1

Linearity and nonlinearity in distributed computation

Glynn Winskel
Cambridge University Computer Laboratory

Abstract

The copying of processes is limited in the context of distributed computation, either as a fact of life, often because remote networks are simply too complicated to have control over, or deliberately, as in the design of security protocols. Roughly, linearity is about how to manage without a presumed ability to copy. The meaning and mathematical consequences of linearity are studied for path-based models of processes which are also models of affine-linear logic. This connection yields an affine-linear language for processes in which processes are typed according to the kind of computation paths they can perform. One consequence is that the affine-linear language automatically respects open-map bisimulation. A range of process operations (from CCS, CCS with process-passing, mobile ambients, and dataflow) can be expressed within the affine-linear language showing the ubiquity of linearity. Of course, process code can be sent explicitly to be copied. Following the discipline of linear logic, suitable nonlinear maps are obtained as linear maps whose domain is under an exponential. Different ways to make assemblies of processes lead to different choices of exponential; the nonlinear maps of only some of which will respect bisimulation.

1.1 Introduction

In the film "Groundhog Day" the main character comes to relive and remember the same day repeatedly, until finally he gets it right (and the girl). Real life isn't like that. The world moves on and we cannot rehearse, repeat or reverse the effects of the more important decisions we take.

As computation becomes increasingly distributed and interactive the more it resembles life in this respect, and the more difficult or impossible it is for a state of computation to be frozen and copied, so that interaction is most often conducted in a single irreversible run. Mathematically, this amounts to a form of linearity, as it is understood within models of linear logic.

Although it can be hard for processes to copy processes, it is generally easy for processes to ignore other processes. For this reason distributed computation often involves affine-linear maps.

To get an idea of the nature of affine-linear maps, imagine a network of interacting processes with a single hole, into which a process, the input process, may be plugged to form a complete process, the output process. The network with the hole represents an affine-linear map; given an input process, one obtains an output process.

The input process cannot be copied, but can be ignored. Consequently, any computation path (or run) of the output process will have depended on at most one computation path (or run) of the input. This property of affine-linear maps rests on an understanding of the nature of the computation paths of a process. The property can be simplified provided we understand a computation path of the input processes to also admit the empty computation path—a computation path obtained even when the input process is ignored. Any output got while ignoring the input process, results from the empty computation path as input. An affine-linear map has the following property:

A computation path of the process arising from the application of an affine-linear map to an input process has resulted from a single computation path, possibly empty, of the input process.

As this suggests, an affine-linear map is determined by its action on single, possibly empty, computation paths.

This article presents models of processes based on computation paths and so can make precise the sense in which many operations on distributed systems are associated with affine-linear maps, investigates the consequences of linearity and affine linearity for the important equivalence of bisimulation, and delineates the boundaries of linearity with respect to one, fairly broad, mathematical model, in which non-deterministic processes are represented as presheaves.

Of course, sometimes code can be sent and copied, which can give rise to maps which are not affine-linear. The presheaf model exposes how

different manners of copying lead to different kinds of nonlinear maps, some respecting bisimulation, others not.

1.2 Path-based models of processes

Consider processes, like those of CCS [24] and CSP [16], which can perform simple atomic actions, among which might be actions of synchronisations. An old idea is to represent the nondeterministic behaviour of a such a process as a "collection" of the computation paths it can perform. If we interpret this idea literally, and may assume that actions occur one at a time, we arrive at one of the early models of processes as *sets* of traces, where a trace is a finite sequence of actions that the process can perform [16]. It was realised very quickly that a problem with the trace model is that it is blind to deadlock; two processes may have the same trace sets and yet one may deadlock while the other does not (see for instance the discussion in [23], Ch.1). To detect possible deadlock, one way or another, one needs to keep track of nondeterministic branching in the representation of processes. An early proposal on how to do this is to represent a process as a tree, where branching stands for nondeterministic choice (*cf.* the synchronisation trees of [23]). The tree is still a sort of "collection" of the process's computation paths but one keeping track of where the paths overlap, through sharing a common subpath as history. Now the model is too concrete; many different trees represent what seems to be esentially the same behaviour, and this led to equivalences such as bisimulation on trees and transition systems [24].

The trace and tree models of processes are based on different ideas of what a "collection" of the computation paths means. A trace set of a process simply expresses whether or not a path of a certain shape is possible for the process. A tree expresses not only what paths are present but also how paths are subpaths, or restrictions, of others. This data, what paths are present and how they restrict to smaller paths, is precisely that caught in a presheaf over a category, a category in which the objects are path shapes and the maps express how one path shape can extend to another. In the category of all such presheaves we can view the tree as a *colimit* of its paths—another kind of "collection" of its paths.

