then because $[\![a^* t]\!]$ is a downwards-closed subset of $!\mathbb{P}'$ which has least element $\varnothing$, we must have $\varnothing \in [\![a^* t]\!]$. Thus $\varnothing \in_{!\mathbb{P}} a^* t$ by the main lemma, which implies the existence of a term t' such that $!\mathbb{P}' : a^* t \xrightarrow{!} t' : \mathbb{P}'$. Thus, $\mathbb{P} : t \xrightarrow{a} t' : \mathbb{P}'$ by Lemma 3.13. □

3.4.2 Full Abstraction w.r.t. Operational Semantics

Adequacy allows an operational formulation of contextual equivalence. In addition to observing arbitrary transitions and transitions of processes of prefix type, we may observe just transitions $!\mathbb{O} : t \xrightarrow{!} t' : \mathbb{O}$ of programs. We'll write this as $t \xrightarrow{!}$. The three notions of observation are equivalent according to

$$\begin{aligned} \exists t'.\ \mathbb{P} : t \xrightarrow{a} t' : \mathbb{P}' &\iff \exists t'.\ !\mathbb{P}' : a^* t \xrightarrow{!} t' : \mathbb{P}' \\ &\iff [a^* t > !x \Rightarrow !\varnothing] \xrightarrow{!} \end{aligned} \tag{3.75}$$

By adequacy, we have $t \xrightarrow{!}$ iff $[\![t]\!] \neq \varnothing$. Hence, two terms t_1 and t_2 with $\Gamma \vdash t_1 : \mathbb{P}$ and $\Gamma \vdash t_2 : \mathbb{P}$ are related by contextual preorder iff for all $(\Gamma, \mathbb{P})$-program contexts C, we have $C(t_1) \xrightarrow{!} \implies C(t_2) \xrightarrow{!}$.

Full abstraction is often formulated in terms of this operational preorder. With t_1 and t_2 as above, the inclusion $[\![t_1]\!] \subseteq [\![t_2]\!]$ holds iff for all $(\Gamma, \mathbb{P})$-program contexts C, we have $C(t_1) \xrightarrow{!} \implies C(t_2) \xrightarrow{!}$.

3.4.3 A Correspondence Result

Consider a process t of prefix type $!\mathbb{P}$ and its successors t' after transitions $!\mathbb{P} : t \xrightarrow{!} t' : \mathbb{P}$. If we take the nondeterministic sum of all the t''s prefixed by $!$, we should end up with a process behaving identically to t, so $\Sigma_{t'} [\![!t']\!] = [\![t]\!]$ where t' ranges over terms such that $!\mathbb{P} : t \xrightarrow{!} t' : \mathbb{P}$. Using Lemma 3.13 and the linear maps $a^* : \mathbb{P} \to !\mathbb{P}'$ we can generalise this to a statement pertaining to all types (Theorem 3.20 below). This result subsumes both soundness and adequacy and to prove it, we need to strengthen the main lemma:

Lemma 3.19 Suppose $\vdash t : \mathbb{P}$ and $p \in \mathbb{P}$. Then $p \in_{\mathbb{P}} t \iff p \in [\![t]\!]$ (and hence $X \subseteq_{\mathbb{P}} t \iff X \subseteq [\![t]\!]$).

Proof. The "$\Rightarrow$" direction follows from the main lemma. The converse is shown by induction on the structure of paths:

Function space. Suppose $P \mapsto q \in_{\mathbb{P} \to \mathbb{Q}} t$. From the proof of full abstraction, we get a term t'_P with $[\![t'_P]\!] = i_{\mathbb{P}} P$, so that $P \subseteq [\![t'_P]\!]$ and thus $P \subseteq_{\mathbb{P}} t'_P$ by the main lemma. The definition of $\in_{\mathbb{P} \to \mathbb{Q}}$ then yields $q \in_{\mathbb{Q}} t\ t'_P$ and by the induction hypothesis we obtain $q \in [\![t\ t'_P]\!]$ from which $P \mapsto q \in [\![t]\!]$ follows.

Sum type. Suppose $\beta p \in_{\Sigma_{\alpha \in A} \mathbb{P}_\alpha} t$. By definition, this means that $p \in_{\mathbb{P}_\beta} \pi_\beta t$, and so by the induction hypothesis, $p \in [\![\pi_\beta t]\!]$ from which $\beta p \in [\![t]\!]$ follows.

Prefix type. Suppose $P \in_{!\mathbb{P}} t$. By definition, there exists a term t' such that $!\mathbb{P} : t \xrightarrow{!} t' : \mathbb{P}$ and $P \subseteq_{\mathbb{P}} t'$. Applying the induction hypothesis to all paths $p \in P$, we get $P \subseteq [\![t']\!]$ from which $P \in [\![!t']\!]$ follows. By soundness, $[\![!t']\!] \subseteq [\![t]\!]$ and so $P \in [\![t]\!]$ as wanted.

Recursive types. Suppose $abs\, p \in_{\mu_j \vec{T}.\vec{\mathbb{T}}} t$. By definition, this means that $p \in_{\mathbb{T}_j[\mu \vec{T}.\vec{\mathbb{T}}/\vec{T}]} rep\, t$, and so by the induction hypothesis, $p \in [\![rep\, t]\!]$ from which $abs\, p \in [\![t]\!]$ follows.

The induction is complete. $\square$

Theorem 3.20 (Correspondence) Let $\vdash t : \mathbb{P}$ and $\mathbb{P} : a : \mathbb{P}'$. Then $\Sigma_{t'} [\![!t']\!] = a^* [\![t]\!]$ where t' ranges over terms such that $\mathbb{P} : t \xrightarrow{a} t' : \mathbb{P}'$.

Proof. The inclusion $\Sigma_{t'}[\![!t']\!] \subseteq a^*[\![t]\!]$ is just a reformulation of soundness. For the converse, assume that $P \in a^*[\![t]\!] = [\![a^*t]\!]$. Then by the main lemma we have $P \in_{!\mathbb{P}'} a^*t$. By definition, this means that there exists a term t' such that $!\mathbb{P}' : a^*t \xrightarrow{!} t' : \mathbb{P}'$ and $P \subseteq_{\mathbb{P}'} t'$. By Lemma 3.13, we get $\mathbb{P} : t \xrightarrow{a} t' : \mathbb{P}'$ and Lemma 3.19 yields $P \subseteq [\![t']\!]$ from which $P \in [\![!t']\!]$ follows. $\square$

In Chapter 6 the correspondence result is strengthened to what we call "strong correspondence", relating derivations in the operational semantics to realisers in presheaf denotations.

3.5 Simulation

We'll consider three standard notions of process equivalence, all based on *typed* simulation. A type-respecting relation R on closed terms is a bisimulation if the following holds. If $\vdash t_1 \; R \; t_2 : \mathbb{P}$, then

1. if $\mathbb{P} : t_1 \xrightarrow{a} t_1' : \mathbb{P}'$, then $\mathbb{P} : t_2 \xrightarrow{a} t_2' : \mathbb{P}'$ for some t_2' such that $\vdash t_1' \; R \; t_2' : \mathbb{P}'$;
2. if $\mathbb{P} : t_2 \xrightarrow{a} t_2' : \mathbb{P}'$, then $\mathbb{P} : t_1 \xrightarrow{a} t_1' : \mathbb{P}'$ for some t_1' such that $\vdash t_1' \; R \; t_2' : \mathbb{P}'$.

As in Section 1.2.1 bisimilarity, written $\sim$, is the largest bisimulation, and if only the second item above is satisfied, then R is called a simulation.

3.5.1 Simulation Equivalence

The path semantics does not capture enough of the branching behaviour of processes to characterise bisimilarity (for that, the presheaf semantics is needed, see Part II). As an example, the processes $!\varnothing + !!\varnothing$ and $!!\varnothing$ have the same denotation, but are clearly not bisimilar. However, using Hennessy-Milner logic we can link path equivalence to simulation. In detail, we consider the fragment of Hennessy-Milner logic given by possibility and finite conjunctions; as noted it is characteristic for simulation equivalence in the case of image-finite processes [31]. With a ranging over actions, formulae are given by the grammar

$$\phi ::= \langle a \rangle \phi \mid \textstyle\bigwedge_{i \leq n} \phi_i \ . \tag{3.76}$$

The empty conjunction is written $\top$ and we sometimes write $\phi_1 \wedge \cdots \wedge \phi_n$ for a conjunction $\bigwedge_{i \leq n} \phi_i$. We type formulae using judgements $\phi : \mathbb{P}$, the idea being that only processes of type $\mathbb{P}$ should be described by formulae of type $\mathbb{P}$.

$$\frac{\mathbb{P} : a : \mathbb{P}' \quad \phi : \mathbb{P}'}{\langle a \rangle \phi : \mathbb{P}} \qquad \frac{\phi_i : \mathbb{P} \quad \text{all } i \leq n}{\bigwedge_{i \leq n} \phi_i : \mathbb{P}} \tag{3.77}$$

A typed notion of satisfaction, written $t \vDash \phi : \mathbb{P}$, is defined by

$$\begin{aligned} t \vDash \langle a \rangle \phi : \mathbb{P} &\iff_{\text{def}} \exists t'.\ \mathbb{P} : t \xrightarrow{a} t' : \mathbb{P}' \text{ and } t' \vDash \phi : \mathbb{P}' \\ t \vDash \textstyle\bigwedge_{i \leq n} \phi_i : \mathbb{P} &\iff_{\text{def}} t \vDash \phi_i : \mathbb{P} \text{ for each } i \leq n \ . \end{aligned} \tag{3.78}$$

Note that $\top : \mathbb{P}$ and $t \vDash \top : \mathbb{P}$ for all $\vdash t : \mathbb{P}$.

Closed terms t_1, t_2 of the same type $\mathbb{P}$ are related by *logical preorder*, written $t_1 \precsim_{\mathrm{L}} t_2$, iff for all formulae $\phi : \mathbb{P}$ we have $t_1 \vDash \phi : \mathbb{P} \implies t_2 \vDash \phi : \mathbb{P}$. If both $t_1 \precsim_{\mathrm{L}} t_2$ and $t_2 \precsim_{\mathrm{L}} t_1$, we say that t_1 and t_2 are *logically equivalent*. Using adequacy and by adapting the proof of full abstraction, we can show that logical equivalence coincides with contextual equivalence as do the associated preorders:

Theorem 3.21 For closed terms t_1 and t_2 of the same type $\mathbb{P}$,

$$t_1 \precsim t_2 \iff t_1 \precsim_{\mathrm{L}} t_2 \ . \tag{3.79}$$

Proof. To each formula $\phi : \mathbb{P}$ we can construct a $(\mathbb{O}, \mathbb{P})$-program context C_ϕ with the property that

$$!\mathbb{O} : C_\phi(t) \xrightarrow{!} \iff t \vDash \phi : \mathbb{P} \ . \tag{3.80}$$

Define

$$\begin{aligned} C_{\langle u \mapsto a\rangle\phi} &\equiv_{\mathrm{def}} C_{\langle a\rangle\phi}(-\ u) \ , & C_{\langle !\rangle\phi} &\equiv_{\mathrm{def}} [- > !x \Rightarrow C_\phi(x)] \ , \\ C_{\langle \beta a\rangle\phi} &\equiv_{\mathrm{def}} C_{\langle a\rangle\phi}(\pi_\beta -) \ , & C_{\langle abs\, a\rangle\phi} &\equiv_{\mathrm{def}} C_{\langle a\rangle\phi}(rep\ -) \ , \\ C_{\bigwedge_{i\leq n}\phi_i} &\equiv_{\mathrm{def}} [C_{\phi_1} > !x_1 \Rightarrow \cdots \Rightarrow [C_{\phi_n} > !x_n \Rightarrow !\varnothing]\cdots] \ . \end{aligned} \tag{3.81}$$

It follows by (3.80) that $t_1 \precsim_{\mathrm{L}} t_2$ iff for all formulae $\phi : \mathbb{P}$ we have that $C_\phi(t_1) \xrightarrow{!}$ implies $C_\phi(t_2) \xrightarrow{!}$. The direction "$\Rightarrow$" then follows by adequacy.

For the converse, we observe that the program contexts C_p used in the full-abstraction proof are all subsumed by the contexts C_ϕ. In detail, using the terms t'_P realising finite sets of paths, we can define actions $\mathbb{P} : a_p : \mathbb{P}'$ and formulae $\phi_p : \mathbb{P}$ by induction on paths $p : \mathbb{P}$ such that $C_p \equiv C_{\langle a_p\rangle\phi_p}$:

$$\begin{aligned} a_{P\mapsto q} &\equiv_{\mathrm{def}} t'_P \mapsto a_q & \phi_{P\mapsto q} &\equiv_{\mathrm{def}} \phi_q \\ a_{\beta p} &\equiv_{\mathrm{def}} \beta a_p & \phi_{\beta p} &\equiv_{\mathrm{def}} \phi_p \\ a_P &\equiv_{\mathrm{def}} ! & \phi_P &\equiv_{\mathrm{def}} \textstyle\bigwedge_{p\in P} \langle a_p\rangle\phi_p \\ a_{abs\, p} &\equiv_{\mathrm{def}} abs\ a_p & \phi_{abs\, p} &\equiv_{\mathrm{def}} \phi_p \end{aligned} \tag{3.82}$$

With $p : \mathbb{P}$ and $\vdash t : \mathbb{P}$ we obtain $p \in [\![t]\!]$ iff $[\![C_{\langle a_p\rangle\phi_p}(t)]\!] \neq \varnothing$ as in the proof of full abstraction, and so by adequacy and (3.80), we have $p \in [\![t]\!]$ iff $t \vDash \langle a_p\rangle\phi_p : \mathbb{P}$. It follows that $t_1 \precsim_{\mathrm{L}} t_2$ implies $[\![t_1]\!] \subseteq [\![t_2]\!]$, and so $t_1 \precsim t_2$. $\square$

We note that the proof above establishes a link between paths and actions:

$$p \in [\![t]\!] \iff \mathbb{P} : t \xrightarrow{a_p} t' : \mathbb{P}' \text{ and } t' \vDash \phi_p : \mathbb{P}' \ . \tag{3.83}$$

Example 3.22 The context $C_{\langle \mathsf{a}!\rangle(\langle \mathsf{b}!\rangle\top \wedge \langle \mathsf{c}!\rangle\top)}$ is given by

$$[\pi_{\mathsf{a}} - > !x \Rightarrow [\pi_{\mathsf{b}} x > !x' \Rightarrow [\pi_{\mathsf{c}} x > !x'' \Rightarrow !\varnothing]]] \ . \tag{3.84}$$

It was used in Example 3.1 to distinguish between the CCS processes $\mathsf{a}.\mathsf{b}.\varnothing + \mathsf{a}.\mathsf{c}.\varnothing$ and $\mathsf{a}.(\mathsf{b}.\varnothing + \mathsf{c}.\varnothing)$. $\square$

3.5.2 Bisimilarity

We start by listing some unsurprising results about bisimilarity, mirroring those of Section 3.2. Types are left out for brevity.

Proposition 3.23 For closed, well-formed terms we have

$$\begin{aligned}
rec\,x.t &\sim t[rec\,x.t/x] \\[1ex]
(\lambda x.t)\ u &\sim t[u/x] \\
\lambda x.(t\ x) &\sim t \\
\lambda x.(\Sigma_{i\in I} t_i) &\sim \Sigma_{i\in I}(\lambda x.t_i) \\
(\Sigma_{i\in I} t_i)\ u &\sim \Sigma_{i\in I}(t_i\ u) \\[1ex]
\pi_\beta(\beta t) &\sim t \\
\pi_\alpha(\beta t) &\sim \varnothing \quad \text{if } \alpha \neq \beta \\
\Sigma_{\alpha\in A}\alpha(\pi_\alpha t) &\sim t \quad \text{where } \vdash t : \Sigma_{\alpha\in A}\mathbb{P}_\alpha \\
\beta(\Sigma_{i\in I} t_i) &\sim \Sigma_{i\in I}(\beta t_i) \\
\pi_\beta(\Sigma_{i\in I} t_i) &\sim \Sigma_{i\in I}(\pi_\beta t_i) \\[1ex]
[!u > !x \Rightarrow t] &\sim t[u/x] \\
[\Sigma_{i\in I} u_i > !x \Rightarrow t] &\sim \Sigma_{i\in I}[u_i > !x \Rightarrow t] \\[1ex]
rep(abs\,t) &\sim t \\
abs(rep\,t) &\sim t \\
abs(\Sigma_{i\in I} t_i) &\sim \Sigma_{i\in I}(abs\,t_i) \\
rep(\Sigma_{i\in I} t_i) &\sim \Sigma_{i\in I}(rep\,t_i)
\end{aligned} \tag{3.85}$$

Proof. In each postulated case $t_1 \sim t_2$, the identity relation extended by the pair (t_1, t_2) is a bisimulation. $\square$

We'll now show that bisimilarity is a congruence for HOPLA. Some notation concerning type-respecting relations is needed. Suppose Γ is an environment list $x_1 : \mathbb{P}_1, \ldots, x_k : \mathbb{P}_k$. A Γ-*closure* is a substitution $[\vec{u}/\vec{x}]$ such that for $1 \leq j \leq k$, $\vdash u_j : \mathbb{P}_j$. If R relates only closed terms, we write R^{o} for its open extension, relating $\Gamma \vdash t_1 : \mathbb{P}$ and $\Gamma \vdash t_2 : \mathbb{P}$ if for all Γ-closures $[\vec{u}/\vec{x}]$ we have $\vdash t_1[\vec{u}/\vec{x}]\ R\ t_2[\vec{u}/\vec{x}] : \mathbb{P}$. Further, we'll write R_{c} for the restriction of a type-respecting relation R to closed terms. For a type-respecting relation R we write R also for the induced relation on actions, given inductively by

$$\frac{\vdash u_1\ R\ u_2 : \mathbb{P} \quad \mathbb{Q} : a_1\ R\ a_2 : \mathbb{P}'}{\mathbb{P} \to \mathbb{Q} : u_1 \mapsto a_1\ R\ u_2 \mapsto a_2 : \mathbb{P}'} \quad \frac{\mathbb{P}_\beta : a_1\ R\ a_2 : \mathbb{P}' \quad \beta \in A}{\Sigma_{\alpha\in A}\mathbb{P}_\alpha : \beta a_1\ R\ \beta a_2 : \mathbb{P}'}$$
$$\frac{}{!\mathbb{P} : !\ R\ ! : \mathbb{P}} \quad \frac{\mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}] : a_1\ R\ a_2 : \mathbb{P}'}{\mu_j\vec{T}.\vec{\mathbb{T}} : abs\,a_1\ R\ abs\,a_2 : \mathbb{P}'} \tag{3.86}$$

Theorem 3.24 Bisimilarity is a congruence.

Proof. We employ Howe's method [38] as adapted to a typed setting by Gordon [28], except that because our typing environments $\Gamma, \Lambda, \ldots$ are lists rather than finite mappings, we need to add structural rules to the definition of the "precongruence candidate", shown in Figure 3.1. Following Howe we now have: (i) $\hat{\sim}$ is reflexive; (ii) $\hat{\sim}$ is operator respecting; (iii) $\sim^{\mathrm{o}} \subseteq \hat{\sim}$; (iv) if $\Gamma \vdash t \mathrel{\hat{\sim}} t' : \mathbb{P}$ and $\Gamma \vdash t' \sim^{\mathrm{o}} w : \mathbb{P}$, then $\Gamma \vdash t \mathrel{\hat{\sim}} w : \mathbb{P}$; (v) if $\Gamma, x : \mathbb{P} \vdash t \mathrel{\hat{\sim}} t' : \mathbb{Q}$ and $\Lambda \vdash u \mathrel{\hat{\sim}} u' : \mathbb{P}$ with Γ and Λ disjoint, then $\Gamma, \Lambda \vdash t[u/x] \mathrel{\hat{\sim}} t'[u'/x] : \mathbb{Q}$; (vi) since $\sim$ is an equivalence relation, the transitive closure $\hat{\sim}^{+}$ of $\hat{\sim}$ is symmetric, and therefore, so is $\hat{\sim}_{\mathrm{c}}^{+}$.

Now we just need to show that $\hat{\sim}_{\mathrm{c}}$ is a simulation, because then $\hat{\sim}_{\mathrm{c}}^{+}$ is a bisimulation by (vi), and so $\hat{\sim}_{\mathrm{c}}^{+} \subseteq \sim$. In particular, $\hat{\sim}_{\mathrm{c}} \subseteq \sim$. By (i) and (v), it follows that $\hat{\sim} \subseteq \sim^{\mathrm{o}}$, and so by (iii), $\hat{\sim} = \sim^{\mathrm{o}}$. Hence, $\sim$ is a congruence because it is an equivalence relation and by (ii) it is operator respecting.

We prove below that $\hat{\sim}_{\mathrm{c}}$ is a simulation by induction on the derivations of the operational semantics. In fact, we need an induction hypothesis slightly stronger than one might expect:

> if $\vdash t_1 \mathrel{\hat{\sim}_{\mathrm{c}}} t_2 : \mathbb{P}$ and $\mathbb{P} : t_1 \xrightarrow{a_1} t_1' : \mathbb{P}'$, then for all actions a_2 with $\mathbb{P} : a_1 \mathrel{\hat{\sim}_{\mathrm{c}}} a_2 : \mathbb{P}'$ we have $\mathbb{P} : t_2 \xrightarrow{a_2} t_2' : \mathbb{P}'$ for some t_2' such that $\vdash t_1' \mathrel{\hat{\sim}_{\mathrm{c}}} t_2' : \mathbb{P}'$.

By (i), $\mathbb{P} : a \mathrel{\hat{\sim}_{\mathrm{c}}} a : \mathbb{P}'$ for all actions $\mathbb{P} : a : \mathbb{P}'$, and so $\hat{\sim}_{\mathrm{c}}$ is a simulation if the above holds. The need for the stronger hypothesis will become clear in the case of application.

Each case is proved in the same way: Assuming $\Gamma \vdash t_1 : \mathbb{P}$ and $\vdash C(t_1) \mathrel{\hat{\sim}_{\mathrm{c}}} t_2 : \mathbb{P}$ for some term-constructor C, possibly involving binding, we obtain from the definition of $\hat{\sim}$ the existence of a term s with $\Gamma \vdash t_1 \mathrel{\hat{\sim}} s : \mathbb{P}$ and $\vdash C(s) \sim t_2 : \mathbb{P}$. Under the assumption $\mathbb{P} : C(t_1) \xrightarrow{a_1} t_1' : \mathbb{P}'$ and with a_2 any action such that $\mathbb{P} : a_1 \mathrel{\hat{\sim}_{\mathrm{c}}} a_2 : \mathbb{P}'$, we show that there is a transition $\mathbb{P} : C(s) \xrightarrow{a_2} s' : \mathbb{P}'$ with $\vdash t_1' \mathrel{\hat{\sim}_{\mathrm{c}}} s' : \mathbb{P}'$. Having showed this, we in *all* cases conclude as follows: since $\vdash C(s) \sim t_2 : \mathbb{P}$ there is a transition $\mathbb{P} : t_2 \xrightarrow{a_2} t_2' : \mathbb{P}'$ with $\vdash s' \sim t_2' : \mathbb{P}$. Hence $\vdash t_1' \mathrel{\hat{\sim}_{\mathrm{c}}} t_2' : \mathbb{P}'$ follows from $\vdash t_1' \mathrel{\hat{\sim}_{\mathrm{c}}} s' : \mathbb{P}'$ by (iv). To avoid repetition, this latter part will be left out below.

Recursive definition. Suppose $\vdash \mathit{rec}\, x.t_1 \mathrel{\hat{\sim}_{\mathrm{c}}} t_2 : \mathbb{P}$ and that $\mathbb{P} : \mathit{rec}\, x.t \xrightarrow{a_1} t_1' : \mathbb{P}'$ is derived from $\mathbb{P} : t_1[\mathit{rec}\, x.t_1/x] \xrightarrow{a_1} t_1' : \mathbb{P}'$. Let a_2 be any action with $\mathbb{P} : a_1 \mathrel{\hat{\sim}_{\mathrm{c}}} a_2 : \mathbb{P}'$. Since $\vdash \mathit{rec}\, x.t_1 \mathrel{\hat{\sim}_{\mathrm{c}}} t_2 : \mathbb{P}$ there exists a term $x : \mathbb{P} \vdash s : \mathbb{P}$ such that $x : \mathbb{P} \vdash t_1 \mathrel{\hat{\sim}} s : \mathbb{P}$ and $\vdash \mathit{rec}\, x.s \sim t_2 : \mathbb{P}$. By (ii) we have $\vdash \mathit{rec}\, x.t_1 \mathrel{\hat{\sim}_{\mathrm{c}}} \mathit{rec}\, x.s : \mathbb{P}$ and using (v) we therefore get $\vdash t_1[\mathit{rec}\, x.t_1/x] \mathrel{\hat{\sim}_{\mathrm{c}}} s[\mathit{rec}\, x.s/x] : \mathbb{P}$. By the induction hypothesis we get $\mathbb{P} : s[\mathit{rec}\, x.s/x] \xrightarrow{a_2} s' : \mathbb{P}'$ with $\vdash t_1' \mathrel{\hat{\sim}_{\mathrm{c}}} s' : \mathbb{P}'$, and hence also $\mathbb{P} : \mathit{rec}\, x.s \xrightarrow{a_2} s' : \mathbb{P}'$.

Nondeterministic sum. Suppose $\vdash \Sigma_{i \in I} t_i \mathrel{\hat{\sim}_{\mathrm{c}}} t_2 : \mathbb{P}$ and that $\mathbb{P} : \Sigma_{i \in I} t_i \xrightarrow{a_1} t_1' : \mathbb{P}'$ is derived from $\mathbb{P} : t_j \xrightarrow{a_1} t_1' : \mathbb{P}'$ for some $j \in I$. Let a_2 be any action

$$\frac{\Gamma \vdash x \sim^{\mathrm{o}} t'' : \mathbb{P}}{\Gamma \vdash x \mathrel{\hat{\sim}} t'' : \mathbb{P}}$$

$$\frac{\Gamma \vdash t \mathrel{\hat{\sim}} t' : \mathbb{Q}}{\Gamma, x : \mathbb{P} \vdash t \mathrel{\hat{\sim}} t' : \mathbb{Q}}$$

$$\frac{\Gamma, y : \mathbb{Q}, x : \mathbb{P}, \Lambda \vdash t \mathrel{\hat{\sim}} t' : \mathbb{R}}{\Gamma, x : \mathbb{P}, y : \mathbb{Q}, \Lambda \vdash t \mathrel{\hat{\sim}} t' : \mathbb{R}}$$

$$\frac{\Gamma, x : \mathbb{P}, y : \mathbb{P} \vdash t \mathrel{\hat{\sim}} t' : \mathbb{Q} \quad z \text{ fresh}}{\Gamma, z : \mathbb{P} \vdash t[z/x, z/y] \mathrel{\hat{\sim}} t'[z/x, z/y] : \mathbb{Q}}$$

$$\frac{\Gamma, x : \mathbb{P} \vdash t \mathrel{\hat{\sim}} t' : \mathbb{P} \quad \Gamma \vdash \mathit{rec}\, x.t' \sim^{\mathrm{o}} t'' : \mathbb{P}}{\Gamma \vdash \mathit{rec}\, x.t \mathrel{\hat{\sim}} t'' : \mathbb{P}}$$

$$\frac{\Gamma \vdash t_j \mathrel{\hat{\sim}} t'_j : \mathbb{P} \quad \text{all } j \in I \quad \Gamma \vdash \Sigma_{i \in I} t'_i \sim^{\mathrm{o}} t'' : \mathbb{P}}{\Gamma \vdash \Sigma_{i \in I} t_i \mathrel{\hat{\sim}} t'' : \mathbb{P}}$$

$$\frac{\Gamma, x : \mathbb{P} \vdash t \mathrel{\hat{\sim}} t' : \mathbb{Q} \quad \Gamma \vdash \lambda x.t' \sim^{\mathrm{o}} t'' : \mathbb{P} \to \mathbb{Q}}{\Gamma \vdash \lambda x.t \mathrel{\hat{\sim}} t'' : \mathbb{P} \to \mathbb{Q}}$$

$$\frac{\Gamma \vdash t \mathrel{\hat{\sim}} t' : \mathbb{P} \to \mathbb{Q} \quad \Delta \vdash u \mathrel{\hat{\sim}} u' : \mathbb{P} \quad \Gamma, \Delta \vdash t'\, u' \sim^{\mathrm{o}} t'' : \mathbb{Q}}{\Gamma, \Delta \vdash t\, u \mathrel{\hat{\sim}} t'' : \mathbb{Q}}$$

$$\frac{\Gamma \vdash t \mathrel{\hat{\sim}} t' : \mathbb{P}_\beta \quad \Gamma \vdash \beta t' \sim^{\mathrm{o}} t'' : \Sigma_{\alpha \in A} \mathbb{P}_\alpha}{\Gamma \vdash \beta t \mathrel{\hat{\sim}} t'' : \Sigma_{\alpha \in A} \mathbb{P}_\alpha}$$

$$\frac{\Gamma \vdash t \mathrel{\hat{\sim}} t' : \Sigma_{\alpha \in A} \mathbb{P}_\alpha \quad \Gamma \vdash \pi_\beta t' \sim^{\mathrm{o}} t'' : \mathbb{P}_\beta}{\Gamma \vdash \pi_\beta t \mathrel{\hat{\sim}} t'' : \mathbb{P}_\beta}$$

$$\frac{\Gamma \vdash t \mathrel{\hat{\sim}} t' : \mathbb{P} \quad \Gamma \vdash !t' \sim^{\mathrm{o}} t'' : !\mathbb{P}}{\Gamma \vdash !t \mathrel{\hat{\sim}} t'' : !\mathbb{P}}$$

$$\frac{\Gamma, x : \mathbb{P} \vdash t \mathrel{\hat{\sim}} t' : \mathbb{Q} \quad \Delta \vdash u \mathrel{\hat{\sim}} u' : !\mathbb{P} \quad \Gamma, \Delta \vdash [u' > !x \Rightarrow t'] \sim^{\mathrm{o}} t'' : \mathbb{Q}}{\Gamma, \Delta \vdash [u > !x \Rightarrow t] \mathrel{\hat{\sim}} t'' : \mathbb{Q}}$$

$$\frac{\Gamma \vdash t \mathrel{\hat{\sim}} t' : \mathbb{T}_j[\mu \vec{T}.\vec{\mathbb{T}}/\vec{T}] \quad \Gamma \vdash \mathit{abs}\, t' \sim^{\mathrm{o}} t'' : \mu_j \vec{T}.\vec{\mathbb{T}}}{\Gamma \vdash \mathit{abs}\, t \mathrel{\hat{\sim}} t'' : \mu_j \vec{T}.\vec{\mathbb{T}}}$$

$$\frac{\Gamma \vdash t \mathrel{\hat{\sim}} t' : \mu_j \vec{T}.\vec{\mathbb{T}} \quad \Gamma \vdash \mathit{rep}\, t' \sim^{\mathrm{o}} t'' : \mathbb{T}_j[\mu \vec{T}.\vec{\mathbb{T}}/\vec{T}]}{\Gamma \vdash \mathit{rep}\, t \mathrel{\hat{\sim}} t'' : \mathbb{T}_j[\mu \vec{T}.\vec{\mathbb{T}}/\vec{T}]}$$

Figure 3.1: The precongruence candidate

with $\mathbb{P} : a_1 \mathrel{\hat{\sim}_{\mathrm{c}}} a_2 : \mathbb{P}'$. Since $\vdash \Sigma_{i\in I} t_i \mathrel{\hat{\sim}_{\mathrm{c}}} t_2 : \mathbb{P}$ there exist terms $\vdash s_i : \mathbb{P}$ such that $\vdash t_i \mathrel{\hat{\sim}_{\mathrm{c}}} s_i : \mathbb{P}$ for each $i \in I$, and $\vdash \Sigma_{i\in I} s_i \sim t_2 : \mathbb{P}$. By the induction hypothesis applied to s_j we get $\mathbb{P} : s_j \xrightarrow{a_2} s' : \mathbb{P}'$ with $\vdash t_1' \mathrel{\hat{\sim}_{\mathrm{c}}} s' : \mathbb{P}'$, and hence also $\mathbb{P} : \Sigma_{i\in I} s_i \xrightarrow{a_2} s' : \mathbb{P}'$.

Abstraction. Suppose $\vdash \lambda x.t_1 \mathrel{\hat{\sim}_{\mathrm{c}}} t_2 : \mathbb{P} \to \mathbb{Q}$ and that $\mathbb{P} \to \mathbb{Q} : \lambda x.t \xrightarrow{u_1 \mapsto a_1} t_1' : \mathbb{P}'$ is derived from $\mathbb{Q} : t_1[u_1/x] \xrightarrow{a_1} t_1' : \mathbb{P}'$. Let $u_2 \mapsto a_2$ be any action with $\mathbb{P} \to \mathbb{Q} : u_1 \mapsto a_1 \mathrel{\hat{\sim}_{\mathrm{c}}} u_2 \mapsto a_2 : \mathbb{P}'$. In particular, $\vdash u_1 \mathrel{\hat{\sim}_{\mathrm{c}}} u_2 : \mathbb{P}$. Since $\vdash \lambda x.t_1 \mathrel{\hat{\sim}_{\mathrm{c}}} t_2 : \mathbb{P}$ there exists a term $x : \mathbb{P} \vdash s : \mathbb{Q}$ such that $x : \mathbb{P} \vdash t_1 \mathrel{\hat{\sim}} s : \mathbb{Q}$ and $\vdash \lambda x.s \sim t_2 : \mathbb{P} \to \mathbb{Q}$. Using (v) we have $\vdash t_1[u_1/x] \mathrel{\hat{\sim}_{\mathrm{c}}} s[u_2/x] : \mathbb{Q}$. By the induction hypothesis we get $\mathbb{Q} : s[u_2/x] \xrightarrow{a_2} s' : \mathbb{P}'$ with $\vdash t_1' \mathrel{\hat{\sim}_{\mathrm{c}}} s' : \mathbb{P}'$, and hence also $\mathbb{P} \to \mathbb{Q} : \lambda x.s \xrightarrow{u_2 \mapsto a_2} s' : \mathbb{P}'$.

Application. Suppose $\vdash t_1\ u_1 \mathrel{\hat{\sim}_{\mathrm{c}}} t_2 : \mathbb{Q}$ and that $\mathbb{Q} : t_1\ u_1 \xrightarrow{a_1} t_1' : \mathbb{P}'$ is derived from $\mathbb{P} \to \mathbb{Q} : t_1 \xrightarrow{u_1 \mapsto a_1} t_1' : \mathbb{P}'$. Let a_2 be any action with $\mathbb{Q} : a_1 \mathrel{\hat{\sim}_{\mathrm{c}}} a_2 : \mathbb{P}'$. Since $\vdash t_1\ u_1 \mathrel{\hat{\sim}_{\mathrm{c}}} t_2 : \mathbb{Q}$ there exist terms $\vdash s_t : \mathbb{P} \to \mathbb{Q}$ and $\vdash s_u : \mathbb{P}$ such that $\vdash t_1 \mathrel{\hat{\sim}_{\mathrm{c}}} s_t : \mathbb{P} \to \mathbb{Q}$ and $\vdash u_1 \mathrel{\hat{\sim}_{\mathrm{c}}} s_u : \mathbb{P}$ and $\vdash s_t\ s_u \sim t_2 : \mathbb{Q}$. By the induction hypothesis we get $\mathbb{P} \to \mathbb{Q} : s_t \xrightarrow{s_u \mapsto a_2} s' : \mathbb{P}'$ with $\vdash t_1' \mathrel{\hat{\sim}_{\mathrm{c}}} s' : \mathbb{P}'$, and hence also $\mathbb{Q} : s_t\ s_u \xrightarrow{a_2} s' : \mathbb{P}'$. Notice how the stronger induction hypothesis allows us to choose the label $s_u \mapsto a_2$ rather than $u_1 \mapsto a_2$ so that we could obtain a transition from $s_t\ s_u$.

Injection and projection. Suppose $\vdash \beta t_1 \mathrel{\hat{\sim}_{\mathrm{c}}} t_2 : \Sigma_{\alpha\in A}\mathbb{P}_\alpha$ and that $\Sigma_{\alpha\in A}\mathbb{P}_\alpha : \beta t_1 \xrightarrow{\beta a_1} t_1' : \mathbb{P}'$ is derived from $\mathbb{P}_\beta : t_1 \xrightarrow{a_1} t_1' : \mathbb{P}'$. Let βa_2 be any action with $\Sigma_{\alpha\in A}\mathbb{P}_\alpha : \beta a_1 \mathrel{\hat{\sim}_{\mathrm{c}}} \beta a_2 : \mathbb{P}'$. Since $\vdash \beta t_1 \mathrel{\hat{\sim}_{\mathrm{c}}} t_2 : \Sigma_{\alpha\in A}\mathbb{P}_\alpha$ there exists a term $\vdash s : \mathbb{P}_\beta$ such that $\vdash t_1 \mathrel{\hat{\sim}_{\mathrm{c}}} s : \mathbb{P}_\beta$ and $\vdash \beta s \sim t_2 : \Sigma_{\alpha\in A}\mathbb{P}_\alpha$. By the induction hypothesis we get $\mathbb{P}_\beta : s \xrightarrow{a_2} s' : \mathbb{P}'$ with $\vdash t_1' \mathrel{\hat{\sim}_{\mathrm{c}}} s' : \mathbb{P}'$, and hence also $\Sigma_{\alpha\in A}\mathbb{P}_\alpha : \beta s \xrightarrow{\beta a_2} s' : \mathbb{P}'$.

Projection is handled symmetrically.

Prefixing. Suppose $\vdash !t_1 \mathrel{\hat{\sim}_{\mathrm{c}}} t_2 : !\mathbb{P}$ so that $!\mathbb{P} : !t_1 \xrightarrow{!} t_1 : \mathbb{P}$. Since $\vdash !t_1 \mathrel{\hat{\sim}_{\mathrm{c}}} t_2 : !\mathbb{P}$ there exists a term $\vdash s : \mathbb{P}$ such that $\vdash t_1 \mathrel{\hat{\sim}_{\mathrm{c}}} s : \mathbb{P}$ and $\vdash !s \sim t_2 : !\mathbb{P}$. We get a transition $!\mathbb{P} : !s \xrightarrow{!} s' : \mathbb{P}$ by the operational rules.

Prefix match. Suppose $\vdash [u_1 > !x \Rightarrow t_1] \mathrel{\hat{\sim}_{\mathrm{c}}} t_2 : \mathbb{Q}$ and that $\mathbb{Q} : [u_1 > !x \Rightarrow t_1] \xrightarrow{a_1} t_1' : \mathbb{P}'$ is derived from $!\mathbb{P} : u_1 \xrightarrow{!} u_1' : \mathbb{P}$ and $\mathbb{Q} : t_1[u_1'/x] \xrightarrow{a_1} t_1' : \mathbb{P}'$. Let a_2 be any action with $\mathbb{Q} : a_1 \mathrel{\hat{\sim}_{\mathrm{c}}} a_2 : \mathbb{P}'$. Since $\vdash [u_1 > !x \Rightarrow t_1] \mathrel{\hat{\sim}_{\mathrm{c}}} t_2 : \mathbb{Q}$ there exist terms $x : \mathbb{P} \vdash s_t : \mathbb{Q}$ and $\vdash s_u : !\mathbb{P}$ such that $x : \mathbb{P} \vdash t_1 \mathrel{\hat{\sim}_{\mathrm{c}}} s_t : \mathbb{Q}$ and $\vdash u_1 \mathrel{\hat{\sim}_{\mathrm{c}}} s_u : !\mathbb{P}$ and $\vdash [s_u > !x \Rightarrow s_t] \sim t_2 : \mathbb{Q}$. By the induction hypothesis applied to u_1, s_u we get $!\mathbb{P} : s_u \xrightarrow{!} s_u' : \mathbb{P}$ such that $\vdash u_1' \mathrel{\hat{\sim}_{\mathrm{c}}} s_u' : \mathbb{P}$. Using (v) we get $\vdash t_1[u_1'/x] \mathrel{\hat{\sim}_{\mathrm{c}}} s_t[s_u'/x] : \mathbb{Q}$ and applying the induction hypothesis in this case yields $\mathbb{Q} : s_t[s_u'/x] \xrightarrow{a_2} s' : \mathbb{P}'$ with $\vdash t_1' \mathrel{\hat{\sim}_{\mathrm{c}}} s' : \mathbb{P}'$, and hence also $\mathbb{Q} : [s_u > !x \Rightarrow s_t] \xrightarrow{a_2} s' : \mathbb{P}'$.

Fold and unfold. Analogous to injection and projection.

The induction is complete. $\square$

Corollary 3.25 The following are equivalent:

(i) $\vdash t_1 \sim t_2 : \mathbb{P} \to \mathbb{Q}$;

(ii) $\vdash t_1\, u_1 \sim t_2\, u_2 : \mathbb{Q}$ for all closed terms u_1, u_2 with $\vdash u_1 \sim u_2 : \mathbb{P}$.

(iii) $\vdash t_1\, u \sim t_2\, u : \mathbb{Q}$ for all closed terms $\vdash u : \mathbb{P}$;

Proof. That (i) implies (ii) is a direct consequence of Theorem 3.24 while reflexivity of bisimilarity yields the implication from (ii) to (iii). We'll show that (iii) implies (i). So suppose that $\vdash t_1\, u \sim t_2\, u : \mathbb{Q}$ for all closed terms $\vdash u : \mathbb{P}$. If $\mathbb{P} \to \mathbb{Q} : t_1 \xrightarrow{u \mapsto a} t_1' : \mathbb{P}'$ for some $\vdash u : \mathbb{P}$, then $\mathbb{Q} : t_1\, u \xrightarrow{a} t_1' : \mathbb{P}'$ by the operational rules and so by the assumption that $t_1\, u$ and $t_2\, u$ are bisimilar, we get $\mathbb{Q} : t_2\, u \xrightarrow{a} t_2' : \mathbb{P}'$ for some t_2' such that $\vdash t_1' \sim t_2' : \mathbb{P}'$. The other requirement is handled symmetrically. □

3.5.3 Applicative Bisimilarity

A type-respecting relation R on closed terms is called an *applicative bisimulation* [1] if the following holds:

1. If $\vdash t_1\, R\, t_2 : \mathbb{P} \to \mathbb{Q}$ then for all $\vdash u : \mathbb{P}$ we have $\vdash t_1\, u\, R\, t_2\, u : \mathbb{Q}$.
2. If $\vdash t_1\, R\, t_2 : \Sigma_{\alpha \in A} \mathbb{P}_\alpha$ then $\vdash \pi_\beta t_1\, R\, \pi_\beta t_2 : \mathbb{P}_\beta$ for all $\beta \in A$.
3. If $\vdash t_1\, R\, t_2 : !\mathbb{P}$, we have

   a. if $!\mathbb{P} : t_1 \xrightarrow{!} t_1' : \mathbb{P}$, then $!\mathbb{P} : t_2 \xrightarrow{!} t_2' : \mathbb{P}$ for some t_2' s.t. $\vdash t_1'\, R\, t_2' : \mathbb{P}$;

   b. if $!\mathbb{P} : t_2 \xrightarrow{!} t_2' : \mathbb{P}$, then $!\mathbb{P} : t_1 \xrightarrow{!} t_1' : \mathbb{P}$ for some t_1' s.t. $\vdash t_1'\, R\, t_2' : \mathbb{P}$.
4. If $\vdash t_1\, R\, t_2 : \mu_j \vec{T}.\vec{\mathbb{T}}$ then $\vdash \mathit{rep}\, t_1\, R\, \mathit{rep}\, t_2 : \mathbb{T}_j[\mu \vec{T}.\vec{\mathbb{T}}/\vec{T}]$.

Applicative bisimilarity, $\sim_{\mathrm{A}}$, is the largest applicative bisimulation.

Proposition 3.26 Bisimilarity and applicative bisimilarity coincide.

Proof. Since $\sim$ is a congruence, it follows that $\sim$ is an applicative bisimulation, and so $\sim \subseteq \sim_{\mathrm{A}}$. For the converse, first observe that if $\vdash t_1 \sim_{\mathrm{A}} t_2 : \mathbb{P}$, then for all actions $\mathbb{P} : a : \mathbb{P}'$, we have $\vdash a^* t_1 \sim_{\mathrm{A}} a^* t_2 : !\mathbb{P}'$. Now suppose $\vdash t_1 \sim_{\mathrm{A}} t_2 : \mathbb{P}$ and consider a transition $\mathbb{P} : t_1 \xrightarrow{a} t_1' : \mathbb{P}'$. Using Lemma 3.13 we get $!\mathbb{P}' : a^* t_1 \xrightarrow{!} t_1' : \mathbb{P}'$, and so $!\mathbb{P}' : a^* t_2 \xrightarrow{!} t_2' : \mathbb{P}'$ for some t_2' such that $\vdash t_1' \sim_{\mathrm{A}} t_2' : \mathbb{P}'$. By Lemma 3.13 again, we get $\mathbb{P} : t_2 \xrightarrow{a} t_2' : \mathbb{P}'$. In a symmetric way we can show that transitions by t_2 are matched by those of t_1, and so we conclude that $\sim_{\mathrm{A}}$ is a bisimulation. But bisimilarity is the largest such thing, and so $\sim_{\mathrm{A}} \subseteq \sim$. □

3.6 Expressive Power

We give a few examples of encodings illustrating the expressive power of HOPLA. We start by encoding the "prefix-sum" construct of [65], useful for subsequent examples.

