Full Abstraction for HOPLA

Mikkel Nygaard[1] and Glynn Winskel[2]

[1] BRICS*, University of Aarhus
[2] Computer Laboratory, University of Cambridge

Abstract. A fully abstract denotational semantics for the higher-order process language HOPLA is presented. It characterises contextual and logical equivalence, the latter linking up with simulation. The semantics is a clean, domain-theoretic description of processes as downwards-closed sets of computation paths: the operations of HOPLA arise as syntactic encodings of canonical constructions on such sets; full abstraction is a direct consequence of expressiveness with respect to computation paths; and simple proofs of soundness and adequacy shows correspondence between the denotational and operational semantics.

1 Introduction

HOPLA (Higher-Order Process LAnguage [19]) is an expressive language for higher-order nondeterministic processes. It has a straightforward operational semantics supporting a standard bisimulation congruence, and can directly encode calculi like CCS, higher-order CCS and mobile ambients with public names. The language came out of work on a linear domain theory for concurrency, based on a categorical model of linear logic and associated comonads [4, 18], the comonad used for HOPLA being an exponential ! of linear logic.

The denotational semantics given in [19] interpreted processes as presheaves. Here we consider a "path semantics" for HOPLA which allows us to characterise operationally the distinguishing power of the notion of computation path underlying the presheaf semantics (in contrast to the distinguishing power of the presheaf structure itself). Path semantics is similar to trace semantics [10] in that 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. Indeed, we characterise contextual equivalence for HOPLA as path equivalence and show that this coincides with logical equivalence for a fragment of Hennessy-Milner logic which is characteristic for simulation equivalence in the case of image-finite processes [8].

To increase the expressiveness of HOPLA (for example, to include the type used in [24] for CCS with late value-passing), while still ensuring that every operation in the language has a canonical semantics, we decompose the "prefix-sum" type $\Sigma_{\alpha \in A} \alpha.\mathbb{P}_\alpha$ in [19] into a sum type $\Sigma_{\alpha \in A} \mathbb{P}_\alpha$ and an anonymous action

---

* Basic Research in Computer Science (`www.brics.dk`),
funded by the Danish National Research Foundation.

prefix type $!\mathbb{P}$. The sum type, also a product, is associated with injection ("tagging") and projection term constructors, βt and $\pi_\beta t$ for $\beta \in A$. The prefix type is associated with constructions of prefixing $!t$ and prefix match $[u > !x \Rightarrow t]$, subsuming the original terms $\beta.t$ and $[u > \beta.x \Rightarrow t]$ using $\beta!t$ and $[\pi_\beta u > !x \Rightarrow t]$.

In Sect. 2 we present a domain theory of path sets, used in Sect. 3 to give a fully abstract denotational semantics to HOPLA. Section 4 presents the operational semantics of HOPLA, essentially that of [19], and relates the denotational and operational semantics with pleasingly simple proofs of soundness and adequacy. Section 5 concludes with a discussion of related and future work.

2 Domain Theory from Path Sets

In the path semantics, processes are intuitively represented as collections of their computation paths. Paths are elements of preorders $\mathbb{P}, \mathbb{Q}, \ldots$ 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*. Path sets $X \subseteq \mathbb{P}$ ordered by inclusion form the elements of the 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 [7], $\widehat{\mathbb{P}}$ 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* [16]. Primes are down-closures $\mathrm{y}_{\mathbb{P}} p = \{p' : p' \leq_{\mathbb{P}} p\}$ 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:

$$X = \textstyle\bigcup_{p \in X} \mathrm{y}_{\mathbb{P}} p \ . \tag{1}$$

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.

$$\begin{tikzcd}
\mathbb{P} \arrow[r, "\mathrm{y}_{\mathbb{P}}"] \arrow[dr, "f"'] & \widehat{\mathbb{P}} \arrow[d, dotted, "f^\dagger"] \\
 & C
\end{tikzcd}
\qquad f^\dagger X = \textstyle\bigcup_{p \in X} fp \ . \tag{2}$$

We call $f^\dagger$ *the extension of* f *along* $\mathrm{y}_{\mathbb{P}}$. Uniqueness of $f^\dagger$ follows from (1).

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), 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:

$$[\mathbb{P}, \widehat{\mathbb{Q}}] \cong [\mathbb{P}, [\mathbb{Q}^{\mathrm{op}}, \mathbf{2}]] \cong [\mathbb{P} \times \mathbb{Q}^{\mathrm{op}}, \mathbf{2}] = [(\mathbb{P}^{\mathrm{op}} \times \mathbb{Q})^{\mathrm{op}}, \mathbf{2}] \cong \widehat{\mathbb{P}^{\mathrm{op}} \times \mathbb{Q}} \ . \qquad (3)$$

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.

2.1 Linear and Continuous Categories

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 [5, 22]. 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 $\mathbf{\&}_{i \in I} \mathbb{P}_i$ 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 $\langle f_i \rangle_{i \in I} : \mathbb{P} \to \mathbf{\&}_{i \in I} \mathbb{P}_i$ 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 as required by Seely's definition of a model of linear logic [22].

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 sum 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

$$\begin{tikzcd} \mathbb{P}_\beta \arrow[r, shift right, "in_\beta"'] & \Sigma_{\alpha \in A} \mathbb{P}_\alpha \arrow[l, shift right, "\pi_\beta"'] \end{tikzcd} \quad \text{such that} \quad \begin{array}{r} \pi_\beta \circ in_\beta = 1_{\mathbb{P}_\beta} \ , \\ \pi_\beta \circ in_\alpha = \varnothing \ \text{ if } \alpha \neq \beta, \text{ and} \\ \Sigma_{\alpha \in A}(in_\alpha \circ \pi_\alpha) = 1_{\Sigma_{\alpha \in A} \mathbb{P}_\alpha} \ . \end{array} \tag{4}$$

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. 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 $\preceq_{\mathbb{P}}$, defined for arbitrary subsets X, Y of $\mathbb{P}$ as follows:

$$X \preceq_{\mathbb{P}} Y \iff_{\mathrm{def}} \forall p \in X. \exists q \in Y. p \leq_{\mathbb{P}} q \ . \tag{5}$$

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 $i_{\mathbb{P}} : !\mathbb{P} \to \widehat{\mathbb{P}}$ sending a finite set $\{p_1, \ldots, p_n\}$ to the join $\mathrm{y}_{\mathbb{P}} p_1 + \cdots + \mathrm{y}_{\mathbb{P}} p_n$. 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:

$$X = \bigcup_{P \preceq_{\mathbb{P}} X} i_{\mathbb{P}} P \ . \tag{6}$$

Further, $\widehat{\mathbb{P}}$ is the *free directed-join completion* of $!\mathbb{P}$ (also known as the ideal completion of $!\mathbb{P}$). 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.

$$\begin{tikzcd} !\mathbb{P} \arrow[r, "i_{\mathbb{P}}"] \arrow[dr, "f"'] & \widehat{\mathbb{P}} \arrow[d, dotted, "f^\ddagger"] \\ & C \end{tikzcd} \qquad f^\ddagger X = \bigcup_{P \preceq_{\mathbb{P}} X} f P \ . \tag{7}$$

Uniqueness of $f^\ddagger$, called the *extension of* f *along* $i_{\mathbb{P}}$, follows from (6). As before, we can replace C by a nondeterministic domain $\widehat{\mathbb{Q}}$ and by the freeness properties (2) and (7), 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.