To illustrate the idea, suppose that actions are drawn from some alphabet L, and consider processes whose computation paths have the shape of strings of actions, so members of L^*. A subpath will be associated with a substring. Regard L^* as a partial order where $s \leq t$ iff s is

a substring of t. (So L^* is also a category where we view there as being an arrow from s to t precisely when $s \leq t$.)

A presheaf over L^* is a functor from the opposite category $(L^*)^{op}$, where all the arrows are reversed, to the category of sets and functions **Set**. Spelt out, a presheaf X over L^* is a function which to each string s gives a set $X(s)$ and which to any pair of strings (s,t), with $s \leq t$, gives a function $X(s,t) : X(t) \to X(s)$ (note the reversal) in such a way that identities and composition are respected: $X(s,s) = 1_{X(s)}$ for each string s, and $X(s,u) = X(s,t) \circ X(t,u)$ whenever $s \leq t \leq u$.

When thinking of a presheaf X as representing a process, for a string s, the set $X(s)$ is the set of computation paths of shape s that the process can perform, and, when $s \leq t$, the function $X(s,t) : X(t) \to X(s)$ tells how paths of shape t restrict to subpaths of shape s. For example, a tree whose branches consist of strings of actions in L is easily viewed as a presheaf X over L^*. The set $X(s)$ consists of all the branches of shape s. The function $X(s,t) : X(t) \to X(s)$ restricts a branch of shape t to its sub-branch of shape s. The presheaf is rooted in the sense that the set $X(\epsilon)$ assigned to the empty string ϵ is a singleton—its only element is the root of the tree. Conversely, it is easy to see that a rooted presheaf over L^* determines a tree.

Suppose that we replace the category of sets used in the definition of presheaves by the simple subcategory $\mathbf{2}$, consisting of two distinct elements, the empty set, $\emptyset$, and the singleton set, $\mathbf{1}$, with the only non-identity map being $\emptyset \subseteq \mathbf{1}$. A functor X from $(L^*)^{op}$ to $\mathbf{2}$ is the same as a monotonic function from the reverse order $(L^*)^{op}$ to the order $\mathbf{2}$, so that if $s \leq t$ then $X(t) \leq X(s)$. When thinking of X as representing a process, $X(s) = \mathbf{1}$ means that the process can perform a path of shape s while $X(s) = \emptyset$ means that it cannot. If $X(t) = \mathbf{1}$ and $s \leq t$, then $X(s) = \mathbf{1}$. The functor X is a characteristic function for a trace set.

So trees and trace sets arise as variants of a common idea, that of representing a process as a (generalised) characteristic function, in the form of a functor from path shapes to measures of the extent to which the path shapes can be realised by the process.

In what follows, we want to broaden computation paths to have more general shapes than sequences of atomic actions, to allow actions to occur concurrently in a computation path, and for individual actions to have a more complicated structure. Later on, processes will be allocated types; the type of a process will specify the shapes of computation path it might perform.

1.3 Processes as trace sets

To allow a broad understanding of the shape of computation paths, we take a path order to be a partial order $\mathbb{P}$ in which the elements are path shapes and the order $p \leq q$ means that p can be extended to q. We obtain a form of nondeterministic domain by imitating the definition of presheaf category, but replacing the use of the category of sets, **Set**, by its much simpler subcategory, **2**. The category **2** is essentially a very simple partial order consisting of $\emptyset$ and **1** ordered by $\emptyset \subseteq \mathbf{1}$. A partial order $\mathbb{P}$ can always be regarded as a category by taking the homset $\mathbb{P}(p, q)$ to be **1** if $p \leq q$, and $\emptyset$ otherwise. Functors between partial orders seen as categories correspond precisely to monotonic maps.

The functor category $\widehat{\mathbb{P}} = [\mathbb{P}^{op}, \mathbf{2}]$ consists of objects the functors from $\mathbb{P}^{op}$ to **2** and maps the natural transformations between them. A functor from $\mathbb{P}^{op}$ to **2** is essentially a monotonic function from $\mathbb{P}^{op}$ to **2**. It is not hard to see that an object X of $\widehat{\mathbb{P}}$ corresponds to a downwards-closed set given by $\{p \in \mathbb{P} \mid X(p) = \mathbf{1}\}$, and that a natural transformation from X to Y in $\widehat{\mathbb{P}}$ corresponds to the inclusion of $\{p \in \mathbb{P} \mid X(p) = \mathbf{1}\}$ in $\{p \in \mathbb{P} \mid Y(p) = \mathbf{1}\}$. So we can identify $\widehat{\mathbb{P}}$ with the partial order of downwards-closed subsets of $\mathbb{P}$, ordered by inclusion; the order $\widehat{\mathbb{P}}$ has joins got simply via unions with the empty join being the least element $\emptyset$. The partial orders obtained in this way are precisely the infinitely-distributive algebraic lattices (see *e.g.*, [27, 28]) and these are just the same as prime algebraic lattices [25], and free join completions of partial orders.