3.6.1 Prefixed Sum

Consider a family of types $(\mathbb{P}_\alpha)_{\alpha\in A}$. Their *prefixed sum* is the type $\Sigma_{\alpha\in A}\alpha.\mathbb{P}_\alpha$ which stands for $\Sigma_{\alpha\in A}!\mathbb{P}_\alpha$. This type describes computation paths in which first an action $\beta \in A$ is performed before resuming as a computation path in $\mathbb{P}_\beta$. The prefixed sum is associated with prefix operations taking a process t of type $\mathbb{P}_\beta$ to $\beta.t \equiv_{\mathrm{def}} \beta(!t)$ of type $\Sigma_{\alpha\in A}\alpha.\mathbb{P}_\alpha$ as well as a prefix match $[u > \beta.x \Rightarrow t] \equiv_{\mathrm{def}} [\pi_\beta u > !x \Rightarrow t]$, where u has prefix-sum type, x has type $\mathbb{P}_\beta$ and t generally involves the variable x.

Proposition 3.27 Using Propositions 3.7 and 3.8, we get:

$$\begin{aligned}
[\![[\beta.u > \beta.x \Rightarrow t]]\!] &= [\![t[u/x]]\!] \\
[\![[\alpha.u > \beta.x \Rightarrow t]]\!] &= \varnothing \quad \text{if } \alpha \neq \beta \\
[\![[\Sigma_{i\in I}u_i > \beta.x \Rightarrow t]]\!] &= [\![\Sigma_{i\in I}[u_i > \beta.x \Rightarrow t]]\!]
\end{aligned} \tag{3.87}$$

We'll write $\alpha_1.\mathbb{P}_{\alpha_1} + \cdots + \alpha_k.\mathbb{P}_{\alpha_k}$ for a typical finite prefixed sum.

Note that the prefixed sum is obtained using the biproduct, so coproduct, of **Lin**. This implies that prefixed sum is a "weak coproduct" in **Cts**. Because of the universal property of the coproduct in **Lin** and using the adjunction between **Lin** and **Cts**, there is a chain of isomorphisms

$$\mathbf{Lin}(\Sigma_{\alpha\in A}!\mathbb{P}_\alpha, \mathbb{Q}) \cong \Pi_{\alpha\in A}\mathbf{Lin}(!\mathbb{P}_\alpha, \mathbb{Q}) \cong \Pi_{\alpha\in A}\mathbf{Cts}(\mathbb{P}_\alpha, \mathbb{Q}) \tag{3.88}$$

—natural in $\mathbb{Q}$. Hence, reading the chain backwards, a tuple $\langle f_\alpha\rangle_{\alpha\in A}$ of maps from the components of the sum to $\mathbb{Q}$ in **Cts** correspond to a unique *linear* map $f : \Sigma_{\alpha\in A}!\mathbb{P}_\alpha \to \mathbb{Q}$ from the prefixed sum in **Cts**. Thus, the prefixed sum is a coproduct in **Cts** but for the fact that the required mediating morphism is unique only within the subcategory of linear maps.

3.6.2 CCS

As in CCS [57], let N be a set of names, $L =_{\mathrm{def}} N \cup \bar{N}$ the set of labels, and $A =_{\mathrm{def}} L \cup \{\tau\}$ the set of actions. The type of CCS processes can then be specified as the solution to the equation $\mathbb{P} = \Sigma_{\alpha\in A}\alpha.\mathbb{P}$ using the prefixed sum. The terms of CCS are translated into HOPLA by the function $\mathcal{H}[\![-]\!]$, defined by structural induction below.

$$\begin{aligned} \mathcal{H}[\![x]\!] &\equiv_{\text{def}} x \\ \mathcal{H}[\![rec\,x.t]\!] &\equiv_{\text{def}} rec\,x.\mathcal{H}[\![t]\!] \\ \mathcal{H}[\![\Sigma_{i\in I}t_i]\!] &\equiv_{\text{def}} \Sigma_{i\in I}\mathcal{H}[\![t_i]\!] \\ \mathcal{H}[\![\alpha.t]\!] &\equiv_{\text{def}} \alpha.\mathcal{H}[\![t]\!] \end{aligned} \qquad \begin{aligned} \mathcal{H}[\![t|u]\!] &\equiv_{\text{def}} Par\ \mathcal{H}[\![t]\!]\ \mathcal{H}[\![u]\!] \\ \mathcal{H}[\![t\setminus S]\!] &\equiv_{\text{def}} Res_S\ \mathcal{H}[\![t]\!] \\ \mathcal{H}[\![t[r]]\!] &\equiv_{\text{def}} Rel_r\ \mathcal{H}[\![t]\!] \end{aligned} \tag{3.89}$$

Here, the operations of parallel composition $Par : \mathbb{P} \to (\mathbb{P} \to \mathbb{P})$ (curried for convenience), restriction $Res_S : \mathbb{P} \to \mathbb{P}$ to names not in $S \subseteq N$, and relabelling $Rel_r : \mathbb{P} \to \mathbb{P}$ using mapping $r : N \to N$ are abbreviations for the following recursively defined processes:

$$\begin{aligned} Par &\equiv_{\text{def}} rec\,f.\lambda x.\lambda y.\Sigma_\alpha[x > \alpha.x' \Rightarrow \alpha.(f\ x'\ y)] + \\ &\qquad\qquad\qquad \Sigma_\alpha[y > \alpha.y' \Rightarrow \alpha.(f\ x\ y')] + \\ &\qquad\qquad\qquad \Sigma_l[x > l.x' \Rightarrow [y > \bar{l}.y' \Rightarrow \tau.(f\ x'\ y')]] \\ Res_S &\equiv_{\text{def}} rec\,f.\lambda x.\Sigma_{\alpha\notin(S\cup\bar{S})}[x > \alpha.x' \Rightarrow \alpha.(f\ x')] \\ Rel_r &\equiv_{\text{def}} rec\,f.\lambda x.[x > \tau.x' \Rightarrow \tau.(f\ x')] + \\ &\qquad\qquad\qquad \Sigma_n[x > n.x' \Rightarrow rn.(f\ x')] + \\ &\qquad\qquad\qquad \Sigma_n[x > \bar{n}.x' \Rightarrow \bar{rn}.(f\ x')] \end{aligned} \tag{3.90}$$

The operational semantics for CCS induced by the translation agrees with that given by Milner:

Proposition 3.28 If $t \xrightarrow{\alpha} t'$ is derivable in CCS then $\mathcal{H}[\![t]\!] \xrightarrow{\alpha!} \mathcal{H}[\![t']\!]$ in HOPLA. Conversely, if $\mathcal{H}[\![t]\!] \xrightarrow{a} u$ in HOPLA, then $a \equiv \alpha!$ and $u \equiv \mathcal{H}[\![t']\!]$ for some α, t' such that $t \xrightarrow{\alpha} t'$ according to CCS.

It follows that the translations of two CCS terms are bisimilar in HOPLA iff they are strongly bisimilar in CCS.

We can recover Milner's expansion law [56] directly from the properties of the prefixed sum. Any process of type $\mathbb{P}$ will—up to path equivalence or bisimilarity—be a sum of prefixed terms. Write $t|u$ for the application $Par\ t\ u$, where t and u are terms of type $\mathbb{P}$. Suppose

$$[\![t]\!] = \Sigma_\alpha\Sigma_{i\in I(\alpha)}\alpha.[\![t_i]\!] \quad \text{and} \quad [\![u]\!] = \Sigma_\alpha\Sigma_{j\in J(\alpha)}\alpha.[\![u_j]\!]\ . \tag{3.91}$$

Using Propositions 3.5 and 3.6, then 3.27, the path set $[\![t|u]\!]$ equals the denotation of the expansion

$$\Sigma_\alpha\Sigma_{i\in I(\alpha)}\alpha.(t_i|u) + \Sigma_\alpha\Sigma_{j\in J(\alpha)}\alpha.(t|u_j) + \Sigma_l\Sigma_{i\in I(l),j\in J(\bar{l})}\tau.(t_i|u_j)\ . \tag{3.92}$$

Using Theorem 3.24 and Proposition 3.23, this can be rephrased with bisimilarity instead of path equivalence.

3.6.3 Higher-Order CCS

The language considered by Hennessy [32] is like CCS but where processes are passed at channels with names $n \in N$; the language can be seen as an extension of Thomsen's CHOCS [88]. For a translation into HOPLA, we follow Hennessy in defining types that satisfy the equations[3]

$$\mathbb{P} = \tau.\mathbb{P} + \Sigma_{n\in N}\bar{n}.\mathbb{C} + \Sigma_{n\in N}n.\mathbb{F} \qquad \mathbb{C} = \mathbb{P} \,\&\, \mathbb{P} \qquad \mathbb{F} = \mathbb{P} \to \mathbb{P}\ . \tag{3.93}$$

We are chiefly interested in the parallel composition of processes, $Par_{\mathbb{P},\mathbb{P}}$ of type $\mathbb{P} \,\&\, \mathbb{P} \to \mathbb{P}$. But parallel composition is really a family of mutually dependent operations also including components such as $Par_{\mathbb{F},\mathbb{C}}$ of type $\mathbb{F} \,\&\, \mathbb{C} \to \mathbb{P}$ to say how abstractions compose in parallel with concretions etc. All these components can be tupled together in a product and parallel composition defined as a simultaneous recursive definition. Writing $(-|-)$ for all the components of the solution, the denotation of a parallel composition $t|u$ of processes equals the denotation of the expansion

$$\begin{aligned}
&\Sigma_\alpha[t > \alpha.x \Rightarrow \alpha.(x|u)]\ + \\
&\Sigma_\alpha[u > \alpha.y \Rightarrow \alpha.(t|y)]\ + \\
&\Sigma_n[t > n.f \Rightarrow [u > \bar{n}.c \Rightarrow \tau.((f\ \ \mathit{fst}\, c)|\ \mathit{snd}\, c)]]\ + \\
&\Sigma_n[t > \bar{n}.c \Rightarrow [u > n.f \Rightarrow \tau.(\mathit{snd}\, c|(f\ \ \mathit{fst}\, c))]]\ .
\end{aligned} \tag{3.94}$$

In the summations, $n \in N$ and α ranges over labels $n, \bar{n}, \tau$.

The bisimulation induced on higher-order CCS terms is perhaps the one to be expected; a corresponding bisimulation relation is defined like an applicative bisimulation but restricted to the types of processes $\mathbb{P}$, concretions $\mathbb{C}$, and abstractions $\mathbb{F}$.

3.6.4 Mobile Ambients with Public Names

We can translate the Ambient Calculus with public names [14] into HOPLA using Winskel's presheaf semantics of mobile ambients [99]. The translation follows similar lines to the process-passing language above. Assume a fixed set of ambient names $n \in N$. Following [15], the syntax of ambients is extended beyond processes (P) to include concretions (C) and abstractions (F):

$$\begin{aligned}
t, u &::= x \mid \mathit{repl}\, t \mid \varnothing \mid t|u \mid n[t] \mid \tau.t \mid \mathit{in}\ n.t \mid \mathit{out}\ n.t \mid \mathit{mvout}\ n.c \mid \\
&\qquad \overline{\mathit{open}\ n}.t \mid \mathit{open}\ n.t \mid \overline{\mathit{mvin}\ n}.c \mid \mathit{mvin}\ n.f \\
c &::= (t, u) \\
f &::= \lambda x.t\ .
\end{aligned} \tag{3.95}$$

The syntax departs a little from that of [15]. We adopt a slightly different notation for concretions, (t, u) instead of $\langle t\rangle u$, and abstractions, $\lambda x.t$ instead

[3]See Page 42 for how to encode the binary product $\mathbb{P} \,\&\, \mathbb{P}$.

of $(x)t$, to make their translation into HOPLA clear. The constructor *repl* means replication. Following a usual convention, any subterm of this form *repl* t is closed. As it is intended to behave as $t|\,\mathit{repl}\,t$ it can be translated into HOPLA using recursion once parallel composition $(-|-)$ is defined.

Types for ambients are given recursively by (n ranges over N):

$$\begin{aligned}
\mathbb{P} &= \tau.\mathbb{P} + \Sigma_n \mathit{in}\ n.\mathbb{P} + \Sigma_n \mathit{out}\ n.\mathbb{P} + \Sigma_n \mathit{mvout}\ n.\mathbb{C} + \\
&\quad \Sigma_n \overline{\mathit{open}\ n}.\mathbb{P} + \Sigma_n \mathit{open}\ n.\mathbb{P} + \Sigma_n \overline{\mathit{mvin}\ n}.\mathbb{C} + \Sigma_n \mathit{mvin}\ n.\mathbb{F} \\
\mathbb{C} &= \mathbb{P} \,\&\, \mathbb{P} \\
\mathbb{F} &= \mathbb{P} \to \mathbb{P}\ .
\end{aligned} \tag{3.96}$$

As in the previous sections, parallel composition is a family of operations, one of which is a binary operation between processes, $Par_{\mathbb{P},\mathbb{P}} : \mathbb{P} \,\&\, \mathbb{P} \to \mathbb{P}$. The family is defined in a simultaneous recursive definition below. Again, we write $(-|-)$ for all components of the solution.

$$\begin{aligned}
t|u = {} & \Sigma_\alpha[t > \alpha.x \Rightarrow \alpha.(x|u)] + \Sigma_\alpha[u > \alpha.y \Rightarrow \alpha.(t|y)] + \\
& \Sigma_n[t > \overline{\mathit{open}\ n}.x \Rightarrow [u > \mathit{open}\ n.y \Rightarrow \tau.(x|y)]] + \\
& \Sigma_n[t > \mathit{open}\ n.x \Rightarrow [u > \overline{\mathit{open}\ n}.y \Rightarrow \tau.(x|y)]] + \\
& \Sigma_n[t > \overline{\mathit{mvin}\ n}.c \Rightarrow [u > \mathit{mvin}\ n.f \Rightarrow \tau.(\mathit{snd}\,c|(f\ \ \mathit{fst}\,c))]] + \\
& \Sigma_n[t > \mathit{mvin}\ n.f \Rightarrow [u > \overline{\mathit{mvin}\ n}.c \Rightarrow \tau.((f\ \ \mathit{fst}\,c)|\,\mathit{snd}\,c)]]\ . \\
f|u = {} & \lambda x.((f\ x)|u) \\
c|u = {} & (\mathit{fst}\,c, (\mathit{snd}\,c|u)) \\
f|c = {} & (f\ \ \mathit{fst}\,c)|\,\mathit{snd}\,c
\end{aligned} \tag{3.97}$$

The remaining cases are given symmetrically. We obtain the obvious expansion law for parallel composition in the same way as for CCS.

Also ambient creation can be defined recursively in HOPLA as a an operation $m[-] : \mathbb{P} \to \mathbb{P}$ where $m \in N$:

$$\begin{aligned}
m[t] = {} & [t > \tau.x \Rightarrow \tau.m[x]] + \\
& \Sigma_n[t > \mathit{in}\ n.x \Rightarrow \overline{\mathit{mvin}\ n}.(m[x], \varnothing)] + \\
& \Sigma_n[t > \mathit{out}\ n.x \Rightarrow \mathit{mvout}\ n.(m[x], \varnothing)] + \\
& [t > \mathit{mvout}\ m.c \Rightarrow \tau.(\mathit{fst}\,c|m[\mathit{snd}\,c])] + \\
& \mathit{open}\ m.t + \\
& \mathit{mvin}\ m.\lambda y.m[t|y]\ .
\end{aligned} \tag{3.98}$$

The denotations of ambients are determined by their capabilities: an ambient $m[t]$ can perform the internal (τ) actions of t, enter a parallel ambient ($\overline{\mathit{mvin}\ n}$) if called upon to do so by an *in* n-action of t, exit an ambient n (*mvout* n) if t so requests through an *out* n-action, be exited if t so requests through an *mvout* m-action, be opened (*open* m), or be entered by an ambient (*mvin* m); initial actions of other forms are restricted away. Ambient creation is at least as complicated as parallel composition. This should not be surprising given that ambient creation corresponds intuitively to putting a process behind (so in parallel with) a wall or membrane which if unopened mediates in the communications the process can do, converting some actions to others and restricting some others away. The tree-containment structure of ambients is captured in the chain of *open* m's that they can perform.

We obtain an expansion theorem for ambient creation. For a process t with $[\![t]\!] = \Sigma_\alpha \Sigma_{i \in I(\alpha)} \alpha.[\![t_i]\!]$, where α ranges over atomic actions of ambients, we have that $[\![m[t]]\!]$ equals the denotation of

$$\begin{array}{l}\Sigma_{i \in I(\tau)} \tau.m[t_i] + \\ \Sigma_n \Sigma_{i \in I(in\ n)} \overline{mvin\ n}.(m[t_i], \varnothing) + \\ \Sigma_n \Sigma_{i \in I(out\ n)} mvout\ n.(m[t_i], \varnothing) + \\ \Sigma_{i \in I(mvout\ m)} \tau.(fst\, t_i | m[snd\, t_i]) + \\ open\ m.t + \\ mvin\ m.(\lambda y.m[t|y])\ .\end{array} \tag{3.99}$$

Again, this can be rephrased with bisimilarity instead of path equivalence.

3.6.5 Extensions

Work is in progress on extending HOPLA with name-generation [101]. We return to this in Chapter 8.

HOPLA has no distinguished invisible action τ, so there is the issue of how to support more abstract operational equivalences such as weak bisimulation. The paper [25] provides a mathematical framework for weak bisimilarity in the context of presheaf models and may be a good starting point.

Being based on atomic actions one at a time, HOPLA does not cope well with independence. In the next chapter, we'll discuss a language much like HOPLA, but built on the affine category **Aff** rather than **Cts**. The tensor operation of **Aff** will allow simple forms of independence to be expressed.

Chapter 4

Affine HOPLA

Affine HOPLA [98, 64, 67] is a typed process language based on the structure of **Aff**. The language adds to HOPLA a tensor operation, $\otimes$, which comes at the price of linearity constraints on variable occurrences. The tensor can be understood as a juxtaposition of independent processes; some intuition can be obtained from nondeterministic dataflow. Consider the processes

$$x : \mathbb{P}, y : \mathbb{Q} \vdash t : \mathbb{R} \quad \text{and} \quad z : \mathbb{S} \vdash u : \mathbb{P} \otimes \mathbb{Q} \ . \tag{4.1}$$

By drawing them as the dataflow networks at the top of Figure 4.1 (labelling wires with the type of values transmitted), we can understand their *tensor product*,

$$x : \mathbb{P}, y : \mathbb{Q}, z : \mathbb{S} \vdash t \otimes u : \mathbb{R} \otimes \mathbb{P} \otimes \mathbb{Q} \tag{4.2}$$

as the juxtaposition in the middle part of the figure, and the *tensor match*

$$z : \mathbb{S} \vdash [u > x \otimes y \Rightarrow t] : \mathbb{R} \tag{4.3}$$

as the composition of t and u as in the bottom part, feeding the output of u into t.

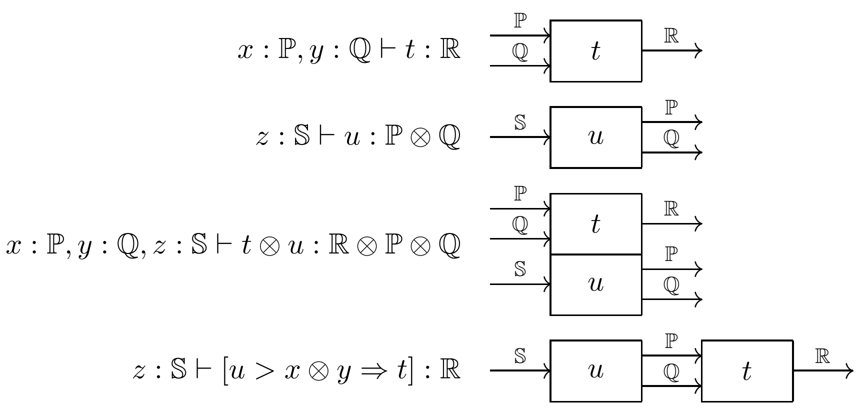

Figure 4.1: Tensor and nondeterministic dataflow

The extension with tensor comes at the price of linearity constraints on the occurrences of variables. In particular, the language will *not* allow us to write an operation that takes a process as input, copies it, and then sets those copies in parallel composition with each other. Such an operation would make use of two computation paths of its input, and would not be affine according to Section 2.3.2.

Note the difference between copying an input process, as a whole, and copying a particular interaction with it. The *Fork*-operation of nondeterministic dataflow (Figure 1.3) copies whatever values is sent to it onto two output streams. Still, it is a linear, and so affine, operation because it has no way of copying the *process* that transmits the input, and so any nondeterministic choices made by that process will be preserved in the output of *Fork*. As an example, attaching *Fork* to a process that nondeterministically chooses between outputting letter a and letter b will give rise to either a or b on *both* output lines, with no possibility of "cross output".

We'll follow the structure of the previous chapter and focus on how Affine HOPLA departs from HOPLA itself. After giving the denotational semantics and proving some expected results about it in Sections 4.1 and 4.2, we prove full abstraction in Section 4.3 by adapting the argument given for HOPLA. An operational semantics for the first-order fragment of Affine HOPLA is provided in Section 4.4 together with a proof of correspondence, so soundness and adequacy. Section 4.5 discusses the expressive power of the language and shows how to encode processes like *Fork*.

It has proved very challenging to extend the operational semantics to higher order. Since a tensor match process $[u > x \otimes y \Rightarrow t]$ is interpreted as composition, an affine t may produce output without obtaining input from u, i.e. $[u > x \otimes y \Rightarrow t]$ is affine in u and so does not distribute over nondeterministic sums like prefix match does. This means that we cannot force u to make a transition in order for the tensor match to make a transition, because even if $u \equiv \varnothing$, the match need not be inactive. In other words, a rule like that for prefix match in HOPLA's operational semantics does not work for tensor match. We seem obliged to work with rather complicated environments, built from tensor matches, in the operational semantics and it is the interaction of the environments with higher-order processes which has been problematic in giving an operational semantics to full Affine HOPLA.

Another problem caused by the environments is that they complicate reasoning about the operational semantics. For instance, we would expect it to satisfy a "diamond property" that if a process can perform an action in the left and then the right component of a tensor, then it can also do so in the opposite order. A direct proof would need to take account of changes made to the environment in transitions, which is not so obvious. We'll return to this point in Chapter 7.

Lacking a satisfactory operational semantics, we have not considered simulation for the affine language.

4.1 Denotational Semantics

Types are given by the grammar

$$\mathbb{T} ::= \mathbb{T}_1 \multimap \mathbb{T}_2 \mid \mathbb{T}_1 \otimes \mathbb{T}_2 \mid \Sigma_{\alpha\in A}\mathbb{T}_\alpha \mid \mathbb{T}_\perp \mid T \mid \mu_j \vec{T}.\vec{\mathbb{T}} \ . \tag{4.4}$$

Closed type expressions are interpreted as path orders $\mathbb{P}, \mathbb{Q}, \ldots$ as before. The exponential $!(-)$ has been replaced by the weakening comonad $(-)_\perp$, but the type $\mathbb{P}_\perp$ is still understood intuitively as the type of processes that may perform an anonymous action, which we'll still call "!", before continuing as a process of type $\mathbb{P}$. For the solution of recursive type definitions we proceed as for HOPLA, replacing finite sets of paths by sets P of size at most one, writing $\perp$ for the empty set.

$$p, q ::= P \mapsto q \mid P \otimes Q \mid \beta p \mid P \mid abs\, p \tag{4.5}$$

Here, $P \otimes Q$ stands for a pair of paths P of $\mathbb{P}_\perp$ and Q of $\mathbb{Q}_\perp$ where at least one is non-$\perp$. Formation rules are displayed below alongside rules defining the ordering. Note that all path orders interpreting types of Affine HOPLA are posets because, unlike the exponential, the comonad $(-)_\perp$ maps posets to posets.

$$\begin{array}{rl}
\dfrac{P : \mathbb{P}_\perp \quad q : \mathbb{Q}}{P \mapsto q : \mathbb{P} \multimap \mathbb{Q}} & \dfrac{P' \preceq_{\mathbb{P}} P \quad q \leq_{\mathbb{Q}} q'}{P \mapsto q \leq_{\mathbb{P}\multimap\mathbb{Q}} P' \mapsto q'} \\
\dfrac{P : \mathbb{P}_\perp \quad Q : \mathbb{Q}_\perp \quad (P, Q) \neq (\perp, \perp)}{P \otimes Q : \mathbb{P} \otimes \mathbb{Q}} & \dfrac{P \preceq_{\mathbb{P}} P' \quad Q \preceq_{\mathbb{Q}} Q'}{P \otimes Q \leq_{\mathbb{P}\otimes\mathbb{Q}} P' \otimes Q'} \\
\dfrac{p : \mathbb{P}_\beta \quad \beta \in A}{\beta p : \Sigma_{\alpha\in A}\mathbb{P}_\alpha} & \dfrac{p \leq_{\mathbb{P}_\beta} p'}{\beta p \leq_{\Sigma_{\alpha\in A}\mathbb{P}_\alpha} \beta p'} \\
\dfrac{}{\perp : \mathbb{P}_\perp} \quad \dfrac{p : \mathbb{P}}{\{p\} : \mathbb{P}_\perp} & \dfrac{P \preceq_{\mathbb{P}} P'}{P \leq_{\mathbb{P}_\perp} P'} \\
\dfrac{p : \mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}]}{abs\, p : \mu_j\vec{T}.\vec{\mathbb{T}}} & \dfrac{p \leq_{\mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}]} p'}{abs\, p \leq_{\mu_j\vec{T}.\vec{\mathbb{T}}} abs\, p'}
\end{array} \tag{4.6}$$

As we continue to use ! for prefixing terms, the only difference from the syntax of HOPLA is the addition of term constructors associated with the tensor type:

$$\begin{array}{lll}
t, u ::= x, y, z, \ldots & \text{(variables)} \\
\quad \mid rec\, x.t & \text{(recursive definition)} \\
\quad \mid \Sigma_{i\in I} t_i & \text{(nondeterministic sum)} \\
\quad \mid \lambda x.t \mid t\ u & \text{(abstraction and application)} \\
\quad \mid t \otimes u \mid [u > x \otimes y \Rightarrow t] & \text{(tensor operation and match)} \\
\quad \mid \beta t \mid \pi_\beta t & \text{(injection and projection)} \\
\quad \mid\ !t \mid [u > !x \Rightarrow t] & \text{(prefix operation and match)} \\
\quad \mid abs\, t \mid rep\, t & \text{(folding and unfolding)}
\end{array} \tag{4.7}$$

The use of a pattern match term $[u > x \otimes y \Rightarrow t]$ for tensor is similar to that in [2]; both variables x and y are binding occurrences with body t.

Let $\mathbb{P}_1, \ldots, \mathbb{P}_k, \mathbb{Q}$ be closed type expressions and $x_1, \ldots, x_k$ distinct variables. A syntactic judgement

$$x_1 : \mathbb{P}_1, \ldots, x_k : \mathbb{P}_k \vdash t : \mathbb{Q} \tag{4.8}$$

stands for a map

$$[\![x_1 : \mathbb{P}_1, \ldots, x_k : \mathbb{P}_k \vdash t : \mathbb{Q}]\!] : \mathbb{P}_1 \otimes \cdots \otimes \mathbb{P}_k \to \mathbb{Q} \tag{4.9}$$

in **Aff**. When the environment list is empty, the corresponding tensor product is the empty path order $\mathbb{O}$. The term-formation rules for the affine language are very similar to those for HOPLA, replacing & by $\otimes$ in the handling of environment lists and the type constructors $!(-)$ and $\to$ by $(-)_\perp$ and $\multimap$. We discuss the remaining differences in the following.

New rules are introduced for the tensor operation:

$$\frac{\Gamma \vdash t : \mathbb{P} \quad \Lambda \vdash u : \mathbb{Q}}{\Gamma, \Lambda \vdash t \otimes u : \mathbb{P} \otimes \mathbb{Q}} \quad \frac{\Gamma \xrightarrow{t} \mathbb{P} \quad \Lambda \xrightarrow{u} \mathbb{Q}}{\Gamma \otimes \Lambda \xrightarrow{t \otimes u} \mathbb{P} \otimes \mathbb{Q}} \tag{4.10}$$

$$\frac{\Gamma, x : \mathbb{P}, y : \mathbb{Q} \vdash t : \mathbb{R} \quad \Lambda \vdash u : \mathbb{P} \otimes \mathbb{Q}}{\Gamma, \Lambda \vdash [u > x \otimes y \Rightarrow t] : \mathbb{R}}$$

$$\frac{\Gamma \otimes \mathbb{P} \otimes \mathbb{Q} \xrightarrow{t} \mathbb{R} \quad \Lambda \xrightarrow{u} \mathbb{P} \otimes \mathbb{Q}}{\Gamma \otimes \Lambda \xrightarrow{1_\Gamma \otimes u} \Gamma \otimes \mathbb{P} \otimes \mathbb{Q} \xrightarrow{t} \mathbb{R}} \tag{4.11}$$

One important difference is the lack of contraction for the affine language. This restricts substitution of a common term into distinct variables, and so copying. The counterpart in the model is the absence of a suitable diagonal map from objects $\mathbb{P}$ to $\mathbb{P} \otimes \mathbb{P}$; for example, the map $X \mapsto X \otimes X$ from $\widehat{\mathbb{P}}$ to $\widehat{\mathbb{P} \otimes \mathbb{P}}$ is not in general a map in **Aff**.[1] Consider a term $t(x, y)$, with its free variables x and y shown explicitly, for which

$$x : \mathbb{P}, y : \mathbb{P} \vdash t(x, y) : \mathbb{Q} \ , \tag{4.12}$$

corresponding to a map $\mathbb{P} \otimes \mathbb{P} \xrightarrow{t} \mathbb{Q}$ in **Aff**. This does not generally entail that $x : \mathbb{P} \vdash t(x, x) : \mathbb{Q}$—there may not be a corresponding map in **Aff**, for example if $t(x, y) = x \otimes y$. Intuitively, if any computation for t involves both inputs, then $x : \mathbb{P} \vdash t(x, x) : \mathbb{Q}$ would use the same input twice and therefore cannot be interpreted in **Aff**. There is a syntactic condition on the

[1] To see this, assume that $\mathbb{P}$ is the prefixed sum $\alpha.\mathbb{O} + \beta.\mathbb{O}$ with paths abbreviated to α, β. Confusing paths with the corresponding primes, the nonempty join $\alpha + \beta$ is sent by $X \mapsto X \otimes X$ to $\alpha \otimes \alpha + \beta \otimes \beta + \alpha \otimes \beta + \beta \otimes \alpha$ instead of $\alpha \otimes \alpha + \beta \otimes \beta$ as would be needed to preserve nonempty joins.

occurrences of variables which ensures that in any computation, at most one of a set of variables is used.

Let v be a raw term. Say a set of variables V is *crossed* in v iff there are subterms of v of the form tensor $t \otimes u$, application $t\ u$, tensor match $[u > x \otimes y \Rightarrow t]$, or prefix match $[u > !x \Rightarrow t]$, for which v has free occurrences of variables from V appearing in both t and u.

If the set $\{x, y\}$ is *not* crossed in $t(x, y)$ above, then t uses at most one of its inputs x, y in each computation; semantically, t is interpreted as a map $\mathbb{P} \otimes \mathbb{P} \to \mathbb{Q}$ of **Aff** which behaves identically on input $X \otimes Y$ and $X \otimes \varnothing + \varnothing \otimes Y$ for all $X, Y \in \widehat{\mathbb{P}}$. In this case $x : \mathbb{P} \vdash t(x, x) : \mathbb{Q}$ holds (cf. Lemma 4.1 below) and is interpreted as the composition

$$\mathbb{P} \xrightarrow{\delta_{\mathbb{P}}} \mathbb{P} \otimes \mathbb{P} \xrightarrow{t} \mathbb{Q} \tag{4.13}$$

—where $\delta_{\mathbb{P}} : \mathbb{P} \to \mathbb{P} \otimes \mathbb{P}$ maps X to $X \otimes \varnothing + \varnothing \otimes X$. It satisfies the usual properties of a diagonal [40], except that it is not a natural transformation. We'll write $\delta^k_{\mathbb{P}} : \mathbb{P} \to \mathbb{P}^k$ for the obvious generalisation to a k-fold tensor product $\mathbb{P}^k = \mathbb{P} \otimes \cdots \otimes \mathbb{P}$. We have $\delta^0 = \varnothing_{\mathbb{P}} : \mathbb{P} \to \mathbb{O}$ and $\delta^1 = 1_{\mathbb{P}} : \mathbb{P} \to \mathbb{P}$.

We can now give the rule for recursively defined processes:

$$\frac{\Gamma, x : \mathbb{P} \vdash t : \mathbb{P} \quad \{x, y\} \text{ not crossed in } t \text{ for any } y \text{ in } \Gamma}{\Gamma \vdash \mathit{rec}\, x.t : \mathbb{P}} \qquad \frac{\Gamma \otimes \mathbb{P} \xrightarrow{t} \mathbb{P}}{\Gamma \xrightarrow{\mathit{fix}\, f} \mathbb{P}} \tag{4.14}$$

Here, *fix* f is the fixed-point in $\mathbf{Aff}(\Gamma, \mathbb{P}) \cong \widehat{\Gamma \multimap \mathbb{P}}$ of the continuous operation f mapping $g : \Gamma \to \mathbb{P}$ in **Aff** to the composition

$$\Gamma \xrightarrow{\delta_\Gamma} \Gamma \otimes \Gamma \xrightarrow{1_\Gamma \otimes g} \Gamma \otimes \mathbb{P} \xrightarrow{t} \mathbb{P}\ . \tag{4.15}$$

4.2 Useful Identities

Counterparts of the results for HOPLA of Section 3.2 can be proved for the affine language. In particular, a general substitution lemma can be formulated as follows:

Lemma 4.1 (Substitution) Suppose $\Gamma, x_1 : \mathbb{P}, \ldots, x_k : \mathbb{P} \vdash t : \mathbb{Q}$ with $\{x_1, \ldots, x_k\}$ not crossed in t. If $\Lambda \vdash u : \mathbb{P}$ with Γ and Λ disjoint, then $\Gamma, \Lambda \vdash t[u/x_1, \ldots, u/x_k] : \mathbb{Q}$ with denotation given by the composition

$$\Gamma \otimes \Lambda \xrightarrow{1_\Gamma \otimes (\delta^k_{\mathbb{P}} \circ u)} \Gamma \otimes \mathbb{P}^k \xrightarrow{t} \mathbb{Q}\ . \tag{4.16}$$

Proof. Similar to the proof for HOPLA, replacing & by $\otimes$ and Δ by δ. Note that naturality of Δ is used in precisely those cases where HOPLA allows variables to be crossed (recursion, application and prefix match). Thus, there is fortunately no need for naturality of δ. We give the cases of the rule-induction for the tensor constructs, now writing h^k for $\delta^k \circ [\![u]\!]$:

Tensor. Suppose that $\Gamma_1', \Gamma_2' \vdash t_1 \otimes t_2 : \mathbb{Q} \otimes \mathbb{R}$ is obtained from $\Gamma_1' \vdash t_1 : \mathbb{Q}$ and $\Gamma_2' \vdash t_2 : \mathbb{R}$. Since the set $\{x_1, \ldots, x_k\}$ by assumption is not crossed in $t_1 \otimes t_2$, all of these variables occur freely in t_1 or they all occur in t_2. Without loss of generality, let's assume the former. So let Γ_1' be a permutation of $\Gamma_1, x_1 : \mathbb{P}, \ldots, x_k : \mathbb{P}$, and Γ a permutation of Γ_1, Γ_2'. Let $\Lambda \vdash u : \mathbb{P}$ with Γ and Λ disjoint. We want to show that $\Gamma, \Lambda \vdash (t_1[u]) \otimes t_2 : \mathbb{Q} \otimes \mathbb{R}$. By the induction hypotheses, we have $\Gamma_1, \Lambda \vdash t_1[u] : \mathbb{Q}$. The typing rule for tensor then yields $\Gamma_1, \Lambda, \Gamma_2 \vdash (t_1[u]) \otimes t_2 : \mathbb{Q} \otimes \mathbb{R}$, and repeated use of exchange then yields $\Gamma, \Lambda \vdash (t_1[u]) \otimes t_2 : \mathbb{Q} \otimes \mathbb{R}$ as wanted.

Up to exchange we must show that $[\![t_1[u]]\!] \otimes [\![t_2]\!] = [\![t_1]\!] \otimes [\![t_2]\!] \circ (1_{\Gamma_1} \otimes h^k \otimes 1_{\Gamma_2})$. This follows immediately by the induction hypothesis.

Tensor match. Suppose that $\Gamma_1', \Gamma_2' \vdash [t_2 > x \otimes y \Rightarrow t_1] : \mathbb{S}$ is obtained from $\Gamma_1', x : \mathbb{Q}, y : \mathbb{R} \vdash t_1 : \mathbb{S}$ and $\Gamma_2' \vdash t_2 : \mathbb{Q} \otimes \mathbb{R}$. Since the set $\{x_1, \ldots, x_k\}$ by assumption is not crossed in $[t_2 > x \otimes y \Rightarrow t_1]$, all of these variables occur freely in t_1 or they all occur in t_2. We assume t_1, as the other case is simpler. So assume that Γ_1' is a permutation of $\Gamma_1, x_1 : \mathbb{P}, \ldots, x_k : \mathbb{P}$ and Γ a permutation of Γ_1, Γ_2'. Let $\Lambda \vdash u : \mathbb{P}$ with Γ and Λ disjoint. By renaming x, y if necessary, we may assume that these variables does not occur freely in u. We want to show that $\Gamma, \Lambda \vdash [t_2 > x \otimes y \Rightarrow t_1[u]] : \mathbb{S}$. By the induction hypothesis, we have $\Gamma_1, x : \mathbb{Q}, y : \mathbb{R}, \Lambda \vdash t_1[u] : \mathbb{S}$. By repeated use of exchange we get $\Gamma_1, \Lambda, x : \mathbb{Q}, y : \mathbb{R} \vdash t_1[u] : \mathbb{S}$ and so $\Gamma_1, \Lambda, \Gamma_2 \vdash [t_2 > x \otimes y \Rightarrow t_1[u]] : \mathbb{S}$ by the typing rule for tensor match. Using exchange again we get $\Gamma, \Lambda \vdash [t_2 > x \otimes y \Rightarrow t_1[u]] : \mathbb{S}$ as wanted.

Up to exchange we must show that $[\![t_1[u]]\!] \circ (1_{\Gamma_1 \otimes \Lambda} \otimes [\![t_2]\!])$ equals $[\![t_1]\!] \circ (1_{\Gamma_1 \otimes \mathbb{P}^k} \otimes [\![t_2]\!]) \circ (1_{\Gamma_1} \otimes h^k \otimes 1_{\Gamma_2})$. But this follows directly from the induction hypothesis.

Incorporating the requirement of non-crossed variables, the remaining cases are very similar to the proof for HOPLA, and we omit them. □

An easy rule-induction on the typing rules shows that the substitution lemma allows single-variable substitutions:

Lemma 4.2 If $\Gamma, x : \mathbb{P}, \Lambda \vdash t : \mathbb{Q}$, then $\{x\}$ is not crossed in t.

Proposition 4.3 Suppose $\Gamma, x : \mathbb{P} \vdash t : \mathbb{P}$. Then

$$[\![rec\, x.t]\!] = [\![t[rec\, x.t/x]]\!] \tag{4.17}$$

Proof. We proceed as in the proof of Proposition 3.5, obtaining $\Gamma', x : \mathbb{P} \vdash t' : \mathbb{P}$ and $\Gamma'', x : \mathbb{P} \vdash t'' : \mathbb{P}$ with Γ' and Γ'' disjoint by renaming variables y of Γ to y' and y''. By the substitution lemma with $k = 1$, we get $\Gamma', \Gamma'' \vdash t'[rec\, x.t''/x] : \mathbb{P}$, denoting

$$\Gamma' \otimes \Gamma'' \xrightarrow{1_{\Gamma'} \otimes rec\, x.t''} \Gamma' \otimes \mathbb{P} \xrightarrow{t'} \mathbb{P} \ . \tag{4.18}$$

Now, since the sets $\{x, y\}$ are not crossed in t, $\{y', y''\}$ are not crossed in $t'[rec\,x.t''/x]$. Hence, by repeated use of exchange and the substitution lemma with $k = 2$, we may perform substitutions $[y/y', y/y'']$ to obtain $\Gamma \vdash t[rec\,x.t/x] : \mathbb{P}$ with denotation

$$\Gamma \xrightarrow{\delta_\Gamma} \Gamma \otimes \Gamma \xrightarrow{1_\Gamma \otimes rec\,x.t} \Gamma \otimes \mathbb{P} \xrightarrow{t} \mathbb{P} \ . \tag{4.19}$$

Again, this is the same as $f(\mathit{fix}\, f) = \mathit{fix}\, f$, the denotation of $rec\,x.t$. □

The properties of abstraction and application, sums, the prefixing constructs, and folding and unfolding are the same as for HOPLA.

Proposition 4.4 (i) Suppose $\Gamma, x : \mathbb{P}, y : \mathbb{Q} \vdash t : \mathbb{R}$ and $\Lambda_1 \vdash u_1 : \mathbb{P}$ and $\Lambda_2 \vdash u_2 : \mathbb{Q}$ with Γ, Λ_1, and Λ_2 pairwise disjoint. Then $\Gamma, \Lambda_1, \Lambda_2 \vdash t[u_1/x, u_2/y] : \mathbb{R}$ and

$$[\![[u_1 \otimes u_2 > x \otimes y \Rightarrow t]]\!] = [\![t[u_1/x, u_2/y]]\!] \ . \tag{4.20}$$

(ii) Suppose $\Gamma, x : \mathbb{P}, y : \mathbb{Q} \vdash t_i : \mathbb{R}$ for all $i \in I$. Then for any suitable u we have

$$[\![[u > x \otimes y \Rightarrow \Sigma_{i \in I} t_i]]\!] = [\![\Sigma_{i \in I} [u > x \otimes y \Rightarrow t_i]]\!] \ . \tag{4.21}$$

(iii) Suppose $\Lambda \vdash u_i : \mathbb{P} \otimes \mathbb{Q}$ for all $i \in I$ and that $I \neq \varnothing$. Then for any suitable t we have

$$[\![[\Sigma_{i \in I} u_i > x \otimes y \Rightarrow t]]\!] = [\![\Sigma_{i \in I} [u_i > x \otimes y \Rightarrow t]]\!] \ . \tag{4.22}$$

(iv) Suppose $\Lambda_1 \vdash u_1 : \mathbb{P}_1 \otimes \mathbb{Q}_1$ and $\Lambda_2, x_1 : \mathbb{P}_1, y_1 : \mathbb{Q}_1 \vdash u_2 : \mathbb{P}_2 \otimes \mathbb{Q}_2$. If further $\Gamma, x_2 : \mathbb{P}_2, y_2 : \mathbb{Q}_2 \vdash t : \mathbb{R}$ where x_1 and x_2 do not occur in Γ, we have

$$\begin{aligned} &[\![[[u_1 > x_1 \otimes y_1 \Rightarrow u_2] > x_2 \otimes y_2 \Rightarrow t]]\!] \\ &\qquad = [\![[u_1 > x_1 \otimes y_1 \Rightarrow [u_2 > x_2 \otimes y_2 \Rightarrow t]]]\!] \ . \end{aligned} \tag{4.23}$$

Proof. All the properties are consequences of tensor match being interpreted as composition in **Aff**. (i) follows by exchange and two applications of the substitution lemma. (ii) and (iii) hold since composition $f \circ g$ in **Aff** is linear in f and affine in g. (iv) follows from associativity of composition. □

4.3 Full Abstraction

As we did for HOPLA, we take a program to be a closed term t of type $\mathbb{O}_\perp$, but because of linearity constraints, program contexts will now have at most one hole. Otherwise, the notion of contextual preorder is the same as in Section 3.3. Again, contextual equivalence coincides with path equivalence:

Theorem 4.5 (Full abstraction) For any terms $\Gamma \vdash t_1 : \mathbb{P}$ and $\Gamma \vdash t_2 : \mathbb{P}$,

$$[\![t_1]\!] \subseteq [\![t_2]\!] \iff t_1 \lesssim t_2 \ . \tag{4.24}$$

Proof. Path "realisers" and "consumers" are defined as in the proof of full abstraction for HOPLA, restricting the terms t'_P and C'_P to the cases where P has at most one element. Terms corresponding to paths of tensor type are defined by

$$\begin{aligned} t_{P\otimes Q} &\equiv t'_P \otimes t'_Q \\ C_{P\otimes Q} &\equiv [- > x \otimes y \Rightarrow [C'_P(x) > !x' \Rightarrow C'_Q(y)]] \end{aligned} \tag{4.25}$$

For any $p : \mathbb{P}$ and $P : \mathbb{P}$ we then have (z being a fresh variable):

$$\begin{aligned} [\![t_p]\!] &= \mathrm{y}_{\mathbb{P}} p & \quad [\![\lambda z.C_p(z)]\!] &= \mathrm{y}_{\mathbb{P}\multimap\mathbb{O}_\perp}(\{p\} \mapsto \varnothing) \\ [\![t'_P]\!] &= j_{\mathbb{P}} P & \quad [\![\lambda z.C'_P(z)]\!] &= \mathrm{y}_{\mathbb{P}\multimap\mathbb{O}_\perp}(P \mapsto \varnothing) \end{aligned} \tag{4.26}$$

We can now proceed as in the proof of Theorem 3.10. □

4.4 Operational Semantics

An operational semantics for the first-order fragment[2] of Affine HOPLA uses actions given by

$$a ::= a \otimes \perp \mid \perp \otimes a \mid \beta a \mid ! \mid abs\, a \ . \tag{4.27}$$

We type actions using typing judgements $\mathbb{P} : a : \mathbb{P}'$ as we did for HOPLA; again, the type $\mathbb{P}'$ is unique given $\mathbb{P}$ and a:

$$\begin{array}{cc} \dfrac{\mathbb{P} : a : \mathbb{P}'}{\mathbb{P}\otimes\mathbb{Q} : a \otimes \perp : \mathbb{P}'\otimes\mathbb{Q}} & \dfrac{\mathbb{Q} : a : \mathbb{Q}'}{\mathbb{P}\otimes\mathbb{Q} : \perp \otimes a : \mathbb{P}\otimes\mathbb{Q}'} \\ \dfrac{\mathbb{P}_\beta : a : \mathbb{P}' \quad \beta \in A}{\Sigma_{\alpha\in A}\mathbb{P}_\alpha : \beta a : \mathbb{P}'} \quad \dfrac{}{\mathbb{P}_\perp : ! : \mathbb{P}} & \dfrac{\mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}] : a : \mathbb{P}'}{\mu_j\vec{T}.\vec{\mathbb{T}} : abs\, a : \mathbb{P}'} \end{array} \tag{4.28}$$

Since we have left out function space actions like $u \mapsto a$, actions now correspond directly to "atomic" paths. Indeed, let atomic paths p', q' be given as the sublanguage of paths (cf. (4.5)) generated by the grammar

$$p', q' ::= p' \otimes \perp \mid \perp \otimes q' \mid \beta p' \mid \perp \mid abs\, p' \ . \tag{4.29}$$

Letting the path $\perp$ of type $\mathbb{P}_\perp$ correspond to the action !, we get an obvious bijective correspondence between atomic paths and the actions above.