2.2 An Adjunction

As linear maps are continuous, **Cts** has **Lin** as a sub-category, one which shares the same objects. We saw above that there is a bijection

$$\mathbf{Lin}(!\mathbb{P}, \mathbb{Q}) \cong \mathbf{Cts}(\mathbb{P}, \mathbb{Q}) \ . \tag{8}$$

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 (7) 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). These maps are the unit and counit, respectively, of the adjunction:

$$\eta_{\mathbb{P}} X = \bigcup_{P \preceq_{\mathbb{P}} X} \mathrm{y}_{!\mathbb{P}} P \qquad \varepsilon_{\mathbb{P}} X = \bigcup_{P \in X} i_{\mathbb{P}} P \tag{9}$$

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 (8) 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 sub-category of **Cts**, the counit is also a map in **Cts**. We have $\varepsilon_{\mathbb{P}} \circ \eta_{\mathbb{P}} = 1_{\mathbb{P}}$ and $1_{!\mathbb{P}} \leq \eta_{\mathbb{P}} \circ \varepsilon_{\mathbb{P}}$ for all objects $\mathbb{P}$.

Right adjoints preserve products, and so **Cts** has 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,

$$k : \mathbb{1} \cong !\mathbb{O} \quad \text{and} \quad m_{\mathbb{P},\mathbb{Q}} : !\mathbb{P} \times !\mathbb{Q} \cong !(\mathbb{P} \mathbin{\&} \mathbb{Q}) \ , \tag{10}$$

with $m_{\mathbb{P},\mathbb{Q}}$ mapping a pair $(P, Q) \in !\mathbb{P} \times !\mathbb{Q}$ to the union $\mathit{in}_1\, P \cup \mathit{in}_2\, Q$; 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 $r^{\mathbf{Cts}}_{\mathbb{Q}} : \mathbb{Q} \mathbin{\&} \mathbb{O} \cong \mathbb{Q}$, making ! symmetric monoidal. It then follows [13] 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

$$l : \mathbb{O} \to \mathbb{1} \quad \text{and} \quad n_{\mathbb{P},\mathbb{Q}} : \mathbb{P} \mathbin{\&} \mathbb{Q} \to \mathbb{P} \times \mathbb{Q} \tag{11}$$

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 $h(\mathit{in}_1\, P \cup \mathit{in}_2\, Q) = i_{\mathbb{P}} P \times i_{\mathbb{Q}} Q$. Also, the unit makes the diagrams below commute and the counit satisfies similar properties.

$$\begin{tikzcd}
 & \mathbb{P} \mathbin{\&} \mathbb{Q} \arrow[dl, "\eta_{\mathbb{P}} \& \eta_{\mathbb{Q}}"'] \arrow[dr, "\eta_{\mathbb{P} \& \mathbb{Q}}"] & \\
!\mathbb{P} \mathbin{\&} !\mathbb{Q} \arrow[r, "n_{!\mathbb{P},!\mathbb{Q}}"'] & !\mathbb{P} \times !\mathbb{Q} \arrow[r, "m_{\mathbb{P},\mathbb{Q}}"'] & !(\mathbb{P} \mathbin{\&} \mathbb{Q})
\end{tikzcd} \qquad \begin{tikzcd}
\mathbb{O} \arrow[r, "l"] \arrow[dr, "\eta_{\mathbb{O}}"'] & \mathbb{1} \arrow[d, "k"] \\
 & !\mathbb{O}
\end{tikzcd} \tag{12}$$

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

$$\mathbb{P} \mathbin{\&} !\mathbb{Q} \xrightarrow{\eta_{\mathbb{P}}\&1_{!\mathbb{Q}}} !\mathbb{P} \mathbin{\&} !\mathbb{Q} \xrightarrow{n_{!\mathbb{P},!\mathbb{Q}}} !\mathbb{P} \times !\mathbb{Q} \xrightarrow{m_{\mathbb{P},\mathbb{Q}}} !(\mathbb{P} \mathbin{\&} \mathbb{Q}) \quad . \tag{13}$$

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 $(- \mathbin{\&} \mathbb{Q})$ in **Cts** via the chain

$$\begin{aligned}\mathbf{Cts}(\mathbb{P} \mathbin{\&} \mathbb{Q}, \mathbb{R}) &\cong \mathbf{Lin}(!(\mathbb{P} \mathbin{\&} \mathbb{Q}), \mathbb{R}) \cong \mathbf{Lin}(!\mathbb{P} \times !\mathbb{Q}, \mathbb{R}) \\ &\cong \mathbf{Lin}(!\mathbb{P}, !\mathbb{Q} \multimap \mathbb{R}) \cong \mathbf{Cts}(\mathbb{P}, !\mathbb{Q} \multimap \mathbb{R}) = \mathbf{Cts}(\mathbb{P}, \mathbb{Q} \to \mathbb{R})\end{aligned} \tag{14}$$

—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 [1, 22]; see also [13] for a recent survey of such models.