We are thinking of $\widehat{\mathbb{P}}$ as a nondeterministic domain [15, 14]. An object in $\widehat{\mathbb{P}}$ is thought of as a denotation of a nondeterministic process which can realise path shapes in $\mathbb{P}$. An object in $\widehat{\mathbb{P}}$ is a trace set, like those originally, in [16], but for general path shapes, standing for the set of computation paths a process can perform. The join operation on $\widehat{\mathbb{P}}$ is a form of nondeterministic sum.

This suggests that we take maps between nondeterministic domains to be join preserving functions, the choice dealt with in the next section. We will however be forced a little beyond this first mathematically obvious choice.

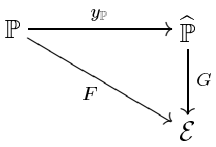
1.3.1 A linear category of domains

We mentioned that $\widehat{\mathbb{P}}$ is a free join completion of a partial order $\mathbb{P}$. We spell this out. There is a monotonic map $y_{\mathbb{P}} : \mathbb{P} \to \widehat{\mathbb{P}}$ which on p yields

$y_{\mathbb{P}}(p) = \mathbb{P}(-, p)$; for $p' \in \mathbb{P}$,

$$y_{\mathbb{P}}(p)(p') = \mathbf{1} \text{ if } p' \leq p, \text{ and } \emptyset \text{ otherwise.}$$

The map $y_{\mathbb{P}}$ satisfies the universal property that for any monotonic map $F : \mathbb{P} \to \mathcal{E}$, where $\mathcal{E}$ is a partial order with all joins, there is a unique join-preserving map $G : \widehat{\mathbb{P}} \to \mathcal{E}$ such that $F = G \circ y_{\mathbb{P}}$:



The proof of the universal property hinges on the fact that every object of $\widehat{\mathbb{P}}$ is the join of the "complete primes", objects $y_{\mathbb{P}}(p)$, below it. We will use an "inner product" notation and describe G above as taking $X \in \widehat{\mathbb{P}}$ to $F \cdot X$.

We can, in particular, instantiate $\mathcal{E}$ to a nondeterministic domain $\widehat{\mathbb{Q}}$, which certainly has all joins. By the universal property, monotonic maps $F : \mathbb{P} \to \widehat{\mathbb{Q}}$ are in 1-1 correspondence with join-preserving maps $G : \widehat{\mathbb{P}} \to \widehat{\mathbb{Q}}$. But monotonic maps $F : \mathbb{P} \to \widehat{\mathbb{Q}}$ are just the same as monotonic maps $F : \mathbb{P} \to [\mathbb{Q}^{op}, \mathbf{2}]$ and, uncurrying, these correspond to monotonic maps $H : \mathbb{P} \times \mathbb{Q}^{op} \to \mathbf{2}$ and so to objects of $\widehat{\mathbb{P}^{op} \times \mathbb{Q}} = [(\mathbb{P}^{op} \times \mathbb{Q})^{op}, \mathbf{2}]$.

So, on mathematical grounds it is natural to consider taking maps between nondeterministic domains as functions which preserve all joins. Such functions (often known as *additive* functions) compose as usual, have identities and give rise to a category rich in structure. Call this category $\mathbf{Lin_2}$; it consists of objects partial orders $\mathbb{P}$, $\mathbb{Q}$, $\cdots$, with maps $G : \mathbb{P} \to \mathbb{Q}$ the join-preserving functions from $\widehat{\mathbb{P}}$ to $\widehat{\mathbb{Q}}$.

As we have just seen, we can regard a map from $\mathbb{P}$ to $\mathbb{Q}$ in $\mathbf{Lin_2}$ in several ways and, in particular, as an object of $\widehat{\mathbb{P}^{op} \times \mathbb{Q}}$. Thus a map in $\mathbf{Lin_2}$ corresponds to a downward-closed subset of $\mathbb{P}^{op} \times \mathbb{Q}$ and so can be viewed as a relation between the partial orders $\mathbb{P}$ and $\mathbb{Q}$, with composition now given as the usual composition of relations.