[2]By this we mean the fragment obtained by leaving out function space, so abstraction and application. We call it the first-order fragment because we are able to handle tensor match which can be viewed as abstraction and immediate application of a first-order function.

Accordingly, we'll interpret an action $\mathbb{P} : a : \mathbb{P}'$ as the corresponding path so that $[\![\mathbb{P} : a : \mathbb{P}']\!] \in \mathbb{P}$. One can observe that $\mathbb{P}'$ is then isomorphic to the poset of those elements of $\mathbb{P}$ that properly extend $[\![\mathbb{P} : a : \mathbb{P}']\!] \in \mathbb{P}$, and from the associated inclusion $i_a : \mathbb{P}'_\perp \hookrightarrow \mathbb{P}$, sending $\perp$ to $[\![\mathbb{P} : a : \mathbb{P}']\!]$, we get a map $a^* : \mathbb{P} \to \mathbb{P}'_\perp$ in **Lin** sending $X \in \widehat{\mathbb{P}}$ to $i_a^{-1}X$. For the HOPLA actions βa, !, and $abs\, a$, the maps a^* are in fact just those defined by (3.59). The action $\mathbb{P} \otimes \mathbb{Q} : a \otimes \perp : \mathbb{P}' \otimes \mathbb{Q}$ and its symmetric version give rise to similar equations, so that we again have $a^*[\![t]\!] = [\![a^* t]\!]$:

$$\begin{aligned} (a \otimes \perp)^* t &\equiv_{\text{def}} [t > x \otimes y \Rightarrow [a^* x > !x' \Rightarrow !(x' \otimes y)]] \\ (a \otimes \perp)^* &= str'_{\mathbb{P}',\mathbb{Q}} \circ (a^* \otimes 1_{\mathbb{Q}}) \end{aligned} \tag{4.30}$$

Here, $str'_{\mathbb{P}',\mathbb{Q}} : \mathbb{P}'_\perp \otimes \mathbb{Q} \to (\mathbb{P}' \otimes \mathbb{Q})_\perp$ is a symmetric version of the monoidal strength defined in Chapter 2. A useful result is that

$$(a \otimes \perp)^* \circ (1_{\mathbb{P}} \otimes f) = (1_{\mathbb{P}'} \otimes f)_\perp \circ (a \otimes \perp)^* \tag{4.31}$$

for any $f : \mathbb{Q}' \to \mathbb{Q}$ of **Aff**:

$$\begin{aligned} &(a \otimes \perp)^* \circ (1_{\mathbb{P}} \otimes f) \\ &= str'_{\mathbb{P}',\mathbb{Q}} \circ (a^* \otimes 1_{\mathbb{Q}}) \circ (1_{\mathbb{P}} \otimes f) \\ &= str'_{\mathbb{P}',\mathbb{Q}} \circ (1_{\mathbb{P}'_\perp} \otimes f) \circ (a^* \otimes 1_{\mathbb{Q}'}) \\ &= (1_{\mathbb{P}'} \otimes f)_\perp \circ str'_{\mathbb{P}',\mathbb{Q}'} \circ (a^* \otimes 1_{\mathbb{Q}'}) \quad \text{(nat. of } str'\text{)} \\ &= (1_{\mathbb{P}'} \otimes f)_\perp \circ (a \otimes \perp)^* \end{aligned}$$

The operational semantics of HOPLA and the proofs of soundness and adequacy benefited from the fact that we only ever needed to apply a^* to closed terms. In the affine case we are not so fortunate because of the tensor match. As $[u > x \otimes y \Rightarrow t]$ is interpreted as the composition $[\![t]\!] \circ (1_\Gamma \otimes [\![u]\!])$ we have

$$a^*[\![[u > x \otimes y \Rightarrow t]]\!] = [\![[u > x \otimes y \Rightarrow a^* t]]\!] \ . \tag{4.32}$$

This suggests that we let t (an open term) take an a-transition in the "environment" $u > x \otimes y$.

Syntactically, environments e are lists of such matches with ϵ the empty list. As a notational convenience, we'll write $[e \Rightarrow t]$ for the term given inductively by

$$\begin{aligned} [\epsilon \Rightarrow t] &\equiv_{\text{def}} t \\ [e, u > x \otimes y \Rightarrow t] &\equiv_{\text{def}} [e \Rightarrow [u > x \otimes y \Rightarrow t]] \end{aligned} \tag{4.33}$$

An environment "exports" a set of variables; the empty environment ϵ exports the empty set, while $e, u > x \otimes y$ exports what e exports except the free variables of u plus x and y, possibly overshadowing variables exported by e of the same names. We may formalise this using a judgement

$$e \vdash x_1 : \mathbb{P}_1, \ldots, x_k : \mathbb{P}_k \tag{4.34}$$

(with the x_i distinct) which denotes the same path set over $\mathbb{P}_1 \otimes \cdots \otimes \mathbb{P}_k$ as the term $[e \Rightarrow x_1 \otimes \cdots \otimes x_k]$.

A term t in environment e will be written $e \mapsto t$. For $e \vdash \Gamma, \Phi$ and $\Gamma \vdash t : \mathbb{P}$, we give the judgement

$$\vdash e \mapsto t : \mathbb{P}; \Phi \tag{4.35}$$

the same denotation as the term $[e \Rightarrow t \otimes x_1 \otimes \cdots \otimes x_k]$ where the x_i are now the variables exported by e but not free in t. We'll avoid some of the book-keeping regarding lists like Φ by treating them as finite mappings from exported variables to types. So when writing $e \vdash \Phi$ or $\vdash e \mapsto t : \mathbb{P}; \Phi$ below, we'll imply no ordering on the variables in Φ.

Incorporating environments, the operational rules are shown in Figure 4.2. They define a transition relation $\mathbb{P} : e \mapsto t \xrightarrow{a} e' \mapsto t'$ with $\vdash e \mapsto t : \mathbb{P}; \Phi$ and $\mathbb{P} : a : \mathbb{P}'$. In the first two rules it is assumed that x, respectively y, is not free in nor overshadowed by the environment e_2. In the rule for tensor match, the variables x and y are implicitly renamed to avoid overshadowing of exported variables in the environment $e, u > x \otimes y$. Transitions do not change the length and variables of environments:

Lemma 4.6 If $\mathbb{P} : e \mapsto t \xrightarrow{a} e' \mapsto t'$ with $e \equiv u_1 > x_1 \otimes y_1, \ldots, u_k > x_k \otimes y_k$, then e' has the form $e \equiv u'_1 > x_1 \otimes y_1, \ldots, u'_k > x_k \otimes y_k$ for some $u'_1, \ldots, u'_k$.

Proof. By an easy rule-induction on the operational rules. □

We'll write $\mathbb{P} : t \xrightarrow{a} t'$ for transitions with empty environment.

The rules are well-typed in the same way as the operational rules of HOPLA, but additionally, transitions do not change the types of exported variables:

Lemma 4.7 Suppose $\vdash e \mapsto t : \mathbb{P}; \Phi$. If $\mathbb{P} : e \mapsto t \xrightarrow{a} e' \mapsto t'$ with $\mathbb{P} : a : \mathbb{P}'$, then $\vdash e' \mapsto t' : \mathbb{P}'; \Phi$.

Proof. By rule-induction on the operational rules.

Variable. Suppose $\vdash e_1, u > x \otimes y, e_2 \mapsto x : \mathbb{P}; \Phi$ and that

$$\mathbb{P} : e_1, u > x \otimes y, e_2 \mapsto x \xrightarrow{a} e'_1, u' > x \otimes y, e_2 \mapsto x \tag{4.36}$$

is derived from

$$\mathbb{P} \otimes \mathbb{Q} : e_1 \mapsto u \xrightarrow{a \otimes \bot} e'_1 \mapsto u' \tag{4.37}$$

with $\mathbb{P} : a : \mathbb{P}'$. We then have $\vdash e_1 \mapsto u : \mathbb{P} \otimes \mathbb{Q}; \Phi_1$ for some set Φ_1. Moreover, $\mathbb{P} \otimes \mathbb{Q} : a \otimes \bot : \mathbb{P}' \otimes \mathbb{Q}$, and so by the induction hypothesis, we have $\vdash e'_1 \mapsto u' : \mathbb{P}' \otimes \mathbb{Q}; \Phi_1$. But then $\vdash e'_1, u' > x \otimes y, e_2 \mapsto x : \mathbb{P}'; \Phi$ by typing as wanted.

$$\frac{\mathbb{P} \otimes \mathbb{Q} : e_1 \mapsto u \xrightarrow{a \otimes \bot} e_1' \mapsto u'}{\mathbb{P} : e_1, u > x \otimes y, e_2 \mapsto x \xrightarrow{a} e_1', u' > x \otimes y, e_2 \mapsto x}$$

$$\frac{\mathbb{P} \otimes \mathbb{Q} : e_1 \mapsto u \xrightarrow{\bot \otimes a} e_1' \mapsto u'}{\mathbb{Q} : e_1, u > x \otimes y, e_2 \mapsto y \xrightarrow{a} e_1', u' > x \otimes y, e_2 \mapsto y}$$

$$\frac{\mathbb{P} : e \mapsto t[rec\, x.t/x] \xrightarrow{a} e' \mapsto t'}{\mathbb{P} : e \mapsto rec\, x.t \xrightarrow{a} e' \mapsto t'}$$

$$\frac{\mathbb{P} : e \mapsto t_j \xrightarrow{a} e' \mapsto t'}{\mathbb{P} : e \mapsto \Sigma_{i \in I} t_i \xrightarrow{a} e' \mapsto t'}\ j \in I$$

$$\frac{\mathbb{P} : e \mapsto t \xrightarrow{a} e' \mapsto t'}{\mathbb{P} \otimes \mathbb{Q} : e \mapsto t \otimes u \xrightarrow{a \otimes \bot} e' \mapsto t' \otimes u}$$

$$\frac{\mathbb{Q} : e \mapsto u \xrightarrow{a} e' \mapsto u'}{\mathbb{P} \otimes \mathbb{Q} : e \mapsto t \otimes u \xrightarrow{\bot \otimes a} e' \mapsto t \otimes u'}$$

$$\frac{\mathbb{P} : e, u > x \otimes y \mapsto t \xrightarrow{a} e', u' > x \otimes y \mapsto t'}{\mathbb{P} : e \mapsto [u > x \otimes y \Rightarrow t] \xrightarrow{a} e' \mapsto [u' > x \otimes y \Rightarrow t']}$$

$$\frac{\mathbb{P}_\beta : e \mapsto t \xrightarrow{a} e' \mapsto t'}{\Sigma_{\alpha \in A} \mathbb{P}_\alpha : e \mapsto \beta t \xrightarrow{\beta a} e' \mapsto t'}$$

$$\frac{\Sigma_{\alpha \in A} \mathbb{P}_\alpha : e \mapsto t \xrightarrow{\beta a} e' \mapsto t'}{\mathbb{P}_\beta : e \mapsto \pi_\beta t \xrightarrow{a} e' \mapsto t'}$$

$$\frac{}{\mathbb{P}_\bot : e \mapsto !t \xrightarrow{!} e \mapsto t}$$

$$\frac{\mathbb{P}_\bot : e \mapsto u \xrightarrow{!} e' \mapsto u' \quad \mathbb{Q} : e' \mapsto t[u'/x] \xrightarrow{a} e'' \mapsto t'}{\mathbb{Q} : e \mapsto [u > !x \Rightarrow t] \xrightarrow{a} e'' \mapsto t'}$$

$$\frac{\mathbb{T}_j[\mu \vec{T}.\vec{\mathbb{T}}/\vec{T}] : e \mapsto t \xrightarrow{a} e' \mapsto t'}{\mu_j \vec{T}.\vec{\mathbb{T}} : e \mapsto abs\, t \xrightarrow{abs\, a} e' \mapsto t'}$$

$$\frac{\mu_j \vec{T}.\vec{\mathbb{T}} : e \mapsto t \xrightarrow{abs\, a} e' \mapsto t'}{\mathbb{T}_j[\mu \vec{T}.\vec{\mathbb{T}}/\vec{T}] : e \mapsto rep\, t \xrightarrow{a} e' \mapsto t'}$$

Figure 4.2: Operational rules for first-order Affine HOPLA

Tensor. Suppose $\vdash e \mapsto t \otimes u : \mathbb{P} \otimes \mathbb{Q}; \Phi$ and that

$$\mathbb{P} \otimes \mathbb{Q} : e \mapsto t \otimes u \xrightarrow{a \otimes \perp} e' \mapsto t' \otimes u \tag{4.38}$$

is derived from

$$\mathbb{P} : e \mapsto t \xrightarrow{a} e' \mapsto t' \tag{4.39}$$

with $\mathbb{P} \otimes \mathbb{Q} : a \otimes \perp : \mathbb{P}' \otimes \mathbb{Q}$. With Φ_u the free variables of u we then have $\vdash e \mapsto t : \mathbb{P}; \Phi, \Phi_u$. Moreover, $\mathbb{P} : a : \mathbb{P}'$ and so by the induction hypothesis, $\vdash e' \mapsto t' : \mathbb{P}'; \Phi, \Phi_u$. Typing now yields $\vdash e' \mapsto t' \otimes u : \mathbb{P}' \otimes \mathbb{Q}; \Phi$ as wanted.

Tensor match. Under the assumption that the variables x and y are renamed if necessary to ensure that they do not overshadow the exported variables of Φ, we have

$$\vdash e \mapsto [u > x \otimes y \Rightarrow t] : \mathbb{P}; \Phi \quad \text{iff} \quad \vdash e, u > x \otimes y \mapsto t : \mathbb{P}; \Phi \ . \tag{4.40}$$

We are done by the induction hypothesis.

The remaining rules are handled similarly. □

We'll write $\mathbb{P} : e \mapsto t \xrightarrow{a} e' \mapsto t' : \mathbb{P}'$ when $\vdash e \mapsto t : \mathbb{P}; \Phi$ and $\mathbb{P} : a : \mathbb{P}'$.

Although we'll not need the syntactic operators a^* below, we show for completeness that the operational semantics validates a result much like Lemma 3.13 saying that observations of a-transitions can be reduced to observations of !-transitions:

Lemma 4.8 Let C be a trivial context generated by the grammar

$$C ::= - \mid [C > x \otimes y \Rightarrow C(x) \otimes C(y)] \ . \tag{4.41}$$

Then $\mathbb{P} : e \mapsto t \xrightarrow{a} e' \mapsto t' : \mathbb{P}'$ iff $\mathbb{P}'_\perp : e \mapsto a^*t \xrightarrow{!} e' \mapsto C(t') : \mathbb{P}'$ for some such context C using fresh variables.

Proof. By structural induction on a. We give the details in the case $a \otimes \perp$. Supposing

$$\mathbb{P} \otimes \mathbb{Q} : e \mapsto t \xrightarrow{a \otimes \perp} e' \mapsto t' : \mathbb{P}' \otimes \mathbb{Q} \ , \tag{4.42}$$

we can derive

$$\mathbb{P} : e, t > x \otimes y \mapsto x \xrightarrow{a} e', t' > x \otimes y \mapsto x : \mathbb{P}' \tag{4.43}$$

using fresh variables x and y. By the induction hypothesis, this means that for some context C,

$$\mathbb{P}'_\perp : e, t > x \otimes y \mapsto a^*x \xrightarrow{!} e', t' > x \otimes y \mapsto C(x) : \mathbb{P}' \ . \tag{4.44}$$

We can the complete a derivation with conclusion

$$\mathbb{P}'_\perp : e \mapsto [t > x \otimes y \Rightarrow [a^* x > !x' \Rightarrow !(x' \otimes y)]] \\ \xrightarrow{!} e' \mapsto [t' > x \otimes y \Rightarrow C(x) \otimes y] \;, \quad (4.45)$$

which is just the same as $\mathbb{P}'_\perp : e \mapsto (a \otimes \perp)^* t \xrightarrow{!} e' \mapsto C'(t') : \mathbb{P}'$ where C' is the context $C' \equiv [- > x \otimes y \Rightarrow C(x) \otimes y]$. The reverse implication follows the same argument backwards. □

4.4.1 A Correspondence Result

We'll prove correspondence, so soundness and adequacy, by exploiting the linearity constraints on variable occurrences in the affine language. This allows us replace the use of logical relations by well-founded induction based on a size measure on terms [63, 64]. Of course, this method works only for the language obtained by replacing general recursion by finite unfoldings. We'll extend the result to cover full recursion afterwards.

The term $rec^n\, x.t$ is typed in the same way as $rec\, x.t$ and interpreted as the n'th approximation to the denotation of $rec\, x.t$. By induction,

$$[\![rec^0\, x.t]\!] =_{\mathrm{def}} \varnothing \quad \text{and} \quad [\![rec^{n+1}\, x.t]\!] =_{\mathrm{def}} [\![t[rec^n\, x.t/x]]\!] \;. \qquad (4.46)$$

The corresponding operational rule is given by

$$\frac{\mathbb{P} : e \mapsto t[rec^n\, x.t/x] \xrightarrow{a} e' \mapsto t'}{\mathbb{P} : e \mapsto rec^{n+1}\, x.t \xrightarrow{a} e' \mapsto t'} \qquad (4.47)$$

We define an ordinal size measure on terms in environments, see Figure 4.3. The need for ordinals arises because of the possibly infinite sum. In the figure, $\oplus$ is the "natural addition" of ordinals (see [49], Def. 2.21); this operation is associative, commutative, has identity 0, and is strictly monotone in each argument.

Because variables that are not crossed occur in different components of a nondeterministic sum, we can prove

Lemma 4.9 Suppose $\Gamma, x_1 : \mathbb{P}, \ldots, x_k : \mathbb{P} \vdash t : \mathbb{Q}$ with $\{x_1, \ldots, x_k\}$ not crossed in t. If $\Lambda \vdash u : \mathbb{P}$ with Γ and Λ disjoint, then $|t[u/x_1, \ldots, u/x_k]| \leq |t| \oplus |u|$.

Proof. By structural induction on t. □

Transitions are accompanied by a decrease in size:

Lemma 4.10 Suppose $\mathbb{P} : e \mapsto t \xrightarrow{a} e' \mapsto t' : \mathbb{P}'$. Then $|e \mapsto t| > |e' \mapsto t'|$.

$$
\begin{aligned}
|x| &=_{\text{def}} 1\\
|\,\mathit{rec}^0\, x.t| &=_{\text{def}} 1\\
|\,\mathit{rec}^{n+1}\, x.t| &=_{\text{def}} |t[\mathit{rec}^n\, x.t/x]| \oplus 1\\
|\Sigma_{i\in I} t_i| &=_{\text{def}} (\sup_{i\in I}|t_i|) \oplus 1\\
|t \otimes u| &=_{\text{def}} |t| \oplus |u|\\
|[u > x \otimes y \Rightarrow t]| &=_{\text{def}} |u| \oplus |t|\\
|\beta t| &=_{\text{def}} |t| \oplus 1\\
|\pi_\beta t| &=_{\text{def}} |t| \oplus 1\\
|!t| &=_{\text{def}} |t| \oplus 1\\
|[u > !x \Rightarrow t]| &=_{\text{def}} |u| \oplus |t|\\
|\,\mathit{abs}\, t| &=_{\text{def}} |t| \oplus 1\\
|\,\mathit{rep}\, t| &=_{\text{def}} |t| \oplus 1
\end{aligned}
$$

$$
\begin{aligned}
|\epsilon| &=_{\text{def}} 0\\
|e, u > x \otimes y| &=_{\text{def}} |e| \oplus |u|
\end{aligned}
$$

$$
|e \mapsto t| =_{\text{def}} |e| \oplus |t|
$$

Figure 4.3: Size measure

Proof. By an easy rule-induction on the operational rules, using Lemma 4.9 for the prefix match rule. □

We can now define a well-founded relation which relates the conclusion to each premise in all rules:

Lemma 4.11 For each rule

$$
\frac{\cdots e_2 \mapsto t_2 \xrightarrow{a_2} e_2' \mapsto t_2' \cdots}{e_1 \mapsto t_1 \xrightarrow{a_1} e_1' \mapsto t_1'} \tag{4.48}
$$

we have $e_1 \mapsto t_1 \succ e_2 \mapsto t_2$ where $\succ$ is the lexicographic order

$$
e \mapsto t \succ e' \mapsto t' \iff \begin{array}{l}(|e \mapsto t| > |e' \mapsto t'|) \text{ or}\\ (|e \mapsto t| = |e' \mapsto t'|) \text{ and } (|t| > |t'|)\end{array} \tag{4.49}
$$

Proof. By a straightforward case analysis, using Lemma 4.10 for the prefix match rule. The rule for tensor match needs the second clause of the definition of $\succ$. □

Lemma 4.12 Assume $\vdash t : \mathbb{P}$ in the finitary language and that $\mathbb{P} : a : \mathbb{P}'$. Then $\Sigma_{t'}[\![!t']\!] = a^*[\![t']\!]$ where t' ranges over terms such that $\mathbb{P} : t \xrightarrow{a} t' : \mathbb{P}'$.

Proof. By well-founded induction on $\succ$ using the induction hypothesis

> Assume $\vdash e \mapsto t : \mathbb{P}; \Phi$ and $\mathbb{P} : a : \mathbb{P}'$ and let $\vec{z}$ be the variables in Φ tensored together. Then we have
>
> $$(a \otimes \bot)^* [\![[e \Rightarrow t \otimes \vec{z}]]\!] = \Sigma_{e',t'} [\![![e' \Rightarrow t' \otimes \vec{z}]]\!] \tag{4.50}$$
>
> —where e', t' range over environments and terms such that $\mathbb{P} : e \mapsto t \xrightarrow{a} e' \mapsto t' : \mathbb{P}'$.

We proceed by case analysis on t.

Variable. There are two possible last rules in derivations, depending on the structure of e. One is

$$\frac{\mathbb{P} \otimes \mathbb{Q} : e_1 \mapsto u \xrightarrow{a \otimes \bot} e_1' \mapsto u'}{\mathbb{P} : e_1, u > x \otimes y, e_2 \mapsto x \xrightarrow{a} e_1', u' > x \otimes y, e_2 \mapsto x} \tag{4.51}$$

It is assumed that the variable x is neither free in nor overshadowed by e_2. Let $\vec{w}$ be the variables exported by e_1 but not free in u. We have

$$\begin{aligned}
&(a \otimes \bot)^* [\![[e_1, u > x \otimes y, e_2 \Rightarrow x \otimes \vec{z}]]\!] \\
&= (a \otimes \bot)^* [\![[e_1, u > x \otimes y \Rightarrow x \otimes [e_2 \Rightarrow \vec{z}]]]\!] && \text{(assumption on } x\text{)} \\
&= (a \otimes \bot)^* \circ (1_{\mathbb{P}} \otimes [\![[e_2 \Rightarrow \vec{z}]]\!]) \circ ([\![u]\!] \otimes 1) \circ [\![e_1]\!] \\
&= (1_{\mathbb{P}'} \otimes [\![[e_2 \Rightarrow \vec{z}]]\!])_\bot \circ (a \otimes \bot)^* \circ [\![[e_1 \Rightarrow u \otimes \vec{w}]]\!] && \text{(by (4.31))} \\
&= (1_{\mathbb{P}'} \otimes [\![[e_2 \Rightarrow \vec{z}]]\!])_\bot \circ \Sigma_{e',u'} [\![![e' \Rightarrow u' \otimes \vec{w}]]\!] && \text{(ind. hyp.)} \\
&= \Sigma_{e',u'} (1_{\mathbb{P}'} \otimes [\![[e_2 \Rightarrow \vec{z}]]\!])_\bot \circ [\![![e' \Rightarrow u' \otimes \vec{w}]]\!] && \text{(linearity)} \\
&= \Sigma_{e',u'} (1_{\mathbb{P}'} \otimes [\![[e_2 \Rightarrow \vec{z}]]\!])_\bot \circ \eta \circ [\![[e' \Rightarrow u' \otimes \vec{w}]]\!] \\
&= \Sigma_{e',u'} \eta \circ (1_{\mathbb{P}'} \otimes [\![[e_2 \Rightarrow \vec{z}]]\!]) \circ [\![[e' \Rightarrow u' \otimes \vec{w}]]\!] && \text{(nat. of } \eta\text{)} \\
&= \Sigma_{e',u'} [\![![e', u' > x \otimes y, e_2 \Rightarrow x \otimes \vec{z}]]\!] && \text{(assumption on } x\text{)}
\end{aligned}$$

—as wanted. The other possible rule is handled symmetrically.

Recursive definition. There is one possible last rule:

$$\frac{\mathbb{P} : e \mapsto t[\mathit{rec}^n\, x.t/x] \xrightarrow{a} e' \mapsto t'}{\mathbb{P} : e \mapsto \mathit{rec}^{n+1}\, x.t \xrightarrow{a} e' \mapsto t'} \tag{4.52}$$

We have

$$\begin{aligned}
&(a \otimes \bot)^* [\![[e \Rightarrow \mathit{rec}^{n+1}\, x.t \otimes \vec{z}]]\!] \\
&= (a \otimes \bot)^* [\![[e \Rightarrow t[\mathit{rec}^n\, x.t/x] \otimes \vec{z}]]\!] \\
&= \Sigma_{e',t'} [\![![e' \Rightarrow t' \otimes \vec{z}]]\!] && \text{(ind. hyp.)}
\end{aligned}$$

—as wanted.

Nondeterministic sum. There is one possible last rule:

$$\frac{\mathbb{P} : e \mapsto t_j \xrightarrow{a} e' \mapsto t'}{\mathbb{P} : e \mapsto \Sigma_{i \in I} t_i \xrightarrow{a} e' \mapsto t'}\ j \in I \tag{4.53}$$

We have

$$
\begin{aligned}
&(a \otimes \bot)^* [\![[e \Rightarrow \Sigma_{i \in I} t_i \otimes \vec{z}]]\!] \\
&= (a \otimes \bot)^* \circ (\Sigma_{i \in I} [\![t_i]\!] \otimes 1) \circ [\![e]\!] \\
&= str' \circ (a^* \otimes 1) \circ (\Sigma_{i \in I} [\![t_i]\!] \otimes 1) \circ [\![e]\!] \\
&= str' \circ (\Sigma_{i \in I} a^* [\![t_i]\!] \otimes 1) \circ [\![e]\!] && \text{(linearity of } a^*) \\
&= \Sigma_{i \in I} str' \circ (a^* [\![t_i]\!] \otimes 1) \circ [\![e]\!] && \text{(linearity of 1st arg. of } str') \\
&= \Sigma_{i \in I} str' \circ (a^* \otimes 1) \circ [\![[e \Rightarrow t_i \otimes \vec{z}]]\!] \\
&= \Sigma_{i \in I} (a \otimes \bot)^* \circ [\![[e \Rightarrow t_i \otimes \vec{z}]]\!] \\
&= \Sigma_{i \in I} \Sigma_{e'_i, t'_i} [\![![e'_i \Rightarrow t'_i \otimes \vec{z}]]\!] && \text{(ind. hyp.)}
\end{aligned}
$$

—as wanted.

Tensor. There are two possible last rules, one of which is:

$$
\frac{\mathbb{P} : e \Mapsto t \xrightarrow{a} e' \Mapsto t'}{\mathbb{P} \otimes \mathbb{Q} : e \Mapsto t \otimes u \xrightarrow{a \otimes \bot} e' \Mapsto t' \otimes u} \tag{4.54}
$$

Let $\vec{w}$ be the free variables of u. We have

$$
\begin{aligned}
&((a \otimes \bot) \otimes \bot)^* [\![[e \Rightarrow (t \otimes u) \otimes \vec{z}]]\!] \\
&= ((a \otimes \bot) \otimes \bot)^* \circ ([\![t]\!] \otimes [\![u]\!] \otimes 1) \circ [\![e]\!] \\
&= ((a \otimes \bot) \otimes \bot)^* \circ (1_{\mathbb{P}} \otimes [\![u]\!] \otimes 1) \circ [\![[e \Rightarrow t \otimes \vec{w}\vec{z}]]\!] \\
&= (1'_{\mathbb{P}} \otimes [\![u]\!] \otimes 1)_\bot \circ (a \otimes \bot)^* \circ [\![[e \Rightarrow t \otimes \vec{w}\vec{z}]]\!] && \text{(by (4.31))} \\
&= (1'_{\mathbb{P}} \otimes [\![u]\!] \otimes 1)_\bot \circ \Sigma_{e', t'} [\![![e' \Rightarrow t' \otimes \vec{w}\vec{z}]]\!] && \text{(ind. hyp.)} \\
&= \Sigma_{e', t'} (1'_{\mathbb{P}} \otimes [\![u]\!] \otimes 1)_\bot \circ [\![![e' \Rightarrow t' \otimes \vec{w}\vec{z}]]\!] && \text{(linearity)} \\
&= \Sigma_{e', t'} (1'_{\mathbb{P}} \otimes [\![u]\!] \otimes 1)_\bot \circ \eta \circ [\![[e' \Rightarrow t' \otimes \vec{w}\vec{z}]]\!] \\
&= \Sigma_{e', t'} \eta \circ (1'_{\mathbb{P}} \otimes [\![u]\!] \otimes 1) \circ [\![[e' \Rightarrow t' \otimes \vec{w}\vec{z}]]\!] && \text{(nat. of } \eta) \\
&= \Sigma_{e', t'} [\![![e' \Rightarrow (t' \otimes u) \otimes \vec{z}]]\!]
\end{aligned}
$$

—as wanted. The other rule is handled symmetrically.

Tensor match. There is one possible last rule:

$$
\frac{\mathbb{P} : e, u > x \otimes y \Mapsto t \xrightarrow{a} e', u' > x \otimes y \Mapsto t'}{\mathbb{P} : e \Mapsto [u > x \otimes y \Rightarrow t] \xrightarrow{a} e' \Mapsto [u' > x \otimes y \Rightarrow t']} \tag{4.55}
$$

It is assumed that the variables x, y are renamed if necessary so that they do not overshadow any variables exported by e. We then have

$$
\begin{aligned}
&(a \otimes \bot)^* [\![[e \Rightarrow [u > x \otimes y \Rightarrow t] \otimes \vec{z}]]\!] \\
&= (a \otimes \bot)^* [\![[e, u > x \otimes y \Rightarrow t \otimes \vec{z}]]\!] && \text{(assumption on } x, y) \\
&= \Sigma_{e', u', t'} [\![![e', u' > x \otimes y \Rightarrow t' \otimes \vec{z}]]\!] && \text{(ind. hyp.)} \\
&= \Sigma_{e', u', t'} [\![![e' \Rightarrow [u' > x \otimes y \Rightarrow t'] \otimes \vec{z}]]\!] && \text{(assumption on } x, y)
\end{aligned}
$$

—as wanted.

Injection. There is one possible last rule:

$$\frac{\mathbb{P}_\beta : e \Mapsto t \xrightarrow{a} e' \Mapsto t'}{\Sigma_{\alpha\in A}\mathbb{P}_\alpha : e \Mapsto \beta t \xrightarrow{\beta a} e' \Mapsto t'} \tag{4.56}$$

We have

$$\begin{aligned}
&(\beta a \otimes \bot)^* [\![[e \Rightarrow \beta t \otimes \vec{z}]]\!] \\
&= str' \circ ((\beta a)^* \otimes 1) \circ ((in_\beta \circ [\![t]\!]) \otimes 1) \circ [\![e]\!] \\
&= str' \circ ((a^* \circ \pi_\beta \circ in_\beta \circ [\![t]\!]) \otimes 1) \circ [\![e]\!] \\
&= str' \circ ((a^* \circ [\![t]\!]) \otimes 1) \circ [\![e]\!] \\
&= (a \otimes \bot)^* \circ [\![[e \Rightarrow t \otimes \vec{z}]]\!] \\
&= \Sigma_{e',t'} [\![![e' \Rightarrow t' \otimes \vec{z}]]\!] && \text{(ind. hyp.)}
\end{aligned}$$

—as wanted.

Projection. There is one possible last rule:

$$\frac{\Sigma_{\alpha\in A}\mathbb{P}_\alpha : e \Mapsto t \xrightarrow{\beta a} e' \Mapsto t'}{\mathbb{P}_\beta : e \Mapsto \pi_\beta t \xrightarrow{a} e' \Mapsto t'} \tag{4.57}$$

We have

$$\begin{aligned}
&(a \otimes \bot)^* [\![[e \Rightarrow \pi_\beta t \otimes \vec{z}]]\!] \\
&= str' \circ (a^* \otimes 1) \circ ((\pi_\beta \circ [\![t]\!]) \otimes 1) \circ [\![e]\!] \\
&= str' \circ ((a^* \circ \pi_\beta \circ [\![t]\!]) \otimes 1) \circ [\![e]\!] \\
&= str' \circ (((\beta a)^* \circ [\![t]\!]) \otimes 1) \circ [\![e]\!] \\
&= (\beta a \otimes \bot)^* \circ [\![[e \Rightarrow t \otimes \vec{z}]]\!] \\
&= \Sigma_{e',t'} [\![![e' \Rightarrow t' \otimes \vec{z}]]\!] && \text{(ind. hyp.)}
\end{aligned}$$

—as wanted.

Prefixing. There is one possible last rule:

$$\frac{}{\mathbb{P}_\bot : e \Mapsto !t \xrightarrow{!} e \Mapsto t} \tag{4.58}$$

We have

$$\begin{aligned}
&(! \otimes \bot)^* [\![[e \Rightarrow !t \otimes \vec{z}]]\!] \\
&= str' \circ (!^* \otimes 1) \circ ((\eta \circ [\![t]\!]) \otimes 1) \circ [\![e]\!] \\
&= str' \circ ((\eta \circ [\![t]\!]) \otimes 1) \circ [\![e]\!] && (!^* \text{ is id}) \\
&= str' \circ (\eta \otimes 1) \circ [\![[e \Rightarrow t \otimes \vec{z}]]\!] \\
&= \eta \circ [\![[e \Rightarrow t \otimes \vec{z}]]\!] && (\text{prop. of } str') \\
&= [\![![e \Rightarrow t \otimes \vec{z}]]\!]
\end{aligned}$$

—as wanted.

Prefix match. There is one possible last rule:

$$\frac{\mathbb{P}_\perp : e \Mapsto u \xrightarrow{!} e' \Mapsto u' \quad \mathbb{Q} : e' \Mapsto t[u'/x] \xrightarrow{a} e'' \Mapsto t'}{\mathbb{Q} : e \Mapsto [u > !x \Rightarrow t] \xrightarrow{a} e'' \Mapsto t'} \tag{4.59}$$

Let $\vec{w}$ be the free variables of t except x. Using symmetry maps we may assume that $[u > !x \Rightarrow t]$ is interpreted as the composition $\overline{[\![t]\!]} \circ str' \circ ([\![u]\!] \otimes 1)$. We then have

$$\begin{array}{ll}
(a \otimes \perp)^*[\![[e \Rightarrow [u > !x \Rightarrow t] \otimes \vec{z}]\!] & \\
\quad = (a \otimes \perp)^* \circ ((\overline{[\![t]\!]} \circ str' \circ ([\![u]\!] \otimes 1)) \otimes 1) \circ [\![e]\!] & \\
\quad = (a \otimes \perp)^* \circ \overline{[\![t]\!]} \circ str' \circ [\![[e \Rightarrow u \otimes \vec{w}\vec{z}]\!] & \\
\quad = (a \otimes \perp)^* \circ \overline{[\![t]\!]} \circ str' \circ (!^* \otimes 1) \circ [\![[e \Rightarrow u \otimes \vec{w}\vec{z}]\!] & (!^* \text{ is id.}) \\
\quad = (a \otimes \perp)^* \circ \overline{[\![t]\!]} \circ (! \otimes \perp)^* \circ [\![[e \Rightarrow u \otimes \vec{w}\vec{z}]\!] & \\
\quad = (a \otimes \perp)^* \circ \overline{[\![t]\!]} \circ \Sigma_{e',u'} [\![![e' \Rightarrow u' \otimes \vec{w}\vec{z}]\!] & (\text{ind. hyp.}) \\
\quad = \Sigma_{e',u'} (a \otimes \perp)^* \circ \overline{[\![t]\!]} \circ [\![![e' \Rightarrow u' \otimes \vec{w}\vec{z}]\!] & (\text{linearity}) \\
\quad = \Sigma_{e',u'} (a \otimes \perp)^* \circ \overline{[\![t]\!]} \circ \eta \circ [\![[e' \Rightarrow u' \otimes \vec{w}\vec{z}]\!] & \\
\quad = \Sigma_{e',u'} (a \otimes \perp)^* \circ [\![t]\!] \circ [\![[e' \Rightarrow u' \otimes \vec{w}\vec{z}]\!] & (\text{univ. prop. of } \eta) \\
\quad = \Sigma_{e',u'} (a \otimes \perp)^* \circ [\![[e' \Rightarrow t[u'/x] \otimes \vec{z}]\!] & (\text{subst. lemma}) \\
\quad = \Sigma_{e',u'} \Sigma_{e'',t'} [\![![e'' \Rightarrow t' \otimes \vec{z}]\!] & (\text{ind. hyp.})
\end{array}$$

—as wanted.

Folding and unfolding. Similar to injection and projection.

By well-founded induction, the proof is complete. □

We'll now prove correspondence for general recursion. We write $t^{(n)}$ for the term obtained by replacing all subterms of the form $rec\, x.u$ in t by $rec^n\, x.u$. We have $[\![t]\!] = \bigcup_{n\in\omega} [\![t^{(n)}]\!]$. Conversely, we write t^- for the term obtained by removing all indices on subterms $rec^n\, x.u$ of t. We clearly have $(t^{(n)})^- \equiv t$ and $[\![t]\!] \subseteq [\![t^-]\!]$.

Theorem 4.13 (Correspondence) Let $\vdash t : \mathbb{P}$ in the first-order fragment with full recursion and $\mathbb{P} : a : \mathbb{P}'$. Then $\Sigma_{t'} [\![!t']\!] = a^*[\![t]\!]$ where t' ranges over terms such that $\mathbb{P} : t \xrightarrow{a} t' : \mathbb{P}'$.

Proof. It follows by linearity of a^* and Lemma 4.12 that

$$a^*[\![t]\!] = a^* \textstyle\bigcup_{n\in\omega} [\![t^{(n)}]\!] = \bigcup_{n\in\omega} a^*[\![t^{(n)}]\!] = \bigcup_{n\in\omega} \Sigma_{t'_n} [\![!t'_n]\!] \ , \tag{4.60}$$

where t'_n ranges over terms such that $\mathbb{P} : t^{(n)} \xrightarrow{a} t'_n : \mathbb{P}'$. Now, each such transition can be matched by a transition $\mathbb{P} : (t^{(n)})^- \xrightarrow{a} (t'_n)^- : \mathbb{P}'$. As $(t^{(n)})^- \equiv t$, the term $(t'_n)^-$ is just one of the terms t' in the statement of the theorem. It follows that $\Sigma_{t'_n} [\![!t'_n]\!] \subseteq \Sigma_{t'} [\![!t']\!]$ for each n and so

$$a^*[\![t]\!] = \textstyle\bigcup_{n\in\omega} \Sigma_{t'_n} [\![!t'_n]\!] \subseteq \Sigma_{t'} [\![!t']\!] \ . \tag{4.61}$$

For the converse, suppose $\mathbb{P} : t \xrightarrow{a} t' : \mathbb{P}'$. The derivation unfolds each subterm $rec\, x.u$ of t a number of times, and because the derivation is finite, there exists $n \in \omega$ such that we have a transition $\mathbb{P} : t^{(n)} \xrightarrow{a} t'_n : \mathbb{P}'$. Hence,

$$\Sigma_{t'}[\![!t']\!] \subseteq \bigcup_{n\in\omega} \Sigma_{t'_n}[\![!t'_n]\!] = a^*[\![t]\!] \tag{4.62}$$

as wanted. □

4.4.2 Independence

As it stands, the operational semantics gives an interleaving model of the first-order fragment, taking atomic steps in one tensor component at a time. This does not fully match the understanding of the tensor operation as a juxtaposition of independent processes. Given transitions as in the upper part of the diamond below one would expect to be able to prove the existence of a term t_2 with transitions as in the lower part, completing the diamond:

$$\begin{tikzcd} & t_1 \arrow[dr, "\bot\otimes a_2"] & \\ t \arrow[ur, "a_1\otimes\bot"] \arrow[dr, dotted, "\bot\otimes a_2"'] & & t' \\ & t_2 \arrow[ur, dotted, "a_1\otimes\bot"'] & \end{tikzcd} \tag{4.63}$$

Formally, one could then exhibit the operational semantics as an asynchronous transition system [5, 84]. We do not have a direct proof of the diamond property, and a treatment of the independence structure of the affine language will have to wait to Chapter 7.

4.4.3 Function Space

Although we don't know how to extend the operational semantics with function space in general, we may reuse the $u \mapsto a$-actions of HOPLA to define the rules below with the side-condition that u be a *closed* term:

$$\frac{\mathbb{Q} : e \Mapsto t[u/x] \xrightarrow{a} e' \Mapsto t'}{\mathbb{P} \multimap \mathbb{Q} : e \Mapsto \lambda x.t \xrightarrow{u\mapsto a} e' \Mapsto t'} \qquad \frac{\mathbb{P} \multimap \mathbb{Q} : e \Mapsto t \xrightarrow{u\mapsto a} e' \Mapsto t'}{\mathbb{Q} : e \Mapsto t\, u \xrightarrow{a} e' \Mapsto t'} \tag{4.64}$$

If u was allowed to be an open term, actions would have free variables and the remaining rules cannot deal with that. As an example, consider an attempted derivation of a transition from $[\lambda z.t \otimes u > x \otimes y \Rightarrow x\, y]$:

$$\cfrac{\cfrac{\cfrac{\cfrac{}{\lambda z.t \otimes u \xrightarrow{(y\mapsto a)\otimes\bot} ?}}{\lambda z.t \otimes u > x \otimes y \Mapsto x \xrightarrow{y\mapsto a} ?}}{\lambda z.t \otimes u > x \otimes y \Mapsto x\, y \xrightarrow{a} ?}}{[\lambda z.t \otimes u > x \otimes y \Rightarrow x\, y] \xrightarrow{a} ?} \tag{4.65}$$

At the top, the correspondence between u and y is lost. We do not presently know how to fix this.

Adding the action $u \mapsto a$ with u closed destroys the property that actions correspond to atomic paths. Still, we may simply use (3.59) and (4.30) as definitions of a^*, and the proof of correspondence can be extended by taking the size of actions into account in defining the well-founded relation:

$$
\begin{aligned}
|\lambda x.t| &=_{\text{def}} |t| \oplus 1 \\
|t\ u| &=_{\text{def}} |t| \oplus |u| \oplus 1 \\[4pt]
|u \mapsto a| &=_{\text{def}} |u| \oplus |a| \\
|a \otimes \bot| = |\bot \otimes a| = |\beta a| = |\,abs\, a| &=_{\text{def}} |a| \\
|!| &=_{\text{def}} 0 \\[4pt]
|e \mapsto t \xrightarrow{a}\,| &=_{\text{def}} |e| \oplus |t| \oplus |a|
\end{aligned}
\tag{4.66}
$$

4.5 Expressive Power

Subject to the linearity constraints on occurrences of variables, the affine language has much of the expressive power of HOPLA. In particular, the calculi discussed in Section 3.6 can be encoded with the restriction that no variable can occur freely on both sides of a parallel composition. The prefixed sum $\Sigma_{\alpha\in A}\alpha.\mathbb{P}_\alpha$ stands for $\Sigma_{\alpha\in A}(\mathbb{P}_\alpha)_\bot$ in Affine HOPLA. Prefixing $\beta.t$ is still translated into $\beta!t$, but now has a different semantics. For example, by replacing $!(-)$ with $(-)_\bot$, the solution of the equation $\mathbb{P} = \Sigma_{\alpha\in A}\alpha.\mathbb{P}_\alpha$ defining the type of CCS processes becomes isomorphic to the path order A^+ of nonempty strings over the alphabet of CCS actions. Thus, the semantics of CCS given by the translation into Affine HOPLA is essentially a trace semantics. This is illustrated by the fact that the two CCS processes $\mathsf{a}.\mathsf{b}.\varnothing + \mathsf{a}.\mathsf{b}.\varnothing$ and $\mathsf{a}.(\mathsf{b}.\varnothing + \mathsf{c}.\varnothing)$ are given the same semantics by Affine HOPLA, but can be told apart using the nonlinear HOPLA context $C_{\langle \mathsf{a}!\rangle(\langle \mathsf{b}!\rangle\top\wedge\langle \mathsf{c}!\rangle\top)}$ as we saw in Examples 3.1 and 3.22.