3 Denotational Semantics

HOPLA is directly suggested by the structure of **Cts**. The language is typed with types given by the grammar

$$\mathbb{T} ::= \mathbb{T}_1 \to \mathbb{T}_2 \mid \Sigma_{\alpha \in A} \mathbb{T}_\alpha \mid !\mathbb{T} \mid T \mid \mu_j \vec{T}.\vec{\mathbb{T}} \quad . \tag{15}$$

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 $\mu_j T_1, \ldots, T_k.(\mathbb{T}_1, \ldots, \mathbb{T}_k)$ and is interpreted as the j-component, for $1 \leq j \leq k$, of “the least” solution to the defining equations $T_1 = \mathbb{T}_1, \ldots, T_k = \mathbb{T}_k$, in which the expressions $\mathbb{T}_1, \ldots, \mathbb{T}_k$ 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 [21, 12]. Here, it will be convenient to give a concrete, inductive characterisation based on a language of *paths*:

$$p, q ::= P \mapsto q \mid \beta p \mid P \mid abs\, p \quad . \tag{16}$$

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 on top of rules defining the ordering on $\mathbb{P}$ using judgements $p \leq_{\mathbb{P}} p'$. Recall that $P \preceq_{\mathbb{P}} P'$ means $\forall p \in P. \exists p' \in P'.\ p \leq_{\mathbb{P}} p'$.

$$\frac{P : !\mathbb{P} \quad q : \mathbb{Q}}{P \mapsto q : \mathbb{P} \to \mathbb{Q}} \qquad \frac{p : \mathbb{P}_\beta \quad \beta \in A}{\beta p : \Sigma_{\alpha \in A} \mathbb{P}_\alpha} \qquad \frac{p_1 : \mathbb{P} \cdots p_n : \mathbb{P}}{\{p_1, \ldots, p_n\} : !\mathbb{P}} \qquad \frac{p : \mathbb{T}_j[\mu \vec{T}.\vec{\mathbb{T}}/\vec{T}]}{abs\, p : \mu_j \vec{T}.\vec{\mathbb{T}}}$$

$$\frac{P' \preceq_{\mathbb{P}} P \quad \leq_{\mathbb{Q}} q'}{P \mapsto q \leq_{\mathbb{P} \mapsto \mathbb{Q}} P' \mapsto q'} \quad \frac{p \leq_{\mathbb{P}_\beta} p'}{\beta p \leq_{\Sigma_{\alpha \in A} \mathbb{P}_\alpha} \beta p'} \qquad \frac{P \preceq_{\mathbb{P}} P'}{P \leq_{!\mathbb{P}} P'} \qquad \frac{p \leq_{\mathbb{T}_j[\mu \vec{T}.\vec{\mathbb{T}}/\vec{T}]} p'}{abs\, p \leq_{\mu_j \vec{T}.\vec{\mathbb{T}}} abs\, p'}$$

Using information systems as in [12] yields the same representation, except for the tagging with *abs* in recursive types, done to help in the proof of adequacy in Sect. 4.1. So rather than the straight equality between a recursive type and its unfolding which we are used to from [12], we get an isomorphism $abs : \mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}] \cong \mu_j\vec{T}.\vec{\mathbb{T}}$ whose inverse we call *rep*.

As an example consider the type of CCS processes given in [19] as the path order $\mathbb{P}$ satisfying $\mathbb{P} = \Sigma_{\alpha\in A}!\mathbb{P}$ where A is a set of CCS actions. The elements of $\mathbb{P}$ then have the form $abs(\beta P)$ where $\beta \in A$ and P is a finite set of paths from $\mathbb{P}$. Intuitively, a CCS process can perform such a path if it can perform the action β and, following that, is able to perform each path in P.

The raw syntax of HOPLA terms is given by

$$t, u ::= x \,|\, rec\, x.t \,|\, \Sigma_{i\in I} t_i \,|\, \lambda x.t \,|\, t\ u \,|\, \beta t \,|\, \pi_\beta t \,|\, !t \,|\, [u > !x \Rightarrow t] \,|\, abs\, t \,|\, rep\, t\ . \tag{17}$$

The variables x in the terms $rec\, x.t$, $\lambda x.t$, and $[u > !x \Rightarrow t]$ are binding occurrences with scope t. We shall take for granted an understanding of free and bound variables, and substitution on raw terms.

Let $\mathbb{P}_1, \ldots, \mathbb{P}_k, \mathbb{Q}$ be closed type expressions and assume that the variables $x_1, \ldots, x_k$ are distinct. A syntactic judgement $x_1 : \mathbb{P}_1, \ldots, x_k : \mathbb{P}_k \vdash t : \mathbb{Q}$ stands for a map $[\![x_1 : \mathbb{P}_1, \ldots, x_k : \mathbb{P}_k \vdash t : \mathbb{Q}]\!] : \mathbb{P}_1 \,\&\, \cdots \,\&\, \mathbb{P}_k \to \mathbb{Q}$ in **Cts**. We'll write Γ, or Λ, for an environment list $x_1 : \mathbb{P}_1, \ldots, x_k : \mathbb{P}_k$ 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. [2]). We assume that the variables in any environment list which appears are distinct.

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

$$\frac{}{x : \mathbb{P} \vdash x : \mathbb{P}} \qquad \frac{}{\mathbb{P} \xrightarrow{1_\mathbb{P}} \mathbb{P}} \tag{18}$$

$$\frac{\Gamma \vdash t : \mathbb{Q}}{\Gamma, x : \mathbb{P} \vdash t : \mathbb{Q}} \qquad \frac{\Gamma \xrightarrow{t} \mathbb{Q}}{\Gamma \,\&\, \mathbb{P} \xrightarrow{t \& \varnothing_\mathbb{P}} \mathbb{Q} \,\&\, \mathbb{O} \xrightarrow{r^{\mathbf{Cts}}_\mathbb{Q}} \mathbb{Q}} \tag{19}$$

$$\frac{\Gamma, y : \mathbb{Q}, x : \mathbb{P}, \Lambda \vdash t : \mathbb{R}}{\Gamma, x : \mathbb{P}, y : \mathbb{Q}, \Lambda \vdash t : \mathbb{R}} \qquad \frac{\Gamma \,\&\, \mathbb{Q} \,\&\, \mathbb{P} \,\&\, \Lambda \xrightarrow{t} \mathbb{R}}{\Gamma \,\&\, \mathbb{P} \,\&\, \mathbb{Q} \,\&\, \Lambda \xrightarrow{t \circ (1_\Gamma \& s^{\mathbf{Cts}}_{\mathbb{P},\mathbb{Q}} \& 1_\Lambda)} \mathbb{R}} \tag{20}$$

$$\frac{\Gamma, x : \mathbb{P}, y : \mathbb{P} \vdash t : \mathbb{Q}}{\Gamma, z : \mathbb{P} \vdash t[z/x, z/y] : \mathbb{Q}} \qquad \frac{\Gamma \,\&\, \mathbb{P} \,\&\, \mathbb{P} \xrightarrow{t} \mathbb{Q}}{\Gamma \,\&\, \mathbb{P} \xrightarrow{1_\Gamma \& \Delta_\mathbb{P}} \Gamma \,\&\, \mathbb{P} \,\&\, \mathbb{P} \xrightarrow{t} \mathbb{Q}} \tag{21}$$

In the formation rule for contraction (21), the variable z must be fresh; the map $\Delta_\mathbb{P}$ is the usual diagonal, given as $\langle 1_\mathbb{P}, 1_\mathbb{P} \rangle$.