This more symmetric, relational presentation exposes an involution central in understanding $\mathbf{Lin_2}$ 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 $\mathbb{P}^{op}$; clearly downward-closed subsets of $\mathbb{P}^{op} \times \mathbb{Q}$ correspond to downward-closed subsets of $(\mathbb{Q}^{op})^{op} \times \mathbb{P}^{op}$, showing how maps $F : \mathbb{P} \to \mathbb{Q}$ correspond to maps $F^{\perp} : \mathbb{Q}^{op} \to \mathbb{P}^{op}$ in $\mathbf{Lin_2}$. The tensor product of $\mathbb{P}$ and $\mathbb{Q}$ is given by the product of partial orders $\mathbb{P} \times \mathbb{Q}$ and the function

space from $\mathbb{P}$ to $\mathbb{Q}$ by $\mathbb{P}^{op} \times \mathbb{Q}$. On objects $\mathbb{P}$ and $\mathbb{Q}$, products and coproducts are both given by $\mathbb{P} + \mathbb{Q}$, the disjoint juxtaposition of $\mathbb{P}$ and $\mathbb{Q}$. One choice of interpretation of the exponential of linear logic is got by taking $!\mathbb{P}$, for a partial order $\mathbb{P}$, to be the partial order obtained as the restriction of $\widehat{\mathbb{P}}$ to its finite (or isolated) elements. (An element of a partial order is finite if whenever it is dominated by a directed join, it is dominated by an element of the join.) The partial order $\widehat{\mathbb{P}}$, with the inclusion $!\mathbb{P} \to \widehat{\mathbb{P}}$, is the free closure of $!\mathbb{P}$ under directed joins (the "ideal completion" of $!\mathbb{P}$.) Consequently, there is 1-1 correspondence between linear maps from $!\mathbb{P}$ to $\mathbb{Q}$ in $\mathbf{Lin_2}$ and Scott continuous (*i.e.*, directed-join preserving) functions from $\widehat{\mathbb{P}}$ to $\widehat{\mathbb{Q}}$. In fact, $!(-)$ extends straightforwardly to a comonad on $\mathbf{Lin_2}$ whose coKleisli category is isomorphic to the category of Scott continuous functions between nondeterministic domains.

Linear maps preserve joins. The join of the empty set is $\emptyset$, to be thought of as a *nil* process, which is unable to perform any computation path. So linear maps always send the *nil* process to the *nil* process. Going back to the intuitions in the introduction, if a network context gave rise to a linear map, then plugging a dead process into the network would always be catastrophic, and lead to the whole network going dead. We could extend to maps from $!\mathbb{P}$ to $\mathbb{Q}$, for objects $\mathbb{P}$ and $\mathbb{Q}$ in $\mathbf{Lin_2}$, but by the properties of the exponential, this would allow arbitrary copying of the argument process. All we often need is to allow maps to ignore their arguments and this can be got much more cheaply, by moving to a model of affine linear logic.

1.3.2 An affine-linear category of domains

A common operation in process algebras is that of prefixing a process by an action, so that a computation path of the prefixed process consists of first performing the action and then resuming by following a computation path of the original process. To understand prefix operations, we first need to lift path shapes by an initial action.

The operation of *lifting* on a partial order $\mathbb{P}$ produces a partial order $\mathbb{P}_\perp$, got by adjoining a new element $\perp$ below a copy of $\mathbb{P}$. Denote by $\lfloor p \rfloor$ the copy in $\mathbb{P}_\perp$ of the original element p in $\mathbb{P}$—each $\lfloor p \rfloor$ is assumed distinct from $\perp$. The order of $\mathbb{P}_\perp$ is given by $\perp \leq \lfloor p \rfloor$, for any $p \in \mathbb{P}$, with $\lfloor p \rfloor \leq \lfloor p' \rfloor$ iff $p \leq p'$ initially in $\mathbb{P}$.

Prefixing operations on processes make essential use of an operation

associated with lifting. The operation is the function

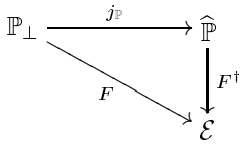
$$\lfloor - \rfloor : \widehat{\mathbb{P}} \to \widehat{\mathbb{P}_\perp}$$

such that $\lfloor X \rfloor(\perp) = \mathbf{1}$ and $\lfloor X \rfloor(\lfloor p \rfloor) = X(p)$ for $X \in \widehat{\mathbb{P}}$. The function $\lfloor - \rfloor$ is not a map from $\mathbb{P}$ to $\mathbb{P}_\perp$ in $\mathbf{Lin_2}$ as it clearly does not preserve joins of the empty set. It does however preserve all nonempty joins (*i.e.*, joins of nonempty sets). This provides a clue as to how to expand the maps of $\mathbf{Lin_2}$.