4.5.1 Nondeterministic Dataflow

More interestingly, the tensor type of Affine HOPLA allows us to define processes of the kind found in treatments of nondeterministic dataflow, cf. Figure 1.3, something which is not possible using HOPLA. To illustrate, define $\mathbb{P}$ recursively as the prefixed sum $\mathbb{P} = \mathsf{a}.\mathbb{P} + \mathsf{b}.\mathbb{P}$, so that $\mathbb{P}$ essentially consists of streams (or sequences) of a's and b's. We can then define dataflow processes whose properties can be determined from the properties of the denotational semantics—in particular using Proposition 4.4:

- A process *Two* of type $\mathbb{P}\otimes\mathbb{P}$ which produces two identical, parallel streams of a's and b's as output:

$$Two \equiv rec\, z.[z > x \otimes y \Rightarrow (\mathsf{a}.x \otimes \mathsf{a}.y) + (\mathsf{b}.x \otimes \mathsf{b}.y)] \ . \tag{4.67}$$

The denotation of *Two* is the set of pairs (s, s') with s and s' strings of a's and b's, such that s is a prefix of s' or vice versa. Notice the "entanglement" between the two sides of the tensor—choices made on one side affect choice on the other.

- A process *Fork* of type $\mathbb{P} \multimap (\mathbb{P} \otimes \mathbb{P})$ which is like *Two*, except it produces its two output streams as copies of the input stream.

$$\begin{aligned} Fork \equiv rec\, f.\lambda z.[z > \mathsf{a}.z' \Rightarrow [f\ z' > x \otimes y \Rightarrow \mathsf{a}.x \otimes \mathsf{a}.y]] \ + \\ [z > \mathsf{b}.z' \Rightarrow [f\ z' > x \otimes y \Rightarrow \mathsf{b}.x \otimes \mathsf{b}.y]] \ . \end{aligned} \tag{4.68}$$

We have e.g. $[\![Fork\ (\mathsf{a}.\mathsf{b}.\varnothing)]\!] = [\![\mathsf{a}.\mathsf{b}.\varnothing \otimes \mathsf{a}.\mathsf{b}.\varnothing]\!]$ and $[\![Fork\ (\mathsf{a}.\varnothing + \mathsf{b}.\varnothing)]\!] = [\![\mathsf{a}.\varnothing \otimes \mathsf{a}.\varnothing + \mathsf{b}.\varnothing \otimes \mathsf{b}.\varnothing]\!]$, the latter *not* containing "cross terms" like $\mathsf{a}.\varnothing \otimes \mathsf{b}.\varnothing$.

- A process *Merge* of type $(\mathbb{P} \otimes \mathbb{P}) \multimap \mathbb{P}$ which merges two streams into one.

$$\begin{aligned} Merge \equiv rec\, f.\lambda z.[z > x \otimes y \Rightarrow \ & [x > \mathsf{a}.x' \Rightarrow \mathsf{a}.f\ (y \otimes x')] \ + \\ & [x > \mathsf{b}.x' \Rightarrow \mathsf{b}.f\ (y \otimes x')]] \ . \end{aligned} \tag{4.69}$$

We have e.g. $[\![Merge\ (\mathsf{a}.\mathsf{a}.\varnothing \otimes \mathsf{b}.\mathsf{b}.\varnothing)]\!] = [\![\mathsf{a}.\mathsf{b}.\mathsf{a}.\mathsf{b}.\varnothing]\!]$.

Using the operational semantics with the extension of Section 4.4.3 we may "execute" the above processes and get transitions like

$$Fork\ (\mathsf{a}.\mathsf{b}.\varnothing) \xrightarrow{\mathsf{a}\otimes\perp} [Fork\ (\mathsf{b}.\varnothing) > x \otimes y \Rightarrow x \otimes \mathsf{a}.y] \tag{4.70}$$

Note how the term "remembers" that an a-action has been made on the left-hand side of the tensor, ensuring that the other side will behave in the same way.

4.5.2 Extensions

Possible extensions with name-generation and invisible actions as discussed for HOPLA apply to Affine HOPLA as well.

The trace operation to represent dataflow processes with feedback loops is not definable in Affine HOPLA, because then we would have obtained a compositional relational semantics of nondeterministic dataflow with feedback, shown impossible by Brock and Ackerman [13]. However, with a more refined notion of "relation", which spells out the different ways in which input and output of a dataflow process are related, such a semantics is in fact possible [33]. This refinement is obtained by moving to the presheaf version of the affine category.

Part II

Presheaf Semantics

Chapter 5

Domain Theory from Presheaves

Consider the two CCS processes $\mathsf{a}.\varnothing$ and $\mathsf{a}.\mathsf{a}.\varnothing$. Their translations into HOPLA and Affine HOPLA denote the path sets $\{\mathsf{a}\varnothing\}$ and $\{\mathsf{a}\varnothing, \mathsf{a}\{\mathsf{a}\varnothing\}\}$, and so we have $[\![\mathsf{a}.\varnothing]\!] \subseteq [\![\mathsf{a}.\mathsf{a}.\varnothing]\!]$. Because nondeterministic sum is interpreted by union, this means that the processes

$$t \equiv \mathsf{a}.\varnothing + \mathsf{a}.\mathsf{a}.\varnothing \quad \text{and} \quad u \equiv \mathsf{a}.\mathsf{a}.\varnothing \tag{5.1}$$

will be represented as the same path set. Still, t and u are not bisimilar, and so neither of the two path semantics captures enough of the branching structure of processes to characterise bisimilarity.

As noted in the introduction, the domain theory of path sets is a simple version of an earlier and more informative domain theory based on presheaves [19, 20]. Recall from Section 2.2 that path sets $X \subseteq \mathbb{P}$ correspond to monotone functions $\mathbb{P}^{\mathrm{op}} \to \mathbf{2}$, so in modelling a process as a path set we are in effect representing the process by a "characteristic function" from paths to truth values $0 < 1$. If instead of these simple truth values we take sets of realisers, so replacing $\mathbf{2}$ by the category of sets and functions, $\mathbf{Set}$, we obtain functors $\mathbb{P}^{\mathrm{op}} \to \mathbf{Set}$, traditionally called presheaves. Viewed as a process, the presheaf $X : \mathbb{P}^{\mathrm{op}} \to \mathbf{Set}$ associates to a path $p \in \mathbb{P}$ the set Xp whose elements we'll think of as standing for the ways in which the path can be realised by the process X. The process t of (5.1) has two different ways of realising the path $\mathsf{a}\varnothing$ while u has only one, and in this way, a presheaf representation will distinguish between the two processes.

Following the intuition above, a nondeterministic sum of processes represented as presheaves is given using *disjoint* union to keep track of the different ways paths are realised. The sum $\Sigma_{i\in I} X_i$ of presheaves X_i over $\mathbb{P}$ has a contribution $\Sigma_{i\in I} X_i p$, the disjoint union of sets, at $p \in \mathbb{P}$. The empty sum of presheaves is the presheaf $\varnothing$ with empty contribution at each $p \in \mathbb{P}$.

The notion of a presheaf makes sense for any small category $\mathbb{P}$ and this extra generality has been exploited in dealing with independence models. In particular, event structures can be seen as special kinds of presheaves over categories of pomsets [100]. When $\mathbb{P}$ is a category, we'll think of an arrow $e : p \to p'$ in $\mathbb{P}$ as expressing the way in which the path p is extended to the path p', and we'll call $\mathbb{P}$ a *path category.*

5.1 Processes as Presheaves

Consider a presheaf X over A^+, the path order of nonempty sequences over some alphabet $A = \{\mathsf{a}, \mathsf{b}, \ldots\}$. It is a functor $X : (A^+)^{\mathrm{op}} \to \mathbf{Set}$ and so assigns to each string $s \in A^+$ a set, Xs, and to each pair s, s' with s a prefix of s', a function $X(s \leq s') : Xs' \to Xs$ in the opposite direction. This can be understood as follows: when $x' \in Xs'$ the process X is capable of performing s' as realised by x' (or, in the way x'). Therefore, X must intuitively also be capable of performing any shorter sequence than s', and so in particular the sequence s. What the function $X(s \leq s')$ does is simply to map the realiser x' for s' to a realiser $x = X(s \leq s')x'$ for s.

The process t of (5.1) can be represented as a presheaf X as follows. Because of the nondeterministic sum, there are two ways in which t may perform the sequence a and one way in which t may perform the sequence aa. Thus, X will map a to a two-element set, say $\{1, 2\}$, and aa to a singleton, say $\{1\}$. No other sequences are possible, so all other elements of A^+ are mapped to the empty set. Just one of the ways of performing a, say 2, may lead to performing the longer sequence aa, and so $X(\mathsf{a} \leq \mathsf{aa}) : X\mathsf{aa} \to X\mathsf{a}$ will map 1 to 2. Likewise, the process u is represented by a presheaf Y mapping a and aa to a singleton, see Figure 5.1. So using presheaves, the representations of t and u become different because the presheaves keep track of nondeterministic branching. This can be further illustrated by noticing that presheaves over A^+ are in bijective correspondence with synchronisation trees over the underlying alphabet A. Indeed, imagine adding the empty sequence ϵ to A^+, forming the path order A^* with least element ϵ. If we agree that any process X can perform the empty sequence in exactly one way, so that $X\epsilon = \{1\}$, we add "roots" to Figure 5.1, and by further turning the branches around, we obtain Figure 5.2. Generally, presheaves X over A^* correspond to sets of such trees, or a "forest", with the set of roots given by $X\epsilon$.

Adding a root to a presheaf amounts to a lifting construction analogous to what we did in Section 2.3.2. Given a path category $\mathbb{P}$, we may add a new initial object to obtain the path category $\mathbb{P}_\perp$. Associated to this construction is an operation taking a presheaf X over $\mathbb{P}$ to $\lfloor X \rfloor$ over $\mathbb{P}_\perp$, such that $\lfloor X \rfloor \perp$ is a singleton and $\lfloor X \rfloor \lfloor p \rfloor = Xp$. Presheaves obtained, to within isomorphism, as images of $\lfloor - \rfloor$ are called *rooted.*

Proposition 5.1 Any presheaf X over $\mathbb{P}_\perp$ has a decomposition as a sum of rooted presheaves $X \cong \Sigma_{i \in X\perp} \lfloor X_i \rfloor$, where, for $i \in X\perp$, the presheaf X_i over $\mathbb{P}$ is, to within isomorphism, given as $X_i p = \{x \in Xp : (X\perp_p)x = i\}$ where $\perp_p$ is the unique arrow $\perp \to p$ in $\mathbb{P}_\perp$.

For a general presheaf over A^*, the decomposition into rooted components exhibits the corresponding forest as a set of trees.

The construction of a tree from a presheaf can also be generalised to arbitrary presheaves. The general construction gives rise to a categorical version of a transition system, called the *category of elements* of the presheaf.

Let $\mathbb{P}$ be a path category and $X : \mathbb{P}^{\mathrm{op}} \to \mathbf{Set}$ a presheaf over $\mathbb{P}$. The *category of elements* of X, written $\mathit{elts}\, X$, has as objects pairs (p, x) with $p \in \mathbb{P}$ and $x \in Xp$ and morphisms of the form $e : (p, x) \to (p', x')$ where $e : p \to p'$ is an arrow of $\mathbb{P}$ such that $(Xe)x' = x$.

In the case of rooted presheaves over A^*, the synchronisation trees obtained as above become categories of elements when closed under identity maps and compositions of edges. Figure 5.3 shows the category of elements of our two presheaves X and Y from above. The identity maps are left out for clarity.

5.2 Presheaf Categories

The presheaves over a path category $\mathbb{P}$ are the objects of the functor category $\widehat{\mathbb{P}} = [\mathbb{P}^{\mathrm{op}}, \mathbf{Set}]$, with arrows being natural transformations. Spelled out, a natural transformation $f : X \to Y$ between presheaves X and Y over $\mathbb{P}$ is a family $(f_p)_{p \in \mathbb{P}}$ of functions $f_p : Xp \to Yp$, satisfying that for any arrow $e : p \to p'$ of $\mathbb{P}$, the square below commutes.

$$
\begin{tikzcd}
p' & Xp' \arrow[r, "f_{p'}"] \arrow[d, "Xe"'] & Yp' \arrow[d, "Ye"] \\
p \arrow[u, "e"] & Xp \arrow[r, "f_p"'] & Yp
\end{tikzcd}
\tag{5.2}
$$

It is perhaps not entirely obvious what process concept this amounts to, so we replay it in terms of categories of elements. Since each f_p is a function $Xp \to Yp$, the natural transformation f induces a map $\mathit{elts}\, f$ between the objects of $\mathit{elts}\, X$ and $\mathit{elts}\, Y$, sending (p, x) to $(p, f_p x)$. Now, naturality of f is simply the same as functoriality of $\mathit{elts}\, f$: that it sends morphisms $(p, x) \xrightarrow{e} (p', x')$ of $\mathit{elts}\, X$ to morphisms $(p, f_p x) \xrightarrow{e} (p', f_{p'} x')$ of $\mathit{elts}\, Y$.

As an example, we can define a natural transformation $f : X \to Y$ between the presheaves given in Figure 5.3, see Figure 5.4. In the figure, $(p, x) \overset{f_p}{\cdots\cdots} (p, y)$ means that $f_p x = y$, or equivalently that $\mathit{elts}\, f$ maps (p, x) to (p, y). Since f preserves transitions in the categories of elements, we can think of it as a functional simulation relation on transition systems.

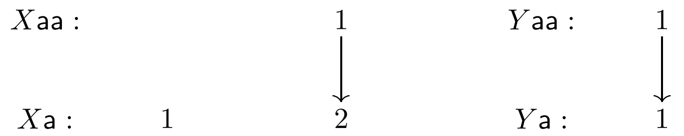

Figure 5.1: Presheaves X and Y

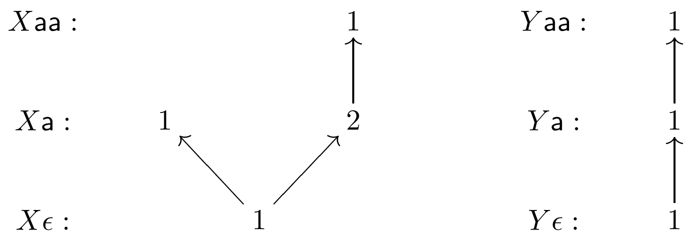

Figure 5.2: Presheaves X and Y as synchronisation trees

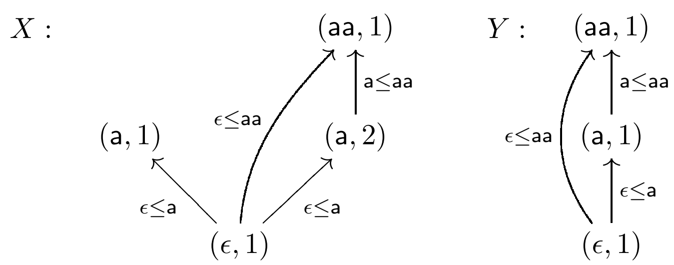

Figure 5.3: Presheaves X and Y as categories of elements

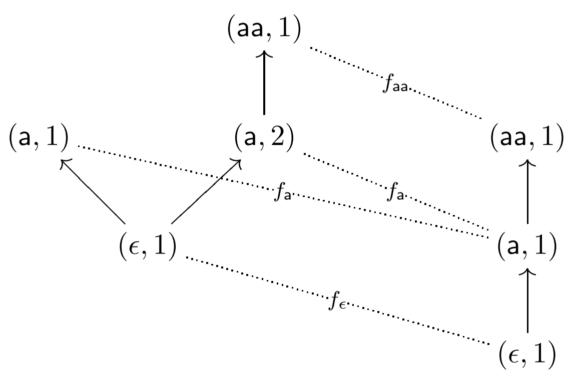

Figure 5.4: Presheaves X and Y and a natural transformation

A presheaf category $\widehat{\mathbb{P}}$ has all limits and colimits given pointwise, at a particular object, by the corresponding limits or colimits of sets. This exposes the nondeterministic sum $\Sigma_{i\in I}X_i$ above as the coproduct of the presheaves X_i, with $\varnothing$ the empty coproduct. Colimits being generalised joins, it is thus reasonable to view presheaf categories as generalised nondeterministic domains [30, 16, 98].

To each presheaf category $\widehat{\mathbb{P}}$ is associated a canonical functor $\mathrm{y}_{\mathbb{P}} : \mathbb{P} \to \widehat{\mathbb{P}}$, standardly called the *Yoneda functor*, saying how to view paths as processes:

$$\mathrm{y}_{\mathbb{P}}p = \mathbb{P}(-, p) \tag{5.3}$$

Images under Yoneda, called *representables*, provide a straightforward generalisation of the notion of primes of the nondeterministic domains of Chapter 2. Intuitively, $\mathrm{y}_{\mathbb{P}}$ maps a computation path p to a process that may do a single computation of shape p, and a path extension $e : p \to p'$ to a simulation of this computation by a longer one. Following this intuition, a natural transformation $\mathrm{y}_{\mathbb{P}}p \to X$ shows how the process X may simulate a process capable of performing just the path p. But then the set Xp should intuitively be the same as the set of such natural transformations. This is the content of

Lemma 5.2 (Yoneda) $\widehat{\mathbb{P}}(\mathrm{y}_{\mathbb{P}}p, X) \cong Xp$, naturally in X and p.

This classic result directly generalises the equivalence $\mathrm{y}_{\mathbb{P}}p \subseteq X$ iff $p \in X$ for path sets. One immediate consequence is that the Yoneda functor is full and faithful, allowing us to view $\mathbb{P}$ as essentially a subcategory of $\widehat{\mathbb{P}}$.

The situation $\mathrm{y}_{\mathbb{P}} : \mathbb{P} \hookrightarrow \widehat{\mathbb{P}}$ is a canonical example of the situation $\mathbb{P} \hookrightarrow \mathbf{M}$ that led Joyal, Nielsen, and Winskel to open-map bisimulation [42]. Using the Yoneda lemma, a map $f : X \to Y$ is open iff whenever we have the situation on the left below, we may "complete the square" as on the right:

$$\begin{tikzcd} & (p', y') \\ (p, x) \arrow[r, dotted, "f_p"'] & (p, y) \arrow[u, "e"'] \end{tikzcd} \qquad \begin{tikzcd} (p', x') \arrow[r, dotted, "f_{p'}"] & (p', y') \\ (p, x) \arrow[u, "e"] \arrow[r, dotted, "f_p"'] & (p, y) \arrow[u, "e"'] \end{tikzcd} \tag{5.4}$$

With the transition system intuition from above, this just requires f to reflect as well as preserve transitions, as expected.

The analogue of (2.6) saying that any path set is a union of primes below it is that any presheaf is a colimit of representables:

$$X \cong \int^{(p,x)\in elts\, X} \mathrm{y}_{\mathbb{P}}p \ . \tag{5.5}$$

The role of the colimit is to "glue together" the paths of X as dictated by the category of elements of X. The counterpart of the freeness property (2.7) is that $\widehat{\mathbb{P}}$ is the free colimit-completion of $\mathbb{P}$. So for any functor $f : \mathbb{P} \to \mathbf{C}$, where $\mathbf{C}$ is a category with all colimits, there is a colimit-preserving functor $f^\dagger : \widehat{\mathbb{P}} \to \mathbf{C}$, determined to within isomorphism, such that $f \cong f^\dagger \circ \mathrm{y}_{\mathbb{P}}$—see e.g. [50], page 43:

$$\begin{tikzcd} \mathbb{P} \arrow[r, "\mathrm{y}_{\mathbb{P}}"] \arrow[dr, "f"'] \arrow[dr, phantom, shift left=3, "\cong"] & \widehat{\mathbb{P}} \arrow[d, dotted, "f^\dagger"] \\ & \mathbf{C} \end{tikzcd} \qquad f^\dagger X = \int^{(p,x)\in elts\, X} fp \ . \tag{5.6}$$

This suggests considering colimit-preserving functors $f : \widehat{\mathbb{P}} \to \widehat{\mathbb{Q}}$ between presheaf categories. By the freeness property, such functors correspond to within isomorphism to functors $\mathbb{P} \to \widehat{\mathbb{Q}}$. We thus have the chain of equivalences:

$$[\mathbb{P}, \widehat{\mathbb{Q}}] = [\mathbb{P}, [\mathbb{Q}^{\mathrm{op}}, \mathbf{Set}]] \cong [\mathbb{P} \times \mathbb{Q}^{\mathrm{op}}, \mathbf{Set}] = \widehat{\mathbb{P}^{\mathrm{op}} \times \mathbb{Q}} \ . \tag{5.7}$$

Once again, the path category $\mathbb{P}^{\mathrm{op}} \times \mathbb{Q}$ provides a function space.

5.3 Linear and Nonlinear Maps

We obtain 2-categories **Lin**, **Cts**, and **Aff** intuitively in the same way as the corresponding categories were obtained in Section 2.3, giving rise to a 2-categorical model of linear logic, and via pseudo-comonads, to models of intuitionistic logic and affine-linear logic, respectively. We give only a brief overview below; details can be found in [20].

Write **Lin** for the 2-category with objects path categories $\mathbb{P}, \mathbb{Q}, \ldots$, arrows colimit-preserving functors between the associated presheaf categories, and natural transformations as two-cells. The equivalence of categories

$$\mathbf{Lin}(\mathbb{P}, \mathbb{Q}) \simeq [\mathbb{P} \times \mathbb{Q}^{\mathrm{op}}, \mathbf{Set}] \tag{5.8}$$

provides a relational exposition of maps of **Lin** (with paths of $\mathbb{P}$ and $\mathbb{Q}$ related in a set of ways, rather than just related or not related, as in the path set case). Again, this exposes $(-)^{\mathrm{op}}$ as an involution of linear logic. It will be useful in Chapter 7 below to use functors $\mathbb{P} \times \mathbb{Q}^{\mathrm{op}} \to \mathbf{Set}$ to stand for maps of **Lin**. Such functors are called *profunctors* (or bimodules, or distributors, see [9] for an elementary introduction). The bicategory of profunctors, **Prof**, is biequivalent to **Lin** via the equivalence above. We'll write $f : \mathbb{P} \to\!\!\!\!\!+ \mathbb{Q}$ when f is a profunctor $\mathbb{P} \times \mathbb{Q}^{\mathrm{op}} \to \mathbf{Set}$. The composition of profunctors $f : \mathbb{P} \to\!\!\!\!\!+ \mathbb{Q}$ and $g : \mathbb{Q} \to\!\!\!\!\!+ \mathbb{R}$ is obtained using the coend formula

$$(g \circ f)(p, r) = \int^{q \in \mathbb{Q}} f(p, q) \times g(q, r) \ . \tag{5.9}$$

Intuitively, the role of the coend is to abstract away from the $q \in \mathbb{Q}$ used in the communication between processes f and g.

An analogue of the exponential ! can be obtained by taking $!\mathbb{P}$ to be the free finite-colimit completion of $\mathbb{P}$. It can then be shown that $\widehat{\mathbb{P}}$ with the inclusion functor $i_{\mathbb{P}} : !\mathbb{P} \to \widehat{\mathbb{P}}$ is the free filtered-colimit completion of $!\mathbb{P}$—see [44]. It follows that maps $!\mathbb{P} \to \mathbb{Q}$ in **Lin** correspond, to within isomorphism, to filtered-colimit preserving (i.e. continuous) functors $\widehat{\mathbb{P}} \to \widehat{\mathbb{Q}}$. The operation ! extends to a 2-functor $! : \mathbf{Cts} \to \mathbf{Lin}$ which is pseudo-left adjoint to the inclusion $\mathbf{Lin} \hookrightarrow \mathbf{Cts}$ of 2-categories. The unit of the adjunction is given by $\eta_{\mathbb{P}} X = \widehat{\mathbb{P}}(i_{\mathbb{P}}-, X)$.

Likewise, $\widehat{\mathbb{P}}$ with the inclusion $j_{\mathbb{P}} : \mathbb{P}_\perp \to \widehat{\mathbb{P}}$ sending $\perp$ to $\varnothing$ and acting as $\mathrm{y}_{\mathbb{P}}$ elsewhere, is the free connected-colimit completion of $\mathbb{P}_\perp$. So maps $\mathbb{P}_\perp \to \mathbb{Q}$ in **Lin** correspond, to within isomorphism, to connected-colimit preserving (i.e. affine) functors $\widehat{\mathbb{P}} \to \widehat{\mathbb{Q}}$. Lifting extends to a 2-functor $(-)_\perp : \mathbf{Aff} \to \mathbf{Lin}$; for $g : \mathbb{P} \to \mathbb{Q}$ in **Aff**, the functor $g_\perp : \mathbb{P}_\perp \to \mathbb{Q}_\perp$ in **Lin** takes $X \in \widehat{\mathbb{P}_\perp}$ with decomposition $\Sigma_{i \in X\perp} \lfloor X_i \rfloor$ to $f_\perp(X) =_{\mathrm{def}} \Sigma_{i \in X\perp} \lfloor f(X_i) \rfloor$, cf. Proposition 5.1. This functor is a pseudo-left adjoint to the inclusion $\mathbf{Lin} \hookrightarrow \mathbf{Aff}$ with unit given by the operation $\lfloor - \rfloor$ from Section 5.1.

Maps $\mathbb{P} \to \mathbb{Q}$ of **Aff** can be represented by profunctors $f : \mathbb{P}_\perp \nrightarrow \mathbb{Q}$ and composition in **Aff** is mirrored by composition of such profunctors. A useful way of defining this composition is as follows. By currying we obtain a functor $\mathit{curry}\, f : \mathbb{P}_\perp \to \widehat{\mathbb{Q}}$ and, using lifting, a profunctor $\mathbb{P}_\perp \nrightarrow \mathbb{Q}_\perp$ sending (P, Q) to $\lfloor (\mathit{curry}\, f) P \rfloor Q$. Now, given also $g : \mathbb{Q}_\perp \nrightarrow \mathbb{R}$, the composition $g \circ f : \mathbb{P}_\perp \nrightarrow \mathbb{R}$ can be obtained as the composition $g \circ (\lfloor (\mathit{curry}\, f) + \rfloor -)$ of profunctors using (5.9):

$$(g \circ f)(P, r) = \int^{Q \in \mathbb{Q}_\perp} \lfloor (\mathit{curry}\, f) P \rfloor Q \times g(Q, r) \ . \tag{5.10}$$

From the above we see that the categorical situation using presheaves is the same as for path sets and so the universal constructions and their properties carry over, only we need to replace straight equality by isomorphism. For Affine HOPLA, the interpretation of types as posets of paths is the same as in the path set case, while for HOPLA itself, we need to move to path categories, as $!\mathbb{P}$ will generally be a category even if $\mathbb{P}$ is just a poset.[1] Solutions to recursive type equations can then be obtained as in [18], characterising solutions up to isomorphism. In fact, our type constructors preserve inclusions of categories in each argument, and so recursive types can be interpreted as limits $\bigcup_n F^n(\varnothing)$ of ω-chains of inclusions

$$\varnothing \subseteq F(\varnothing) \subseteq \cdots \subseteq F^n(\varnothing) \subseteq \cdots \ . \tag{5.11}$$

This makes the interpretation of recursive types identical to that of their unfoldings, and so we may elide the use of the *abs* and *rep* isomorphisms. We'll exploit this in the next chapter.

[1]The operator $!(-)$ preserves the property of being *essentially small*, i.e. equivalent to a category whose objects form a set. This is sufficient to ensure that $\widehat{!\mathbb{P}}$ will always be a category.

With one exception, all the properties of Chapter 2 and Sections 3.2 and 4.2 can be shown to hold up to isomorphism in the presheaf setting. The exception is the equation

$$[\![[\Sigma_{i\in I} u_i > x \otimes y \Rightarrow t]]\!] = [\![\Sigma_{i\in I}[u_i > x \otimes y \Rightarrow t]]\!] \quad \text{if } I \neq \varnothing, \tag{5.12}$$

from Section 4.2. It fails because tensor match is affine in the matched term, but "affine" now means preserving *connected* colimits, not just nonempty joins, and sums (coproducts) are manifestly not connected.

This difference in the notion of affine when moving from path sets to presheaves is central to the distinguishing power of the presheaf semantics. The interpretation of the term $\mathsf{a}.x$ is an affine map, and so allowing affine maps to preserve nonempty joins validates both

$$\begin{aligned} \mathsf{a}.(\mathsf{b}.\varnothing + \mathsf{c}.\varnothing) &= \mathsf{a}.\mathsf{b}.\varnothing + \mathsf{a}.\mathsf{c}.\varnothing \quad \text{and} \\ \mathsf{a}.\mathsf{a}.\varnothing = \mathsf{a}.(\varnothing + \mathsf{a}.\varnothing) &= \mathsf{a}.\varnothing + \mathsf{a}.\mathsf{a}.\varnothing \ . \end{aligned} \tag{5.13}$$

The presheaf semantics of Affine HOPLA distinguishes the terms in both cases. Prefixing is interpreted by the lifting construct $\lfloor - \rfloor$ of Section 5.1, and as indicated there, it is analogous to adding a root to a synchronisation forest, forming a tree. Thus, the presheaf semantics of Affine HOPLA interprets these processes as the presheaves corresponding to their usual synchronisation tree representations (Figures 1.2 and 5.2). It follows immediately that the presheaf semantics of HOPLA and Affine HOPLA are not fully abstract with respect to contextual equivalence, because by the full abstraction results of Part I, there are no contexts distinguishing between the terms $\mathsf{a}.\varnothing + \mathsf{a}.\mathsf{a}.\varnothing$ and $\mathsf{a}.\mathsf{a}.\varnothing$.

Chapter 6

Strong Correspondence

Despite the correspondence results (Theorems 3.20 and 4.13), the path semantics does not really explain the operational semantics of HOPLA and Affine HOPLA. Indeed, because the processes t and u of (5.1) are equated in the path semantics, these results would still hold if the operational semantics had an extra rule like e.g.

$$\frac{}{\mathsf{a.a.a.}\varnothing \xrightarrow{\mathsf{a!}} \mathsf{a.}\varnothing + \mathsf{a.a.}\varnothing} \tag{6.1}$$

—although this is "clearly wrong". This chapter provides an explanation for the operational rules of Chapters 3 and 4 by relating the realisers of presheaf denotations to derivations in the operational semantics.

For terms $\vdash t : \mathbb{P}$ and actions $\mathbb{P} : a : \mathbb{P}'$ the correspondence results state the equality

$$\Sigma_{t'}[\![!t']\!] = a^*[\![t]\!] \tag{6.2}$$

—where t' ranges terms such that $\mathbb{P} : t \xrightarrow{a} t' : \mathbb{P}'$. A stronger correspondence can be obtained for the presheaf semantics by replacing the sum $\Sigma_{t'}[\![!t']\!]$ over successor terms by the sum $\Sigma_d[\![!t_d]\!]$ over *derivations* d with conclusion $\mathbb{P} : t \xrightarrow{a} t_d : \mathbb{P}'$. We also need to replace equality by isomorphism,

$$a^*[\![t]\!] \cong \Sigma_d[\![!t_d]\!] \ . \tag{6.3}$$

The operational rules are designed with this "strong correspondence" in mind [64, 65]. Even by itself, the rule (6.1) would violate strong correspondence because

$$(\mathsf{a!})^*[\![\mathsf{a.a.a.}\varnothing]\!] \cong [\![!\mathsf{a.a.}\varnothing]\!] \not\cong [\![!(\mathsf{a.}\varnothing + \mathsf{a.a.}\varnothing)]\!] \ . \tag{6.4}$$

In Section 6.1 we prove the result for finitary HOPLA (leaving out recursive types and recursive process definitions). We have not yet succeeded in extending the proof to full HOPLA. Work in progress is outlined in Section 6.2. Strong correspondence for Affine HOPLA is much easier, see Section 6.3.

6.1 Finitary HOPLA

Without recursive types we can give a simple proof of strong correspondence for HOPLA. The proof can easily be extended to handle recursively defined processes, but since these are not very interesting without recursive types, we leave out recursion on terms as well as on types. The proof uses logical predicates $A_{\mathbb{P}}$ on closed terms of type $\mathbb{P}$, defined by structural induction on types:

$$\begin{aligned} A_{\mathbb{P}\to\mathbb{Q}}(t) &\iff_{\text{def}} \forall u.\ (A_{\mathbb{P}}(u) \implies A_{\mathbb{Q}}(t\ u)) \\ A_{\Sigma_{\alpha\in A}\mathbb{P}_\alpha}(t) &\iff_{\text{def}} \forall \beta \in A.\ A_{\mathbb{P}_\beta}(\pi_\beta t) \\ A_{!\mathbb{P}}(t) &\iff_{\text{def}} \begin{cases} (\llbracket t \rrbracket \cong \Sigma_d \llbracket !t_d \rrbracket) \text{ and} \\ (!\mathbb{P} : t \xrightarrow{!} t' : \mathbb{P} \implies A_{\mathbb{P}}(t')) \end{cases} \end{aligned} \tag{6.5}$$

—where in the sum above d ranges over derivations of $!\mathbb{P} : t \xrightarrow{!} t_d : \mathbb{P}$. The logical predicates extend to actions as follows:

$$\frac{A_{\mathbb{P}}(u) \quad A(\mathbb{Q} : a : \mathbb{P}')}{A(\mathbb{P} \to \mathbb{Q} : u \mapsto a : \mathbb{P}')} \quad \frac{A(\mathbb{P}_\beta : a : \mathbb{P}') \quad \beta \in A}{A(\Sigma_{\alpha\in A}\mathbb{P}_\alpha : \beta a : \mathbb{P}')}$$
$$\frac{}{A(!\mathbb{P} : ! : \mathbb{P})} \quad \frac{A(\mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}] : a : \mathbb{P}')}{A(\mu_j\vec{T}.\vec{\mathbb{T}} : abs\, a : \mathbb{P}')} \tag{6.6}$$

By structural induction on types we have

Lemma 6.1 Suppose $\vdash t : \mathbb{P}$. Then $A_{\mathbb{P}}(t)$ iff for all actions $\mathbb{P} : a : \mathbb{P}'$ with $A(\mathbb{P} : a : \mathbb{P}')$ we have $A_{!\mathbb{P}'}(a^* t)$.

Recall also Lemma 3.13 which says $!\mathbb{P}' : a^*t \xrightarrow{!} t' : \mathbb{P}'$ iff $\mathbb{P} : t \xrightarrow{a} t' : \mathbb{P}'$. In fact, such derivations are in bijective correspondence, and we'll use this fact freely below. In addition, we'll make use of presheaf versions of Propositions 3.6, 3.7, and 3.8 saying that, in the presheaf semantics, we have isomorphisms

$$\llbracket (\lambda x.t)\ u \rrbracket \cong \llbracket t[u/x] \rrbracket \tag{6.7}$$
$$\llbracket \pi_\beta(\beta t) \rrbracket \cong \llbracket t \rrbracket \tag{6.8}$$
$$\llbracket [!u > !x \Rightarrow t] \rrbracket \cong \llbracket t[u/x] \rrbracket \tag{6.9}$$

Lemma 6.2 (Main Lemma) For all terms $\vdash t : \mathbb{P}$ of finitary HOPLA we have $A_{\mathbb{P}}(t)$.

Proof. By structural induction on terms using the induction hypothesis

> Suppose $x_1 : \mathbb{P}_1, \ldots, x_k : \mathbb{P}_k \vdash t : \mathbb{P}$ and let $\vdash s_j : \mathbb{P}_j$ with $A_{\mathbb{P}_j}(s_j)$ for $1 \le j \le k$. Then $A_{\mathbb{P}}(t[s_1/x_1, \ldots, s_k/x_k])$.

We'll abbreviate $x_1 : \mathbb{P}_1, \ldots, x_k : \mathbb{P}_k$ to Γ and $[s_1/x_1, \ldots, s_k/x_k]$ to $[s]$.

Variable. Immediate.

Nondeterministic sum. Let $\Gamma \vdash \Sigma_{i\in I} t_i : \mathbb{P}$ and $\vdash s_j : \mathbb{P}_j$ with $A_{\mathbb{P}_j}(s_j)$ for $1 \leq j \leq k$. Using Lemma 6.1 we must show that whenever $A(\mathbb{P} : a : \mathbb{P}')$, we have $A_{!\mathbb{P}'}(a^* \Sigma_{i\in I} t_i[s])$. Letting d range over derivations $\mathbb{P} : \Sigma_{i\in I} t_i[s] \xrightarrow{a} t_d : \mathbb{P}'$ and d_i over derivations $\mathbb{P} : t_i[s] \xrightarrow{a} t_{d_i} : \mathbb{P}'$ for each $i \in I$, we have $\Sigma_d [\![!t_d]\!] \cong \Sigma_{i\in I} \Sigma_{d_i} [\![!t_{d_i}]\!]$ by the operational rules, and hence:

$$\begin{aligned}
&[\![a^* \Sigma_{i\in I} t_i[s]]\!] \\
&\cong a^* \Sigma_{i\in I} [\![t_i[s]]\!] \\
&\cong \Sigma_{i\in I} [\![a^* t_i[s]]\!] \quad \text{(linearity of } a^*) \\
&\cong \Sigma_{i\in I} \Sigma_{d_i} [\![!t_{d_i}]\!] \quad \text{(ind. hyp.)} \\
&\cong \Sigma_d [\![!t_d]\!]
\end{aligned} \tag{6.10}$$

Further, if $\mathbb{P} : \Sigma_{i\in I} t_i[s] \xrightarrow{a} t' : \mathbb{P}'$, then for some $j \in I$ we have $\mathbb{P} : t_j[s] \xrightarrow{a} t' : \mathbb{P}'$ and so by the induction hypothesis, $A_{\mathbb{P}'}(t')$ as wanted.

Abstraction. Let $\Gamma \vdash \lambda x.t : \mathbb{P} \to \mathbb{Q}$ and $\vdash s_j : \mathbb{P}_j$ with $A_{\mathbb{P}_j}(s_j)$ for $1 \leq j \leq k$. Using Lemma 6.1 we must show that whenever $A(\mathbb{P} \to \mathbb{Q} : u \mapsto a : \mathbb{P}')$—so in particular $A_{\mathbb{P}}(u)$—we have $A_{!\mathbb{P}'}((u \mapsto a)^*(\lambda x.t[s]))$. Letting d range over derivations $\mathbb{P} \to \mathbb{Q} : \lambda x.t[s] \xrightarrow{u \mapsto a} t_d : \mathbb{P}'$ and d' over derivations $\mathbb{P} : t[s][u/x] \xrightarrow{a} t_{d'} : \mathbb{P}'$, we have $\Sigma_d [\![!t_d]\!] \cong \Sigma_{d'} [\![!t_{d'}]\!]$ by the operational rules, and hence:

$$\begin{aligned}
&[\![(u \mapsto a)^*(\lambda x.t[s])]\!] \\
&\cong [\![a^*(\lambda x.t[s]\ u)]\!] \quad \text{(def. of } (u \mapsto a)^*) \\
&\cong [\![a^*(t[s][u/x])]\!] \quad \text{(by (6.7))} \\
&\cong \Sigma_{d'} [\![!t_{d'}]\!] \quad \text{(ind. hyp.)} \\
&\cong \Sigma_d [\![!t_d]\!]
\end{aligned} \tag{6.11}$$

Further, if $\mathbb{P} \to \mathbb{Q} : \lambda x.t[s] \xrightarrow{u \mapsto a} t' : \mathbb{P}'$, then $\mathbb{Q} : t[s][u/x] \xrightarrow{a} t' : \mathbb{P}'$ and so by the induction hypothesis, $A_{\mathbb{P}'}(t')$ as wanted.

Application. Let $\Gamma \vdash t\ u : \mathbb{Q}$ and $\vdash s_j : \mathbb{P}_j$ with $A_{\mathbb{P}_j}(s_j)$ for $1 \leq j \leq k$. By the induction hypothesis, $A_{\mathbb{P}\to\mathbb{Q}}(t[s])$ and $A_{\mathbb{P}}(u[s])$ for some $\mathbb{P}$. The definition of $A_{\mathbb{P}\to\mathbb{Q}}$ yields $A_{\mathbb{Q}}(t[s]\ u[s])$ as wanted.

Injection. Let $\Gamma \vdash \beta t : \Sigma_{\alpha\in A} \mathbb{P}_\alpha$ and $\vdash s_j : \mathbb{P}_j$ with $A_{\mathbb{P}_j}(s_j)$ for $1 \leq j \leq k$. Using Lemma 6.1 we must show that whenever $A(\Sigma_{\alpha\in A} \mathbb{P}_\alpha : \beta a : \mathbb{P}')$ we have $A_{!\mathbb{P}'}((\beta a)^*(\beta t[s]))$. Letting d range over derivations $\Sigma_{\alpha\in A} \mathbb{P}_\alpha : \beta t[s] \xrightarrow{\beta a} t_d : \mathbb{P}'$ and d' over derivations $\mathbb{P}_\beta : t \xrightarrow{a} t_{d'} : \mathbb{P}'$, we have $\Sigma_d [\![!t_d]\!] \cong \Sigma_{d'} [\![!t_{d'}]\!]$ by the operational rules, and hence:

$$\begin{aligned}
&[\![(\beta a)^*(\beta t[s])]\!] \\
&\cong [\![a^*(\pi_\beta(\beta t[s]))]\!] \quad \text{(def. of } (\beta a)^*) \\
&\cong [\![a^*(t[s])]\!] \quad \text{(by (6.8))} \\
&\cong \Sigma_{d'} [\![!t_{d'}]\!] \quad \text{(ind. hyp.)} \\
&\cong \Sigma_d [\![!t_d]\!]
\end{aligned} \tag{6.12}$$

Further, if $\Sigma_{\alpha\in A}\mathbb{P}_\alpha : \beta t[s] \xrightarrow{\beta a} t' : \mathbb{P}'$, then $\mathbb{P}_\beta : t[s] \xrightarrow{a} t' : \mathbb{P}'$ and so by the induction hypothesis, $A_{\mathbb{P}'}(t')$ as wanted.

Projection. Let $\Gamma \vdash \pi_\beta t : \mathbb{P}_\beta$ and $\vdash s_j : \mathbb{P}_j$ with $A_{\mathbb{P}_j}(s_j)$ for $1 \leq j \leq k$. By the induction hypothesis, $A_{\Sigma_{\alpha\in A}\mathbb{P}_\alpha}(t)$ and so by definition, $A_{\mathbb{P}_\beta}(\pi_\beta t)$ as wanted.

Prefixing. Let $\Gamma \vdash !t : !\mathbb{P}$ and $\vdash s_j : \mathbb{P}_j$ with $A_{\mathbb{P}_j}(s_j)$ for $1 \leq j \leq k$. There is just one derivation $!\mathbb{P} : !t[s] \xrightarrow{!} t[s] : \mathbb{P}$ and we have the identity $[\![!^*(!t[s])]\!] = [\![!t[s]]\!]$ by definition, and $A_\mathbb{P}(t[s])$ by the induction hypothesis.

Prefix match Let $\Gamma \vdash [u > !x \Rightarrow t] : \mathbb{Q}$ and $\vdash s_j : \mathbb{P}_j$ with $A_{\mathbb{P}_j}(s_j)$ for $1 \leq j \leq k$. Using Lemma 6.1, and renaming x if necessary to make it distinct from the x_j, we must show that whenever $A(\mathbb{Q} : a : \mathbb{P}')$ we have $A_{!\mathbb{P}'}(a^*[u[s] > !x \Rightarrow t[s]])$. The induction hypothesis for u yields $[\![!^*u[s]]\!] = [\![u[s]]\!] \cong \Sigma_{d'}[\![!u'_{d'}]\!]$ and $A_\mathbb{P}(u'_{d'})$ for derivations d' of $!\mathbb{P} : u[s] \xrightarrow{!} u'_{d'} : \mathbb{P}$. Letting d range over derivations $\mathbb{Q} : [u[s] > !x \Rightarrow t[s]] \xrightarrow{a} t_d : \mathbb{P}'$ and $d_{d'}$ over derivations $\mathbb{Q} : t[s][u'_{d'}/x] \xrightarrow{a} t_{d_{d'}} : \mathbb{P}'$, we have $\Sigma_d[\![!t_d]\!] \cong \Sigma_{d'}\Sigma_{d_{d'}}[\![!t_{d_{d'}}]\!]$ by the operational rules and hence,

$$
\begin{array}{ll}
[\![a^*[u[s] > !x \Rightarrow t[s]]]\!] & \\
\cong [\![a^*[\Sigma_{d'}!u'_{d'} > !x \Rightarrow t[s]]]\!] & \text{(compositionality)} \\
\cong a^*\Sigma_{d'}[\![t[s][u'_{d'}/x]]\!] & \text{(by (6.9))} \\
\cong \Sigma_{d'}[\![a^*(t[s][u'_{d'}/x])]\!] & \text{(linearity of } a^*) \\
\cong \Sigma_{d'}\Sigma_{d_{d'}}[\![!t_{d_{d'}}]\!] & \text{(ind. hyp.)} \\
\cong \Sigma_d[\![!t_d]\!] &
\end{array}
\tag{6.13}
$$

Further, if $\mathbb{Q} : [u[s] > !x \Rightarrow t[s]] \xrightarrow{a} t' : \mathbb{P}'$, then $!\mathbb{P} : u[s] \xrightarrow{!} u' : \mathbb{P}$ for some u' such that $A_\mathbb{P}(u')$ by the induction hypothesis for u, and then $\mathbb{Q} : t[s][u'/x] \xrightarrow{a} t' : \mathbb{P}'$ with $A_\mathbb{Q}(t')$ by the induction hypothesis for t, as wanted.