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 $\bigcup_{n\in\omega} f^n(\varnothing)$. Below, $\mathit{fix}\, f$ is the fixpoint in $\mathbf{Cts}(\mathit{\Gamma},\mathbb{P}) \cong \widehat{\mathit{\Gamma} \to \mathbb{P}}$ of the continuous operation f mapping $g : \mathit{\Gamma} \to \mathbb{P}$ in **Cts** to the composition $[\![t]\!] \circ (1_{\mathit{\Gamma}} \,\&\, g) \circ \mathit{\Delta}_{\mathit{\Gamma}}$.

$$\frac{\mathit{\Gamma}, x : \mathbb{P} \vdash t : \mathbb{P}}{\mathit{\Gamma} \vdash \mathit{rec}\, x.t : \mathbb{P}} \qquad \frac{\mathit{\Gamma} \,\&\, \mathbb{P} \xrightarrow{t} \mathbb{P}}{\mathit{\Gamma} \xrightarrow{\mathit{fix}\, f} \mathbb{P}} \tag{22}$$

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

$$\frac{\mathit{\Gamma} \vdash t_j : \mathbb{P} \quad \text{all } j \in I}{\mathit{\Gamma} \vdash \mathit{\Sigma}_{i\in I} t_i : \mathbb{P}} \qquad \frac{\mathit{\Gamma} \xrightarrow{t_j} \mathbb{P} \quad \text{all } j \in I}{\mathit{\Gamma} \xrightarrow{\langle t_i \rangle_{i\in I}} \boldsymbol{\&}_{i\in I}\, \mathbb{P} \xrightarrow{\mathit{\Sigma}} \mathbb{P}} \tag{23}$$

Function space. As noted at the end of Sect. 2.2, the category **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 **Cts**; its inverse is called *uncurry*. We obtain application, $\mathit{app} : (\mathbb{P} \to \mathbb{Q}) \,\&\, \mathbb{P} \to \mathbb{Q}$ as $\mathit{uncurry}(1_{\mathbb{P}\to\mathbb{Q}})$.

$$\frac{\mathit{\Gamma}, x : \mathbb{P} \vdash t : \mathbb{Q}}{\mathit{\Gamma} \vdash \lambda x.t : \mathbb{P} \to \mathbb{Q}} \qquad \frac{\mathit{\Gamma} \,\&\, \mathbb{P} \xrightarrow{t} \mathbb{Q}}{\mathit{\Gamma} \xrightarrow{\mathit{curry}\, t} \mathbb{P} \to \mathbb{Q}} \tag{24}$$

$$\frac{\mathit{\Gamma} \vdash t : \mathbb{P} \to \mathbb{Q} \quad \mathit{\Lambda} \vdash u : \mathbb{P}}{\mathit{\Gamma}, \mathit{\Lambda} \vdash t\; u : \mathbb{Q}} \qquad \frac{\mathit{\Gamma} \xrightarrow{t} \mathbb{P} \to \mathbb{Q} \quad \mathit{\Lambda} \xrightarrow{u} \mathbb{P}}{\mathit{\Gamma} \,\&\, \mathit{\Lambda} \xrightarrow{t \& u} (\mathbb{P} \to \mathbb{Q}) \,\&\, \mathbb{P} \xrightarrow{\mathit{app}} \mathbb{Q}} \tag{25}$$

Sum type. The category **Cts** does not have coproducts, but we can build a useful sum type out of the biproduct of **Lin**. The properties of (4) are obviously also satisfied in **Cts**, even though the construction is universal only in the subcategory of linear maps because composition is generally not bilinear in **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 [19] 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 $\pi_1 t$, $\pi_2 t$.

$$\frac{\mathit{\Gamma} \vdash t : \mathbb{P}_\beta \quad \beta \in A}{\mathit{\Gamma} \vdash \beta t : \mathit{\Sigma}_{\alpha\in A} \mathbb{P}_\alpha} \qquad \frac{\mathit{\Gamma} \xrightarrow{t} \mathbb{P}_\beta \quad \beta \in A}{\mathit{\Gamma} \xrightarrow{t} \mathbb{P}_\beta \xrightarrow{\mathit{in}_\beta} \mathit{\Sigma}_{\alpha\in A} \mathbb{P}_\alpha} \tag{26}$$

$$\frac{\mathit{\Gamma} \vdash t : \mathit{\Sigma}_{\alpha\in A} \mathbb{P}_\alpha \quad \beta \in A}{\mathit{\Gamma} \vdash \pi_\beta t : \mathbb{P}_\beta} \qquad \frac{\mathit{\Gamma} \xrightarrow{t} \mathit{\Sigma}_{\alpha\in A} \mathbb{P}_\alpha \quad \beta \in A}{\mathit{\Gamma} \xrightarrow{t} \mathit{\Sigma}_{\alpha\in A} \mathbb{P}_\alpha \xrightarrow{\pi_\beta} \mathbb{P}_\beta} \tag{27}$$

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. We'll say that $\eta_{\mathbb{P}}$ maps u of type $\mathbb{P}$ to a "prefixed" process $!u$ of type $!\mathbb{P}$; intuitively, the process $!u$ will be able to perform an action, which we call $!$, before continuing as u.

$$\frac{\mathit{\Gamma} \vdash u : \mathbb{P}}{\mathit{\Gamma} \vdash !u : !\mathbb{P}} \qquad \frac{\mathit{\Gamma} \xrightarrow{u} \mathbb{P}}{\mathit{\Gamma} \xrightarrow{u} \mathbb{P} \xrightarrow{\eta_{\mathbb{P}}} !\mathbb{P}} \tag{28}$$

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 write $[u > !x \Rightarrow t]$ for $\bar{t}u$. Intuitively, this construction "tests" or matches u against the pattern $!x$ and passes the results of successful matches for x on to t. Indeed, first prefixing a term u of type $\mathbb{P}$ and then matching yields a successful match u for x as $\bar{t}(\eta_{\mathbb{P}} u) = tu$. By linearity of $\bar{t}$, the possibly multiple results of successful matches are nondeterministically summed together; the denotations of $[\Sigma_{i\in I} u_i > !x \Rightarrow t]$ and $\Sigma_{i\in I}[u_i > !x \Rightarrow t]$ are identical.