To accommodate the functions $\lfloor - \rfloor$ we move to a slightly broader category, though fortunately one that inherits a good many properties from $\mathbf{Lin_2}$. The category $\mathbf{Aff_2}$ has the same objects, partial orders, but its maps from $\mathbb{P}$ to $\mathbb{Q}$, written $F : \mathbb{P} \to \mathbb{Q}$, are functions $F : \widehat{\mathbb{P}} \to \widehat{\mathbb{Q}}$ which need only preserve *nonempty* joins.

A map $F : \mathbb{P} \to \mathbb{Q}$ in $\mathbf{Aff_2}$ is really a nonempty-join preserving function from $\widehat{\mathbb{P}}$ to $\widehat{\mathbb{Q}}$, so takes a (denotation of a) nondeterministic process with computation paths in $\mathbb{P}$ as input and yields a (denotation of a) nonderministic process with computation paths in $\mathbb{Q}$ as output. Because the map need only preserve nonempty joins it is at liberty to ignore the input process in giving non-trivial output. Because the map preserves all nonempty joins, if a computation path of the resulting output process requires computation of the input process, then it only requires a single computation path of the input process. The input process does not need to be copied to explore the range of computation paths it might follow.

A map in $\mathbf{Aff_2}$ is determined by its action on computation paths extended to include the empty path $\perp$. There is an embedding $j_{\mathbb{P}} : \mathbb{P}_\perp \to \widehat{\mathbb{P}}$ which takes $\perp$ to $\emptyset$ and any $\lfloor p \rfloor$ to $y_{\mathbb{P}}(p)$. The map $j_{\mathbb{P}}$ satisfies the universal property that for any monotonic map $F : \mathbb{P}_\perp \to \mathcal{E}$, where $\mathcal{E}$ is a partial order with all nonempty joins, there is a unique nonempty-join preserving map $F^\dagger : \widehat{\mathbb{P}} \to \mathcal{E}$ such that $F = F^\dagger \circ j_{\mathbb{P}}$:



The proof of the universal property rests on the fact that every $X \in \widehat{\mathbb{P}}$ is the nonempty join of the set consisting of all $j_{\mathbb{P}}(p')$, where $p' \in \mathbb{P}_\perp$, that X dominates—this set is nonempty because it contains $j_{\mathbb{P}}(\perp)$ $(= \emptyset)$.

By instantiating $\mathcal{E}$ to $\widehat{\mathbb{Q}}$ where $\mathbb{Q}$ is a partial order, we see that maps

$\mathbb{P} \to \mathbb{Q}$ in $\mathbf{Aff_2}$ are in 1-1 correspondence with maps $\mathbb{P}_\perp \to \mathbb{Q}$ in $\mathbf{Lin_2}$, and so to elements in $\widehat{(\mathbb{P}_\perp)^{op}} \times \mathbb{Q}$.

There is a unique (and obvious) way to extend the lifting operation $(-)_\perp$ to a functor from $\mathbf{Aff_2}$ to $\mathbf{Lin_2}$ so that the correspondence

$$\mathbf{Aff_2}(\mathbb{P}, \mathbb{Q}) \cong \mathbf{Lin_2}(\mathbb{P}_\perp, \mathbb{Q})$$

is natural in $\mathbb{P} \in \mathbf{Aff_2}$ and $\mathbb{Q} \in \mathbf{Lin_2}$. This exhibits the functor $(-)_\perp$ as a left adjoint to the inclusion functor $\mathbf{Lin_2} \hookrightarrow \mathbf{Aff_2}$. Composing the two adjoints, we obtain a comonad $(-)_\perp$ on $\mathbf{Lin_2}$ whose coKliesli category is isomorphic to $\mathbf{Aff_2}$. Clearly, a map in $\mathbf{Aff_2}$ also belongs to the subcategory $\mathbf{Lin_2}$ iff it is strict in the sense of preserving $\emptyset$.