The structural induction is complete. □

Theorem 6.3 (Strong Correspondence) Suppose $\vdash t : \mathbb{P}$ and $\mathbb{P} : a : \mathbb{P}'$ in finite HOPLA. Then $a^*[\![t]\!] \cong \Sigma_d[\![!t_d]\!]$ where d ranges over derivations of $\mathbb{P} : t \xrightarrow{a} t_d : \mathbb{P}'$.

Proof. By the main lemma, we have $A_{!\mathbb{P}'}(a^*t)$. Hence, $[\![a^*t]\!] \cong \Sigma_{d'}[\![!t_{d'}]\!]$ where d' ranges over derivations of $!\mathbb{P}' : a^*t \xrightarrow{!} t_{d'} : \mathbb{P}'$. The result then follows by Lemma 3.13. □

6.2 Full HOPLA

In extending the above result to the full language, in particular including recursive types, it would be reasonable to try to imitate the use of logical relations in Section 3.4.1. The relation $X \subseteq_\mathbb{P} t$ between a path set $X \subseteq \mathbb{P}$ and a closed term $\vdash t : \mathbb{P}$ was understood intuitively as saying that all paths p in X are operationally realised by t, so $p \in_{\mathbb{P}} t$. Generalising this to presheaves, we are led to say not only whether or not a path may be realised, but to give the set of ways it may be realised, i.e. we are led to turn the logical relations $(- \in_{\mathbb{P}} t)$ into presheaves $\widehat{t} \in \widehat{\mathbb{P}}$, and then replace $X \subseteq_{\mathbb{P}} t$ by $\widehat{\mathbb{P}}(X, \widehat{t})$. By a straightforward realisability interpretation of (3.70), we would obtain

$$\begin{aligned} \widehat{t}(P \mapsto q) &= \Pi_u[\widehat{\mathbb{P}}(i_{\mathbb{P}}P, \widehat{u}), \widehat{t\,u}q] \\ \widehat{t}(\beta p) &= \widehat{\pi_\beta t}p \\ \widehat{t}P &= \Sigma_d \widehat{\mathbb{P}}(i_{\mathbb{P}}P, \widehat{t_d}) \\ \widehat{t}(abs\, p) &= \widehat{rep\, t}p \end{aligned} \tag{6.14}$$

—where d ranges over derivations of $!\mathbb{P} : t \xrightarrow{!} t_d : \mathbb{P}$. In particular, the set $\widehat{t}(P \mapsto q)$ is obtained by replacing universal quantification with product and implication by function space in the formula $\forall u.\ P \subseteq_{\mathbb{P}} u \implies q \in_{\mathbb{Q}} t\, u$.

As it stands, there is a problem with this. Since $!\mathbb{P}$ is now the free finite-colimit completion of $\mathbb{P}$, paths P should be interpreted as finite colimits, not as finite sets of paths. It is therefore not immediately obvious how to define a formal language of paths (and morphisms between paths) on which structural induction would make sense.

Because of this problem, we have considered a slightly roundabout approach, starting with the definition of a nondeterministic evaluation relation $\mathbb{P} : t \Downarrow v$. Here, $\vdash t : \mathbb{P}$ and v is a closed, well-formed term generated by the grammar

$$v ::= \lambda x.t \mid \beta t \mid\ !t \mid abs\, v\ . \tag{6.15}$$

The evaluation relation is defined by the rules

$$\begin{gathered} \frac{}{\mathbb{P} : v \Downarrow v} \quad \frac{\mathbb{P} : t[rec\, x.t/x] \Downarrow v}{\mathbb{P} : rec\, x.t \Downarrow v} \quad \frac{\mathbb{P} : t_j \Downarrow v}{\mathbb{P} : \Sigma_{i \in I} t_i \Downarrow v}\ j \in I \\ \frac{\mathbb{P} \to \mathbb{Q} : t \Downarrow \lambda x.t' \quad \mathbb{P} : t'[u/x] \Downarrow v}{\mathbb{Q} : t\, u \Downarrow v} \quad \frac{\Sigma_{\alpha \in A} \mathbb{P}_\alpha : t \Downarrow \beta t' \quad \mathbb{P}_\beta : t' \Downarrow v}{\mathbb{P}_\beta : \pi_\beta t \Downarrow v} \\ \frac{!\mathbb{P} : u \Downarrow\ !u' \quad \mathbb{Q} : t[u'/x] \Downarrow v}{\mathbb{Q} : [u > !x \Rightarrow t] \Downarrow v} \\ \frac{\mathbb{T}_j[\mu \vec{T}.\vec{\mathbb{T}}/\vec{T}] : t \Downarrow v}{\mu_j \vec{T}.\vec{\mathbb{T}} : abs\, t \Downarrow abs\, v} \quad \frac{\mu_j \vec{T}.\vec{\mathbb{T}} : t \Downarrow abs\, v}{\mathbb{T}_j[\mu \vec{T}.\vec{\mathbb{T}}/\vec{T}] : rep\, t \Downarrow v} \end{gathered} \tag{6.16}$$

By rule-induction we have that $\mathbb{P} : t \Downarrow v$ implies $\vdash v : \mathbb{P}$ as expected.

Lemma 6.4 Suppose $\vdash t : \mathbb{P}$. There is a bijective correspondence between pairs of derivations $\mathbb{P} : t \Downarrow v$ and $\mathbb{P} : v \xrightarrow{a} t' : \mathbb{P}'$ and derivations $\mathbb{P} : t \xrightarrow{a} t' : \mathbb{P}'$.

Proof. By induction on d, Figure 6.1 defines a map θ from pairs of derivations (d, d') with d a derivation of $\mathbb{P} : t \Downarrow v$ and d' a derivation of $\mathbb{P} : v \xrightarrow{a} t' : \mathbb{P}'$

to derivations $\theta(d,d')$ of $\mathbb{P} : t \xrightarrow{a} t' : \mathbb{P}'$. In the figure, types are omitted for clarity, and we use the notation $\overset{d}{t \Downarrow v}$ to mean that d is a derivation of $t \Downarrow v$.

Conversely, by induction on d, Figure 6.2 defines a map ϕ from derivations d of $\mathbb{P} : t \xrightarrow{a} t' : \mathbb{P}'$ to pairs $\phi d = (\phi_1 d, \phi_2 d)$ where $\phi_1 d$ is a derivation of $\mathbb{P} : t \Downarrow v$ and $\phi_2 d$ is a derivation of $\mathbb{P} : v \xrightarrow{a} t' : \mathbb{P}'$. It is exploited in the definition that the unique subderivation of $\phi_2 d$ deriving $v_0 \xrightarrow{a_0} t'$ with $v \equiv abs^n\, v_0$ and $a \equiv abs^n\, a_0$ is also a subderivation of d itself, so that we may apply ϕ inductively to its strict subderivations.

The maps θ and ϕ are shown to be inverses by two straightforward inductions on derivations. $\square$

For non-value $\vdash t : \mathbb{P}$, we now define $\widehat{t} =_{\mathrm{def}} \Sigma_d \widehat{v_d}$ where d ranges over derivations d of $\mathbb{P} : t \Downarrow v_d$. For values v, the presheaves $\widehat{v}$ are defined by well-founded induction using a lexicographic product of a well-founded relation on path objects and morphisms and the structural order on values. Here are first the defining equations:

$$
\begin{aligned}
\widehat{\lambda x.t}(P \mapsto q) &=_{\mathrm{def}} \Pi_u[\widehat{\mathbb{P}}(i_{\mathbb{P}}P, \widehat{u}), \widehat{t[u/x]}q] \\
\widehat{\lambda x.t}(M \mapsto n : P \mapsto q \to P' \mapsto q') &=_{\mathrm{def}} \Pi_u[\widehat{\mathbb{P}}(i_{\mathbb{P}}M, \widehat{u}), \widehat{t[u/x]}n] \\
\widehat{\beta t}(\beta p) &=_{\mathrm{def}} \widehat{t}p \\
\widehat{\beta t}(\beta m : \beta p \to \beta p') &=_{\mathrm{def}} \widehat{t}m \\
\widehat{!t}P &=_{\mathrm{def}} \widehat{\mathbb{P}}(i_{\mathbb{P}}P, \widehat{t}) \\
\widehat{!t}(M : P \to P') &=_{\mathrm{def}} \widehat{\mathbb{P}}(i_{\mathbb{P}}M, \widehat{t}) = (- \circ i_{\mathbb{P}}M) \\
\widehat{abs\, v}p &=_{\mathrm{def}} \widehat{v}p \\
\widehat{abs\, v}(m : p \to p') &=_{\mathrm{def}} \widehat{v}m
\end{aligned}
\tag{6.17}
$$

Here, P ranges over finite colimits, i.e. objects in $!\mathbb{P} \hookrightarrow \widehat{\mathbb{P}}$ while $P \mapsto q$ is used as notation for pairs of $\mathbb{P} \to \mathbb{Q} = (!\mathbb{P})^{\mathrm{op}} \times \mathbb{Q}$ and βp as notation for images of the injection map $in_\beta : \mathbb{P}_\beta \to \Sigma_{\alpha\in A}\mathbb{P}_\alpha$ between path categories. The map $\widehat{\lambda x.t}(M \mapsto n)$ maps a tuple $\langle f_u : \widehat{\mathbb{P}}(i_{\mathbb{P}}P, \widehat{u}) \to \widehat{t[u/x]}q'\rangle_u$ to the tuple $\langle \widehat{t[u/x]}n \circ f_u \circ (- \circ i_{\mathbb{P}}M)\rangle_u$. We'll drop the injection $i_{\mathbb{P}}$ below for brevity.

Lemma 6.5 The presheaves $\widehat{v}$ are well-defined.

Proof. A well-founded relation on paths is obtained using the axiom of foundation saying that the relation $\in$ is well-founded. In a set-theoretic encoding, the pairs $P \mapsto q$ are sets $\{P, \{P, q\}\}$ such that $P \in P \mapsto q$ and $q \in^2 P \mapsto q$. The injects βp are pairs (β, p), so sets $\{\beta, \{\beta, p\}\}$, so that

$$\theta(\overline{v \Downarrow v}, d') =_{\mathrm{def}} d'$$

$$\theta\Big(\dfrac{\overset{d}{t_j \Downarrow v}}{\Sigma_{i\in I} t_i \Downarrow v}\ j \in I,\ \overset{d'}{v \xrightarrow{a} t'}\Big) =_{\mathrm{def}} \dfrac{\overset{\theta(d,d')}{t_j \xrightarrow{a} t'}}{\Sigma_{i\in I} t_i \xrightarrow{a} t'}\ j \in I$$

$$\theta\Big(\dfrac{\overset{d}{t[rec\,x.t/x] \Downarrow v}}{rec\,x.t \Downarrow v},\ \overset{d'}{v \xrightarrow{a} t'}\Big) =_{\mathrm{def}} \dfrac{\overset{\theta(d,d')}{t[rec\,x.t/x] \xrightarrow{a} t'}}{rec\,x.t \xrightarrow{a} t'}$$

$$\theta\Big(\dfrac{\overset{d_1}{t \Downarrow \lambda x.t''} \quad \overset{d_2}{t''[u/x] \Downarrow v}}{t\ u \Downarrow v},\ \overset{d'}{v \xrightarrow{a} t'}\Big) =_{\mathrm{def}} \dfrac{\theta\Big(d_1, \dfrac{\overset{\theta(d_2,d')}{t''[u/x] \xrightarrow{a} t'}}{\lambda x.t'' \xrightarrow{u \mapsto a} t'}\Big) \atop t \xrightarrow{u \mapsto a} t'}{t\ u \xrightarrow{a} t'}$$

$$\theta\Big(\dfrac{\overset{d_1}{t \Downarrow \beta t''} \quad \overset{d_2}{t'' \Downarrow v}}{\pi_\beta t \Downarrow v},\ \overset{d'}{v \xrightarrow{a} t'}\Big) =_{\mathrm{def}} \dfrac{\theta\Big(d_1, \dfrac{\overset{\theta(d_2,d')}{t'' \xrightarrow{a} t'}}{\beta t'' \xrightarrow{\beta a} t'}\Big) \atop t \xrightarrow{\beta a} t'}{\pi_\beta t \xrightarrow{a} t'}$$

$$\theta\Big(\dfrac{\overset{d_1}{u \Downarrow\, !u'} \quad \overset{d_2}{t[u'/x] \Downarrow v}}{[u > !x \Rightarrow t] \Downarrow v},\ \overset{d'}{v \xrightarrow{a} t'}\Big) =_{\mathrm{def}} \dfrac{\overset{\theta(d_1, \overline{!u' \xrightarrow{!} u'})}{u \xrightarrow{!} u'} \quad \overset{\theta(d_2,d')}{t[u'/x] \xrightarrow{a} t'}}{[u > !x \Rightarrow t] \xrightarrow{a} t'}$$

$$\theta\Big(\dfrac{\overset{d}{t \Downarrow v}}{abs\,t \Downarrow abs\,v},\ \dfrac{\overset{d'}{v \xrightarrow{a} t'}}{abs\,v \xrightarrow{abs\,a} t'}\Big) =_{\mathrm{def}} \dfrac{\overset{\theta(d,d')}{t \xrightarrow{a} t'}}{abs\,t \xrightarrow{abs\,a} t'}$$

$$\theta\Big(\dfrac{\overset{d}{t \Downarrow abs\,v}}{rep\,t \Downarrow v},\ \overset{d'}{v \xrightarrow{a} t'}\Big) =_{\mathrm{def}} \dfrac{\theta\Big(d, \dfrac{\overset{d'}{v \xrightarrow{a} t'}}{abs\,v \xrightarrow{abs\,a} t'}\Big) \atop t \xrightarrow{abs\,a} t'}{rep\,t \xrightarrow{a} t'}$$

Figure 6.1: Definition of θ

$$\phi(\overset{d}{v \xrightarrow{a} t'}) =_{\mathrm{def}} (\overline{v \Downarrow v}, d)$$

$$\phi\Big(\frac{\overset{d}{t_j \xrightarrow{a} t'}}{\Sigma_{i\in I} t_i \xrightarrow{a} t'}\ j \in I\Big) =_{\mathrm{def}} \Big(\frac{\overset{\phi_1 d}{t_j \Downarrow v}}{\Sigma_{i\in I} t_i \Downarrow v}\ j \in I, \overset{\phi_2 d}{v \xrightarrow{a} t'}\Big)$$

$$\phi\Big(\frac{\overset{d}{t[rec\, x.t/x] \xrightarrow{a} t'}}{rec\, x.t \xrightarrow{a} t'}\Big) =_{\mathrm{def}} \Big(\frac{\overset{\phi_1 d}{t[rec\, x.t/x] \Downarrow v}}{rec\, x.t \Downarrow v}, \overset{\phi_2 d}{v \xrightarrow{a} t'}\Big)$$

$$\phi\Big(\frac{\overset{d}{t \xrightarrow{u \mapsto a} t'}}{t\ u \xrightarrow{a} t'}\Big) =_{\mathrm{def}} \Big(\frac{\overset{\phi_1 d}{t \Downarrow \lambda x.t''} \quad \overset{\phi_1 d'}{t''[u/x] \Downarrow v}}{t\ u \Downarrow v}, \overset{\phi_2 d'}{v \xrightarrow{a} t'}\Big)$$

—where d' is a subderivation of $\phi_2 d = \dfrac{\overset{d'}{t''[u/x] \xrightarrow{a} t'}}{\lambda x.t'' \xrightarrow{u \mapsto a} t'}$

$$\phi\Big(\frac{\overset{d}{t \xrightarrow{\beta a} t'}}{\pi_\beta t \xrightarrow{a} t'}\Big) =_{\mathrm{def}} \Big(\frac{\overset{\phi_1 d}{t \Downarrow \beta t''} \quad \overset{\phi_1 d'}{t'' \Downarrow v}}{\pi_\beta t \Downarrow v}, \overset{\phi_2 d'}{v \xrightarrow{a} t'}\Big)$$

—where d' is a subderivation of $\phi_2 d = \dfrac{\overset{d'}{t'' \xrightarrow{a} t'}}{\beta t'' \xrightarrow{\beta a} t'}$

$$\phi\Big(\frac{\overset{d}{t \xrightarrow{a} t'}}{abs\ t \xrightarrow{abs\ a} t'}\Big) =_{\mathrm{def}} \Big(\frac{\overset{\phi_1 d}{t \Downarrow v}}{abs\ t \Downarrow abs\ v}, \frac{\overset{\phi_2 d}{v \xrightarrow{a} t'}}{abs\ v \xrightarrow{abs\ a} t'}\Big)$$

$$\phi\Big(\frac{\overset{d}{t \xrightarrow{abs\ a} t'}}{rep\ t \xrightarrow{a} t'}\Big) =_{\mathrm{def}} \Big(\frac{\overset{\phi_1 d}{t \Downarrow abs\ v}}{rep\ t \Downarrow v}, d'\Big)$$

—where d' is a subderivation of $\phi_2 d = \dfrac{\overset{d'}{v \xrightarrow{a} t'}}{abs\ v \xrightarrow{abs\ a} t'}$

Figure 6.2: Definition of ϕ

$p \in^2 \beta p$. Finite colimits P are presheaves, so functors, so pairs of functions, so sets $\{P_0, \{P_0, P_1\}\}$. Here, P_0 is a function on objects, so a set of pairs (p, x) where $x \in Pp$. Hence, $p \in^3 P$ whenever $Pp \neq \varnothing$. We therefore have a well-founded order $\succ = \in^+$ on paths satisfying

$$\begin{aligned} P \mapsto q &\succ P \\ P \mapsto q &\succ q \\ \beta p &\succ p \\ P &\succ p \qquad \text{if } Pp \neq \varnothing \end{aligned} \tag{6.18}$$

We extend this order to morphisms using a product construction:

$$(m : p \to p') \succ (n : q \to q') \iff_{\text{def}} (p \succ q) \text{ and } (p' \succ q') \tag{6.19}$$

We now have $p \succ q \iff 1_p \succ 1_q$ and we can relate paths and morphisms using $p \succ m \iff_{\text{def}} 1_p \succ m$ and $m \succ p \iff_{\text{def}} m \succ 1_p$.

Now, in the first three clauses of the definition of the presheaves $\widehat{v}$, we have that paths/morphisms mentioned on the left-hand side are related by $\succ$ to those needed on the right-hand side. In particular, to obtain the set $\widehat{\mathbb{P}}(P, \widehat{u})$ we need only consider path objects p of $\mathbb{P}$ such that $Pp \neq \varnothing$ and morphisms between such paths. In the fourth clause, the path/morphism mentioned on the two sides are the same, but the structural size of the value decreases. Thus, by well-founded induction using the lexicographic product of $\succ$ with the structural order on values, the presheaves $\widehat{v}$ are well-defined. $\square$

Using this definition of $\widehat{t}$ we can show a result corresponding to the main lemma (Lemma 3.15) of Section 3.4. We'll need some auxiliary notation. First, consider a family $\langle f^i : X_i \to Y_i \rangle_{i \in I}$ with each $f^i \in \widehat{\mathbb{P}}(X_i, Y_i)$. We'll write $\Sigma_{i \in I} f^i$ for the obvious map $\Sigma_{i \in I} X_i \to \Sigma_{i \in I} Y_i$. Second, let $f : X \to \widehat{t}$ and $g : Y \to \widehat{u}$ be maps in $\widehat{\mathbb{P} \to \mathbb{Q}}$ and $\widehat{\mathbb{P}}$, respectively. We want a map $f \cdot g : XY \to \widehat{t\,u}$. Application yields a map $fg : XY \to \widehat{t}\widehat{u}$. At path q, the application $\widehat{t}\widehat{u}$ is given by the coend formula

$$\begin{aligned} &\textstyle\int^{P \in !\mathbb{P}} \widehat{\mathbb{P}}(P, \widehat{u}) \times \widehat{t}(P \mapsto q) \\ &= \textstyle\int^{P \in !\mathbb{P}} \widehat{\mathbb{P}}(P, \widehat{u}) \times \Sigma_d \widehat{\lambda x.t_d}(P \mapsto q) \\ &\cong \textstyle\int^{P \in !\mathbb{P}} \Sigma_d (\widehat{\mathbb{P}}(P, \widehat{u}) \times \widehat{\lambda x.t_d}(P \mapsto q)) \\ &\cong \Sigma_d \textstyle\int^{P \in !\mathbb{P}} \widehat{\mathbb{P}}(P, \widehat{u}) \times \widehat{\lambda x.t_d}(P \mapsto q) \\ &= \Sigma_d \textstyle\int^{P \in !\mathbb{P}} \widehat{\mathbb{P}}(P, \widehat{u}) \times \Pi_u [\widehat{\mathbb{P}}(P, \widehat{u}), \widehat{t_d[u/x]} q] \end{aligned}$$

For each derivation d of $\mathbb{P} \to \mathbb{Q} : t \Downarrow \lambda x.t_d$ there is a unique morphism h_d from the above coend to $\widehat{t_d[u/x]}$ because the latter is the vertex of a wedge from the same functor. Indeed, for any $M : P \to P'$ in $!\mathbb{P}$, the square

$$\begin{tikzcd}
\widehat{\mathbb{P}}(P', \widehat{u}) \times \Pi_{u'}[\widehat{\mathbb{P}}(P, \widehat{u'}), \widehat{t_d[u'/x]q}] \arrow[r, "(-\circ M)\times 1"] \arrow[d, "1\times\langle f_{u'}\mapsto f_{u'}\circ(-\circ M)\rangle_{u'}"'] & \widehat{\mathbb{P}}(P, \widehat{u}) \times \Pi_{u'}[\widehat{\mathbb{P}}(P, \widehat{u'}), \widehat{t_d[u'/x]q}] \arrow[d, "app_u"] \\
\widehat{\mathbb{P}}(P', \widehat{u}) \times \prod_{u'}[\widehat{\mathbb{P}}(P', \widehat{u'}), \widehat{t_d[u'/x]q}] \arrow[r, "app_u"'] & \widehat{t_d[u/x]q}
\end{tikzcd}$$

commutes. Here, app_u maps $(g, \langle f_{u'}\rangle_{u'})$ to $f_u g$. We take $f\cdot g$ to be $(\Sigma_d h_d)\circ fg : XY \to \Sigma_d \widehat{t_d[u/x]} = \widehat{t\,u}$.

Lemma 6.6 (Main Lemma) For any $\vdash t : \mathbb{P}$ the set $\widehat{\mathbb{P}}([\![t]\!], \widehat{t})$ is nonempty.

Proof. By structural induction on open terms using the induction hypothesis

> Suppose t has free variables among $x_1 : \mathbb{P}_1, \ldots, x_k : \mathbb{P}_k$ and that there are closed terms $\vdash s_j : \mathbb{P}_j$ for each $1 \leq j \leq k$. Then, there is a map
>
> $$H_t : \Pi_j \widehat{\mathbb{P}_j}(X_j, \widehat{s_j}) \to \widehat{\mathbb{P}}([\![t]\!](X_1, \ldots, X_k), \widehat{t[s_1/x_1, \ldots, s_k/x_k]}) \ , \tag{6.20}$$
>
> natural in the X_i.

We'll abbreviate $(X_1, \ldots, X_k)$ to X and $[s_1/x_1, \ldots, s_k/x_k]$ to s, and we'll suppress the free variables of t whenever possible.

Variable. Immediate as the projections from the product $\Pi_j \widehat{\mathbb{P}_j}(X_j, \widehat{s_j})$ are natural in the X_i.

Recursion. We want a map $f : [\![rec\,x.t]\!] \to \widehat{rec\,x.t}$. We'll start by showing that for each $n \in \omega$ there is a map $f^n : [\![rec^n\,x.t]\!] \to \widehat{rec\,x.t}$. Here, $[\![rec^n\,x.t]\!]$ is the n'th approximation to the denotation of $[\![rec\,x.t]\!]$.

Basis. $[\![rec^0\,x.t]\!]$ is the empty presheaf, and so we may use the empty natural transformation.

Step. Suppose we have $f^n : [\![rec^n\,x.t]\!] \to \widehat{rec\,x.t}$. By the induction hypothesis for the structural induction, there is a natural transformation $H_t(f^n) : [\![t]\!]([\![rec^n\,x.t]\!]) \to \widehat{t[rec\,x.t/x]}$. Now, $[\![t]\!]([\![rec^n\,x.t]\!]) = [\![rec^{n+1}\,x.t]\!]$ and there is an obvious isomorphism $\theta : \widehat{t[rec\,x.t/x]} \cong \widehat{rec\,x.t}$ and so we can take $f^{n+1} = \theta \circ H_t(f^n)$.

The mathematical induction is complete.

For each $n \in \omega$, there is a map $g^n : [\![rec^n\,x.t]\!] \to [\![rec^{n+1}\,x.t]\!]$ given by induction on n so that g^0 is the empty natural transformation while $g^{n+1} = [\![t]\!]g^n$. Thus, we have a diagram of shape ω in $\widehat{\mathbb{P}}$ whose colimit is the denotation of $rec\,x.t$. Therefore, if we can show that $\widehat{rec\,x.t}$ is the vertex of a corresponding cocone, we are done as we can take f to be the unique mediating morphism from the colimit to $\widehat{rec\, x.t}$. Clearly, it is enough to show that the following triangle commutes for any $n \in \omega$:

$$\begin{tikzcd}
\llbracket rec^n\, x.t \rrbracket \arrow[dd, "g^n"'] \arrow[dr, "f^n"] & \\
& \widehat{rec\, x.t} \\
\llbracket rec^{n+1}\, x.t \rrbracket \arrow[ur, "f^{n+1}"'] &
\end{tikzcd}$$

We proceed by mathematical induction. The base case is obvious since g^0 and f^0 are both empty. For the induction step, suppose that the above diagram commutes. We need to show that $f^{n+2} \circ g^{n+1} = f^{n+1}$. Now, $f^{n+2} \circ g^{n+1} = \theta \circ H_t(f^{n+1}) \circ \llbracket t \rrbracket g^n$ while $f^{n+1} = \theta \circ H_t(f^n)$ and so the wanted equality follows by naturality of H_t and the commutativity of the diagram above. The mathematical induction is complete, and $\widehat{rec\, x.t}$ is the vertex of a cocone as wanted.

Nondeterministic sum. We want a map $f : \llbracket \Sigma_{i \in I} t_i \rrbracket \to \widehat{\Sigma_{i \in I} t_i}$. We have $\llbracket \Sigma_{i \in I} t_i \rrbracket \cong \Sigma_{i \in I} \llbracket t_i \rrbracket$ and $\widehat{\Sigma_{i \in I} t_i} \cong \Sigma_{i \in I} \widehat{t_i}$. By the induction hypothesis, there is a map $f^i : \llbracket t_i \rrbracket \to \widehat{t_i}$ for each $i \in I$ and we can thus take $f = \Sigma_{i \in I} f_i$.

Abstraction. We want a map $f : \llbracket \lambda x.t \rrbracket \to \widehat{\lambda x.t}$. So we need a function

$$f_{P \mapsto q} : \llbracket \lambda x.t \rrbracket (P \mapsto q) \to \Pi_u [\widehat{\mathbb{P}}(P, \widehat{u}), \widehat{t[u/x]} q] \tag{6.21}$$

natural in $P \mapsto q$. Note that $\llbracket \lambda x.t \rrbracket (P \mapsto q) \cong (\llbracket t \rrbracket P) q$. By the induction hypothesis, H_t provides for each u and $g : P \to \widehat{u}$ a natural transformation $H_t(g) : \llbracket t \rrbracket (P) \to \widehat{t[u/x]}$. So given $y \in (\llbracket t \rrbracket P) q$ we take $f_{P \mapsto q} y$ to be at component u the map $g \mapsto H_t(g)_q y$. Naturality in q follows from naturality of $H_t(g)$ while naturality in P follows from naturality of H_t.

Application. We want a map $f : \llbracket t\ u \rrbracket \to \widehat{t\, u}$. By the induction hypothesis, there are maps $g : \llbracket t \rrbracket \to \widehat{t}$ and $h : \llbracket u \rrbracket \to \widehat{u}$. We can then take $f = g \cdot h$.

Prefixing. We want a map $f : \llbracket !t \rrbracket \to \widehat{!t}$. By the induction hypothesis, there is a map $g : \llbracket t \rrbracket \to \widehat{t}$. We have $\llbracket !t \rrbracket \cong \eta_{\mathbb{P}} \llbracket t \rrbracket$ and $\widehat{!t} = \widehat{\mathbb{P}}(i_{\mathbb{P}}-, \widehat{t}) \cong \eta_{\mathbb{P}} \widehat{t}$, and so we take $f = \eta_{\mathbb{P}} g$.

Prefix match. We want a map $f : \llbracket [u > !x \Rightarrow t] \rrbracket \to [\widehat{u > !x \Rightarrow} t]$. By the induction hypothesis for u there is a map $g : \llbracket u \rrbracket \to \widehat{u}$. Letting d range over derivations $!\mathbb{P} : u \Downarrow !u_d$, we have $\llbracket u \rrbracket \cong \Sigma_d X_d$ and $g \cong \Sigma_d g_d$ such that $g_d : X_d \to \widehat{!u_d}$. Note that since $\widehat{!u_d} \cong \eta_{\mathbb{P}} \widehat{u_d}$ we have $\varepsilon_{\mathbb{P}} g_d : \varepsilon_{\mathbb{P}} X_d \to \widehat{u_d}$. From the denotational semantics, we have

$$
\begin{aligned}
&[\![[u > !x \Rightarrow t]]\!]q \\
&\cong \int^{P\in !\mathbb{P}} [\![u]\!]P \times ([\![t]\!]P)q \\
&\cong \int^{P\in !\mathbb{P}} (\Sigma_d X_d)P \times ([\![t]\!]P)q \\
&\cong \Sigma_d \int^{P\in !\mathbb{P}} X_d P \times ([\![t]\!]P)q
\end{aligned}
$$

For each derivation d, the canonical map $X_d \to \eta_{\mathbb{P}}(\varepsilon_{\mathbb{P}} X_d)$ (corresponding to the fact that $X \subseteq \eta_{\mathbb{P}}(\varepsilon_{\mathbb{P}} X)$ for path sets) induces a unique mediating map h_d from the above coend to the coend

$$
\begin{aligned}
&\int^{P\in !\mathbb{P}} (\eta_{\mathbb{P}}(\varepsilon_{\mathbb{P}} X_d))P \times ([\![t]\!]P)q \\
&\cong \int^{P\in !\mathbb{P}} \widehat{\mathbb{P}}(P, \varepsilon_{\mathbb{P}} X_d) \times ([\![t]\!]P)q \\
&\cong ([\![t]\!](\varepsilon_{\mathbb{P}} X_d))q
\end{aligned}
$$

By the induction hypothesis for t we get from $H_t(\varepsilon_{\mathbb{P}} g_d)$ a natural transformation $f_d : [\![t]\!](\varepsilon_{\mathbb{P}} X_d) \to \widehat{t[u_d/x]}$ and so we may take $f = \Sigma_d(f_d \circ h_d)$ because $\widehat{[u > !x \Rightarrow t]} \cong \Sigma_d \widehat{t[u_d/x]}$.

The remaining cases are treated similarly. $\square$

What we would like to do now is to combine the above result with the approach from Section 6.1, using a logical predicate $B_{\mathbb{P}}$ on natural transformations $f : X \to \widehat{t}$ which at prefix type would ensure that we got not just any map $[\![t]\!] \to \widehat{t}$, but a suitable bijection giving strong correspondence from the main lemma.

We expect to define the predicates $B_{\mathbb{P}}(f : X \to \widehat{v})$ as $\forall p \in \mathbb{P}.\ B_{\mathbb{P}}(f, p)$ where the predicates $B_{\mathbb{P}}(f : X \to \widehat{v}, p)$ are defined by well-founded induction on the lexicographic product used to show well-definedness of the presheaves $\widehat{v}$. For $f : X \to \widehat{t}$ where t is not a value, we have that each derivation d of $t \Downarrow v_d$ provides a map $f_d : X_d \to \widehat{v_d}$ with $f \cong \Sigma_d f_d$, and we would then define $B_{\mathbb{P}}(f)$ in terms of $B_{\mathbb{P}}(f_d)$. A proof along these lines is work in progress.

An alternative way of obtaining the main lemma is also possible, avoiding the nondeterministic evaluation. Rather, it is based on a direct definition of the presheaves $\widehat{t}$ much as in (6.14), but with a different treatment of recursive types. This approach uses a realisability version of Lemma 3.17 which is flexible enough that we may be able to adjust to make it behave well with respect to suitable logical predicates $B_{\mathbb{P}}$.

6.3 Affine HOPLA

If $\vdash t : \mathbb{P}$ and $\mathbb{P} : a : \mathbb{P}'$ in Affine HOPLA, then $a^*[\![t]\!]$ is a presheaf over $\mathbb{P}'_\perp$. By Proposition 5.1, $a^*[\![t]\!]$ has a rooted component decomposition,

$$\Sigma_{i\in(a^*[\![t]\!]\perp)} \lfloor X_i \rfloor \tag{6.22}$$

with each $X_i \in \widehat{\mathbb{P}'}$. As lifting interprets prefixing, proving strong correspondence amounts to showing that the elements i of $a^*[\![t]\!]\bot$ correspond bijectively to derivations d_i of $\mathbb{P} : t \xrightarrow{a} t_{d_i} : \mathbb{P}'$ with $[\![t_{d_i}]\!] \cong X_i$.

Theorem 6.7 (Strong Correspondence) Suppose $\vdash t : \mathbb{P}$ and $\mathbb{P} : a : \mathbb{P}'$ in Affine HOPLA. Then $a^*[\![t]\!] \cong \Sigma_d[\![!t_d]\!]$ where d ranges over derivations of $\mathbb{P} : t \xrightarrow{a} t_d : \mathbb{P}'$.

Proof. (Sketch) For Affine HOPLA with finite unfoldings of recursively defined processes, we can use well-founded induction on the size measure (4.49), essentially repeating the proof of Lemma 4.12 showing correspondence for finite unfoldings. In the presheaf semantics, recursively defined processes of type $\mathbb{P}$ are interpreted using ω-colimits of the form $\int^{n\in\omega} f^n\varnothing$ where $f : \mathbb{P} \to \mathbb{P}$ is composed from the universal constructions of **Aff**. Now, all these constructions preserve natural transformations whose components are injections, and so the colimit has contribution at $p \in \mathbb{P}$ given up to isomorphism by $\bigcup_{n\in\omega}(f^n\varnothing)p$. This allows us to use the same kind of reasoning here as in the proof of Theorem 4.13, extending the result to full recursion. $\square$

Chapter 7

Event-Structure Representation

We now study the independence structure of the tensor operation of Affine HOPLA by giving an event-structure semantics to the language [63]. At first order it agrees with the presheaf semantics with realisers of presheaves corresponding to finite configurations of the representing event structure. The representation of open terms of Affine HOPLA associates to each such configuration the minimal input needed for the corresponding realisation. This is analogous to Berry's stability requirement (cf. Section 1.1.7). There, stability was meant to avoid functions like "parallel or", but both HOPLA and Affine HOPLA seem to allow such functions to be expressed. Consider the following term:

$$\lambda x.[x > \mathsf{a}.x' \Rightarrow \mathsf{c}.\varnothing] + [x > \mathsf{b}.x' \Rightarrow \mathsf{c}.\varnothing] \ . \tag{7.1}$$

In the path semantics, this term is interpreted as a function mapping $\varnothing$ to $\varnothing$ and both $\{\mathsf{a}\varnothing\}$ and $\{\mathsf{b}\varnothing\}$ to $\{\mathsf{c}\varnothing\}$. The function is thus not stable, because the path set $\{\mathsf{a}\varnothing, \mathsf{b}\varnothing\}$ dominates the inputs, yet there is no unique minimal input associated to the output $\{\mathsf{c}\varnothing\}$.

But intuitively, each way of realising that output, of which there are two, *is* associated with minimal input paths $\mathsf{a}\varnothing$ and $\mathsf{b}\varnothing$, respectively. Again, the presheaf semantics is needed to distinguish between these different realisations of the same computation path. In the presheaf semantics there is not just one $\mathsf{c}\varnothing$-output, but rather one for each realisation of $\mathsf{a}\varnothing$ in the input *and* one for each realisation of $\mathsf{b}\varnothing$ in the input.

Section 7.1 reviews event structures, some basic constructions, and notions of morphism. This is put to use in Section 7.2 to provide representations of the path orders used in the first-order fragment of Affine HOPLA and certain kinds of profunctors between them. Section 7.3 then shows that the representable profunctors include those that are denotable in the first-order language. This provides an alternative denotational semantics of the fragment.

An alternative operational semantics for the first-order fragment is considered in Section 7.4. It allows an easy proof of the diamond property discussed in Section 4.4.2. Section 7.5 briefly outlines how the event-structure denotational semantics extends to higher types, at the loss of correspondence to the presheaf semantics. A tentative extension of the operational semantics with fully general rules for function space is also provided.

7.1 Event Structures

We repeat the definition of event structures from the introduction, but use Girard's notation: consistency $\frown\!\!\!\!\smile$ is the complement of conflict $\smile = \#$, the reflexive closure of which is written $\asymp$ with complement $\frown$. Specifying one relation clearly determines all of them. So an event structure is a triple $\mathbb{E} = (E, \leq, \smile)$ where E is a set of events upon which a partial order $\leq$ of causality and a binary, symmetric, irreflexive relation $\smile$ of conflict is defined. This data must satisfy

(i) $\forall e \in E.\ \lceil e \rceil =_{\mathrm{def}} \{e' : e' \leq e\}$ is finite.

(ii) $\forall e, e', e'' \in E.\ e \smile e' \leq e'' \implies e \smile e''$.

A configuration of $\mathbb{E}$ is a subset $x \subseteq E$ which is

(i) downwards-closed: $\forall e \in x.\ \lceil e \rceil \subseteq x$.

(ii) consistent: $\forall e, e' \in x.\ e \frown\!\!\!\!\smile e'$.

We'll write $\mathcal{C}(\mathbb{E})$ for the partial order of finite configurations of $\mathbb{E}$, ordered by inclusion. Note that the empty set belongs to $\mathcal{C}(\mathbb{E})$ (so $\mathcal{C}(\mathbb{E})$ is pointed) as does $\lceil e \rceil$ for any $e \in E$. Two configurations x and x' of $\mathcal{C}(\mathbb{E})$ are said to be *compatible*, written $x \uparrow x'$, if there exists a configuration $y \in \mathcal{C}(\mathbb{E})$ with $x \subseteq y$ and $x' \subseteq y$.

Event structures can be viewed as the elements of a cpo and so admit least fixed-points of continuous operations [94]. Let $\mathbb{E}_1 = (E_1, \leq_1, \smile_1)$ and $\mathbb{E}_2 = (E_2, \leq_2, \smile_2)$ be event structures. We say that $\mathbb{E}_1$ is a *substructure* of $\mathbb{E}_2$, written $\mathbb{E}_1 \trianglelefteq \mathbb{E}_2$, if

(i) $E_1 \subseteq E_2$

(ii) $\forall e \in E_1.\ \lceil e \rceil_1 = \lceil e \rceil_2$

(iii) $\forall e, e' \in E_1.\ e \smile_1 e' \iff e \smile_2 e'$

Clearly, if $\mathbb{E}_1 \trianglelefteq \mathbb{E}_2$ and $E_1 = E_2$, then $\mathbb{E}_1$ and $\mathbb{E}_2$ are the same event structure. Event structures ordered by $\trianglelefteq$ form a large cpo (i.e. a cpo except for the fact that the elements form a proper class rather than a set). The least element is $\mathbb{O} = (\varnothing, \varnothing, \varnothing)$, and the least upper bound of an ω-chain $(\mathbb{E}_n)_{n \in \omega}$

with $\mathbb{E}_n = (E_n, \leq_n, \smile_n)$ is $(\bigcup_n E_n, \bigcup_n \leq_n, \bigcup_n \smile_n)$. This extends to tuples of event structures in the obvious point-wise way. An operation f on tuples of event structures is therefore monotone (continuous) iff it is monotone (continuous) in each argument separately. So it suffices to check monotonicity or continuity of unary operations, and a monotone, unary operation f is continuous iff it is *continuous on events*, so for any ω-chain as above, each event of $f(\bigcup_n \mathbb{E}_n)$ is an event of $\bigcup_n f(\mathbb{E}_n)$.

We list some basic constructions on event structures which are clearly monotone:

Sum. Given $\mathbb{E}_i = (E_i, \leq_i, \smile_i)$ for $i \in I$, define $\Sigma_{i\in I}\mathbb{E}_i$ to have

Events: the disjoint union of the E_i. For concreteness, let $E = \bigcup_{i\in I} \{i\} \times E_i$.

Causality: $(i, e) \leq (i', e') \iff_{\mathrm{def}} i = i'$ and $e \leq_i e'$.

Conflict: $(i, e) \smile (i', e') \iff_{\mathrm{def}} i \neq i'$ or ($i = i'$ and $e \smile_i e'$).

A subset of E is a configuration iff it is of the form $\{i\} \times x_i$ where $x_i \in \mathcal{C}(\mathbb{E}_i)$ for some $i \in I$. The poset $\mathcal{C}(\Sigma_{i\in I}\mathbb{E}_i)$ is therefore the coalesced sum of posets $\mathcal{C}(\mathbb{E}_i)$.

For each $i \in I$, assume an ω-chain $(\mathbb{E}_i^n)_{n\in\omega}$. Events of $\Sigma_{i\in I}\bigcup_n \mathbb{E}_i^n$ have the form (i, e) where $i \in I$ and e is an event of $\bigcup_n \mathbb{E}_i^n$, so that for some $n \in \omega$, we have that e is an event of $\mathbb{E}_i^n$. But then for that same n, the pair (i, e) is an event of $\Sigma_{i\in I}\mathbb{E}_i^n$, and so (i, e) is an event of $\bigcup_n \Sigma_{i\in I}\mathbb{E}_i^n$. Since $\bigcup_n \Sigma_{i\in I}\mathbb{E}_i^n \trianglelefteq \Sigma_{i\in I}\bigcup_n \mathbb{E}_i^n$ by monotonicity, we have $\bigcup_n \Sigma_{i\in I}\mathbb{E}_i^n = \Sigma_{i\in I}\bigcup_n \mathbb{E}_i^n$ as wanted.

Tensor. Given $\mathbb{E}_i = (E_i, \leq_i, \smile_i)$ for $i = 1, 2$, define $\mathbb{E}_1 \otimes \mathbb{E}_2$ to have

Events: $E = \{1\} \times E_1 \cup \{2\} \times E_2$, the disjoint union of the E_i.

Causality: $(i, e) \leq (i', e') \iff_{\mathrm{def}} i = i'$ and $e \leq_i e'$.

Conflict: $(i, e) \smile (i', e') \iff_{\mathrm{def}} i = i'$ and $e \smile_i e'$.

A subset of E is a configuration iff it is of the form $\{1\} \times x_1 \cup \{2\} \times x_2$ where $x_i \in \mathcal{C}(\mathbb{E}_i)$ for $i = 1, 2$. Therefore, $\mathcal{C}(\mathbb{E}_1 \otimes \mathbb{E}_2) \cong \mathcal{C}(\mathbb{E}_1) \times \mathcal{C}(\mathbb{E}_2)$.

Given an ω-chain $(\mathbb{E}_n)_{n\in\omega}$, an event of $(\bigcup_n \mathbb{E}_n) \otimes \mathbb{E}'$ may have the form $(1, e)$ where e is an event of $\mathbb{E}_n$ for some $n \in \omega$. Then for that same n, the pair $(1, e)$ is an event of $\mathbb{E}_n \otimes \mathbb{E}'$ and so of $\bigcup_n (\mathbb{E}_n \otimes \mathbb{E}')$. All other events of $(\bigcup_n \mathbb{E}_n) \otimes \mathbb{E}'$ have the form $(2, e)$ where e is an event of $\mathbb{E}'$. Then $(2, e)$ is an event of $\mathbb{E}_n \otimes \mathbb{E}'$ for each n and so an event of $\bigcup_{n\in\omega} \mathbb{E}_n \otimes \mathbb{E}'$. Thus, $(- \otimes \mathbb{E}')$ is continuous. Continuity in the other argument is obtained symmetrically.