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 (13):

$$\frac{\Gamma, x : \mathbb{P} \vdash t : \mathbb{Q} \quad \Lambda \vdash u : !\mathbb{P}}{\Gamma, \Lambda \vdash [u > !x \Rightarrow t] : \mathbb{Q}} \qquad \frac{\Gamma \mathbin{\&} \mathbb{P} \xrightarrow{t} \mathbb{Q} \quad \Lambda \xrightarrow{u} !\mathbb{P}}{\Gamma \mathbin{\&} \Lambda \xrightarrow{1_\Gamma \& u} \Gamma \mathbin{\&} !\mathbb{P} \xrightarrow{str_{\Gamma,\mathbb{P}}} !(\Gamma \mathbin{\&} \mathbb{P}) \xrightarrow{\bar{t}} \mathbb{Q}} \tag{29}$$

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

$$\frac{\Gamma \vdash t : \mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}]}{\Gamma \vdash abs\, t : \mu_j\vec{T}.\vec{\mathbb{T}}} \qquad \frac{\Gamma \xrightarrow{t} \mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}]}{\Gamma \xrightarrow{t} \mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}] \xrightarrow{abs} \mu_j\vec{T}.\vec{\mathbb{T}}} \tag{30}$$

$$\frac{\Gamma \vdash t : \mu_j\vec{T}.\vec{\mathbb{T}}}{\Gamma \vdash rep\, t : \mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}]} \qquad \frac{\Gamma \xrightarrow{t} \mu_j\vec{T}.\vec{\mathbb{T}}}{\Gamma \xrightarrow{t} \mu_j\vec{T}.\vec{\mathbb{T}} \xrightarrow{rep} \mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}]} \tag{31}$$

3.1 Useful Equivalences

We provide some technical results about the path semantics which are used in the proof of soundness, Proposition 10. Proofs can be found in [20].

Lemma 1 (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 $[\![t]\!] \circ (1_\Gamma \mathbin{\&} [\![u]\!])$.*

Corollary 2. *If $\Gamma, x : \mathbb{P} \vdash t : \mathbb{P}$, then $\Gamma \vdash t[rec\, x.t/x] : \mathbb{P}$ and $[\![rec\, x.t]\!] = [\![t[rec\, x.t/x]]\!]$ so recursion amounts to unfolding.*

Corollary 3. *Application amounts to substitution. In the situation of the substitution lemma, we have $[\![(\lambda x.t)\ u]\!] = [\![t[u/x]]\!]$.*

Proposition 4. *From the properties of the biproduct and by linearity of injections and projections, we get:*

$$\begin{aligned} [\![\pi_\beta(\beta t)]\!] &= [\![t]\!] \\ [\![\pi_\alpha(\beta t)]\!] &= \varnothing \qquad \text{if } \alpha \neq \beta \\ [\![\Sigma_{\alpha\in A}\alpha(\pi_\alpha(t))]\!] &= [\![t]\!] \end{aligned} \qquad \begin{aligned} [\![\beta(\Sigma_{i\in I} t_i)]\!] &= [\![\Sigma_{i\in I}(\beta t_i)]\!] \\ [\![\pi_\beta(\Sigma_{i\in I} t_i)]\!] &= [\![\Sigma_{i\in I}(\pi_\beta t_i)]\!] \end{aligned} \tag{32}$$

Proposition 5. *The prefix match satisfies the properties*

$$\begin{aligned} [\![[!u > !x \Rightarrow t]]\!] &= [\![t[u/x]]\!] \\ [\![[\Sigma_{i\in I} u_i > !x \Rightarrow t]]\!] &= [\![\Sigma_{i\in I}[u_i > !x \Rightarrow t]]\!] \end{aligned} \tag{33}$$

3.2 Full Abstraction

We define a *program* to be a closed term t of type $!\mathbb{O}$. 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 denotational semantics gives rise to a type-respecting contextual preorder [15]:

Definition 6. *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* $t_1 \mathrel{\underset{\sim}{\sqsubset}} t_2$, *iff for all* $(\Gamma,\mathbb{P})$*-program contexts* C, *we have* $[\![C(t_1)]\!] \neq \varnothing \implies [\![C(t_2)]\!] \neq \varnothing$. *If both* $t_1 \mathrel{\underset{\sim}{\sqsubset}} t_2$ *and* $t_2 \mathrel{\underset{\sim}{\sqsubset}} t_1$, *we say that* t_1 *and* t_2 *are* contextually equivalent.

Contextual equivalence coincides with path equivalence:

Theorem 7 (Full abstraction). *For any terms* $\Gamma \vdash t_1 : \mathbb{P}$ *and* $\Gamma \vdash t_2 : \mathbb{P}$,

$$[\![t_1]\!] \subseteq [\![t_2]\!] \iff t_1 \mathrel{\underset{\sim}{\sqsubset}} t_2 \ . \tag{34}$$

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 monotonicity, and so $t_1 \mathrel{\underset{\sim}{\sqsubset}} t_2$ as wanted.

Now suppose that $t_1 \mathrel{\underset{\sim}{\sqsubset}} t_2$. With $p : \mathbb{P}$ we define closed terms t_p of type $\mathbb{P}$ and $(\mathbb{O},\mathbb{P})$-program contexts C_p that respectively "realise" and "consume" the path p, by induction on the structure of p. We'll also need realisers t'_P and consumers C'_P of finite sets of paths:

$$\begin{aligned}
t_{P\mapsto q} &\equiv \lambda x.[C'_P(x) > !x' \Rightarrow t_q] & C_{P\mapsto q} &\equiv C_q(-\ t'_P)\\
t_{\beta p} &\equiv \beta t_p & C_{\beta p} &\equiv C_p(\pi_\beta -)\\
t_P &\equiv !t'_P & C_P &\equiv [- > !x \Rightarrow C'_P(x)]\\
t_{abs\,p} &\equiv abs\, t_p & C_{abs\,p} &\equiv C_p(rep\ -)\\
t'_{\{p_1,\ldots,p_n\}} &\equiv t_{p_1} + \cdots + t_{p_n}\\
C'_{\{p_1,\ldots,p_n\}} &\equiv [C_{p_1} > !x \Rightarrow \cdots \Rightarrow [C_{p_n} > !x \Rightarrow !\varnothing]\cdots]
\end{aligned} \tag{35}$$

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):

$$\begin{aligned}
[\![t_p]\!] &= \mathrm{y}_{\mathbb{P}} p & [\![\lambda z.C_p(z)]\!] &= \mathrm{y}_{\mathbb{P}\to!\mathbb{O}}(\{p\} \mapsto \varnothing)\\
[\![t'_P]\!] &= i_{\mathbb{P}} P & [\![\lambda z.C'_P(z)]\!] &= \mathrm{y}_{\mathbb{P}\to!\mathbb{O}}(P \mapsto \varnothing)
\end{aligned} \tag{36}$$