With the help of the comonad $(-)_\perp$ we have turned the model of linear logic $\mathbf{Lin_2}$ into a model $\mathbf{Aff_2}$ of *affine* linear logic (a model of intuitionistic linear logic in which the structural rule of weakening is satisfied through the unit of the tensor also being a terminal object—[17]). Its operations are defined in terms of the corresponding operations in $\mathbf{Lin_2}$. For example, its tensor $\otimes$ is defined so that

$$(\mathbb{P} \otimes \mathbb{Q})_\perp = \mathbb{P}_\perp \times \mathbb{Q}_\perp \ ,$$

a product, and so tensor in $\mathbf{Lin_2}$, of partial orders $\mathbb{P}_\perp$ and $\mathbb{Q}_\perp$. Its function space is given by $(\mathbb{P}_\perp)^{op} \times \mathbb{Q}$. The category $\mathbf{Aff_2}$ has the same products as $\mathbf{Lin_2}$. It does not have coproducts, though we will later see a form of prefixed sum which is useful in giving semantics to process languages.

1.4 Processes as presheaves

In order to take account of the branching structure of nondeterministic processes we move from a representation of a process as characteristic function from computation-path shapes to $\emptyset$ or $\mathbf{1}$ in the partial order $\mathbf{2}$, and explore the variation where we measure the presence of a computation-path shape in a process by a set in the category $\mathbf{Set}$ of sets.

It is useful for later to broaden our understanding of shapes of computation paths to be objects in a small category $\mathbb{P}$. In our applications, the category $\mathbb{P}$ is thought of as a path category, consisting of shapes of paths, where a map $e : p \to p'$ expresses how the path p extends to the path p'. Let $\mathbb{P}$ be a small category. The category of *presheaves over* $\mathbb{P}$, written $\widehat{\mathbb{P}}$, is the category $[\mathbb{P}^{op}, \mathbf{Set}]$ with objects the functors from $\mathbb{P}^{op}$

(the opposite category) to the category of sets, and maps the natural transformations between them.

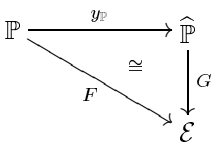
A presheaf $X : \mathbb{P}^{op} \to \mathbf{Set}$ specifies for a typical path p the set $X(p)$ of computation paths of shape p. The presheaf X acts on a map $e : p \to p'$ in $\mathbb{P}$ to give a function $X(e)$ saying how p'-paths in X restrict to p-paths in X—several paths may restrict to the same path. In this way a presheaf can model the nondeterministic branching of a process.

A presheaf category has all limits and colimits given pointwise, at a particular object, by the corresponding limits or colimits of sets. In particular, a presheaf category has all sums (coproducts) of presheaves; the sum $\Sigma_{i\in I} X_i$ of presheaves X_i, $i \in I$, over $\mathbb{P}$ has a contribution $\Sigma_{i\in I} X_i(p)$, the disjoint union of sets, at p an object of $\mathbb{P}$. The empty sum of presheaves is the presheaf $\emptyset$ with empty contribution at each p in $\mathbb{P}$. In process terms, a sum of presheaves represents a nondeterministic sum of processes.

1.4.1 A linear category of presheaf models

A category of presheaves, $\widehat{\mathbb{P}}$, is accompanied by the *Yoneda embedding*, a functor $y_{\mathbb{P}} : \mathbb{P} \to \widehat{\mathbb{P}}$, which fully and faithfully embeds $\mathbb{P}$ in the category of presheaves. For every object p of $\mathbb{P}$, the Yoneda embedding yields $y_{\mathbb{P}}(p) = \mathbb{P}(-, p)$. Presheaves isomorphic to images of objects of $\mathbb{P}$ under the Yoneda embedding are called *representables*.

Via the Yoneda embedding we can regard $\mathbb{P}$ essentially as a full subcategory of $\widehat{\mathbb{P}}$. Moreover $\widehat{\mathbb{P}}$ is characterized (up to equivalence of categories) as the free colimit completion of $\mathbb{P}$. In other words, the Yoneda embedding $y_{\mathbb{P}}$ satisfies the universal property that for any functor $F : \mathbb{P} \to \mathcal{E}$, where $\mathcal{E}$ is a category with all colimits, there is a colimit preserving functor $G : \widehat{\mathbb{P}} \to \mathcal{E}$, determined to within isomorphism, such that $F \cong G \circ y_{\mathbb{P}}$:



The proof rests on the fact that any presheaf is a colimit of representables—see *e.g.*, [22] P.43. We will describe G above as $F \cdot -$.

In particular, we can take $\mathcal{E}$ to be a presheaf category $\widehat{\mathbb{Q}}$. As the universal property suggests, colimit-preserving functors between presheaf categories are useful. Define the category $\mathbf{Lin}_{\mathbf{Set}}$ to consist of small cat-

egories $\mathbb{P}$, $\mathbb{Q}$, $\cdots$, with maps $G : \mathbb{P} \to \mathbb{Q}$ the colimit-preserving functors from $\widehat{\mathbb{P}}$ to $\widehat{\mathbb{Q}}$.