Lifting. Given $\mathbb{E} = (E, \leq, \smile)$, define $\mathbb{E}_\perp$ to have

Events: $E_\perp = \{\varnothing\} \cup \{\lceil e \rceil : e \in E\}$.

Causality: given by inclusion.

Conflict: given by incompatibility.

A subset of $E_\perp$ is a configuration iff it is empty or of the form $\lfloor x \rfloor = \{\varnothing\} \cup \{\lceil e \rceil : e \in x\}$ where $x \in \mathcal{C}(\mathbb{E})$. Therefore, $\mathcal{C}(\mathbb{E}_\perp) \cong \mathcal{C}(\mathbb{E})_\perp$.

Given an ω-chain $(\mathbb{E}_n)_{n\in\omega}$, an event of $(\bigcup_n \mathbb{E}_n)_\perp$ may be of the form $\lceil e \rceil$ where e is an event of $\bigcup_n \mathbb{E}_n$ and so of $\mathbb{E}_n$ for some $n \in \omega$. For this same n, we have that $\lceil e \rceil$ is an event of $(\mathbb{E}_n)_\perp$ and so of $\bigcup_n (\mathbb{E}_{n\perp})$. The only other event of $(\bigcup_n \mathbb{E}_n)_\perp$ is $\varnothing$ which is an event of $\mathbb{E}_{n\perp}$ for all n and so an event of $\bigcup_n (\mathbb{E}_{n\perp})$. Thus lifting is continuous.

7.2 Representations

We may use the above constructions to give meaning to the types of the first-order fragment of Affine HOPLA. Let ζ range over finite maps from type variables to event structures. We then define

$$\begin{aligned}
[\![\mathbb{T}_1 \otimes \mathbb{T}_2]\!]\zeta &=_{\mathrm{def}} [\![T_1]\!]\zeta \otimes [\![T_2]\!]\zeta \\
[\![\Sigma_{\alpha\in A}\mathbb{T}_\alpha]\!]\zeta &=_{\mathrm{def}} \Sigma_{\alpha\in A}[\![\mathbb{T}_\alpha]\!]\zeta \\
[\![\mathbb{T}_\perp]\!]\zeta &=_{\mathrm{def}} ([\![\mathbb{T}]\!]\zeta)_\perp \\
[\![T]\!]\zeta &=_{\mathrm{def}} \zeta T \\
[\![\mu_j \vec{T}.\vec{\mathbb{T}}]\!]\zeta &=_{\mathrm{def}} \mu_j f
\end{aligned} \tag{7.2}$$

—where $\mu_j f$ is the j'th component, for $1 \le j \le k$, of the least fixed-point of the mapping f sending a k-tuple of event structures $\vec{\mathbb{E}}$ to the k-tuple

$$([\![\mathbb{T}_1]\!](\zeta[\vec{T} \mapsto \vec{\mathbb{E}}]), \ldots, [\![T_k]\!](\zeta[\vec{T} \mapsto \vec{\mathbb{E}}])) \ . \tag{7.3}$$

We'll confuse a closed expression $\mathbb{P}$ for an event structure with the event structure itself. By induction on the syntax of types we can show that $\mathcal{C}(\mathbb{P})$ is isomorphic to the corresponding path order $\mathbb{P}_\perp$ according to Chapter 4. We'll use the path notation from that chapter for the configurations of the event structure $\mathbb{P}$ with $p : \mathbb{P}$ ranging over nonempty configurations and $P : \mathbb{P}$ over all configurations. This has the unfortunate consequence that P now both stands for nonempty configurations of $\mathbb{P}_\perp$ and possibly empty configurations of $\mathbb{P}$. We'll use $!P : \mathbb{P}_\perp$ for the former to avoid confusion. We'll write $P \le_{\mathbb{P}} P'$ when P is a subconfiguration of P'.

As for terms of Affine HOPLA, consider first a presheaf $X : \mathbb{Q}^{\mathrm{op}} \to \mathbf{Set}$ over some path order $\mathbb{Q}$. By constructing its category of elements we obtain a monotone function (both $\mathbb{Q}$ and *elts* X are posets) $o : \mathit{elts}\, X \to \mathbb{Q}$ sending (q, x) to q. This map is a *discrete fibration.* In general, a functor $o : \mathbf{X} \to \mathbb{Q}$ is a discrete fibration if whenever $q \le q'$ in $\mathbb{Q}$ and $ox' = q'$ for some object $x' \in \mathbf{X}$, there is a unique arrow $x'|_q \to x'$ in $\mathbf{X}$ mapped by o to $q \le q'$:

$$\begin{tikzcd} \mathbf{X} \arrow[d, "o"] \\ \mathbb{Q} \end{tikzcd} \qquad \begin{tikzcd} x'|_q \arrow[r, dotted] \arrow[d, mapsto] & x' \arrow[d, mapsto] \\ q \arrow[r] & q' \end{tikzcd} \tag{7.4}$$

The "discreteness" of this fibration lies in the uniqueness of the arrow into x' above $q \leq q'$. In the case $\mathbf{X} = elts\, X$, we take $(q', x')|_q = (q, X(q \leq q')x')$.

Via the functor *elts*, the category of presheaves $\widehat{\mathbb{Q}}$ is equivalent to the category of discrete fibrations over $\mathbb{Q}$ [29]. The right adjoint to *elts* maps a discrete fibration $o : \mathbf{X} \to \mathbb{Q}$ to a presheaf $X \in \widehat{\mathbb{Q}}$ by taking (for x' above q')

$$Xq = \{x \in \mathbf{X} : ox = q\} \quad \text{and} \quad X(q \leq q')x' = x'|_q \ . \tag{7.5}$$

We can represent discrete fibrations, and so presheaves, using the "strict morphisms" of [100], here called *output morphisms*. An output morphism $o : \mathbb{E}_1 \to \mathbb{E}_2$ is a map $o : E_1 \to E_2$ such that $e \frown_1 e'$ implies $oe \frown_2 oe'$ and $o\lceil e \rceil = \lceil oe \rceil$ for all $e, e' \in E_1$. Output morphisms compose and extend directly to strict and monotone maps of configurations. The identity morphism $1_{\mathbb{E}} : \mathbb{E} \to \mathbb{E}$ is an output morphism as is the inclusion $\mathbb{E}_1 \hookrightarrow \mathbb{E}_2$ associated with the substructure relation $\mathbb{E}_1 \trianglelefteq \mathbb{E}_2$.

Lemma 7.1 Let $o : \mathbb{E} \to \mathbb{Q}$ be an output morphism. The extension to configurations $o : \mathcal{C}(\mathbb{E}) \to \mathcal{C}(\mathbb{Q})$ is a discrete fibration.

Proof. Consider two configurations $Q \subseteq Q' = ox'$ of $\mathbb{Q}$. We must show that there is a unique subconfiguration $x = x'|_Q$ of x' with $ox = Q$. Let x be the subset $\{e \in x' : oe \in Q\} \subseteq x'$. Since o is injective on consistent subsets, this is the unique subset of x' with image Q, and so we just need to show that x is itself a configuration of $\mathbb{E}$. When $e \in x$ we have $e \in x'$ and $oe \in Q$. Since x' and Q are configurations and so downwards-closed, $\lceil e \rceil \subseteq x'$ and $\lceil oe \rceil \subseteq Q$. As o is an output morphism we have $\lceil oe \rceil = o\lceil e \rceil$ from which it follows that $\lceil e \rceil \subseteq x$, and x is downwards-closed. Consistency of x is immediate as x is a subset of the configuration x'. $\square$

So output morphisms $o : \mathbb{E} \to \mathbb{Q}$ give rise to discrete fibrations $o : \mathcal{C}(\mathbb{E}) \to \mathcal{C}(\mathbb{Q})$ and so to presheaves over the path order $\mathbb{Q}_\perp \cong \mathcal{C}(\mathbb{Q})$. Because o is strict, each such presheaf will be rooted and thus isomorphic to $\lfloor X \rfloor$ for some presheaf X over the path order $\mathbb{Q}$ such that $elts\lfloor X \rfloor \cong \mathcal{C}(\mathbb{E})$. While not all presheaves over $\mathbb{Q}$ can be represented in this way, we are able to represent all closed terms of Affine HOPLA at ground type $\mathbb{Q}$.

Of course, to give a compositional event-structure semantics to the language, we need to show how to interpret open terms, so we need a representation of profunctors $\mathbb{P}_\perp \rightarrow\!\!\!\!+ \mathbb{Q}$, for path orders $\mathbb{P}_\perp$ and $\mathbb{Q}$, using event structures. Again, we can get some help from category theory. We ignore lifting for a moment. The category $\mathbf{Prof}(\mathbb{P}, \mathbb{Q})$ of profunctors $f : \mathbb{P} \rightarrow\!\!\!\!+ \mathbb{Q}$ is equivalent to the category $\mathbf{Bifib}(\mathbb{P}, \mathbb{Q})$ of discrete bifibrations[1] from $\mathbb{P}$ to $\mathbb{Q}$ [76]. Starting from a profunctor f, one constructs a "category of elements" *elts* f by taking as objects triples (p, x, q) with $x \in f(p, q)$ and arrows

[1] Thanks to Marcelo Fiore for making us aware of this connection.

$(p, x, q) \leq (p', x', q')$ whenever $p \leq_{\mathbb{P}} p'$ and $q \leq_{\mathbb{Q}} q'$ and $f(p \leq_{\mathbb{P}} p', q)x = f(p', q \leq_{\mathbb{Q}} q')x'$. The obvious projections $\iota : \mathit{elts}\, f \to \mathbb{P}$ and $o : \mathit{elts}\, f \to \mathbb{Q}$ yield a span $\mathbb{P} \xleftarrow{\iota} \mathit{elts}\, f \xrightarrow{o} \mathbb{Q}$ with ι an opfibration (a dual notion of fibration, associated with an operator $|^{p'}$ for objects above $p \leq p'$) and o a fibration. The span satisfies certain coherence and discreteness axioms to the effect that starting from such a span $\mathbb{P} \xleftarrow{\iota} \mathbf{X} \xrightarrow{o} \mathbb{Q}$, we can construct a profunctor $g : \mathbb{P} \mathrel{-\!\!\!+\!\!\!\rightarrow} \mathbb{Q}$ by taking

$$\begin{aligned} g(p, q) &= \{x \in \mathbf{X} : \iota x = p \text{ and } ox = q\} \qquad \text{and} \\ g(p \leq_{\mathbb{P}} p', q \leq_{\mathbb{Q}} q')x' &= (x'|^{p'})|_q = (x'|_q)|^{p'} \end{aligned} \tag{7.6}$$

for x' above p and q'. If $\mathbf{X}$ is $\mathit{elts}\, f$, we have $f \cong g$. Bifibrations compose by first constructing the pullback

$$\begin{tikzcd} & & \mathbf{Y} \arrow[dl, dotted, "\pi_1"'] \arrow[dr, dotted, "\pi_2"] & & \\ & \mathbf{X}_1 \arrow[dl, "\iota_1"'] \arrow[dr, "o_1"] & & \mathbf{X}_2 \arrow[dl, "\iota_2"'] \arrow[dr, "o_2"] & \\ \mathbb{P} & & \mathbb{Q} & & \mathbb{R} \end{tikzcd} \tag{7.7}$$

—and then quotienting $\mathbf{Y}$ in a way similar to what happens in the coend formula (5.9) to ensure that the span given by $\iota = \iota_1 \circ \pi_1$ and $o = o_2 \circ \pi_2$ is again a bifibration. We'll not go into more detail here, because we lack a way of representing opfibrations using maps of event structures.

But consider *stable* profunctors $f : \mathbb{P}_\perp \mathrel{-\!\!\!+\!\!\!\rightarrow} \mathbb{Q}$, those that, when curried to functors $\mathit{curry}\, f : \mathbb{P}_\perp \to \widehat{\mathbb{Q}}$, preserve pullbacks. Then, because $\mathbb{P}_\perp$ has all pullbacks (via the isomorphism $\mathbb{P}_\perp \cong \mathcal{C}(\mathbb{P})$, they are given by intersection of compatible configurations), for each $x \in f(P, q)$ there is a unique, minimal $P_0 \leq_{\mathbb{P}_\perp} P$ such that $x_0 \in f(P_0, q)$ and $f(P_0 \leq P, q)x_0 = x$. Accordingly, we "collapse" the category of elements $\mathit{elts}\, f$ on the input side, and retain only those triples (P_0, x_0, q) which are input-minimal. This turns ι into an ordinary monotone map and restores o as a discrete fibration, because the collapse is isomorphic to the category of elements of the presheaf $\int^{P \in \mathbb{P}_\perp} (\mathit{curry}\, f)P \in \widehat{\mathbb{Q}}$. Thus, we can reuse the output morphisms on the output side while the ι map will be represented by an *input morphism*:

Let $\mathbb{E}_1 = (E_1, \leq_1, \smile_1)$ and $\mathbb{E}_2 = (E_2, \leq_2, \smile_2)$ be event structures. An input morphism $\iota : \mathbb{E}_1 \to \mathbb{E}_2$ is a map $\iota : E_1 \to \mathcal{C}(\mathbb{E}_2)$ such that $e \leq_1 e' \implies \iota e \subseteq \iota e'$ and $e \mathrel{\frown\!\!\!\!\smile}_1 e' \implies \iota e \uparrow \iota e'$ for all $e, e' \in E_1$. The extension to a map $\mathcal{C}(\mathbb{E}_1) \to \mathcal{C}(\mathbb{E}_2)$, got by $\iota^\dagger x =_{\mathrm{def}} \bigcup_{e \in x} \iota e$, is monotone and strict. Input morphisms $\iota_1 : \mathbb{E}_1 \to \mathbb{E}_2$ and $\iota_2 : \mathbb{E}_2 \to \mathbb{E}_3$ compose as $\iota_2^\dagger \circ \iota_1$. The map $\lceil - \rceil : \mathbb{E} \to \mathcal{C}(\mathbb{E})$ is an input morphism $\mathbb{E} \to \mathbb{E}$ for any event structure $\mathbb{E}$; its extension $\lceil - \rceil^\dagger$ is the identity on $\mathcal{C}(\mathbb{E})$. Note also that the composition $\iota \circ o$ of an input morphism and an output morphism is again an input morphism.

In this way we are led to consider spans $\mathbb{P} \xleftarrow{\iota} \mathbb{E} \xrightarrow{o} \mathbb{Q}$ of event structures with ι and o input and output morphisms, respectively. Guided by the above, such a span induces a rooted, stable profunctor $f : \mathcal{C}(\mathbb{P}) \longrightarrow\!\!\!\!\!\!\!\!+\;\;\; \mathcal{C}(\mathbb{Q})$ by taking

$$\begin{aligned} f(P,Q) = \{x \in \mathcal{C}(\mathbb{E}) : \iota^\dagger x \subseteq P \text{ and } ox = Q\} \qquad \text{and} \\ f(P \subseteq P', Q \subseteq Q')x' = x'|_Q \ . \end{aligned} \tag{7.8}$$

Note that $|^{P'}$ is replaced by the identity map because $f(P \subseteq P', Q)$ is an inclusion for all $Q \in \mathcal{C}(\mathbb{Q})$. That f is rooted means that $f(P, \bot)$ is a singleton for any $P \in \mathcal{C}(\mathbb{P})$. Via the isomorphism $\mathcal{C}(\mathbb{P}) \cong \mathbb{P}_\bot$, rooted profunctors $\mathcal{C}(\mathbb{P}) \longrightarrow\!\!\!\!\!\!\!\!+\;\;\; \mathcal{C}(\mathbb{Q})$ correspond to profunctors $\mathbb{P}_\bot \longrightarrow\!\!\!\!\!\!\!\!+\;\;\; \mathbb{Q}$ between path orders. We'll use the notation $\langle \iota, \mathbb{E}, o\rangle : \mathbb{P} \to \mathbb{Q}$ for a span $\mathbb{P} \xleftarrow{\iota} \mathbb{E} \xrightarrow{o} \mathbb{Q}$ and write $\overline{\langle \iota, \mathbb{E}, o\rangle}$ for the represented profunctor $\mathbb{P}_\bot \longrightarrow\!\!\!\!\!\!\!\!+\;\;\; \mathbb{Q}$. When $f = \overline{\langle \iota, \mathbb{E}, o\rangle}$ we have

$$elts \lfloor \textstyle\int^{P \in \mathbb{P}_\bot} (curry\, f)P \rfloor \cong \mathcal{C}(\mathbb{E}) \ . \tag{7.9}$$

We'll write $\mathbf{Spn}(\mathbb{P}, \mathbb{Q})$ for the category whose objects are spans $\langle \iota, \mathbb{E}, o\rangle : \mathbb{P} \to \mathbb{Q}$ and whose arrows $o : \langle \iota_1, \mathbb{E}_1, o_1\rangle \to \langle \iota_2, \mathbb{E}_2, o_2\rangle$ are output morphisms between vertices of spans such that the diagram below commutes up to use of $\supseteq$ on the input side:

$$\begin{tikzcd} & \mathbb{E}_1 \arrow[dl, "\iota_1"'] \arrow[dr, "o_1"] \arrow[dd, "o"] & \\ \mathbb{P} \arrow[r, phantom, "\supseteq"] & & \mathbb{Q} \\ & \mathbb{E}_2 \arrow[ul, "\iota_2"] \arrow[ur, "o_2"'] & \end{tikzcd} \qquad \iota_1 e \supseteq \iota_2(oe) \quad \text{and} \quad o_1 e = o_2(oe) \tag{7.10}$$

Such morphisms of spans correspond to natural transformations between the represented profunctors.[2]

Proposition 7.2 The functor $\overline{(-)} : \mathbf{Spn}(\mathbb{P}, \mathbb{Q}) \to \mathbf{Prof}(\mathbb{P}_\bot, \mathbb{Q})$ is full and faithful.

Proof. Consider a morphism of spans as in (7.10) above and write

$$f_1 = \overline{\langle \iota_1, \mathbb{E}_1, o_1\rangle} \quad \text{and} \quad f_2 = \overline{\langle \iota_2, \mathbb{E}_2, o_2\rangle} \tag{7.11}$$

for the represented profunctors. We construct a natural transformation $\overline{o} : f_1 \to f_2$ by taking $\overline{o}_{P,Q}$ to be o extended to a map on configurations. If $x \in f_1(P,Q)$ then $\iota_1^\dagger x \subseteq P$ and $o_1 x = Q$ and we have

$$\iota_2^\dagger(ox) = \textstyle\bigcup_{e \in x} \iota_2(oe) \subseteq \bigcup_{e \in x} \iota_1 e = \iota_1^\dagger x \subseteq P \tag{7.12}$$

[2] One can also show that the span morphisms that make the diagram above commute on the nose correspond to natural transformations whose naturality squares are pullbacks. Since this condition is just a rephrasing of Berry's stability requirement, such morphisms may be more appropriate.

and $o_2(ox) = o_1 x = Q$. Hence, $ox \in f_2(P, Q)$ as wanted. As for naturality of $\overline{o}$, consider the square

$$\begin{array}{ccc} P & & Q \\ \downarrow & & \uparrow \\ P' & & Q' \end{array} \qquad \begin{tikzcd} f_1(P,Q) \arrow[r, "\overline{o}_{P,Q}"] \arrow[d, "(-)|_{Q'}"'] & f_2(P,Q) \arrow[d, "(-)|_{Q'}"] \\ f_1(P',Q') \arrow[r, "\overline{o}_{P',Q'}"'] & f_2(P',Q') \end{tikzcd} \tag{7.13}$$

Let $x \in f_1(P, Q)$. Since $o_2 \circ o = o_1$ we have

$$(ox)|_{Q'} = \{e \in ox : o_2 e \in Q'\} = o\{e \in x : o_2(oe) \in Q'\} = o(x|_{Q'}) \tag{7.14}$$

—as wanted.

Conversely, consider a natural transformation $o : f_1 \to f_2$. Given any event e of $\mathbb{E}_1$ we have $\lceil e \rceil \in f_1(\iota_1 e, o_1\lceil e \rceil)$ and so $o\lceil e \rceil \in f_2(\iota_1 e, o_1\lceil e \rceil)$ since o is a natural transformation. With o_1 an output morphism, this means that $o_2(o\lceil e \rceil) = o_1\lceil e \rceil = \lceil o_1 e \rceil$. As o_2 is also an output morphism, we have that $o\lceil e \rceil = \lceil e' \rceil$ for some e' such that $o_2 e' = o_1 e$. We now define $oe = e'$ and immediately obtain $o_2 \circ o = o_1$ and $\lceil oe \rceil = \lceil e' \rceil = o\lceil e \rceil$. Further, since $\lceil oe \rceil = o\lceil e \rceil \in f_2(\iota_1 e, o_1\lceil e \rceil)$, we have $\iota_2(oe) = \iota_2^\dagger \lceil oe \rceil \subseteq \iota_1 e$. Thus the diagram of (7.10) commutes in the way wanted. Finally, o preserves consistency as it maps configurations to configurations, and if $oe = oe'$, then $o_1 e = o_2(oe) = o_2(oe') = o_1 e'$ from which $e = e'$ follows as o_1 is an output morphism. Thus o preserves $\frown_1$ and is therefore an output morphism.

The above two maps are inverses. The direction starting from a morphism of spans is obvious. For the converse, we need to show that a natural transformation $o : f_1 \to f_2$ is completely determined by its values on prime configurations. So let $x \in f_1(P, Q)$. For any $e' \in ox$, naturality of o implies that $(ox)|_{\lceil o_2 e' \rceil} = \lceil e' \rceil = o\lceil e \rceil$ for some $e \in x$. Thus, $ox = \bigcup_{e \in x} o\lceil e \rceil$ and we are done. $\square$

So $\mathbf{Spn}(\mathbb{P}, \mathbb{Q})$ embeds in $\mathbf{Prof}(\mathbb{P}_\perp, \mathbb{Q})$. It would be nice at this point to give an independent characterisation of the represented profunctors, perhaps allowing us to work more abstractly below. However, we don't have that yet.

Note that by restricting morphisms of $\mathbf{Spn}(\mathbb{P}, \mathbb{Q})$ to those induced by the substructure relation, we obtain a large cpo ordered by $\langle \iota_1, \mathbb{E}_1, o_1 \rangle \trianglelefteq \langle \iota_2, \mathbb{E}_2, o_2 \rangle$ iff $\mathbb{E}_1 \trianglelefteq \mathbb{E}_2$ and ι_1, o_1 are the restrictions of ι_2, o_2 to the events of $\mathbb{E}_1$. This allows us to give meaning to recursively defined spans.

Our spans compose by adjoining a pullback in a way similar to bifibrations, but without the need for quotienting at the vertex. Given spans

$$\begin{tikzcd} & \mathbb{E}_1 \arrow[dl, "\iota_1"'] \arrow[dr, "o_1"] & & \mathbb{E}_2 \arrow[dl, "\iota_2"'] \arrow[dr, "o_2"] & \\ \mathbb{P} & & \mathbb{Q} & & \mathbb{R} \end{tikzcd} \tag{7.15}$$

with $\mathbb{E}_i = (E_i, \leq_i, \smile_i)$ for $i = 1, 2$, we construct first a span $\langle \pi_1, \mathbb{E}, \pi_2 \rangle : \mathbb{E}_1 \to \mathbb{E}_2$ by defining $\mathbb{E}$ to have

Events: $E = \{(x, e) \in \mathcal{C}(\mathbb{E}_1) \times E_2 : o_1 x = \iota_2 e\}$.

Causality: $(x, e) \leq (x', e') \iff_{\text{def}} x \subseteq x'$ and $e \leq_2 e'$.

Consistency: $(x, e) \frown\!\!\!\!\smile (x', e') \iff_{\text{def}} x \uparrow x'$ and $e \frown\!\!\!\!\smile_2 e'$.

The maps π_1, π_2 are the first and second projections from $\mathcal{C}(\mathbb{E}_1) \times E_2$. As the proposition below shows, this amounts to adjoining a pullback to the diagram above, working within the category of posets and monotone maps.

Proposition 7.3 The map $\pi_1 : \mathbb{E} \to \mathbb{E}_1$ is an input morphism and $\pi_2 : \mathbb{E} \to \mathbb{E}_2$ is an output morphism. The map $(\pi_1^\dagger -, \pi_2 -)$ is an isomorphism $\mathcal{C}(\mathbb{E}) \cong \{(x_1, x_2) \in \mathcal{C}(\mathbb{E}_1) \times \mathcal{C}(\mathbb{E}_2) : o_1 x_1 = \iota_2^\dagger x_2\}$.

Proof. First note that if c is a configuration of $\mathbb{E}$, then

$$\forall (x, e), (x', e) \in E.\ (x, e) \in c \text{ and } (x', e) \in c \implies x = x' \ . \tag{7.16}$$

Indeed, if x and x' are compatible, so that $x \cup x'$ is a consistent subset of E_2, there can be only one subset of $x \cup x'$ with image $\iota_2 e$ under o_1 as this is an output morphism. Hence, we must have $x = x'$.

The map $\pi_1 : \mathbb{E} \to \mathbb{E}_1$ is an input morphism by construction. To show that π_2 is an output morphism, suppose $(x, e) \frown (x', e')$. Then $x \uparrow x'$ and $e \frown\!\!\!\!\smile e'$ but $(x, e) \neq (x', e')$. By (7.16), $e = e'$ implies $x = x'$ which is a contradiction and so $e \neq e'$. But then $e \frown_2 e'$ as wanted. Now consider $e' \in \lceil \pi_2(x, e) \rceil = \lceil e \rceil$. We need to show that $e' \in \pi_2 \lceil (x, e) \rceil$. By definition of the events of $\mathbb{E}$ we have $o_1 x = \iota_2 e$ and so $\iota_2 e' \subseteq o_1 x$ which implies the existence of a unique subconfiguration $x' = x|_{\iota_2 e'}$ of x with $o_1 x' = \iota_2 e'$. Hence $(x', e') \in E$ with $(x', e') \leq (x, e)$ and so $e' \in \pi_2 \lceil (x, e) \rceil$ as wanted. For the converse, it suffices to observe that π_2 preserves causality.

Let $c \in \mathcal{C}(\mathbb{E})$. By the above, $\pi_1^\dagger c \in \mathcal{C}(\mathbb{E}_1)$ and $\pi_2 c \in \mathcal{C}(\mathbb{E}_2)$. Further, $o_1(\pi_1^\dagger c) = \iota_2^\dagger(\pi_2 c)$ because this is true for any subset of E. Hence, mapping c to the pair $(\pi_1^\dagger c, \pi_2 c)$, we get one part of the wanted isomorphism.

For the converse, consider $x_1 \in \mathcal{C}(\mathbb{E}_1)$ and $x_2 \in \mathcal{C}(\mathbb{E}_2)$ with $o_1 x_1 = \iota_2^\dagger x_2$. Define $c \subseteq E$ by $c =_{\text{def}} \{(x, e) \in E.\ x \subseteq x_1 \text{ and } e \in x_2\}$. Consistency of c follows because all first components are compatible, being subsets of x_1, and all second components are consistent, being elements of x_2. Downwards-closure of c follows from downwards-closure of x_2. Hence, $c \in \mathcal{C}(\mathbb{E})$.

We just need to show that the above two mappings are mutual inverses. Mapping $c \in \mathcal{C}(\mathbb{E})$ to $(\pi_1^\dagger c, \pi_2 c)$ and then to $c' = \{(x, e) \in E.\ x \subseteq \pi_1^\dagger c$ and $e \in \pi_2 c\}$, we have at least $c \subseteq c'$. Given any $(x', e) \in c'$, we have $(x, e) \in c$ for some x, and so $x = x'$ by (7.16). Hence $c' = c$. Conversely, mapping the pair $(x_1, x_2) \in \mathcal{C}(\mathbb{E}_1) \times \mathcal{C}(\mathbb{E}_2)$ with $o_1 x_1 = \iota_2^\dagger x_2$ to $c = \{(x, e) \in E.\ x \subseteq x_1$ and

$e \in x_2\}$, and then to $(\pi_1^\dagger c, \pi_2 c)$, we have at least $\pi_1^\dagger c \subseteq x_1$ and $\pi_2 c \subseteq x_2$. If $e \in x_2$, then $\iota_2 e \subseteq \iota_2^\dagger x_2 = o_1 x_1$, and so there exists a unique subconfiguration $x = x_1|_{\iota_2 e}$ of x_1 with $o_1 x = \iota_2 e$. Thus, $(x, e) \in E$ and so $(x, e) \in c$ such that $e \in \pi_2 c$. We conclude $\pi_2 c = x_2$. But now we have $o_1(\pi_1^\dagger c) = \iota_2^\dagger(\pi_2 c) = \iota_2^\dagger x_2$, but there is only one subset of x_1 with this property, namely x_1 itself. So $x_1 = \pi_1^\dagger c$ and we are done. □

Because of the way input and output morphisms compose, we have that the maps $\iota : \mathbb{E} \to \mathbb{P}$ and $o : \mathbb{E} \to \mathbb{R}$, given by $\iota = \iota_1^\dagger \circ \pi_1$ and $o = o_2 \circ \pi_2$, form a span $\langle \iota, \mathbb{E}, o\rangle : \mathbb{P} \to \mathbb{R}$ which we'll take as the composition of the two spans of (7.15).

Proposition 7.4 In the situation above, $\overline{\langle \iota, \mathbb{E}, o\rangle} \cong \overline{\langle \iota_2, \mathbb{E}_2, o_2\rangle} \circ \overline{\langle \iota_1, \mathbb{E}_1, o_2\rangle}$.

Proof. With $f_i = \overline{\langle \iota_i, \mathbb{E}_i, o_i\rangle}$, the composition $f_2 \circ f_1$ is given at (P, r) by the coend (5.10),

$$\int^{Q\in\mathbb{Q}_\perp} \lfloor(\mathit{curry}\, f_1)P\rfloor Q \times f_2(Q, r)\ . \tag{7.17}$$

Being a coend in set, this is isomorphic to a suitable quotient of

$$\bigcup_{Q\in\mathbb{Q}_\perp} \lfloor(\mathit{curry}\, f_1)P\rfloor Q \times \{Q\} \times f_2(Q, r)\ . \tag{7.18}$$

As f_2 is stable, we can use the minimal inputs of f_2 as representatives of the equivalence classes, and describe the composition as the profunctor

$$(P, r) \mapsto \{(x_1, x_2) : \begin{array}{l} x_1 \in \lfloor(\mathit{curry}\, f_1)P\rfloor Q_0 \\ x_2 \in f_2(Q_0, r) \text{ input-minimal} \end{array} \}\ . \tag{7.19}$$

By the properties of the representations of f_1 and f_2, this is isomorphic to the profunctor

$$(P, r) \mapsto \{(x_1, x_2) : \begin{array}{l} x_1 \in \mathcal{C}(\mathbb{E}_1) \text{ and } \iota_1^\dagger x_1 \subseteq P \\ x_2 \in \mathcal{C}(\mathbb{E}_2) \text{ and } o_2 x_2 = r \\ o_1 x_1 = \iota_2^\dagger x_2 \end{array} \}\ . \tag{7.20}$$

Now, using Proposition 7.3, this yields at each (P, r) at set isomorphic to

$$\{c \in \mathcal{C}(\mathbb{E}) : \iota^\dagger c \subseteq P \text{ and } oc = r\} \tag{7.21}$$

—as wanted. Naturality in P is immediate because each naturality square is built from inclusions and the above isomorphism. As for naturality in r, suppose that $r' \subseteq r$. If (x_1, x_2) is a pair in (7.20) with $o_2 x_2 = r$, then $x_2' = x_2|_{r'}$ is the unique subconfiguration of x_2 with $o_2 x_2' = r'$. This induces a unique subconfiguration $x_1' = x_1|_{\iota_2^\dagger x_2'}$ of x_1 with $o_1 x_1' = \iota_2^\dagger x_2'$. Since $\iota_1^\dagger x_1 \subseteq P$ we also have $\iota_1^\dagger x_1' \subseteq P$, and so (x_1', x_2') belongs to the image of (P, r') of the profunctor (7.20). We need to show that the corresponding configurations

c and c' with $(x_1, x_2) = (\pi_1^\dagger c, \pi_2 c)$ and $(x_1', x_2') = (\pi_1^\dagger c', \pi_2 c')$ are related as $c' = c|_{r'}$, and for this we just need to show that $c' \subseteq c$ and $oc' = r'$. For the former, we have $\pi_1^\dagger c' = x_1' \subseteq x_1 = \pi_1^\dagger c$ and $\pi_2 c' = x_2' \subseteq x_2 = \pi_2 c$. Hence, if $(x', e) \in c'$, then $(x, e) \in c$ for some x with $x, x' \subseteq x_1$ and so $x' = x$ by (7.16). So $c' \subseteq c$. The latter is obtained as $oc' = o_2 \circ \pi_2 c' = o_2 x_2' = r'$. $\square$

The identity profunctor $\mathbb{P} \mathrel{\longrightarrow\!\!\!\!\!+} \mathbb{P}$, given by $\mathbb{P}(-, +)$, is represented by the span $\langle \lceil - \rceil, \mathbb{P}, 1_\mathbb{P} \rangle$. Using this identity and the above composition, we expect **Spn** to be biequivalent to a suitable sub-bicategory of **Prof**.

7.3 Stable Denotational Semantics

We can mirror the constructions on maps of **Aff** from Section 2.3.2 using $\trianglelefteq$-continuous constructions on spans to get an event structure version of the presheaf denotational semantics of Affine HOPLA at first order. Spans $\langle \iota, \mathbb{E}, o \rangle : \mathbb{P} \to \mathbb{Q}$ represent maps $\mathbb{P} \to \mathbb{Q}$ of **Aff** via the equivalence $\mathbf{Aff}(\mathbb{P}, \mathbb{Q}) \simeq \mathbf{Prof}(\mathbb{P}_\perp, \mathbb{Q})$. We'll use the notation $\overline{\langle \iota, \mathbb{E}, o \rangle}$ also for the represented map of **Aff**.

Identity. The span representing the identity map $\mathbb{P} \to \mathbb{P}$ of **Aff** is given by $\langle \lceil - \rceil, \mathbb{P}, 1_\mathbb{P} \rangle : \mathbb{P} \to \mathbb{P}$.

Products. The product path order $\&_{i \in I} \mathbb{P}_i$ is represented using the sum $\Sigma_{i \in I} \mathbb{P}_i$ of event structures. Projections $\pi_j : \&_{i \in I} \mathbb{P}_i \to \mathbb{P}_j$ are represented by spans $\langle \lceil (j, -) \rceil, \mathbb{P}_j, 1_{\mathbb{P}_j} \rangle$. Given a family of spans $\langle \iota_i, \mathbb{E}_i, o_i \rangle$ representing maps $f_i : \mathbb{P} \to \mathbb{P}_i$, $i \in I$ in **Aff**, we can construct a representation of the unique-up-to-isomorphism map $\langle f_i \rangle_{i \in I} : \mathbb{P} \to \&_{i \in I} \mathbb{P}_i$ in the form of a span $\langle \iota, \Sigma_{i \in I} \mathbb{E}_i, o \rangle$ by defining $\iota(i, e) = \iota_i e$ and $o(i, e) = (i, o_i e)$.

The empty product is given by the empty event structure $\mathbb{O}$ and the unique map $\varnothing_\mathbb{P} : \mathbb{P} \to \mathbb{O}$ of **Aff** is represented by the empty span $\langle \varnothing, \mathbb{O}, \varnothing \rangle$.

The nondeterministic sum map $\Sigma : \&_{i \in I} \mathbb{P} \to \mathbb{P}$ is represented by the span $\langle \lceil - \rceil, \Sigma_{i \in I} \mathbb{P}, o \rangle$ with $o(i, e) = e$.

Tensor. Given spans $\langle \iota_i, \mathbb{E}_i, o_i \rangle : \mathbb{P}_i \to \mathbb{Q}_i$ representing the maps $f_i : \mathbb{P}_i \to \mathbb{Q}_i$ for $i = 1, 2$, we obtain a representation of the tensor product $f_1 \otimes f_2 : \mathbb{P}_1 \otimes \mathbb{P}_2 \to \mathbb{Q}_1 \otimes \mathbb{Q}_2$ using the span $\langle \iota_1 \otimes \iota_2, \mathbb{E}_1 \otimes \mathbb{E}_2, o_1 \otimes o_2 \rangle$ where $(\iota_1 \otimes \iota_2)(i, e) = \{i\} \times \iota_i e$ and $(o_1 \otimes o_2)(i, e) = (i, o_i e)$.

The unit for the tensor construction is the empty event structure $\mathbb{O}$. Right identities $r_\mathbb{P}^{\mathbf{Aff}} : \mathbb{P} \otimes \mathbb{O} \to \mathbb{P}$ are represented by spans $\langle \lceil (1, -) \rceil, \mathbb{P}, 1_\mathbb{P} \rangle$. Left identities are represented similarly.

The symmetry map $s_{\mathbb{P}_0, \mathbb{P}_1}^{\mathbf{Aff}} : \mathbb{P}_0 \otimes \mathbb{P}_1 \to \mathbb{P}_1 \otimes \mathbb{P}_0$ is represented by the span $\langle \lceil - \rceil, \mathbb{P}_0 \otimes \mathbb{P}_1, (1 - i, -) \rangle$.

The weak diagonals $\delta_\mathbb{P} : \mathbb{P} \to \mathbb{P} \otimes \mathbb{P}$ are represented using spans $\langle \iota, \mathbb{P} + \mathbb{P}, o \rangle : \mathbb{P} \to \mathbb{P} \otimes \mathbb{P}$ defined by $\iota(i, e) = \lceil e \rceil$ and $o(i, e) = (i, e)$.

Sums. The sum type path order $\Sigma_{\alpha\in A}\mathbb{P}_\alpha$ and its projections are represented in the same way as for the product above. Injections $in_\beta : \mathbb{P}_\beta \to \Sigma_{\alpha\in A}\mathbb{P}_\alpha$ are represented by spans $\langle\lceil - \rceil, \mathbb{P}_\beta, (\beta, -)\rangle$.

Prefixing. The prefix type path order $\mathbb{P}_\perp$ is represented by the lifting of the corresponding event structure. The unit $\eta_\mathbb{P} : \mathbb{P} \to \mathbb{P}_\perp$ of **Aff** is represented by the span $\langle\iota, \mathbb{P}_\perp, 1_{\mathbb{P}_\perp}\rangle$ with ι the obvious inclusion of events of $\mathbb{P}_\perp$ as configurations of $\mathbb{P}$. Conversely, the counit $\varepsilon_\mathbb{P} : \mathbb{P}_\perp \to \mathbb{P}$ is represented by the span $\langle\lfloor\lceil - \rceil\rfloor, \mathbb{P}, 1_\mathbb{P}\rangle$.

If $f : \mathbb{P} \to \mathbb{Q}$ is represented by the span $\langle\iota, \mathbb{E}, o\rangle$, then $f_\perp : \mathbb{P}_\perp \to \mathbb{Q}_\perp$ is represented by the span $\langle\lfloor\iota^\dagger - \rfloor, \mathbb{E}_\perp, o\rangle : \mathbb{P}_\perp \to \mathbb{Q}_\perp$. Then a nonempty configuration $\lfloor x \rfloor \in \mathcal{C}(\mathbb{E}_\perp)$, corresponding to $x \in \mathcal{C}(\mathbb{E})$, has image $\lfloor\iota^\dagger x\rfloor$ under the input morphism and $\lfloor ox \rfloor$ under the output morphism.

The strength maps $str_{\mathbb{P},\mathbb{Q}} : \mathbb{P}\otimes\mathbb{Q}_\perp \to (\mathbb{P}\otimes\mathbb{Q})_\perp$ are represented by spans $\langle\iota_{str}, \mathbb{P}\otimes\mathbb{Q}, \lceil - \rceil\rangle$ with $\iota_{str}(1, e) = \lceil e \rceil \otimes \{\varnothing\}$ and $\iota_{str}(2, e) = \perp \otimes \lceil e \rceil$. With this definition, a nonempty configuration $P \otimes Q$ of the vertex has image $P \otimes !Q$ under $\iota^\dagger_{str}$.

Recursive definitions. By the remarks in the proof of Theorem 6.7, the least fixed-points in $\mathbf{Spn}(\mathbb{P}, \mathbb{P})$ yield the same results as ω-colimits in $\mathbf{Aff}(\mathbb{P}, \mathbb{P})$ for the constructions above.

With $\Gamma \vdash t : \mathbb{Q}$ we'll write $\mathcal{P}[\![t]\!]$ for the map $\Gamma \to \mathbb{Q}$ of **Aff** obtained by the same rules that defined the path semantics in Section 4.1, but now using the presheaf version. Similarly, we'll write $\mathcal{E}[\![t]\!]$ for the span $\Gamma \to \mathbb{Q}$ obtained by the rules, interpreting morphisms as spans. By a straightforward induction on the typing derivation of t we then have

Proposition 7.5 Suppose t is a well-formed term of the first-order fragment of Affine HOPLA. Then $\overline{\mathcal{E}[\![t]\!]} \cong \mathcal{P}[\![t]\!]$.

Let $\langle\iota, \mathbb{E}, o\rangle$ be the span interpreting $\Gamma \vdash t : \mathbb{P}$ with $\Gamma \equiv x_1 : \mathbb{P}_1, \ldots, x_k : \mathbb{P}_k$. If $\Gamma \equiv P_1 \otimes \cdots \otimes P_k$ with $P_j : \mathbb{P}_j$ for $1 \le j \le k$, and $P : \mathbb{P}$ we'll write $\mathcal{E}[\![t]\!](\gamma, P)$ for the set of configurations $x \in \mathbb{E}$ with $\iota^\dagger x = \gamma$ and $ox = P$. Propositions 7.6 and 7.7 below provide a characterisation of these sets.

Proposition 7.6 (i) For any $\Gamma \vdash t : \mathbb{P}$, we have $\mathcal{E}[\![t]\!](\perp, \perp) = \{\varnothing\}$.

(ii) $\mathcal{E}[\![x : \mathbb{P} \vdash x : \mathbb{P}]\!](P, P) = \{P\}$ while $\mathcal{E}[\![x : \mathbb{P} \vdash x : \mathbb{P}]\!](P, P') = \varnothing$ if $P \neq P'$.

(iii) If $\Gamma, x : \mathbb{P} \vdash t : \mathbb{Q}$ is obtained from $\Gamma \vdash t : \mathbb{Q}$ by weakening, then $\mathcal{E}[\![\Gamma, x : \mathbb{P} \vdash t : \mathbb{Q}]\!](\gamma \otimes \perp, Q) \cong \mathcal{E}[\![\Gamma \vdash t : \mathbb{Q}]\!](\gamma, Q)$, while $\mathcal{E}[\![\Gamma, x : \mathbb{P} \vdash t : \mathbb{Q}]\!](\gamma \otimes p, Q) = \varnothing$ for any $p : \mathbb{P}$.

(iv) If $\Gamma, x : \mathbb{P}, y : \mathbb{Q}, \Delta \vdash t : \mathbb{R}$ is obtained from $\Gamma, y : \mathbb{Q}, x : \mathbb{P}, \Delta \vdash t : \mathbb{R}$ by exchange, then $\mathcal{E}[\![\Gamma, x : \mathbb{P}, y : \mathbb{Q}, \Delta \vdash t : \mathbb{R}]\!](\gamma \otimes P \otimes Q \otimes \delta, R) \cong \mathcal{E}[\![\Gamma, y : \mathbb{Q}, x : \mathbb{P}, \Delta \vdash t : \mathbb{R}]\!](\gamma \otimes Q \otimes P \otimes \delta, R)$.