Suppose t_1 and t_2 are closed. Given any $p \in [\![t_1]\!]$ we have $[\![C_p(t_1)]\!] \neq \varnothing$ and so using $t_1 \mathrel{\underset{\sim}{\sqsubset}} t_2$, 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 $\Gamma \equiv x_1 : \mathbb{P}_1, \ldots, x_k : \mathbb{P}_k$. 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

$$t_1 \mathrel{\underset{\sim}{\sqsubset}} t_2 \implies \lambda\vec{x}.t_1 \mathrel{\underset{\sim}{\sqsubset}} \lambda\vec{x}.t_2 \implies [\![\lambda\vec{x}.t_1]\!] \subseteq [\![\lambda\vec{x}.t_2]\!] \implies [\![t_1]\!] \subseteq [\![t_2]\!] \ . \tag{37}$$

The proof is complete. $\square$

$$\frac{\mathbb{P} : t[\mathit{rec}\, x.t/x] \xrightarrow{a} t'}{\mathbb{P} : \mathit{rec}\, x.t \xrightarrow{a} t'} \qquad \frac{\mathbb{P} : t_j \xrightarrow{a} t'}{\mathbb{P} : \Sigma_{i \in I} t_i \xrightarrow{a} t'}\, j \in I \qquad \frac{\mathbb{Q} : t[u/x] \xrightarrow{a} t'}{\mathbb{P} \to \mathbb{Q} : \lambda x.t \xrightarrow{u \mapsto a} t'} \qquad \frac{\mathbb{P} \to \mathbb{Q} : t \xrightarrow{u \mapsto a} t'}{\mathbb{Q} : t\, u \xrightarrow{a} t'}$$

$$\frac{\mathbb{P}_\beta : t \xrightarrow{a} t'}{\Sigma_{\alpha \in A} \mathbb{P}_\alpha : \beta t \xrightarrow{\beta a} t'} \qquad \frac{\Sigma_{\alpha \in A} \mathbb{P}_\alpha : t \xrightarrow{\beta a} t'}{\mathbb{P}_\beta : \pi_\beta t \xrightarrow{a} t'} \qquad \frac{}{!\mathbb{P} : !t \xrightarrow{!} t} \qquad \frac{!\mathbb{P} : u \xrightarrow{!} u' \quad \mathbb{Q} : t[u'/x] \xrightarrow{a} t'}{\mathbb{Q} : [u > !x \Rightarrow t] \xrightarrow{a} t'}$$

$$\frac{\mathbb{T}_j[\mu \vec{T}.\vec{\mathbb{T}}/\vec{T}] : t \xrightarrow{a} t'}{\mu_j \vec{T}.\vec{\mathbb{T}} : \mathit{abs}\, t \xrightarrow{\mathit{abs}\, a} t'} \qquad \frac{\mu_j \vec{T}.\vec{\mathbb{T}} : t \xrightarrow{\mathit{abs}\, a} t'}{\mathbb{T}_j[\mu \vec{T}.\vec{\mathbb{T}}/\vec{T}] : \mathit{rep}\, t \xrightarrow{a} t'}$$

Fig. 1. Operational rules

4 Operational Semantics

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

$$a ::= u \mapsto a \mid \beta a \mid ! \mid \mathit{abs}\, a \ . \tag{38}$$

We assign types to actions a using a judgement 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 $\mathbb{P}'$.

$$\frac{\vdash u : \mathbb{P} \quad \mathbb{Q} : a : \mathbb{P}'}{\mathbb{P} \to \mathbb{Q} : u \mapsto a : \mathbb{P}'} \qquad \frac{\mathbb{P}_\beta : a : \mathbb{P}' \quad \beta \in A}{\Sigma_{\alpha \in A} \mathbb{P}_\alpha : \beta a : \mathbb{P}'} \qquad \frac{}{!\mathbb{P} : ! : \mathbb{P}} \qquad \frac{\mathbb{T}_j[\mu \vec{T}.\vec{\mathbb{T}}/\vec{T}] : a : \mathbb{P}'}{\mu_j \vec{T}.\vec{\mathbb{T}} : \mathit{abs}\, a : \mathbb{P}'} \tag{39}$$

Notice that in $\mathbb{P} : a : \mathbb{P}'$, the type $\mathbb{P}'$ is unique given $\mathbb{P}$ and a. The operational rules of Fig. 1 define a relation $\mathbb{P} : t \xrightarrow{a} t'$ where $\vdash t : \mathbb{P}$ and $\mathbb{P} : a : \mathbb{P}'$.[3] An easy rule induction shows

Proposition 8. *If* $\mathbb{P} : t \xrightarrow{a} t'$ *with* $\mathbb{P} : a : \mathbb{P}'$*, then* $\vdash t' : \mathbb{P}'$.

Accordingly, we'll write $\mathbb{P} : t \xrightarrow{a} t' : \mathbb{P}'$ when $\mathbb{P} : t \xrightarrow{a} t'$ and $\mathbb{P} : a : \mathbb{P}'$.

4.1 Soundness and Adequacy

For each action $\mathbb{P} : a : \mathbb{P}'$ we define a linear map $a^* : \mathbb{P} \to !\mathbb{P}'$ which intuitively maps a process t of type $\mathbb{P}$ to a representation of its possible successors after performing the action a. In order to distinguish between, say, the successor $\varnothing$ and no successors, a^* embeds into the type $!\mathbb{P}'$ rather than using $\mathbb{P}'$ itself. For instance, the successors after action ! of the processes $!\varnothing$ and $\varnothing$ are, respectively, $!^*[\![!\varnothing]\!] = 1_{!\mathbb{P}}(\eta_{\mathbb{P}}\varnothing) = \eta_{\mathbb{P}}\varnothing$ and $!^*[\![\varnothing]\!] = 1_{!\mathbb{P}}\varnothing = \varnothing$. It will be convenient to treat a^* as a syntactic operation and so we define a term a^*t such that $[\![a^*t]\!] = a^*[\![t]\!]$:

$$\begin{aligned} (u \mapsto a)^* &= a^* \circ \mathit{app} \circ (- \,\&\, [\![u]\!]) & (u \mapsto a)^* t &\equiv a^*(t\, u) \\ (\beta a)^* &= a^* \circ \pi_\beta & (\beta a)^* t &\equiv a^*(\pi_\beta t) \\ !^* &= 1_{!\mathbb{P}} & !^* t &\equiv t \\ (\mathit{abs}\, a)^* &= a^* \circ \mathit{rep} & (\mathit{abs}\, a)^* t &\equiv a^*(\mathit{rep}\, t) \end{aligned} \tag{40}$$

---

[3] The explicit types in the operational rules were missing in the rules given in [19]. They are needed to ensure that the types of t and a agree in transitions.