By the universal property, colimit-preserving functors $G : \widehat{\mathbb{P}} \to \widehat{\mathbb{Q}}$ correspond to within isomorphism to functors $F : \mathbb{P} \to \widehat{\mathbb{Q}}$, and such functors are in 1-1 correspondence with profunctors $\bar{F} : \mathbb{P} \to \mathbb{Q}$. Recall that the category of profunctors from $\mathbb{P}$ to $\mathbb{Q}$, written $\mathbf{Prof}(\mathbb{P}, \mathbb{Q})$, is the functor category $[\mathbb{P} \times \mathbb{Q}^{op}, \mathbf{Set}]$, which clearly equals the category of presheaves $\widehat{\mathbb{P}^{op} \times \mathbb{Q}}$, and is isomorphic to the functor category $[\mathbb{P}, \widehat{\mathbb{Q}}]$. We thus have the chain of equivalences:

$$\mathbf{Lin}_{\mathbf{Set}}(\mathbb{P}, \mathbb{Q}) \simeq [\mathbb{P}, \widehat{\mathbb{Q}}] \cong \mathbf{Prof}(\mathbb{P}, \mathbb{Q}) = \widehat{\mathbb{P}^{op} \times \mathbb{Q}} \ .$$

Exactly analogously with the domain model $\mathbf{Lin_2}$ in Section 1.3.1, one can build a model of linear logic out of $\mathbf{Lin}_{\mathbf{Set}}$, though there are now subtleties, as what were previously functors must now be pseudo functors (preserving composition only up to coherent isomorphism). In particular, the involution of linear logic takes a map $F : \mathbb{P} \to \mathbb{Q}$ to a map $F^{\perp} : \mathbb{Q}^{op} \to \mathbb{P}^{op}$ in $\mathbf{Lin}_{\mathbf{Set}}$ via

$$\mathbf{Lin}_{\mathbf{Set}}(\mathbb{P}, \mathbb{Q}) \simeq \widehat{\mathbb{P}^{op} \times \mathbb{Q}} \cong \widehat{(\mathbb{Q}^{op})^{op} \times \mathbb{P}^{op}} \simeq \mathbf{Lin}_{\mathbf{Set}}(\mathbb{Q}^{op}, \mathbb{P}^{op}) \ .$$

On objects, the tensor product of $\mathbb{P}$ and $\mathbb{Q}$ is given by the product of categories $\mathbb{P} \times \mathbb{Q}$ and the function space from $\mathbb{P}$ to $\mathbb{Q}$ by $\mathbb{P}^{op} \times \mathbb{Q}$. On objects $\mathbb{P}$ and $\mathbb{Q}$, products and coproducts are both given by $\mathbb{P} + \mathbb{Q}$, the sum of categories $\mathbb{P}$ and $\mathbb{Q}$. As for the exponential ! of linear logic, there are many possible choices—see Section 1.8.2.

Just as in the domain case, maps in $\mathbf{Lin}_{\mathbf{Set}}$ are most often too restrictive. Maps in $\mathbf{Lin}_{\mathbf{Set}}$ preserve colimits and, in particular, sums and the empty colimit $\emptyset$, a property which is only the case for rather special operations on processes.

1.4.2 An affine-linear category of presheaf models

Many operations associated with process languages do not preserve sums, so arbitrary colimits. Prefixing operations only preserve connected colimits (colimits of nonempty connected diagrams). Prefixing operations derive from the functor $\lfloor - \rfloor : \widehat{\mathbb{P}} \to \widehat{\mathbb{P}_{\perp}}$. The lifted category $\mathbb{P}_{\perp}$ comprises a copy of $\mathbb{P}$—the copy in $\mathbb{P}_{\perp}$ of the original object p is written $\lfloor p \rfloor$—to which a new initial object $\perp$ has been adjoined. (This definition extends to categories the earlier definition of lifting on partial orders, Section 1.3.2.) The functor $\lfloor - \rfloor : \widehat{\mathbb{P}} \to \widehat{\mathbb{P}_{\perp}}$ adjoins a "root" to a presheaf X in $\widehat{\mathbb{P}}$ in the sense that $\lfloor X \rfloor(\lfloor p \rfloor)$ is $X(p)$, for any p in $\mathbb{P}$, while $\lfloor X \rfloor(\perp)$

is the singleton set $\{*\}$, the new root being $*$; the restriction maps are extended so that restriction to $\bot$ sends elements to $*$. A map from X to Y in $\widehat{\mathbb{P}}$ is sent to its obvious extension from $\lfloor X \rfloor$ to $\lfloor Y \rfloor$ in $\widehat{\mathbb{P}_\bot}$. Presheaves that to within isomorphism can be obtained as images under $\lfloor - \rfloor$ are called *rooted* [19].