Proof. (i) The empty configuration is the unique configuration with empty image under an output morphism. (ii) The identity span has vertex $\mathbb{P}$ and input and output morphisms are both identity maps on configurations. (iii) If $\Gamma \vdash t : \mathbb{Q}$ is interpreted by $\langle \iota, \mathbb{E}, o \rangle$, then using weakening, $\Gamma, x : \mathbb{P} \vdash t : \mathbb{Q}$ is interpreted using the composition

$$\begin{tikzcd}[column sep=small] & \Gamma \otimes \mathbb{O} \arrow[dl, "\lceil - \rceil \otimes \varnothing"'] \arrow[dr, "1_\Gamma \otimes \varnothing"] & & \Gamma \arrow[dl, "\lceil (1,-) \rceil"'] \arrow[dr, "1_\Gamma"] & & \mathbb{E} \arrow[dl, "\iota"'] \arrow[dr, "o"] & \\ \Gamma \otimes \mathbb{P} & & \Gamma \otimes \mathbb{O} & & \Gamma & & \mathbb{Q} \end{tikzcd} \tag{7.22}$$

By Proposition 7.3, the configurations of the composition correspond bijectively to triples $((\gamma_1, \perp), \gamma_2, c)$, with γ_1 and γ_2 configurations of Γ and $c \in \mathcal{C}(\mathbb{E})$, satisfying $(1_\Gamma \otimes \varnothing)(\gamma_1, \varnothing) = \lceil (1,-) \rceil^\dagger \gamma_2$ and $1_\Gamma \gamma_2 = \iota^\dagger c$. Then $\gamma_1 = \lceil - \rceil^\dagger \gamma_2 = \gamma_2 = \iota^\dagger c$. Thus, the configurations of the composition are in bijective correspondence with the configurations of $\mathbb{E}$. Let $c \in \mathcal{C}(\mathbb{E})$. The configuration of the composition corresponding to c is sent by the input morphism to $(\lceil - \rceil \otimes \varnothing)^\dagger (\iota^\dagger c \otimes \perp) = \iota^\dagger c \otimes \perp$ and by the output morphism to oc as wanted. (iv) Similar to (iii). □

Proposition 7.7

$$\begin{aligned}
\mathcal{E}[\![rec\, x.t]\!](\gamma, p) &\cong \mathcal{E}[\![t[rec\, x.t/x]]\!](\gamma, p) \\
\mathcal{E}[\![\Sigma_{i \in I} t_i]\!](\gamma, p) &\cong \Sigma_{i \in I} \mathcal{E}[\![t_i]\!](\gamma, p) \\
\mathcal{E}[\![t \otimes u]\!](\gamma \otimes \delta, P \otimes Q) &\cong \mathcal{E}[\![t]\!](\gamma, P) \times \mathcal{E}[\![u]\!](\delta, Q) \\
\mathcal{E}[\![[u > x \otimes y \Rightarrow t]]\!](\gamma \otimes \delta, r) &\cong \Sigma_{P,Q} \mathcal{E}[\![t]\!](\gamma \otimes P \otimes Q, r) \times \mathcal{E}[\![u]\!](\delta, P \otimes Q) \\
\mathcal{E}[\![\beta t]\!](\gamma, \beta p) &\cong \mathcal{E}[\![t]\!](\gamma, p) \\
\mathcal{E}[\![\pi_\beta t]\!](\gamma, p) &\cong \mathcal{E}[\![t]\!](\gamma, \beta p) \\
\mathcal{E}[\![!t]\!](\gamma, !P) &\cong \mathcal{E}[\![t]\!](\gamma, P) \\
\mathcal{E}[\![[u > !x \Rightarrow t]]\!](\gamma \otimes \delta, q) &\cong \Sigma_P \mathcal{E}[\![t]\!](\gamma \otimes P, q) \times \mathcal{E}[\![u]\!](\delta, !P) \\
\mathcal{E}[\![abs\, t]\!](\gamma, abs\, p) &\cong \mathcal{E}[\![t]\!](\gamma, p) \\
\mathcal{E}[\![rep\, t]\!](\gamma, p) &\cong \mathcal{E}[\![t]\!](\gamma, abs\, p)
\end{aligned} \tag{7.23}$$

Proof. We give the proof for prefix match, the other cases are handled similarly. In Section 4.1 we defined $[\![[u > !x \Rightarrow t]]\!]$ to be the composition $\varepsilon_\mathbb{Q} \circ [\![t]\!]_\perp \circ str_{\Gamma,\mathbb{P}} \circ (1_\Gamma \otimes [\![u]\!])$. Thus, if $\mathcal{E}[\![u]\!] = \langle \iota_u, \mathbb{E}_u, o_u \rangle$ and $\mathcal{E}[\![t]\!] = \langle \iota_t, \mathbb{E}_t, o_t \rangle$, we obtain $\mathcal{E}[\![[u > !x \Rightarrow t]]\!]$ by composition of four spans, the first two of which are

$$\begin{tikzcd}[column sep=small] & \Gamma \otimes \mathbb{E}_u \arrow[dl, "\lceil - \rceil \otimes \iota_u"'] \arrow[dr, "1_\Gamma \otimes o_u"] & & \Gamma \otimes \mathbb{P} \arrow[dl, "\iota_{str}"'] \arrow[dr, "\lceil - \rceil"] & \\ \Gamma \otimes \Delta & & \Gamma \otimes \mathbb{P}_\perp & & (\Gamma \otimes \mathbb{P})_\perp \end{tikzcd} \tag{7.24}$$

By Proposition 7.3, the nonempty configurations of the composition correspond to pairs $((\gamma_1, y), (\gamma_2, P))$ where $(1_\Gamma \otimes o_u)(\gamma_1 \otimes y) = \iota^\dagger_{str}(\gamma_2, P) = \gamma_2 \otimes !P$. So $\gamma_1 = \gamma_2$ and $o_u y = !P$. Thus, the nonempty configurations are in bijective correspondence with pairs (γ, y) with γ a configuration of Γ and $y \in \mathcal{C}(\mathbb{E}_u)$ such that $o_u y = !P$ for some P. Such a pair is associated with input $\gamma \otimes \iota^\dagger_u y$ and output $\lfloor \gamma \otimes P \rfloor$.

The two remaining spans are given by

$$\begin{tikzcd}
 & (\mathbb{E}_t)_\perp \arrow[dl, "\lfloor \iota^\dagger_t - \rfloor"'] \arrow[dr, "o_t"] & & \mathbb{Q} \arrow[dl, "\lfloor \lceil - \rceil \rfloor"'] \arrow[dr, "1_\mathbb{Q}"] & \\
(\Gamma \otimes \mathbb{P})_\perp & & \mathbb{Q}_\perp & & \mathbb{Q}
\end{tikzcd} \tag{7.25}$$

Again by Proposition 7.3, the nonempty configurations of this span are in bijective correspondence with pairs $(\lfloor x \rfloor, q)$ with $x \in \mathcal{C}(\mathbb{E}_t)$ and $q : \mathbb{Q}$ such that $o_t x = q$. Thus, the nonempty configurations are in bijective correspondence with the configurations of $\mathbb{E}_t$, and $x \in \mathcal{C}(\mathbb{E}_t)$ is associated with input $\lfloor \iota_t x \rfloor$ and output $o_t x$.

By a final use of Proposition 7.3, we see that the nonempty configurations of $\mathcal{E}[\![[u > !x \Rightarrow t]]\!]$ are in bijective correspondence with pairs $((\gamma, y), x)$ with $o_u y = !P$ and $\lceil - \rceil(\gamma \otimes P) = \lfloor \gamma \otimes P \rfloor = \lfloor \iota_t x \rfloor$, so that $\iota^\dagger_t x = \gamma \otimes P$. Thus, the nonempty configurations are in bijective correspondence with pairs (x, y) with $x \in \mathcal{C}(\mathbb{E}_t)$ and $y \in \mathcal{C}(\mathbb{E}_u)$ such that $\iota^\dagger_t x = \gamma \otimes P$ and $o_u x = !P$ for some γ, P. The associated input is given by $\gamma \otimes \iota^\dagger_u y$ and the output is given by $o_t x$. We are done. $\square$

7.4 Stable Operational Semantics

The above propositions suggest an alternative operational semantics for first-order Affine HOPLA, obviating the need for the environments of Section 4.4 by exploiting stability. It is at the cost of having transitions between *open* terms. Consider an open term $\Gamma \vdash t : \mathbb{P}$ and its interpretation as a span $\mathcal{E}[\![t]\!] = \langle \iota, \mathbb{E}, o \rangle : \Gamma \to \mathbb{P}$. For each element x of the set $\mathcal{E}[\![t]\!](\gamma, P)$ we'll have a derivation of a transition of the form $\gamma \vdash t \xrightarrow{P} t'$. Moreover, $\mathcal{E}[\![t']\!]$ will to within isomorphism be obtained from $\mathcal{E}[\![t]\!]$ as the "span above x", defined in Section 7.4.1.

Thus, the operational rules not only derive transitions but also a syntactic representation of the unique minimal input needed for each derived transition. Hence we call the operational semantics "stable".

7.4.1 The "Above" Operation

Given $\mathbb{E} = (E, \leq, \smile)$ and a configuration $x \in \mathcal{C}(\mathbb{E})$, define $\mathbb{E}/x$ to have

Events: $E/x = \{e \in E : \forall e' \in x.\ e \frown e'\}$

Causality: obtained by restricting $\leq$

Conflict: obtained by restricting $\smile$

A subset $x' \subseteq E/x$ is a configuration iff $x \cup x'$ is a configuration of $\mathbb{E}$. Moreover, x and x' are disjoint, so the poset $\mathcal{C}(\mathbb{E}/x)$ is isomorphic to the poset $(\mathcal{C}(\mathbb{E}))/x$ of elements of $\mathcal{C}(\mathbb{E})$ above x. This construction extends to spans:

Lemma 7.8 Given $s = \langle \iota, \mathbb{E}, o\rangle : \mathbb{P} \to \mathbb{Q}$ and $x \in \mathcal{C}(\mathbb{E})$ we get a span $s/x = \langle \iota/x, \mathbb{E}/x, o/x\rangle : \mathbb{P}/(\iota^\dagger x) \to \mathbb{Q}/(ox)$ by taking $(\iota/x)e = \iota e \setminus \iota^\dagger x$ and $(o/x)e = oe$.

Proof. Let e be an event of $\mathbb{E}/x$. We must show that $(\iota/x)e = \iota e \setminus \iota^\dagger x$ is a configuration of $\mathbb{P}/(\iota^\dagger x)$. Now P is a configuration here iff $\iota^\dagger x \cup P$ is a configuration of $\mathbb{P}$. We have $\iota^\dagger x \cup (\iota e \setminus \iota^\dagger x) = \iota^\dagger x \cup \iota e = \iota^\dagger(x \cup \{e\})$. Since ι is an input morphism $\mathbb{E} \to \mathbb{P}$, this is a configuration of $\mathbb{P}$ if $x \cup \{e\}$ is consistent, which it is since $e \frown_{\mathbb{E}} e'$ for all $e' \in x$. So ι/x maps events of $\mathbb{E}/x$ to configurations of $\mathbb{P}/(\iota^\dagger x)$. That ι/x is an input morphism follows immediately from ι being an input morphism.

We must also show that $(o/x)e = oe$ is an event of $\mathbb{Q}/(ox)$. So given any $e_{\mathbb{Q}} \in ox$, we must show that $oe \frown_{\mathbb{Q}} e_{\mathbb{Q}}$. Now, $e_{\mathbb{Q}} = oe'$ for some $e' \in x$, and by definition of $\mathbb{E}/x$, we have $e \frown_{\mathbb{E}} e'$ from which $oe \frown_{\mathbb{Q}} oe' = e_{\mathbb{Q}}$ follows as o is an output morphism. So o/x sends events of $\mathbb{E}/x$ to events of $\mathbb{Q}/(ox)$. Moreover, o/x preserves causality and $\frown$ because o does, and finally, if $e_{\mathbb{Q}} \in \lceil (o/x)e\rceil$, then $e_{\mathbb{Q}} \in \lceil oe\rceil \setminus (ox)$ and so $e_{\mathbb{Q}} \in o\lceil e\rceil \setminus (ox)$ which means that $e_{\mathbb{Q}} \in (o/x)\lceil e\rceil$ as wanted. □

For a configuration $P : \mathbb{P}$ the event structure $\mathbb{P}/P$ is again the interpretation of a type for which $\mathbb{P}/P$ is taken to be a synonym according to

$$\begin{aligned}
\mathbb{P}/\bot &\equiv_{\mathrm{def}} \mathbb{P} \\
(\mathbb{P} \otimes \mathbb{Q})/(P \otimes Q) &\equiv_{\mathrm{def}} (\mathbb{P}/P) \otimes (\mathbb{Q}/Q) \\
(\Sigma_{\alpha\in A}\mathbb{P}_\alpha)/(\beta p) &\equiv_{\mathrm{def}} \mathbb{P}_\beta/p \\
\mathbb{P}_\bot/(!P) &\equiv_{\mathrm{def}} \mathbb{P}/P \\
(\mu_j \vec{T}.\vec{\mathbb{T}})/(abs\, p) &\equiv_{\mathrm{def}} (\mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}])/p
\end{aligned} \tag{7.26}$$

7.4.2 Operational Rules

Consider an environment list $\Gamma \equiv x_1 : \mathbb{P}_1, \ldots, x_k : \mathbb{P}_k$. We'll write $\gamma : \Gamma$ if the path list $\gamma \equiv x_1 : P_1, \ldots, x_k : P_k$ has $P_j : \mathbb{P}_j$ for $1 \leq j \leq k$. In this case, we shall write Γ/γ for the environment list $x_1 : \mathbb{P}_1/P_1, \ldots, x_k : \mathbb{P}_k/P_k$, and confuse γ with the configuration $P_1 \otimes \cdots \otimes P_k$ of $\mathbb{P}_1 \otimes \cdots \otimes \mathbb{P}_k$ which in turn is confused with Γ.

A reasonable way to define the operational rules would be to have transitions of the form $\gamma \vdash t \xrightarrow{P} t'$ with $\Gamma \vdash t : \mathbb{P}$ and $\gamma : \Gamma$ and $P : \mathbb{P}$. However, then one would need rules like those below for handling weakening and exchange.

$$\frac{\gamma \vdash t \xrightarrow{Q} t'}{\gamma, x : \bot \vdash t \xrightarrow{Q} t'} \quad \frac{\gamma, y : Q, x : P, \delta \vdash t \xrightarrow{R} t'}{\gamma, x : P, y : Q, \delta \vdash t \xrightarrow{R} t'} \tag{7.27}$$

Clearly, with such rules we have no hope of obtaining a bijective correspondence between derivations and configurations. Therefore, we'll use a global type environment Φ, assigning types to *all* variables, and corresponding path maps $\phi : \Phi$ with ϕx a path of type Φx for all variables x. The everywhere $\bot$ path map is itself written $\bot$, and we have $\bot : \Phi$ for any Φ. We'll write $\Gamma \subset \Phi$ if Γ is an environment list which agrees with Φ in the sense that if $x : \mathbb{P}$ occurs in Γ, then $\Phi(x) = \mathbb{P}$. Any path map $\gamma : \Phi$ which assigns $\bot$ to all variables not mentioned in Γ can be viewed as a path list $\gamma : \Gamma$, and conversely, any path list $\gamma : \Gamma$ extends to a path map $\gamma : \Phi$ by mapping all variables not in Γ to $\bot$. If $\gamma : \Phi$ and $\delta : \Phi$ restrict to $\gamma : \Gamma$ and $\delta : \Delta$ with Γ and Δ disjoint, we'll write $\gamma, \delta : \Phi$ for the path map obtained from the concatenation of the lists γ and δ.

The operational rules are shown in Figure 7.1. The semantics is parameterised by a global typing environment Φ which is left implicit. For each such Φ, the transition relation $\gamma \vdash \mathbb{P} : t \xrightarrow{P} t'$ is defined for path maps γ, terms t, and paths P such that for some $\Gamma \subset \Phi$, we have $\gamma : \Gamma$ and $\Gamma \vdash t : \mathbb{P}$. The rules are type-correct:

Proposition 7.9 Suppose $\gamma \vdash \mathbb{P} : t \xrightarrow{P} t'$ using global typing environment Φ. For any $\Gamma \subset \Phi$ such that $\gamma : \Gamma$ and $\Gamma \vdash t : \mathbb{P}$, we have $\Gamma/\gamma \vdash t' : \mathbb{P}/P$.

Proof. By rule-induction on the operational rules.

Bottom. Consider the transition $\bot \vdash \mathbb{P} : t \xrightarrow{\bot} t$. If $\Gamma \subset \Phi$ and $\Gamma \vdash t : \mathbb{P}$ then since $\Gamma/\bot \equiv \Gamma$ and $\mathbb{P}/\bot \equiv \mathbb{P}$, we are done.

Variable. Consider the transition $x : p \vdash \mathbb{P} : x \xrightarrow{p} x$. If $\Gamma \subset \Phi$ with $\Gamma \vdash x : \mathbb{P}$, then $\Gamma \equiv \Gamma_1, x : \mathbb{P}, \Gamma_2$ for some Γ_1, Γ_2. We have $p : \mathbb{P}$ and so $\Gamma_1/\bot, x : \mathbb{P}/p, \Gamma_2/bot \vdash x : \mathbb{P}/p$ as wanted.

Tensor. Suppose that $\gamma, \delta \vdash \mathbb{P} \otimes \mathbb{Q} : t \otimes u \xrightarrow{P \otimes Q} t' \otimes u'$ is derived from $\gamma \vdash \mathbb{P} : t \xrightarrow{P} t'$ and $\delta \vdash \mathbb{Q} : u \xrightarrow{Q} u'$. By the induction hypotheses we get $\Gamma/\gamma \vdash t' : \mathbb{P}/P$ and $\Delta/\delta \vdash u' : \mathbb{Q}/Q$ and so by the typing rule for tensor, $\Gamma/\gamma, \Delta/\delta \vdash t' \otimes u' : \mathbb{P}/P \otimes \mathbb{Q}/Q$. Since $\mathbb{P}/P \otimes \mathbb{Q}/Q \equiv (\mathbb{P} \otimes \mathbb{Q})/(P \otimes Q)$ we are done.

Tensor match. Suppose that $\gamma, \delta \vdash \mathbb{R} : [u > x \otimes y \Rightarrow t] \xrightarrow{r} [u' > x \otimes y \Rightarrow t']$ is derived from $\gamma, x : P, y : Q \vdash \mathbb{R} : t \xrightarrow{r} t'$ and $\delta \vdash \mathbb{P} \otimes \mathbb{Q} : u \xrightarrow{P \otimes Q} u'$.

$$\frac{}{\perp \vdash \mathbb{P} : t \xrightarrow{\perp} t} \qquad \frac{}{x : p \vdash \mathbb{P} : x \xrightarrow{p} x}$$

$$\frac{\gamma \vdash \mathbb{P} : t[rec\, x.t/x] \xrightarrow{p} t'}{\gamma \vdash \mathbb{P} : rec\, x.t \xrightarrow{p} t'} \qquad \frac{\gamma \vdash \mathbb{P} : t_j \xrightarrow{p} t'}{\gamma \vdash \mathbb{P} : \Sigma_{i \in I} t_i \xrightarrow{p} t'}\ j \in I$$

$$\frac{\gamma \vdash \mathbb{P} : t \xrightarrow{P} t' \quad \delta \vdash \mathbb{Q} : u \xrightarrow{Q} u' \quad (P, Q) \neq (\perp, \perp)}{\gamma, \delta \vdash \mathbb{P} \otimes \mathbb{Q} : t \otimes u \xrightarrow{P \otimes Q} t' \otimes u'}$$

$$\frac{\gamma, x : P, y : Q \vdash \mathbb{R} : t \xrightarrow{r} t' \quad \delta \vdash \mathbb{P} \otimes \mathbb{Q} : u \xrightarrow{P \otimes Q} u'}{\gamma, \delta \vdash \mathbb{R} : [u > x \otimes y \Rightarrow t] \xrightarrow{r} [u' > x \otimes y \Rightarrow t']}$$

$$\frac{\gamma \vdash \mathbb{P}_\beta : t \xrightarrow{p} t'}{\gamma \vdash \Sigma_{\alpha \in A} \mathbb{P}_\alpha : \beta t \xrightarrow{\beta p} t'} \qquad \frac{\gamma \vdash \Sigma_{\alpha \in A} \mathbb{P}_\alpha : t \xrightarrow{\beta p} t'}{\gamma \vdash \mathbb{P}_\beta : \pi_\beta t \xrightarrow{p} t'}$$

$$\frac{\gamma \vdash \mathbb{P} : t \xrightarrow{P} t'}{\gamma \vdash \mathbb{P}_\perp : !t \xrightarrow{!P} t'} \qquad \frac{\gamma, x : P \vdash \mathbb{Q} : t \xrightarrow{q} t' \quad \delta \vdash \mathbb{P}_\perp : u \xrightarrow{!P} u'}{\gamma, \delta \vdash \mathbb{Q} : [u > !x \Rightarrow t] \xrightarrow{q} t'[u'/x]}$$

$$\frac{\gamma \vdash \mathbb{T}_j[\mu \vec{T}.\vec{\mathbb{T}}/\vec{T}] : t \xrightarrow{p} t'}{\gamma \vdash \mu_j \vec{T}.\vec{\mathbb{T}} : abs\, t \xrightarrow{abs\, p} t'} \qquad \frac{\gamma \vdash \mu_j \vec{T}.\vec{\mathbb{T}} : t \xrightarrow{abs\, p} t'}{\gamma \vdash \mathbb{T}_j[\mu \vec{T}.\vec{\mathbb{T}}/\vec{T}] : rep\, t \xrightarrow{p} t'}$$

Figure 7.1: Stable operational semantics for first-order Affine HOPLA

By the induction hypotheses for t, we get $\Gamma/\gamma, x : \mathbb{P}/P, y : \mathbb{Q}/Q \vdash t' : \mathbb{R}/r$, while the hypothesis for u yields $\Delta/\delta \vdash u' : (\mathbb{P} \otimes \mathbb{Q})/(P \otimes Q)$. Since $(\mathbb{P} \otimes \mathbb{Q})/(P \otimes Q) \equiv (\mathbb{P}/P) \otimes (\mathbb{Q}/Q)$, the typing rule for tensor match yields $\Gamma/\gamma, \Delta/\delta \vdash [u' > x \otimes y \Rightarrow t'] : \mathbb{R}/r$ as wanted.

Prefixing. Suppose that $\gamma \vdash \mathbb{P}_\perp : !t \xrightarrow{!P} t'$ is derived from $\gamma \vdash \mathbb{P} : t \xrightarrow{P} t'$. By the induction hypothesis, we get $\Gamma/\gamma \vdash t' : \mathbb{P}/P$ and since $\mathbb{P}/P \equiv \mathbb{P}_\perp/(!P)$ we are done.

Prefix match. Suppose that $\gamma, \delta \vdash \mathbb{Q} : [u > !x \Rightarrow t] \xrightarrow{q} t'[u'/x]$ is derived from $\gamma, x : P \vdash \mathbb{Q} : t \xrightarrow{q} t'$ and $\delta \vdash \mathbb{P}_\perp : u \xrightarrow{!P} u'$. By the induction hypothesis for t, we get $\Gamma/\gamma, x : \mathbb{P}/P \vdash t' : \mathbb{Q}/q$, while the hypothesis for u yields $\Delta/\delta \vdash u' : \mathbb{P}_\perp/(!P)$. As $\mathbb{P}_\perp/(!P) \equiv \mathbb{P}/P$, the substitution lemma yields $\Gamma/\gamma, \Delta/\delta \vdash t'[u'/x] : \mathbb{Q}/q$ as wanted.

The remaining rules are handled similarly. □

A strong correspondence result relating derivations to configurations is obtained directly from Propositions 7.6 and 7.7.

Theorem 7.10 (Strong Correspondence) Let $\Gamma \vdash t : \mathbb{P}$ with $\Gamma \subset \Phi$. There is a bijection between the set $D_{\gamma,t,P}$ of derivations of $\gamma \vdash \mathbb{P} : t \xrightarrow{P} t'$, for some t', using Φ and the set $\mathcal{E}[\![\Gamma \vdash t : \mathbb{P}]\!](\gamma, P)$.

Proof. For any $\Gamma \vdash t : \mathbb{P}$ there is a unique derivation of $\bot \vdash \mathbb{P} : t \xrightarrow{\bot} t$, because using any other last rule than the axiom for $\bot$-transitions, the path on the transition is non-$\bot$. This derivation is mapped to the unique element of $\mathcal{E}[\![t]\!](\bot, \bot)$ cf. Proposition 7.6, part (i).

Hence, we just need to relate other derivations to nonempty configurations. Recursion is treated using finite unfoldings as in Section 4.4.1 and we omit those details here. The proof otherwise proceeds by structural induction on t:

Identity. For any $p : \mathbb{P}$ there is a unique derivation of $x : p \vdash \mathbb{P} : x \xrightarrow{p} x$. Writing γ for the path list $\gamma : \Gamma$ obtained by extending $x : p$, we map this derivation to the unique configuration corresponding to p in $\mathcal{E}[\![\Gamma \vdash x : \mathbb{P}]\!](\gamma, p)$ cf. Proposition 7.6, parts (ii) and (iii).

Nondeterministic sum. We have

$$\begin{array}{ll} D_{\gamma,\Sigma_{i\in I}t_i,p} & \\ \cong \Sigma_{i\in I} D_{\gamma,t_i,p} & \\ \cong \Sigma_{i\in I}\mathcal{E}[\![t_i]\!](\gamma, p) & \text{(ind. hyp.)} \\ \cong \mathcal{E}[\![\Sigma_{i\in I}t_i]\!](\gamma, p) & \text{(Prop. 7.7)} \end{array}$$

—as wanted.

Tensor. We have

$$\begin{array}{ll} D_{(\gamma,\delta),t\otimes u,P\otimes Q} & \\ \cong D_{\gamma,t,P} \times D_{\delta,u,Q} & \\ \cong \mathcal{E}[\![t]\!](\gamma, P) \times \mathcal{E}[\![u]\!](\delta, Q) & \text{(ind. hyp.)} \\ \cong \mathcal{E}[\![t \otimes u]\!](P, Q) & \text{(Prop. 7.7)} \end{array}$$

—as wanted.

Tensor match. We have

$$\begin{array}{ll} D_{(\gamma,\delta),[u>x\otimes y\Rightarrow t],r} & \\ \cong \Sigma_{P,Q} D_{(\gamma,x:P,y:Q),t,r} \times D_{\delta,u,P\otimes Q} & \\ \cong \Sigma_{P,Q}\mathcal{E}[\![t]\!](\gamma \otimes P \otimes Q, r) \times \mathcal{E}[\![u]\!](\delta, P \otimes Q) & \text{(ind. hyp.)} \\ \cong \mathcal{E}[\![[u > x \otimes y \Rightarrow t]]\!](\gamma \otimes \delta, r) & \text{(Prop. 7.7)} \end{array}$$

—as wanted.

Prefix. We have

$$\begin{array}{ll} D_{\gamma,!t,!P} & \\ \cong D_{\gamma,t,P} & \\ \cong \mathcal{E}[\![t]\!](\gamma, P) & \text{(ind. hyp.)} \\ \cong \mathcal{E}[\![!t]\!](\gamma, !P) & \text{(Prop. 7.7)} \end{array}$$

—as wanted.

Prefix match. We have

$$\begin{array}{ll} D_{(\gamma,\delta),[u>!x\Rightarrow t],q} & \\ \cong \Sigma_P D_{(\gamma,x:P),t,q} \times D_{\delta,u,!P} & \\ \cong \Sigma_P \mathcal{E}[\![t]\!](\gamma \otimes P, q) \times \mathcal{E}[\![u]\!](\delta, !P) & \text{(ind. hyp.)} \\ \cong \mathcal{E}[\![[u > !x \Rightarrow t]]\!](\gamma \otimes \delta, q) & \text{(Prop. 7.7)} \end{array}$$

—as wanted.

The remaining cases are treated similarly. □

The theorem says nothing about the successor term t' of transitions $\gamma \vdash t \xrightarrow{P} t'$. However, the next section shows that we are able to compose derivations and to restrict them to obtain derivations with smaller output. As this matches the corresponding operations on configurations, a soundness result will follow, cf. Proposition 7.12 below.

7.4.3 Operations on Derivations

We'll now consider composition and restriction as syntactic operations on derivations. The former corresponds to extension of configurations by the addition of more compatible events, and the latter corresponds to the restriction operation on configurations induced by output maps. For defining composition on derivations, we need a technical lemma:

Lemma 7.11 Suppose $\Gamma, x : \mathbb{P} \vdash t : \mathbb{Q}$. If $\Delta \vdash u : \mathbb{P}$ with Γ and Δ disjoint, then $\gamma, \delta \vdash \mathbb{Q} : t[u/x] \xrightarrow{Q} t''$ iff $\gamma, x : P \vdash \mathbb{Q} : t \xrightarrow{Q} t'$ and $\delta \vdash \mathbb{P} : u \xrightarrow{P} u'$, and moreover, $t'' \equiv t'[u'/x]$.

Proof. For $Q \not\equiv \perp$ the result is proved by structural induction on t using the induction hypothesis

> Suppose $\Gamma, x_1 : \mathbb{P}, \ldots, x_k : \mathbb{P} \vdash t : \mathbb{Q}$ with $\{x_1, \ldots, x_k\}$ not crossed in t. If $\Delta \vdash u : \mathbb{P}$ with Γ and Δ disjoint, then $\gamma, \delta \vdash \mathbb{Q} : t[u/x_1, \ldots, u/x_k] \xrightarrow{q} t''$ iff $\gamma, x_j : P, \perp \vdash \mathbb{Q} : t \xrightarrow{q} t'$, some $1 \leq j \leq k$, and $\delta \vdash \mathbb{P} : u \xrightarrow{P} u'$, and moreover, $t'' \equiv t'[u'/x_1, \ldots, u'/x_k]$.

For $Q \equiv \perp$, the result follows using the rule for $\perp$-transitions. □

Given $P : \mathbb{P}$ and $P' : (\mathbb{P}/P)$ we'll write $P \,;\, P'$ for the path of $\mathbb{P}$ obtained as the configuration $P \cup P'$. We treat ; as a syntactic operator on paths according to:

$$\begin{aligned} \perp \,;\, P \equiv_{\text{def}} P \,;\, \perp &\equiv_{\text{def}} P \\ (P \otimes Q) \,;\, (P' \otimes Q') &\equiv_{\text{def}} (P \,;\, P') \otimes (Q \,;\, Q') \\ (\beta p) \,;\, p' &\equiv_{\text{def}} \beta(p \,;\, p') \\ (!P) \,;\, p' &\equiv_{\text{def}} !(P \,;\, p') \\ (\mathit{abs}\, p) \,;\, p' &\equiv_{\text{def}} \mathit{abs}(p \,;\, p') \end{aligned} \tag{7.28}$$

Composition of paths extends to composition of path lists in the obvious way. Suppose d is a derivation of $\gamma \vdash \mathbb{P} : t \xrightarrow{P} t'$ and d' is a derivation of $\gamma' \vdash \mathbb{P} : t' \xrightarrow{P'} t''$. Then $d ; d'$ as defined in Figures 7.2 and 7.3 is a derivation of $\gamma ; \gamma' \vdash \mathbb{P} : t \xrightarrow{P;P'} t''$. To save space, typing information is omitted in the figures.

With d a derivation of $\gamma' \vdash \mathbb{P} : t \xrightarrow{P'} t''$ with $P \leq_{\mathbb{P}} P'$ we'll write $d|_P$ for the restricted derivation of $\gamma \vdash \mathbb{P} : t \xrightarrow{P} t'$, defined in Figures 7.4 and 7.5. Note that we have $d ; d' \equiv d''$ for some d' iff $d \equiv d''|_P$ for some P. The "residual" derivation d' can be obtained by reading the equations $d ; d' \equiv_{\mathrm{def}} d''$ defining composition as a definition of d'.

These operations allow us to extend the bijection θ from the strong correspondence result to a complete representation of spans using derivations. If $d'' = d ; d'$ then θd and $\theta d'$ are disjoint with $\theta d'' = \theta d \cup \theta d'$, and $\theta(d|_P) = (\theta d)|_P$. This is enough to determine spans up to isomorphism. Consider the interpretation $\langle \iota, \mathbb{E}, o\rangle$ of $\Gamma \vdash t : \mathbb{P}$. By the strong correspondence result, and the properties of output morphisms, events of $\mathbb{E}$ correspond to derivations $\gamma \vdash \mathbb{P} : t \xrightarrow{p} t'$ where p is a prime configuration of $\mathbb{P}$. The path list γ and label p yield the images of the event under ι and o. If events e and e' are represented by derivations d_e and $d_{e'}$, we have $e \leq e'$ iff $d_e \equiv d_{e'}|_{oe}$, and $e \frown e'$ iff there exists a derivation d with $d|_{oe} \equiv d_e$ and $d|_{oe'} \equiv d_{e'}$.

We can now prove

Proposition 7.12 (Soundness) Let $\Gamma \vdash t : \mathbb{P}$ with $\Gamma \subset \Phi$. Given any derivation d of $\gamma \vdash t \xrightarrow{P} t'$ using Φ we have $\mathcal{E}[\![\Gamma/\gamma \vdash t' : \mathbb{P}/P]\!] \cong \mathcal{E}[\![\Gamma \vdash t : \mathbb{P}]\!]/(\theta d)$.

Proof. Let $\mathcal{E}[\![\Gamma \vdash t : \mathbb{P}]\!]$ be the span $\langle \iota, \mathbb{E}, o\rangle$ and $\mathcal{E}[\![\Gamma/\gamma \vdash t' : \mathbb{P}/P]\!]$ the span $\langle \iota', \mathbb{E}', o'\rangle$. By the above, the latter is represented by the derivations d' of $\gamma' \vdash \mathbb{P}/P : t' \xrightarrow{P'} t''$. All such derivations can be precomposed with the derivation d and thereby represent precisely the configurations of $\mathbb{E}$ extending θd. Thus, we have $\mathbb{E}' \cong \mathbb{E}/(\theta d)$. Consider an event e of $\mathbb{E}'$. By the isomorphism $\mathbb{E}' \cong \mathbb{E}/(\theta d)$ we know that $x = \{e\} \cup \theta d$ belongs to $\mathbb{E}$ (using e also for the event of $\mathbb{E}$ corresponding to e of $\mathbb{E}'$). Hence we have a derivation d_x of $\gamma_x \vdash \mathbb{P} : t \xrightarrow{p_x} t_x$ representing x so that $\iota^\dagger x = \gamma_x$ and $ox = p_x$. This derivation restricts to d since x restricts to θd. Thus, we have a residual derivation of $\gamma_e \vdash \mathbb{P}/P : t' \xrightarrow{p_e} t_x$ with $\gamma ; \gamma_e \equiv \gamma_x$ and $P ; p_e \equiv p_x$, and we also have $oe = p_e$. Moreover, d_x restricts to $d_{\lceil e\rceil}$ as x restricts to $o\lceil e\rceil$. This means that we have a derivation $\gamma_{\lceil e\rceil} \vdash \mathbb{P} : t \xrightarrow{p_e} t_{\lceil e\rceil}$ with $\gamma_{\lceil e\rceil} \cup \gamma = \gamma_x$. Thus,

$$\begin{array}{c} \iota' e = \gamma_e = \gamma_x \setminus \gamma = \gamma_{\lceil e\rceil} \setminus \gamma = \iota e \setminus \iota^\dagger(\theta d) \qquad \text{and} \\ o' e = p_e = oe \ . \end{array} \tag{7.29}$$

Using Lemma 7.8, we are done. □

$$\frac{}{\perp \vdash t \xrightarrow{\perp} t} \; ; \; d \equiv_{\mathrm{def}} d \equiv_{\mathrm{def}} d \; ; \; \frac{}{\perp \vdash t \xrightarrow{\perp} t}$$

$$\frac{}{x : p \vdash x \xrightarrow{p} x} \; ; \; \frac{}{x : p' \vdash x \xrightarrow{p'} x} \equiv_{\mathrm{def}} \frac{}{x : p \; ; \; p' \vdash x \xrightarrow{p;p'} x}$$

$$\frac{\begin{array}{c} d \\ \gamma \vdash t[rec\,x.t/x] \xrightarrow{p} t' \end{array}}{\gamma \vdash rec\,x.t \xrightarrow{p} t'} \; ; \; \begin{array}{c} d' \\ \gamma' \vdash t' \xrightarrow{p'} t'' \end{array} \equiv_{\mathrm{def}} \frac{\begin{array}{c} d \; ; \; d' \\ \gamma \; ; \; \gamma' \vdash t[rec\,x.t/x] \xrightarrow{p;p'} t'' \end{array}}{\gamma \; ; \; \gamma' \vdash rec\,x.t \xrightarrow{p;p'} t''}$$

$$\frac{\begin{array}{c} d \\ \gamma \vdash t_j \xrightarrow{p} t' \end{array}}{\gamma \vdash \Sigma_{i \in I} t_i \xrightarrow{p} t'} \; j \in I \; ; \; \begin{array}{c} d' \\ \gamma' \vdash t' \xrightarrow{p'} t'' \end{array} \equiv_{\mathrm{def}} \frac{\begin{array}{c} d \; ; \; d' \\ \gamma \; ; \; \gamma' \vdash t_j \xrightarrow{p;p'} t'' \end{array}}{\gamma \; ; \; \gamma' \vdash \Sigma_{i \in I} t_i \xrightarrow{p;p'} t''} \; j \in I$$

$$\frac{\begin{array}{c} d_t \\ \gamma \vdash t \xrightarrow{P} t' \end{array} \quad \begin{array}{c} d_u \\ \delta \vdash u \xrightarrow{Q} u' \end{array}}{\gamma, \delta \vdash t \otimes u \xrightarrow{P \otimes Q} t' \otimes u'} \; ; \; \frac{\begin{array}{c} d'_t \\ \gamma' \vdash t' \xrightarrow{P'} t'' \end{array} \quad \begin{array}{c} d'_u \\ \delta' \vdash u' \xrightarrow{Q'} u'' \end{array}}{\gamma', \delta' \vdash t' \otimes u' \xrightarrow{P' \otimes Q'} t'' \otimes u''}$$

$$\equiv_{\mathrm{def}} \frac{\begin{array}{c} d_t \; ; \; d'_t \\ \gamma \; ; \; \gamma' \vdash t \xrightarrow{P;P'} t'' \end{array} \quad \begin{array}{c} d_u \; ; \; d'_u \\ \delta \; ; \; \delta' \vdash u \xrightarrow{Q;Q'} u'' \end{array}}{(\gamma \; ; \; \gamma'), (\delta \; ; \; \delta') \vdash t \otimes u \xrightarrow{(P;P') \otimes (Q;Q')} t'' \otimes u''}$$

$$\frac{\begin{array}{c} d_t \\ \gamma, x : P, y : Q \vdash t \xrightarrow{r} t' \end{array} \quad \begin{array}{c} d_u \\ \delta \vdash u \xrightarrow{P \otimes Q} u' \end{array}}{\gamma, \delta \vdash [u > x \otimes y \Rightarrow t] \xrightarrow{r} [u' > x \otimes y \Rightarrow t']}$$

$$; \; \frac{\begin{array}{c} d'_t \\ \gamma', x : P', y : Q' \vdash t' \xrightarrow{r'} t'' \end{array} \quad \begin{array}{c} d'_u \\ \delta' \vdash u' \xrightarrow{P' \otimes Q'} u'' \end{array}}{\gamma', \delta' \vdash [u' > x \otimes y \Rightarrow t'] \xrightarrow{r'} [u'' > x \otimes y \Rightarrow t'']}$$

$$\equiv_{\mathrm{def}} \frac{\begin{array}{c} d_t \; ; \; d'_t \\ (\gamma \; ; \; \gamma'), x : P \; ; \; P', y : Q \; ; \; Q' \vdash t \xrightarrow{r;r'} t'' \end{array} \quad \begin{array}{c} d_u \; ; \; d'_u \\ \delta \; ; \; \delta' \vdash u \xrightarrow{(P;P') \otimes (Q;Q')} u'' \end{array}}{(\gamma \; ; \; \gamma'), (\delta \; ; \; \delta') \vdash [u > x \otimes y \Rightarrow t] \xrightarrow{r;r'} [u'' > x \otimes y \Rightarrow t'']}$$

Figure 7.2: Composition of derivations

$$\frac{\overset{d}{\gamma \vdash t \xrightarrow{p} t'}}{\gamma \vdash \beta t \xrightarrow{\beta p} t'} \;;\; \overset{d'}{\gamma' \vdash t' \xrightarrow{p'} t''} \equiv_{\mathrm{def}} \frac{\overset{d\,;\,d'}{\gamma\,;\,\gamma' \vdash t \xrightarrow{p;p'} t''}}{\gamma\,;\,\gamma' \vdash \beta t \xrightarrow{\beta(p;p')} t''}$$

$$\frac{\overset{d}{\gamma \vdash t \xrightarrow{\beta p} t'}}{\gamma \vdash \pi_\beta t \xrightarrow{p} t'} \;;\; \overset{d'}{\gamma' \vdash t' \xrightarrow{p'} t''} \equiv_{\mathrm{def}} \frac{\overset{d\,;\,d'}{\gamma\,;\,\gamma' \vdash t \xrightarrow{\beta(p;p')} t''}}{\gamma\,;\,\gamma' \vdash \pi_\beta t \xrightarrow{p;p'} t''}$$

$$\frac{\overset{d}{\gamma \vdash t \xrightarrow{P} t'}}{\gamma \vdash\, !t \xrightarrow{!P} t'} \;;\; \overset{d'}{\gamma' \vdash t' \xrightarrow{P'} t''} \equiv_{\mathrm{def}} \frac{\overset{d\,;\,d'}{\gamma\,;\,\gamma' \vdash t \xrightarrow{P;P'} t''}}{\gamma\,;\,\gamma' \vdash\, !t \xrightarrow{!(P;P')} t''}$$

$$\frac{\overset{d_t}{\gamma, x : P \vdash t \xrightarrow{q} t'} \quad \overset{d_u}{\delta \vdash u \xrightarrow{!P} u'}}{\gamma, \delta \vdash [u > !x \Rightarrow t] \xrightarrow{q} t'[u'/x]} \;;\; \overset{d}{\gamma', \delta' \vdash t'[u'/x] \xrightarrow{q'} v}$$

$$\equiv_{\mathrm{def}} \frac{\overset{d_t\,;\,d'_t}{(\gamma\,;\,\gamma'), x : P\,;\,P' \vdash t \xrightarrow{q;q'} t''} \quad \overset{d_u\,;\,d'_u}{\delta\,;\,\delta' \vdash u \xrightarrow{!(P;P')} u''}}{(\gamma\,;\,\gamma'), (\delta\,;\,\delta') \vdash [u > !x \Rightarrow t] \xrightarrow{q;q'} t''[u''/x]}$$

—where $\overset{d'_t}{\gamma', x : P' \vdash t' \xrightarrow{q'} t''}$ and $\overset{d'_u}{\delta' \vdash u' \xrightarrow{P'} u''}$, with $v \equiv t''[u''/x]$, are induced by d using Lemma 7.11.

$$\frac{\overset{d}{\gamma \vdash t \xrightarrow{p} t'}}{\gamma \vdash \mathit{abs}\, t \xrightarrow{\mathit{abs}\, p} t'} \;;\; \overset{d'}{\gamma' \vdash t' \xrightarrow{p'} t''} \equiv_{\mathrm{def}} \frac{\overset{d\,;\,d'}{\gamma\,;\,\gamma' \vdash t \xrightarrow{p;p'} t''}}{\gamma\,;\,\gamma' \vdash \mathit{abs}\, t \xrightarrow{\mathit{abs}(p;p')} t''}$$

$$\frac{\overset{d}{\gamma \vdash t \xrightarrow{\mathit{abs}\, p} t'}}{\gamma \vdash \mathit{rep}\, t \xrightarrow{p} t'} \;;\; \overset{d'}{\gamma' \vdash t' \xrightarrow{p'} t''} \equiv_{\mathrm{def}} \frac{\overset{d\,;\,d'}{\gamma\,;\,\gamma' \vdash t \xrightarrow{\mathit{abs}(p;p')} t''}}{\gamma\,;\,\gamma' \vdash \mathit{rep}\, t \xrightarrow{p;p'} t''}$$

Figure 7.3: Composition of derivations (continued)

$$d|_\perp \equiv_{\mathrm{def}} d$$

$$\left.\frac{}{\perp, x : p' \vdash x \xrightarrow{p'} x}\right|_p \equiv_{\mathrm{def}} \frac{}{\perp, x : p \vdash x \xrightarrow{p} x}$$

$$\left.\frac{\begin{array}{c} d \\ \gamma' \vdash t[\mathit{rec}\, x.t/x] \xrightarrow{p'} t'' \end{array}}{\gamma' \vdash \mathit{rec}\, x.t \xrightarrow{p'} t''}\right|_p \equiv_{\mathrm{def}} \frac{\begin{array}{c} d|_p \\ \gamma \vdash t[\mathit{rec}\, x.t/x] \xrightarrow{p} t' \end{array}}{\gamma \vdash \mathit{rec}\, x.t \xrightarrow{p} t'}$$

$$\left.\frac{\begin{array}{c} d \\ \gamma' \vdash t_j \xrightarrow{p'} t'' \end{array}}{\gamma' \vdash \Sigma_{i\in I} t_i \xrightarrow{p'} t''}\ j \in I\right|_p \equiv_{\mathrm{def}} \frac{\begin{array}{c} d|_p \\ \gamma \vdash t_j \xrightarrow{p} t' \end{array}}{\gamma \vdash \Sigma_{i\in I} t_j \xrightarrow{p} t'}\ j \in I$$

$$\left.\frac{\begin{array}{cc} d_t & d_u \\ \gamma' \vdash t \xrightarrow{P'} t'' & \delta' \vdash u \xrightarrow{Q'} u'' \end{array}}{\gamma', \delta' \vdash t \otimes u \xrightarrow{P'\otimes Q'} t'' \otimes u''}\right|_{P\otimes Q} \equiv_{\mathrm{def}} \frac{\begin{array}{cc} d_t|_P & d_u|_Q \\ \gamma \vdash t \xrightarrow{P} t' & \delta \vdash u \xrightarrow{Q} u' \end{array}}{\gamma, \delta \vdash t \otimes u \xrightarrow{P\otimes Q} t' \otimes u'}$$

$$\left.\frac{\begin{array}{cc} d_t & d_u \\ \gamma', x : P', y : Q' \vdash t \xrightarrow{r'} t'' & \delta' \vdash u \xrightarrow{P'\otimes Q'} u'' \end{array}}{\gamma', \delta' \vdash [u > x \otimes y \Rightarrow t] \xrightarrow{r'} [u'' > x \otimes y \Rightarrow t'']}\right|_r$$

$$\equiv_{\mathrm{def}} \frac{\begin{array}{cc} d_t|_r & d_u|_{P\otimes Q} \\ \gamma, x : P, y : Q \vdash t \xrightarrow{r} t' & \delta \vdash u \xrightarrow{P\otimes Q} u' \end{array}}{\gamma, \delta \vdash [u > x \otimes y \Rightarrow t] \xrightarrow{r} [u' > x \otimes y \Rightarrow t']}$$

Figure 7.4: Restriction of derivations

$$\left.\frac{\begin{array}{c} d \\ \gamma' \vdash t \xrightarrow{p'} t'' \end{array}}{\gamma' \vdash \beta t \xrightarrow{\beta p'} t''}\right|_{\beta p} \equiv_{\mathrm{def}} \frac{\begin{array}{c} d|_p \\ \gamma \vdash t \xrightarrow{p} t' \end{array}}{\gamma \vdash \beta t \xrightarrow{\beta p} t'}$$

$$\left.\frac{\begin{array}{c} d \\ \gamma' \vdash t \xrightarrow{\beta p'} t'' \end{array}}{\gamma' \vdash t \xrightarrow{p'} t''}\right|_{p} \equiv_{\mathrm{def}} \frac{\begin{array}{c} d|_{\beta p} \\ \gamma \vdash t \xrightarrow{\beta p} t' \end{array}}{\gamma \vdash \pi_\beta t \xrightarrow{p} t'}$$

$$\left.\frac{\begin{array}{c} d \\ \gamma' \vdash t \xrightarrow{P'} t'' \end{array}}{\gamma' \vdash !t \xrightarrow{!P'} t''}\right|_{!P} \equiv_{\mathrm{def}} \frac{\begin{array}{c} d|_P \\ \gamma \vdash t \xrightarrow{P} t' \end{array}}{\gamma \vdash !t \xrightarrow{!P} t'}$$

$$\left.\frac{\begin{array}{c} d_t \\ \gamma', x : P' \vdash t \xrightarrow{q'} t'' \end{array} \quad \begin{array}{c} d_u \\ \delta' \vdash u \xrightarrow{!P'} u'' \end{array}}{\gamma', \delta' \vdash [u > !x \Rightarrow t] \xrightarrow{q'} t''[u''/x]}\right|_{q} \equiv_{\mathrm{def}} \frac{\begin{array}{c} d_t|_q \\ \gamma, x : P \vdash t \xrightarrow{q} t' \end{array} \quad \begin{array}{c} d_u|_{!P} \\ \delta \vdash u \xrightarrow{!P} u' \end{array}}{\gamma, \delta \vdash [u > !x \Rightarrow t] \xrightarrow{q} t'[u'/x]}$$

$$\left.\frac{\begin{array}{c} d \\ \gamma' \vdash t \xrightarrow{p'} t'' \end{array}}{\gamma' \vdash \mathit{abs}\, t \xrightarrow{\mathit{abs}\, p'} t''}\right|_{\mathit{abs}\, p} \equiv_{\mathrm{def}} \frac{\begin{array}{c} d|_p \\ \gamma \vdash t \xrightarrow{p} t' \end{array}}{\gamma \vdash \mathit{abs}\, t \xrightarrow{\mathit{abs}\, p} t'}$$

$$\left.\frac{\begin{array}{c} d \\ \gamma' \vdash t \xrightarrow{\mathit{abs}\, p'} t'' \end{array}}{\gamma' \vdash \mathit{rep}\, t \xrightarrow{p'} t''}\right|_{p} \equiv_{\mathrm{def}} \frac{\begin{array}{c} d|_{\mathit{abs}\, p} \\ \gamma \vdash t \xrightarrow{\mathit{abs}\, p} t' \end{array}}{\gamma \vdash \mathit{rep}\, t \xrightarrow{p} t'}$$

Figure 7.5: Restriction of derivations (continued)

7.4.4 Relating the Operational Semantics

We relate the stable operational semantics to the original operational semantics given in Chapter 4. Note first that the actions $\mathbb{P} : a : \mathbb{P}'$ of that chapter correspond to "atomic" paths $a : \mathbb{P}$. By strong correspondence, derivations $\vdash \mathbb{P} : t \xrightarrow{a} t'$ in the stable semantics correspond to configurations in the event-structure representation and so to elements of the presheaf denotation of t. Strong correspondence for the original operational semantics further relates to derivations there. Thus we have a bijective correspondence between derivations $\vdash \mathbb{P} : t \xrightarrow{a} t'$ in the stable semantics and derivations $\mathbb{P} : t \xrightarrow{a} t''$ in the original semantics.