The role of a^* is to reduce the action a to a prefix action. Formally the reduction is captured by the lemma below, proved by structural induction on a:

Lemma 9. $\mathbb{P} : t \xrightarrow{a} t' : \mathbb{P}' \iff\ !\mathbb{P}' : a^* t \xrightarrow{!} t' : \mathbb{P}'$.

Note that the reduction is done uniformly at all types using deconstructor contexts: application, projection, and unfolding. This explains the somewhat mysterious function space actions $u \mapsto a$. A similar use of labels to carry context information appears e.g. in [6].

Soundness says that the operational notion of "successor" is included in the semantic notion. The proof is by rule induction on the transition rules, see [20].

Proposition 10 (Soundness). *If* $\mathbb{P} : t \xrightarrow{a} t' : \mathbb{P}'$*, then* $\eta_{\mathbb{P}'}[\![t']\!] \subseteq a^*[\![t]\!]$.

We obtain a corresponding adequacy result using logical relations $X \trianglelefteq_{\mathbb{P}} t$ between subsets $X \subseteq \mathbb{P}$ and closed terms of type $\mathbb{P}$. Intuitively, $X \trianglelefteq_{\mathbb{P}} t$ 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 [23], 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:

$$\begin{aligned} X \trianglelefteq_{\mathbb{P}} t &\iff_{\mathrm{def}} \forall p \in X.\ p \in_{\mathbb{P}} t \\ P \mapsto q \in_{\mathbb{P}\to\mathbb{Q}} t &\iff_{\mathrm{def}} \forall u.\ (P \trianglelefteq_{\mathbb{P}} u \implies q \in_{\mathbb{Q}} t\ u) \\ \beta p \in_{\Sigma_{\alpha\in A}\mathbb{P}_\alpha} t &\iff_{\mathrm{def}} p \in_{\mathbb{P}_\beta} \pi_\beta t \\ P \in_{!\mathbb{P}} t &\iff_{\mathrm{def}} \exists t'.\ !\mathbb{P} : t \xrightarrow{!} t' : \mathbb{P} \text{ and } P \trianglelefteq_{\mathbb{P}} t' \\ \mathit{abs}\, p \in_{\mu_j \vec{T}.\vec{\mathbb{T}}} t &\iff_{\mathrm{def}} p \in_{\mathbb{T}_j[\mu\vec{T}.\vec{\mathbb{T}}/\vec{T}]} \mathit{rep}\, t \end{aligned} \tag{41}$$

The main lemma below is proved by structural induction on terms, see [20].

Lemma 11. *Suppose* $\vdash t : \mathbb{P}$*. Then* $[\![t]\!] \trianglelefteq_{\mathbb{P}} t$.

Proposition 12 (Adequacy). *Suppose* $\vdash t : \mathbb{P}$ *and* $\mathbb{P} : a : \mathbb{P}'$*. Then*

$$a^*[\![t]\!] \neq \varnothing \iff \exists t'.\ \mathbb{P} : t \xrightarrow{a} t' : \mathbb{P}' \tag{42}$$

Proof. The "$\Leftarrow$" direction follows from soundness. Assume $a^*[\![t]\!] \neq \varnothing$. 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 Lemma 11, which implies the existence of a term t' such that $!\mathbb{P}' : a^* t \xrightarrow{!} t' : \mathbb{P}'$. By Lemma 9 we have $\mathbb{P} : t \xrightarrow{a} t' : \mathbb{P}'$. $\square$

4.2 Full Abstraction w.r.t. Operational Semantics

Adequacy allows an operational formulation of contextual equivalence. If t is a program, we write $t\xrightarrow{!}$ if there exists t' such that $!\mathbb{O} : t \xrightarrow{!} t' : \mathbb{O}$. We then have $t\xrightarrow{!}$ iff $[\![t]\!] \neq \varnothing$ by adequacy. 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 $[\![t_1]\!] \subseteq_{\Gamma} [\![t_2]\!]$ holds iff for all $(\Gamma, \mathbb{P})$-program contexts C, we have the implication $C(t_1)\xrightarrow{!} \implies C(t_2)\xrightarrow{!}$.

4.3 Simulation

The path semantics does not capture enough of the branching behaviour of processes to characterise bisimilarity (for that, the presheaf semantics is needed, see [11, 19]). 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; it is characteristic for simulation equivalence in the case of image-finite processes [8]. With a ranging over actions, formulae are given by

$$\phi ::= \langle a \rangle \phi \mid \textstyle\bigwedge_{i \le n} \phi_i \ . \tag{43}$$

The empty conjunction is written $\top$. We type formulae using judgements $\phi : \mathbb{P}$, the idea being that only processes of type $\mathbb{P}$ should be described by $\phi : \mathbb{P}$.

$$\frac{\mathbb{P} : a : \mathbb{P}' \quad \phi : \mathbb{P}'}{\langle a \rangle \phi : \mathbb{P}} \qquad \frac{\phi_i : \mathbb{P} \quad \text{all } i \le n}{\bigwedge_{i \le n} \phi_i : \mathbb{P}} \tag{44}$$

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

$$t \vDash \langle a \rangle \phi : \mathbb{P} \iff \exists t'.\, \mathbb{P} : t \xrightarrow{a} t' : \mathbb{P}' \text{ and } t' \vDash \phi : \mathbb{P}' \tag{45}$$

$$t \vDash \textstyle\bigwedge_{i \le n} \phi_i : \mathbb{P} \iff t \vDash \phi_i : \mathbb{P} \text{ for each } i \le n \ . \tag{46}$$

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

Definition 13. *Closed terms t_1, t_2 of the same type $\mathbb{P}$ are related by* logical preorder*, written $t_1 \mathrel{\underset{\sim}{\sqsubset}}_{\mathrm{L}} t_2$, iff for all formulae $\phi : \mathbb{P}$ we have $t_1 \vDash \phi : \mathbb{P} \implies t_2 \vDash \phi : \mathbb{P}$. If both $t_1 \mathrel{\underset{\sim}{\sqsubset}}_{\mathrm{L}} t_2$ and $t_2 \mathrel{\underset{\sim}{\sqsubset}}_{\mathrm{L}} t_1$, we say that t_1 and t_2 are* logically equivalent.

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

$$C_\phi(t)\xrightarrow{!} \iff t \vDash \phi : \mathbb{P} \ . \tag{47}$$

Define

$$\begin{gathered}
C_{\langle u \mapsto a \rangle \phi} \equiv C_{\langle a \rangle \phi}(-\ u) \ , \qquad C_{\langle ! \rangle \phi} \equiv [- > !x \Rightarrow C_\phi(x)] \ , \\
C_{\langle \beta a \rangle \phi} \equiv C_{\langle a \rangle \phi}(\pi_\beta -) \ , \qquad C_{\langle abs\, a \rangle \phi} \equiv C_{\langle a \rangle \phi}(rep\ -) \ , \\
C_{\bigwedge_{i \le n} \phi_i} \equiv [C_{\phi_1} > !x \Rightarrow \cdots \Rightarrow [C_{\phi_n} > !x \Rightarrow !\varnothing] \cdots] \ .
\end{gathered} \tag{48}$$

Corollary 14. *For closed terms t_1 and t_2 of the same type,*

$$t_1 \underset{\sim}{\sqsubset} t_2 \iff t_1 \underset{\sim}{\sqsubset}_{\mathrm{L}} t_2 \ . \tag{49}$$

Proof. The direction "⇒" follows from (47) and the remarks of Sect. 4.2. As for the converse, we observe that the program contexts C_p of the full abstraction proof in Sect. 3.2 are all subsumed by the contexts above. Thus, if $t_1 \underset{\sim}{\sqsubset}_{\mathrm{L}} t_2$, then $[\![t_1]\!] \subseteq [\![t_2]\!]$ and so $t_1 \underset{\sim}{\sqsubset} t_2$ by full abstraction. □

5 Related and Future Work

Matthew Hennessy's fully abstract semantics for higher-order CCS [9] is a path semantics, and what we have presented here can be seen as a generalisation of his work via the translation of higher-order CCS into HOPLA, see [19].

The presheaf semantics originally given for HOPLA is a refined version of the path semantics. A path set $X \in \widehat{\mathbb{P}}$ can be seen to give a "yes/no answer" to the question of whether or not a path $p \in \mathbb{P}$ can be realised by the process (cf. the representation in Sect. 2 of path sets as monotone maps $\mathbb{P}^{\mathrm{op}} \to \mathbf{2}$). A presheaf over $\mathbb{P}$ is a functor $\mathbb{P}^{\mathrm{op}} \to \mathbf{Set}$ to the category of sets and functions, and gives instead a set of "realisers", saying *how* a path may be realised. This extra information can be used to obtain refined versions of the proofs of soundness and adequacy, giving hope of extending the full abstraction result to a characterisation of bisimilarity, possibly in terms of *open maps* [11].

Replacing the exponential ! by a "lifting" comonad yields a model **Aff** of affine linear logic and an affine version of HOPLA, again with a fully abstract path semantics [20]. The tensor operation of **Aff** can be understood as a simple parallel composition of event structures [17]. Thus, the affine language holds promise of extending our approach to "independence" models like Petri nets or event structures in which computation paths are partial orders of events. Work is in progress to provide an operational semantics for this language together with results similar to those obtained here.

Being a higher-order process language, HOPLA allows process passing and so can express certain forms of mobility, in particular that present in the ambient calculus with public names [3, 19]. Another kind of mobility, mobility of communication links, arises from name-generation as in the π-calculus [14]. Inspired by HOPLA, Francesco Zappa Nardelli and GW have defined a higher-order process language with name-generation, allowing encodings of full ambient calculus and π-calculus. Bisimulation properties and semantic underpinnings are being developed [25].

References

1. P. N. Benton. A mixed linear and non-linear logic: proofs, terms and models (extended abstract). In *Proc. CSL'94*, LNCS 933.
2. T. Bräuner. *An Axiomatic Approach to Adequacy.* Ph.D. Dissertation, University of Aarhus, 1996. BRICS Dissertation Series DS-96-4.

3. L. Cardelli and A. D. Gordon. Anytime, anywhere: modal logics for mobile ambients. In *Proc. POPL'00.*
4. G. L. Cattani and G. Winskel. Profunctors, open maps and bisimulation. Manuscript, 2000.
5. J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–102, 1987.
6. A. D. Gordon. Bisimilarity as a theory of functional programming. In *Proc. MFPS'95*, ENTCS 1.
7. M. C. B. Hennessy and G. D. Plotkin. Full abstraction for a simple parallel programming language. In *Proc. MFCS'79*, LNCS 74.
8. M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1):137–161, 1985.
9. M. Hennessy. A fully abstract denotational model for higher-order processes. *Information and Computation*, 112(1):55–95, 1994.
10. C. A. R. Hoare. *A Model for Communicating Sequential Processes.* Technical monograph, PRG-22, University of Oxford Computing Laboratory, 1981.
11. A. Joyal, M. Nielsen, and G. Winskel. Bisimulation from open maps. *Information and Computation*, 127:164–185, 1996.
12. K. G. Larsen and G. Winskel. Using information systems to solve recursive domain equations effectively. In *Proc. Semantics of Data Types, 1984*, LNCS 173.
13. P.-A. Melliès. Categorical models of linear logic revisited. Submitted to *Theoretical Computer Science*, 2002.
14. R. Milner, J. Parrow and D. Walker. A calculus of mobile processes, parts I and II. *Information and Computation*, 100(1):1–77, 1992.
15. J. H. Morris. *Lambda-Calculus Models of Programming Languages.* PhD thesis, MIT, 1968.
16. M. Nielsen, G. Plotkin and G. Winskel. Petri nets, event structures and domains, part I. *Theoretical Computer Science*, 13(1):85–108, 1981.
17. M. Nygaard. Towards an operational understanding of presheaf models. Progress report, University of Aarhus, 2001.
18. M. Nygaard and G. Winskel. Linearity in process languages. In *Proc. LICS'02.*
19. M. Nygaard and G. Winskel. HOPLA—a higher-order process language. In *Proc. CONCUR'02*, LNCS 2421.
20. M. Nygaard and G. Winskel. Domain theory for concurrency. Submitted to *Theoretical Computer Science*, 2003.
21. D. S. Scott. Domains for denotational semantics. In *Proc. ICALP'82*, LNCS 140.
22. R. A. G. Seely. Linear logic, $*$-autonomous categories and cofree coalgebras. In *Proc. Categories in Computer Science and Logic*, 1987.
23. G. Winskel. *The Formal Semantics of Programming Languages.* MIT Press, 1993.
24. G. Winskel. A presheaf semantics of value-passing processes. In *Proc. CONCUR'96*, LNCS 1119.
25. G. Winskel and F. Zappa Nardelli. Manuscript, 2003.