Note that because the actions are atomic paths, they correspond to singleton configurations of $\mathbb{P}$. By the properties of output morphisms, each configuration mapped to a singleton configuration is itself a singleton. Thus, derivations $t \xrightarrow{a} t'$ in either semantics correspond to events of the vertex of $\mathcal{E}[\![t]\!]$ with image a under the output morphism.

To complete the picture, we should be able to relate sequences of transitions in the original operational semantics to configurations. This involves the ability to ignore the ordering of occurrences of independent events (i.e. events related by $\frown$ and unrelated by causality), and so the ordering of the corresponding transitions. Now, if two events of the vertex of $\mathcal{E}[\![t]\!]$ are independent, then they have independent images under the output morphism. Independence of events of type $\mathbb{P}$ induces a syntactic independence relation on actions $\mathbb{P} : a : \mathbb{P}'$ for some $\mathbb{P}'$, whose reflexive closure is given as the least congruence I on actions satisfying $a \otimes \bot \; I \; \bot \otimes b$ for any actions a and b. Thus, to ignore ordering of transitions corresponding to independent events, we need the diamond property discussed in Section 4.4.2. That will be a corollary of the result below which shows that a-labelled derivations $\vdash \mathbb{P} : t \xrightarrow{a} t'$ and $\mathbb{P} : t \xrightarrow{a} t''$ can be matched up so that the successor terms t' and t'' are identical.

Proposition 7.13 Suppose $\vdash t : \mathbb{P}$. Then $\mathbb{P} : t \xrightarrow{a} t'$ in the original operational semantics of Chapter 4 iff $\vdash \mathbb{P} : t \xrightarrow{a} t'$ in the stable operational semantics.

Proof. By rule-induction on the original operational rules, using the induction hypothesis

> Suppose $\Gamma \vdash t : \mathbb{P}$ and $\vdash e : \Gamma, \Phi$. Then
>
> $$\mathbb{P} : e \mapsto t \xrightarrow{a} e' \mapsto t' \iff \gamma \vdash \mathbb{P} : t \xrightarrow{a} t' \text{ and } A_\gamma(e, e') \quad (7.30)$$
>
> —where $A_\gamma(e, e')$ is defined for path maps $\gamma : \Gamma$ and environments $\vdash e : \Gamma$ and $\vdash e' : \Gamma/\gamma$ as follows: $A_\bot(\epsilon, \epsilon)$ is true and,

inductively,

$$A_{\gamma,x:P,y:Q}((e, u > x \otimes y), (e', u' > x \otimes y)) \iff_{\mathrm{def}} \\ \exists \delta.\ \delta \vdash \mathbb{P} \otimes \mathbb{Q} : u \xrightarrow{P\otimes Q} u' \text{ and } A_{\gamma,\delta}(e, e')\ . \quad (7.31)$$

We'll omit the typing information on transitions below. Note that we have $A_{\gamma;\gamma'}(e, e'')$ iff there exists e' such that $A_\gamma(e, e')$ and $A_{\gamma'}(e', e'')$.

Variable. Suppose that $e_1, u > x \otimes y, e_2 \mapsto x \xrightarrow{a} e'_1, u' > x \otimes y, e_2 \mapsto x$ is derived from $e_1 \mapsto u \xrightarrow{a\otimes\bot} e'_1 \mapsto u'$. By the induction hypothesis, the premise is equivalent to the existence of a derivation $\gamma \vdash u \xrightarrow{a\otimes\bot} u'$ for some γ such that $A_\gamma(e_1, e'_1)$. This is then equivalent to $A_{x:a,y:\bot}((e_1, u > x \otimes y, e_2), (e'_1, u' > x \otimes y, e_2))$ by definition, and as the stable operational rules yields a derivation $x : a \vdash x \xrightarrow{a} x$, we are done. The other rule for variables is handled symmetrically.

Tensor. Suppose that $e \mapsto t \otimes u \xrightarrow{a\otimes\bot} e' \mapsto t' \otimes u$ is derived from $e \mapsto t \xrightarrow{a} e' \mapsto t'$. By the induction hypothesis, the premise is equivalent to the existence of a derivation d_t of $\gamma \vdash t \xrightarrow{a} t'$ for some γ such that $A_\gamma(e, e')$. By the stable operational semantics, d_t exists iff there is a derivation of $\gamma \vdash t \otimes u \xrightarrow{a\otimes\bot} t' \otimes u$, as wanted.

Tensor match. Suppose $e \mapsto [u > x \otimes y \Rightarrow t] \xrightarrow{a} e' \mapsto [u' > x \otimes y \Rightarrow t']$ is derived from $e, u > x \otimes y \mapsto t \xrightarrow{a} e', u' > x \otimes y \Rightarrow t'$. By the induction hypothesis, the premise is equivalent to the existence of a derivation d_t of $\gamma, x : P, y : Q \vdash t \xrightarrow{a} t'$ with $A_{\gamma,x:P,y:Q}((e, u > x \otimes y), (e', u' > x \otimes y))$. By definition, the latter means that we have a derivation d_u of $\delta \vdash u \xrightarrow{P\otimes Q} u'$ and $A_{\gamma,\delta}(e, e')$ for some δ. By the stable operational semantics, we have d_t and d_u iff we have a derivation of $\gamma, \delta \vdash [u > x \otimes y \Rightarrow t] \xrightarrow{a} [u' > x \otimes y \Rightarrow t']$, as wanted.

Prefix match. Suppose $e \mapsto [u > !x \Rightarrow t] \xrightarrow{a} e'' \mapsto v$ is derived from $e \mapsto u \xrightarrow{!} e' \mapsto u'$ and $e' \mapsto t[u'/x] \xrightarrow{a} e'' \mapsto v$. By the induction hypothesis, the first premise is equivalent to the existence of a derivation d_u of $\delta \vdash u \xrightarrow{!\bot} u'$ for some δ such that $A_\delta(e, e')$. By the induction hypothesis, the second premise is equivalent to the existence of a derivation d of $\gamma, \delta' \vdash t[u'/x] \xrightarrow{a} v$ such that $A_{\gamma,\delta'}(e', e'')$. Lemma 7.11 implies that d exists iff we have both a derivation d_t of $\gamma, x : P \vdash t \xrightarrow{a} t'$ and a derivation d'_u of $\delta' \vdash u' \xrightarrow{P} u''$ with $v \equiv t''[u''/x]$. Now, $d_u\ ;\ d'_u$ is a derivation of $\delta\ ;\ \delta' \vdash u \xrightarrow{!P} u''$, and using d_t, we get a derivation of $\gamma, (\delta\ ;\ \delta') \vdash [u > !x \Rightarrow t] \xrightarrow{a} t''[u''/x]$ as wanted. Conversely, such a derivation induces the derivations d_t and $d_u\ ;\ d'_u$ above. Finally, we have $A_{\gamma,(\delta;\delta')}(e, e'')$ iff $A_\delta(e, e')$ and $A_{\gamma,\delta'}(e', e'')$, and so we are done.

The remaining rules are handled similarly. $\square$

Corollary 7.14 The diamond property holds for the original operational semantics.

Proof. We omit typing information. If we have derivations of $t \xrightarrow{a\otimes\perp} t_1$ and $t_1 \xrightarrow{\perp\otimes b} t'$ in the original semantics, we have corresponding derivations d_a and d_b in the stable operational semantics by Proposition 7.13. The composition $d_a \,;\, d_b$ is a derivation of $\vdash t \xrightarrow{a\otimes b} t'$, and by restriction, we obtain a derivation $(d_a \,;\, d_b)|_{\perp\otimes b}$ of $\vdash t \xrightarrow{\perp\otimes b} t_2$ for some t_2 together with a residual derivation of $\vdash t_2 \xrightarrow{a\otimes\perp} t'$. Using Proposition 7.13 again, we get corresponding derivations in original semantics as wanted. $\square$

It follows easily that whenever $a_1 \; I \; a_2$ and we have transitions $t \xrightarrow{a_1} t_1$ and $t_1 \xrightarrow{a_2} t'$ in the original semantics, then we also have transitions $t \xrightarrow{a_2} t_2$ and $t_2 \xrightarrow{a_1} t'$ for some t_2. By replacing actions by the events of $\mathcal{E}[\![t]\!]$ sitting above them, we exhibit the original operational semantics as an asynchronous transition system [5, 84].

7.5 Higher-Order Processes

The event structure semantics extends to higher-order processes, though at the price that the correspondence to the presheaf semantics fails.

Given event structures $\mathbb{E}_i = (E_i, \leq_i, \smile_i)$ for $i = 1, 2$, we define the event structure $\mathbb{E}_1 \multimap \mathbb{E}_2$ by first describing its configurations. Given any subset $f \subseteq \mathcal{C}(\mathbb{E}_1) \times E_2$, write $\pi_2(f, E)$ for $\{e \in E_2 : \exists x \in \mathbb{E}_1.\ x \subseteq E \text{ and } (x, e) \in f\}$. Then $\pi_2 f = \pi_2(f, \pi_1^\dagger f)$. Now, f is a configuration if

(i) $\pi_1^\dagger f \in \mathcal{C}(\mathbb{E}_1)$;

(ii) $\forall x \in \mathcal{C}(\mathbb{E}_1).\ \pi_2(f, x) \in \mathcal{C}(\mathbb{E}_2)$;

(iii) $\forall (x, e), (x', e) \in \mathcal{C}(\mathbb{E}_1) \times E_2.\ (x, e) \in f \text{ and } (x', e) \in f \implies x = x'$.

This defines a *stable family of configurations* $F_{\mathbb{E}_1 \multimap \mathbb{E}_2}$ over the event set $\mathcal{C}(\mathbb{E}_1) \times E_2$ [94]. Any event structure $\mathbb{E}$ over events E induces a stable family of configurations over the same events by taking the configurations of $\mathbb{E}$ as the elements of the family. This construction is part of an adjunction between the categories of event structures and stable families; the right adjoint constructs from a stable family an event structure by taking as events the prime configurations of the family.

The prime configurations of $F_{\mathbb{E}_1 \multimap \mathbb{E}_2}$ are the nonempty subsets $f \subseteq \mathcal{C}(\mathbb{E}_1) \times E_2$ satisfying the above, but with (ii) strengthened by demanding that $\pi_2(f, x)$ is either empty or of the form $\lceil e \rceil$.

The adjunction yields a bijective correspondence between output morphisms $\mathbb{E} \to (\mathbb{P} \multimap \mathbb{Q})$ and morphisms of stable families $\mathbb{E} \to F_{\mathbb{P} \multimap \mathbb{Q}}$. Moreover, the latter morphisms are in bijective correspondence with spans from

$\mathbb{E}$. In one direction of the correspondence, a span $s = \langle \iota, \mathbb{E}, o \rangle$ is sent to the map $s : \mathbb{E} \to F_{\mathbb{P} \multimap \mathbb{Q}}$ given by $e \mapsto \{(\iota e', oe') : e' \leq_{\mathbb{E}} e\}$. By the adjunction, s then corresponds to an output morphism $\mathbb{E} \to (\mathbb{P} \multimap \mathbb{Q})$, and since these are in bijective correspondence with spans $\mathbf{Spn}(\mathbb{O}, \mathbb{P} \multimap \mathbb{Q})$ we obtain an isomorphism between the categories $\mathbf{Spn}(\mathbb{O}, \mathbb{P} \multimap \mathbb{Q})$ and $\mathbf{Spn}(\mathbb{P}, \mathbb{Q})$. This indicates how the function space internalises spans; the category $\mathbf{Spn}(\mathbb{O}, \mathbb{P} \multimap \mathbb{Q})$ plays the role of $\widehat{\mathbb{P}^{\mathrm{op}} \times \mathbb{Q}}$ in showing how the function space of **Lin** internalises maps of **Lin**.

The families of configurations defining $\mathbb{P} \otimes \mathbb{Q} \multimap \mathbb{R}$ and $\mathbb{P} \multimap (\mathbb{Q} \multimap \mathbb{R})$ are the same, but for a trivial renaming of events given by the isomorphism $\mathcal{C}(\mathbb{P} \otimes \mathbb{Q}) \times R \cong \mathcal{C}(\mathbb{P}) \times \mathcal{C}(\mathbb{Q}) \times R$ where R is the event set of $\mathbb{R}$. This induces an isomorphism between $\mathbb{P} \otimes \mathbb{Q} \multimap \mathbb{R}$ and $\mathbb{P} \multimap (\mathbb{Q} \multimap \mathbb{R})$. We therefore obtain a chain of isomorphisms

$$\begin{aligned} \mathbf{Spn}(\mathbb{P} \otimes \mathbb{Q}, \mathbb{R}) &\cong \mathbf{Spn}(\mathbb{O}, \mathbb{P} \otimes \mathbb{Q} \multimap \mathbb{R}) \\ &\cong \mathbf{Spn}(\mathbb{O}, \mathbb{P} \multimap (\mathbb{Q} \multimap \mathbb{R})) \cong \mathbf{Spn}(\mathbb{P}, \mathbb{Q} \multimap \mathbb{R}) \ . \end{aligned} \tag{7.32}$$

We take this as evidence that **Spn** has monoidal closed structure and omit further details. The denotational rules for function space of Section 3.1 can now be used to provide the full language Affine HOPLA with a stable denotational semantics.

The stable operational semantics can be extended to full Affine HOPLA by allowing configurations of functions spaces as labels and using transitions of the form $\gamma \vdash^f t \xrightarrow{P} t'$ where the f-tag is a configuration of $\Gamma \multimap \mathbb{P}$ such that $\pi_1^\dagger f = \gamma$ and $\pi_2 f = P$. This extra information is needed in a rule for lambda-abstraction:

$$\frac{\gamma, x : P \vdash \mathbb{Q} : t \xrightarrow{q} t'}{\gamma \vdash \mathbb{P} \multimap \mathbb{Q} : \lambda x.t \xrightarrow{?} \lambda x.t'} \tag{7.33}$$

Here the label of the transition of the conclusion should be the currying of the configuration $f \in \mathcal{C}(\Gamma \otimes \mathbb{P} \multimap \mathbb{Q})$ represented by the premise, but the projections $\gamma \otimes P = \pi_1^\dagger f$ and $q = \pi_2 f$ are not enough to determine f.

With complicated labels and the extra, semantic information needed, such an operational semantics will be rather denotational in nature. It seems reasonable to look for simple labels to replace function-space configurations. An obvious attempt would be to use the paths $P \mapsto q$ from Section 4.1. This leads to the following rules:

$$\begin{gathered} \frac{\gamma, x : P \vdash \mathbb{Q} : t \xrightarrow{q} t'}{\gamma \vdash \mathbb{P} \multimap \mathbb{Q} : \lambda x.t \xrightarrow{P \mapsto q} \lambda x.t'} \\ \frac{\gamma \vdash \mathbb{P} \multimap \mathbb{Q} : t \xrightarrow{P \mapsto q} t' \quad \delta \vdash \mathbb{P} : u \xrightarrow{P} u'}{\gamma, \delta \vdash \mathbb{Q} : t\ u \xrightarrow{q} t'\ u'} \end{gathered} \tag{7.34}$$

While these rules seem to behave as wanted, we don't yet have a proof pinpointing the correspondence with the event-structure semantics. The problem is that the paths $P \mapsto q$ contain too little information to describe a configuration of $\mathbb{P} \multimap \mathbb{Q}$. Indeed, there will generally be many configurations f with $\pi_1^\dagger f = P$ and $\pi_2 f = q$, and so, at best, derivations $\gamma \vdash \mathbb{P} \multimap \mathbb{Q} : t \xrightarrow{P \mapsto q} t'$ will correspond to equivalence classes of configurations of the vertex of $\mathcal{E}[\![t]\!]$, the equivalence relating two configurations iff they have the same images under the projections. We hope to be able to adapt the proofs of strong correspondence and soundness accordingly.

Chapter 8

Conclusion

Section 8.1 summarises the thesis and Section 8.2 discusses related work aiming at extending our basic domain theory to a fuller coverage of concurrent computation.

8.1 Summary

We have given a simple domain theory for concurrent processes within the framework of Scott and Strachey. Processes are represented as sets of computation paths as in Hennessy's work on full abstraction for CCS with process passing. Ordered by inclusion, path sets form nondeterministic domains, and two expressive metalanguages for concurrency, HOPLA and Affine HOPLA, are based on canonical constructions on these domains. Their denotational semantics are fully abstract with contextual equivalence characterised as equality of path set denotations.

One may argue that nondeterministic processes like those we have considered are a better fit to Scott's domain theory than sequential programs. Domains are constructed from computation paths which form an intuitive basis of prime elements, useful for giving simple adequacy proofs; the somewhat subtle information ordering is replaced by the very concrete inclusion of path sets; metalanguages arise more directly from canonical constructions on domains as there is no extra notion of evaluation strategy that needs to be catered for by use of lifting; and full abstraction is an easy consequence of expressiveness with respect to computation paths.

The domain theory based on path sets is a simple version of an earlier and more informative domain theory based on presheaves. Compared to the path set semantics of a process, a presheaf says not only whether or not a given computation path may be performed by the process, but yields the set of ways it may be realised. This extra structure of presheaves takes the nondeterministic branching behaviour of processes into account. The presheaf semantics of the two metalanguages inform operational semantics with derivations corresponding to the realisers of presheaves. This strengthens the correspondence results obtained using the path semantics, themselves subsuming more standard soundness and adequacy results.

The domain theory and the two metalanguages have highlighted the role of linearity in concurrent computation, distinguishing between linear, affine, and continuous process operations that use respectively exactly one, at most one, and any finite number of computation paths of their input in providing a path of output. Continuous operations, as embodied in HOPLA, receive the code of other processes and are free to make copies of that code in order to explore the behaviour of the input in different contexts. Affine operations, as embodied in Affine HOPLA, do not allow copying and can only examine the input in a single context. Such operations are more realistic in a distributed setting where processes interact as peers rather than being sent to each other as code. Linear operations are further restricted in that they are not even allowed to ignore their input, if they are to have nontrivial behaviour themselves. So linear operations alone are not enough. It can nevertheless be useful for reasoning about processes to be attentive to linearity as it amounts to preserving nondeterministic sums, leading e.g. to expansion laws.

Both metalanguages are shown expressive enough to handle nondeterminism, prefixing and higher-order processes to the extent of allowing direct encodings of CCS, CCS with process passing, and mobile ambients with public names (the affine language with suitable restrictions on variable occurrences). This is remarkable in that the only process constructs of the metalanguages, as far as these encodings are concerned, are nondeterministic sums and prefix operations. In particular, parallel composition and ambient creation can be defined in the languages and need not be primitive constructs. Affine HOPLA has an additional construct, a tensor operation, which allows the affine metalanguage to encode dataflow processes with multiple communication lines. By an event-structure representation of the first-order fragment of the affine language, we have exposed the tensor as a juxtaposition of independent processes.

8.2 Related Work

Three main extensions of the work above concern name-generation, independence models, and bisimulation.

8.2.1 Name-Generation

The encodings of CCS with process passing and mobile ambients with public names show that our metalanguages can express certain forms of mobility of processes by virtue of allowing process passing. Another kind of mobility, mobility of communication links, arises from name-generation as in the π-calculus [58]. Inspired by HOPLA, Winskel and Zappa Nardelli have defined a higher-order process language with name-generation, allowing encodings of full Ambient Calculus and π-calculus [101]. The extensions are to add a type of names $\mathcal{N}$, function spaces, as well as a type $\delta\mathbb{P}$ supporting new-name generation through the abstraction *new* $x.t$. The denotational semantics of the extension is currently being developed; this addresses the question of when function spaces exist in the obvious model (extending that of [17]). There is already an operational semantics; it is like that of HOPLA but given at stages indexed by the current set of names.

8.2.2 Independence

The event-structure representation of Affine-HOPLA exposes that the affine language yields an independence model for concurrency. Still, the coverage of independence models is rather limited.

Spans of event structures like those used in Chapter 7 allow defining the trace operation of nondeterministic dataflow. Indeed, the representable profunctors, which are stable and rooted, were called "stable port profunctors" in [33] and used to give a fully abstract denotational semantics to nondeterministic dataflow with trace.

As an operation on spans, trace maps $\langle \iota, \mathbb{E}, o\rangle : \mathbb{P} \otimes \mathbb{R} \to \mathbb{Q} \otimes \mathbb{R}$ to a span $\langle \iota', \mathbb{E}', o'\rangle : \mathbb{P} \to \mathbb{Q}$ with the events of $\mathbb{E}'$ being prime configurations of $\mathbb{E}$ satisfying a certain "securedness" requirement [98] to the effect that all input of type $\mathbb{R}$ has been previously output. However, conflict on the event structure $\mathbb{E}'$ cannot in general be given by a binary relation, and so the spans have to be generalised to allow the vertices to be more general event structures with a consistency predicate replacing the conflict relation [93].

It was noted in Section 4.5.2 that trace cannot be expressed in Affine HOPLA, and how best to extend the syntax and operational semantics of the affine language to accommodate trace is not known.

Perhaps more importantly, one can show the event-structure denotations of Affine HOPLA are too impoverished to coincide with the standard independence semantics of CCS as e.g. given in [96]. Indeed, given any term t and the corresponding interpretation $\mathcal{E}[\![t]\!] = \langle \iota, \mathbb{E}, o\rangle$, we have that for all events e of $\mathbb{E}$, the poset $\lceil e \rceil$ is a linear order. By contrast, in the standard semantics, a parallel composition $(\mathsf{a}.\mathsf{c}.\varnothing)|(\mathsf{b}.\bar{\mathsf{c}}.\varnothing)$ will be modelled as an event structure with three mutually consistent events a, b, c such that $a \leq c$ and $b \leq c$, but with a and b unrelated.

Again, we have to move beyond Affine HOPLA for defining such semantics. Guidelines on what's lacking in the affine language can be got from work on presheaf models for concurrency [16], where the ingredients of product of presheaves, pomset augmentation and cartesian liftings (extending the match operators of Affine HOPLA) all play a critical role. This work suggests exploring other event-structure representations, based on more general spans of event structures, and perhaps a new comonad yielding a less rigid form of prefixing.

As a general point, the categories obtained from presheaves are very rich in structure, pointing towards more expressive languages than HOPLA and Affine HOPLA. In particular, the affine category accommodates the independence model of event structures to the extent of supporting the standard event-structure semantics of CCS and related languages, as well as the trace of nondeterministic dataflow.

8.2.3 Bisimulation

Presheaf models come with a built-in, semantic notion of bisimulation, derived from open maps, and for general reasons, this equivalence is a congruence for both metalanguages.

One of these general reasons is a remarkable result by Cattani and Winskel [16, 19, 20] which says that the morphisms of **Lin** preserve open maps with respect to the situation $\mathrm{y}_{\mathbb{P}} : \mathbb{P} \hookrightarrow \widehat{\mathbb{P}}$, i.e. using a diagram like the one on the left below in the definition of what it means for $f : X \to Y$ to be open.

$$\begin{tikzcd} \mathrm{y}_{\mathbb{P}}p \arrow[r] \arrow[d] & X \arrow[d, "f"] \\ \mathrm{y}_{\mathbb{P}}q \arrow[r] \arrow[ur, dotted] & Y \end{tikzcd} \qquad \begin{tikzcd} j_{\mathbb{P}}P \arrow[r] \arrow[d] & X \arrow[d, "f"] \\ j_{\mathbb{P}}Q \arrow[r] \arrow[ur, dotted] & Y \end{tikzcd} \qquad \begin{tikzcd} i_{\mathbb{P}}P \arrow[r] \arrow[d] & X \arrow[d, "f"] \\ i_{\mathbb{P}}Q \arrow[r] \arrow[ur, dotted] & Y \end{tikzcd} \tag{8.1}$$

Cattani and Winskel also show that the morphisms of **Aff** preserve open maps with respect to the situation $j_{\mathbb{P}} : \mathbb{P}_{\perp} \hookrightarrow \widehat{\mathbb{P}}$ (center diagram above), and one can further show that morphisms of **Cts** preserve open maps with respect to the situation $i_{\mathbb{P}} : !\mathbb{P} \hookrightarrow \widehat{\mathbb{P}}$ (right diagram above). We'll return to this below. Thus, open-map bisimilarity is preserved in all cases, giving congruence results for free when process languages are interpreted in **Lin**, **Aff**, or **Cts**.

Part of the motivation for Winskel's original work on Affine HOPLA [98] was the hope that an operational semantics for full Affine HOPLA would provide an operational, coinductive characterisation of open-map bisimilarity. Unfortunately, our operational semantics for the affine language has not been extended to higher-order processes in general, and the stable operational semantics, even if correct, is not helpful as the underlying event structure semantics already diverges from the presheaf semantics.

We do have a full operational semantics for HOPLA, but here the notion of open map degenerates into isomorphisms and the congruence result is trivial [20]. The collapse of open maps in **Cts** suggests that one should look at other choices of exponential [20, 65]. The abstract presentation of HOPLA using universal constructions should be helpful in evaluating candidates.

As it stands, a general operational understanding of open-map bisimulation has not been obtained. In fact, the metalanguages may not even be able to provide such an understanding.

Recall from (5.7) that the maps of **Lin** correspond to profunctors $\mathbb{P} \times \mathbb{Q}^{\mathrm{op}} \to \mathbf{Set}$. The bicategory **Prof** of profunctors is biequivalent to **Lin**, and like **Lin** the bicategory **Prof** has an involution so that maps $f : \mathbb{P} \to \mathbb{Q}$ correspond to their dual $f^{\perp} : \mathbb{Q}^{\perp} \to \mathbb{P}^{\perp}$. Indeed, again just as in **Lin**, a map $f : \mathbb{P} \to \mathbb{Q}$ corresponds to a map $f' : \mathbb{P} \times \mathbb{Q}^{\perp} \to \mathbb{1}$, in which we have "dualised" the output to input. It is because of this duality that open maps and open-map bisimulation for higher-order processes take as much account of input as they do output. Most often two higher-order processes are defined to be bisimilar iff they yield bisimilar outputs on any common input (cf. Section 3.5.3). But this simply won't do within a type discipline in which all nontrivial output can be "dualised" to input. On the other hand, traditional process languages and their types don't support this duality.

One line towards understanding open-map bisimulation at higher order is to design a process language in which this duality is present. The language could support the types of **Prof** extended by a suitable pseudo comonad. Ideally one would obtain a coinductive characterisation of open map bisimulation at higher order based on an operational semantics.

Bibliography

[1] S. Abramsky. The lazy lambda calculus. In D. Turner, editor. *Research Topics in Functional Programming*. Addison-Wesley, 1990.

[2] S. Abramsky. Computational interpretations of linear logic. *Theoretical Computer Science*, 111(1–2):3–57, 1993.

[3] S. Abramsky and A. Jung. Domain theory. In S. Abramsky et al, editors. *Handbook of Logic in Computer Science. Volume 3. Semantic Structures*. Oxford University Press, 1994.

[4] S. Abramsky. Game semantics for programming languages. In *Proc. MFCS'97*, LNCS 1295.

[5] M. A. Bednarczyk. *Categories of Asynchronous Systems*. PhD thesis, University of Sussex, 1988.

[6] H. P. Barendregt. *The Lambda Calculus. Its Syntax and Semantics*. Revised edition, North-Holland, 1984.

[7] P. N. Benton. A mixed linear and non-linear logic: proofs, terms and models (extended abstract). In *Proc. CSL'94*, LNCS 933.

[8] G. Berry. *Modèles complètement adéquats et stables des lambda-calculs typés*. Thèse de Doctorat d'Etat, Université Paris VII, 1979.

[9] F. Borceux. *Handbook of Categorical Algebra I. Basic Category Theory*. Cambridge University Press, 1994.

[10] W. Brauer, W. Reisig and G. Rozenberg, editors. *Petri Nets: Central Models and Their Properties*. LNCS 254.

[11] W. Brauer, W. Reisig and G. Rozenberg, editors. *Petri Nets: Applications and Relationships to Other Models of Concurrency*. LNCS 255.

[12] T. Bräuner. *An Axiomatic Approach to Adequacy*. Ph.D. Dissertation, University of Aarhus, 1996. BRICS Dissertation Series DS-96-4.

[13] J. D. Brock and W. B. Ackerman. Scenarios: a model of non-determinate computation. In *Proc. Formalization of Programming Concepts 1981*, LNCS 107.

[14] L. Cardelli and A. D. Gordon. Anytime, anywhere: modal logics for mobile ambients. In *Proc. POPL'00*.

[15] L. Cardelli and A. D. Gordon. A commitment relation for the ambient calculus. Note. October 6th, 2000. Available from `http://research.microsoft.com/~adg/`.

[16] G. L. Cattani and G. Winskel. Presheaf models for concurrency. In *Proc. CSL'96*, LNCS 1258.

[17] G. L. Cattani, I. Stark and G. Winskel. Presheaf models for the π-calculus. In *Proc. CTCS'97*, LNCS 1290.

[18] G. L. Cattani, M. Fiore, and G. Winskel. A theory of recursive domains with applications to concurrency. In *Proc. LICS'98*.

[19] G. L. Cattani. *Presheaf Models for Concurrency*. Ph.D. Dissertation, University of Aarhus, 1999. BRICS Dissertation Series DS-99-1.

[20] G. L. Cattani and G. Winskel. Profunctors, open maps and bisimulation. Manuscript, 2003. Available from `http://www.cl.cam.ac.uk/~gw104/`.

[21] A. Church. An unsolvable problem in elementary number theory. *American Journal of Mathematics*, 58:354–363, 1936.

[22] F. Crazzolara and G. Winskel. Events in security protocols. In *Proc. 8th ACM Conference on Computer and Communications Security*, 2001.

[23] S. Dal Zilio. Mobile processes: a commented bibliography. In F. Cassez et al, editors. *Modelling and Verification of Parallel Processes*, LNCS 2067, 2001.

[24] J. B. Dennis. Data flow computation. In M. Broy, editor. *Control Flow and Data Flow: Concepts of Distributed Programming*. Springer-Verlag, 1985.

[25] M. Fiore, G. L. Cattani, and G. Winskel. Weak bisimulation and open maps. In *Proc. LICS'99*.

[26] S. Furber, editor. *Proc. Eighth International Symposium on Asynchronous Circuits and Systems*. IEEE, 2002.

[27] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–102, 1987.

[28] A. D. Gordon. Bisimilarity as a theory of functional programming. In *Proc. MFPS'95*, ENTCS 1.

[29] J. Gray. Fibred and cofibred categories. In *Proc. La Jolla Conference on Categorical Algebra*, Springer-Verlag, 1966.

[30] M. C. B. Hennessy and G. D. Plotkin. Full abstraction for a simple parallel programming language. In *Proc. MFCS'79*, LNCS 74.

[31] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1):137–161, 1985.

[32] M. Hennessy. A fully abstract denotational model for higher-order processes. *Information and Computation*, 112(1):55–95, 1994.

[33] T. Hildebrandt, P. Panangaden and G. Winskel. A relational model of non-deterministic dataflow. In *Proc. CONCUR'98*, LNCS 1466.

[34] T. T. Hildebrandt. A fully abstract presheaf semantics for SCCS with finite delay. In *Proc. CTCS'99*, ENTCS 29.

[35] T. T. Hildebrandt. *Categorical Models for Concurrency: Independence, Fairness and Dataflow.* Ph.D. Dissertation, University of Aarhus, 2000. BRICS Dissertation Series DS-00-1.

[36] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.

[37] C. A. R. Hoare. *A Model for Communicating Sequential Processes.* Technical monograph, PRG-22, University of Oxford Computing Laboratory, 1981.

[38] D. J. Howe. Proving congruence of bisimulation in functional programming languages. *Information and Computation*, 124(2):103–112, 1996.

[39] J. M. E. Hyland and C.-H. L. Ong. On full abstraction for PCF, parts I, II, and III. *Information and Computation*, 163(2):285–408, 2000.

[40] B. Jacobs. Semantics of weakening and contraction. *Annals of Pure and Applied Logic*, 69(1):73–106, 1994.

[41] A. Joyal and I. Moerdijk. A completeness theorem for open maps. *Annals of Pure and Applied Logic*, 70:51–86, 1994.

[42] A. Joyal, M. Nielsen, and G. Winskel. Bisimulation from open maps. *Information and Computation*, 127:164–185, 1996.

[43] G. Kahn. The semantics of a simple language for parallel programming. *Information Processing*, 74:471–475, 1974.

[44] G. M. Kelly. *Basic concepts of enriched category theory.* London Math. Soc. Lecture Note Series 64, Cambridge University Press, 1982.

[45] S. C. Kleene. λ-definability and recursiveness. *Duke Mathematical Journal*, 2:340–353, 1936.

[46] J. Lambek and P. L. Scott. *Introduction to higher order categorical logic.* Cambridge University Press, 1986.

[47] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

[48] K. G. Larsen and G. Winskel. Using information systems to solve recursive domain equations effectively. In *Proc. Semantics of Data Types, 1984*, LNCS 173.

[49] A. Levy. *Basic Set Theory.* Springer-Verlag, 1979.

[50] S. Mac Lane and I. Moerdijk. *Sheaves in Geometry and Logic.* Springer-Verlag, 1992.

[51] S. Mac Lane. *Categories for the Working Mathematician.* Second edition, Springer, 1998.

[52] A. Mazurkiewicz. Basic notions of trace theory. In *Proc. REX summer school 1988*, LNCS 354.

[53] G. McCusker. A fully abstract relational model of syntactic control of interference. In *Proc. CSL'02*, LNCS 2471.

[54] P.-A. Melliès. Categorical models of linear logic revisited. Submitted to *Theoretical Computer Science*, 2002.

[55] M. Merro and F. Zappa Nardelli. Bisimulation proof methods for mobile ambients. In *Proc. ICALP'03*, LNCS 2719.

[56] R. Milner. *A Calculus of Communicating Systems.* LNCS 92, Springer-Verlag, 1980.

[57] R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[58] R. Milner, J. Parrow and D. Walker. A calculus of mobile processes, parts I and II. *Information and Computation*, 100(1):1–77, 1992.

[59] R. Milner, M. Tofte, R. Harper and D. MacQueen. *The Definition of Standard ML (Revised).* The MIT Press, 1997.

[60] R. Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, 1999.

[61] J. H. Morris. *Lambda-Calculus Models of Programming Languages*. PhD thesis, MIT, 1968.

[62] M. Nielsen, G. Plotkin and G. Winskel. Petri nets, event structures and domains, part I. *Theoretical Computer Science*, 13(1):85–108, 1981.

[63] M. Nygaard. Towards an operational understanding of presheaf models. Progress report, University of Aarhus, 2001. Available from `http://www.daimi.au.dk/~nygaard/`.

[64] M. Nygaard and G. Winskel. Linearity in process languages. In *Proc. LICS'02*.

[65] M. Nygaard and G. Winskel. HOPLA—a higher-order process language. In *Proc. CONCUR'02*, LNCS 2421.

[66] M. Nygaard and G. Winskel. Full abstraction for HOPLA. In *Proc. CONCUR'03*, LNCS 2761.

[67] M. Nygaard and G. Winskel. Domain theory for concurrency. To appear in *Theoretical Computer Science* special issue on domain theory.

[68] D. Park. Concurrency and automata on infinite sequences. In *Proc. Theoretical Computer Science: 5th GI-Conference*, LNCS 104, 1981.

[69] D. A. Peled, V. R. Pratt and G. J. Holzmann, editors. *Proc. Partial Order Methods in Verification 1996*. DIMACS 29, American Mathematical Society, 1997.

[70] B. C. Pierce. *Basic Category Theory for Computer Scientists*. The MIT Press.

[71] B. C. Pierce. *Types and Programming Languages*. The MIT Press, 2002.

[72] A. M. Pitts. Relational properties of domains. *Information and Computation*, 127(2):66–90, 1996.

[73] G. Plotkin. A powerdomain construction. *SIAM Journal on Computing*, 5(3):452–487, 1976.

[74] G. Plotkin. LCF considered as a programming language. *Theoretical Computer Science* 5(3): 225-255, 1977.

[75] G. Plotkin. A structural approach to operational semantics. Lecture notes, DAIMI FN-19, Department of Computer Science, University of Aarhus, 1981.

[76] R. Rosebrugh and R. J. Wood. Proarrows and cofibrations. *Journal of Pure and Applied Algebra*, 53:271-296, 1988.

[77] D. Sangiorgi. *Expressing Mobility in Process Algebra. First-Order and Higher-Order Paradigms*. PhD thesis, University of Edinburgh, 1992.

[78] D. Sangiorgi. Bisimulation for higher-order process calculi. *Information and Computation*, 131(2):141–178, 1996.

[79] D. Sangiorgi and D. Walker. *The π-calculus. A Theory of Mobile Processes*. Cambridge University Press, 2001.

[80] D. S. Scott. Outline of a mathematical theory of computation. In *Proc. 4th Annual Princeton Conference on Information Sciences and Systems*, 1970.

[81] D. S. Scott. Domains for denotational semantics. In *Proc. ICALP'82*, LNCS 140.

[82] D. S. Scott and C. Strachey. Towards a mathematical semantics for computer languages. In *Proc. Symposium on Computers and Automata*, Microwave Research Institute Symposia Series, vol. 21, 1971.

[83] R. A. G. Seely. Linear logic, $*$-autonomous categories and cofree coalgebras. In *Proc. Categories in Computer Science and Logic, 1987*, Contemporary Mathematics 92.

[84] M. W. Shields. Concurrent machines. *Computer Journal*, 28:449–465, 1985.

[85] C. Strachey. Fundamental concepts in programming languages. *Higher-Order Logic and Symbolic Computation*, 13:11–49, 2000.

[86] P. Taylor. *Practical Foundations of Mathematics*. Cambridge University Press, 1999.

[87] F. Javier Thayer, J. C. Herzog and J. D. Guttman. Strand spaces: why is a security protocol correct? In *Proc. IEEE Symposium on Security and Privacy 1998.*

[88] B. Thomsen. A calculus of higher-order communicating systems. In *Proc. POPL'89.*

[89] A. Turing. Computability and λ-definability. *Journal of Symbolic Logic*, 2:153–163, 1937.

[90] R. J. van Glabbeek. The linear time — branching time spectrum. In *Proc. CONCUR'90*, LNCS 458.

[91] G. Winskel. *Events in Computation*. PhD thesis, University of Edinburgh, 1980. Available from `http://www.cl.cam.ac.uk/~gw104/`.

[92] G. Winskel. On powerdomains and modality. *Theoretical Computer Science*, 36:127–137, 1985.

[93] G. Winskel. Event Structures. In [11].

[94] G. Winskel. An introduction to event structures. In *Proc. REX summer school 1988*, LNCS 354.

[95] G. Winskel. *The Formal Semantics of Programming Languages. An Introduction*. The MIT Press, 1993.

[96] G. Winskel and M. Nielsen. Models for concurrency. In S. Abramsky et al, editors. *Handbook of Logic in Computer Science. Volume 4. Semantic Modelling*. Oxford University Press, 1995.

[97] G. Winskel. A presheaf semantics of value-passing processes. In *Proc. CONCUR'96*, LNCS 1119.

[98] G. Winskel. A linear metalanguage for concurrency. In *Proc. AMAST'98*, LNCS 1548.

[99] G. Winskel. A presheaf semantics for mobile ambients. Manuscript, 1998.

[100] G. Winskel. Event structures as presheaves—two representation theorems. In *Proc. CONCUR'99*, LNCS 1664.

[101] G. Winskel and F. Zappa Nardelli. Manuscript, 2003.

Recent BRICS Dissertation Series Publications

DS-03-13 Mikkel Nygaard. *Domain Theory for Concurrency*. November 2003. PhD thesis. xiii+161 pp.

DS-03-12 Paulo B. Oliva. *Proof Mining in Subsystems of Analysis*. September 2003. PhD thesis. xii+198 pp.

DS-03-11 Maciej Koprowski. *Cryptographic Protocols Based on Root Extracting*. August 2003. PhD thesis. xii+138 pp.

DS-03-10 Serge Fehr. *Secure Multi-Player Protocols: Fundamentals, Generality, and Efficiency*. August 2003. PhD thesis. xii+125 pp.

DS-03-9 Mads J. Jurik. *Extensions to the Paillier Cryptosystem with Applications to Cryptological Protocols*. August 2003. PhD thesis. xii+117 pp.

DS-03-8 Jesper Buus Nielsen. *On Protocol Security in the Cryptographic Model*. August 2003. PhD thesis. xiv+341 pp.

DS-03-7 Mario José Cáccamo. *A Formal Calculus for Categories*. June 2003. PhD thesis. xiv+151.

DS-03-6 Rasmus K. Ursem. *Models for Evolutionary Algorithms and Their Applications in System Identification and Control Optimization*. June 2003. PhD thesis. xiv+183 pp.

DS-03-5 Giuseppe Milicia. *Applying Formal Methods to Programming Language Design and Implementation*. June 2003. PhD thesis. xvi+211.

DS-03-4 Federico Crazzolara. *Language, Semantics, and Methods for Security Protocols*. May 2003. PhD thesis. xii+160.

DS-03-3 Jiří Srba. *Decidability and Complexity Issues for Infinite-State Processes*. 2003. PhD thesis. xii+171 pp.

DS-03-2 Frank D. Valencia. *Temporal Concurrent Constraint Programming*. February 2003. PhD thesis. xvii+174.

DS-03-1 Claus Brabrand. *Domain Specific Languages for Interactive Web Services*. January 2003. PhD thesis. xiv+214 pp.

DS-02-5 Rasmus Pagh. *Hashing, Randomness and Dictionaries*. October 2002. PhD thesis. x+167 pp.

DS-02-4 Anders Møller. *Program Verification with Monadic Second-Order Logic & Languages for Web Service Development*. September 2002. PhD thesis. xvi+337 pp.