Proposition 1.1 *Any presheaf Y in $\widehat{\mathbb{P}_\bot}$ has a decomposition as a sum of rooted presheaves*

$$Y \cong \Sigma_{i \in Y(\bot)} \lfloor Y_i \rfloor \ ,$$

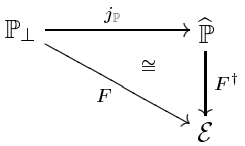
where, for $i \in Y(\bot)$, the presheaf Y_i in $\widehat{\mathbb{P}}$ is the restriction of Y to the elements which are sent under Y to the element i over $\bot$, i.e., for $p \in \mathbb{P}$,

$$Y_i(p) = \{x \in Y(\lfloor p \rfloor) \mid (Ye_p)(x) = i\}$$

where $e_p : \bot \to \lfloor p \rfloor$ is the unique map from $\bot$ to p in $\mathbb{P}$.

Intuitively, thinking of presheaves as processes, the presheaves Y_i, where $i \in Y(\bot)$, in the decomposition of Y a presheaf over $\mathbb{P}_\bot$, are those processes that Y can become after performing the initial action $\bot$.

The *strict Yoneda embedding* $j_{\mathbb{P}_\bot} : \mathbb{P}_\bot \to \widehat{\mathbb{P}}$, sends $\bot$ to $\emptyset$ and elsewhere acts like $y_{\mathbb{P}}$. The presheaf category $\widehat{\mathbb{P}}$, with the strict Yoneda embedding $j_{\mathbb{P}}$ is a free connected-colimit completion of $\mathbb{P}_\bot$. Together they satisfy the universal property that for any functor $F : \mathbb{P} \to \mathcal{E}$, where $\mathcal{E}$ is a category with all connected colimits, there is a connected-colimit preserving functor $F^\dagger : \widehat{\mathbb{P}} \to \mathcal{E}$, determined to within isomorphism, such that $F \cong F^\dagger \circ j_{\mathbb{P}}$:



The central observation on which the proof relies is that any presheaf is a connected colimit of representables, $j_{\mathbb{P}}(\lfloor p \rfloor)$ with p in $\mathbb{P}$, together with $j_{\mathbb{P}}(\bot) = \emptyset$, the empty presheaf.

The universal property suggests the importance of connected-colimit preserving functors. Define $\mathbf{Aff}_{\mathbf{Set}}$ to be the category consisting of objects small categories $\mathbb{P}, \mathbb{Q}, \cdots$, with maps $G : \mathbb{P} \to \mathbb{Q}$ the connected-colimit preserving functors $G : \widehat{\mathbb{P}} \to \widehat{\mathbb{Q}}$ between the associated presheaf categories, and composition the usual composition of functors. A functor which preserves colimits certainly preserves connected colimits, so

$\mathbf{Lin}_{\mathbf{Set}}$ is a subcategory of $\mathbf{Aff}_{\mathbf{Set}}$. The two categories, $\mathbf{Lin}_{\mathbf{Set}}$ and $\mathbf{Aff}_{\mathbf{Set}}$, share the same objects. We can easily characterise those maps in $\mathbf{Aff}_{\mathbf{Set}}$ which are in $\mathbf{Lin}_{\mathbf{Set}}$:

Proposition 1.2 *Suppose $F : \widehat{\mathbb{P}} \to \widehat{\mathbb{Q}}$ is a functor which preserves connected colimits. The following properties are equivalent:*

(i) F preserves all colimits,
(ii) F preserves all coproducts (sums),
(iii) F is strict, i.e., F sends the empty presheaf to the empty presheaf.

Because $\widehat{\mathbb{P}}$ is the free connected-colimit completion of $\mathbb{P}_\perp$, we obtain the equivalence

$$\mathbf{Aff}_{\mathbf{Set}}(\mathbb{P}, \mathbb{Q}) \simeq [\mathbb{P}_\perp, \widehat{\mathbb{Q}}] \ ,$$

and consequently the equivalence

$$\mathbf{Aff}_{\mathbf{Set}}(\mathbb{P}, \mathbb{Q}) \simeq \mathbf{Lin}_{\mathbf{Set}}(\mathbb{P}_\perp, \mathbb{Q}) \ .$$

The equivalence is part of an adjunction between $\mathbf{Aff}_{\mathbf{Set}}$ and $\mathbf{Lin}_{\mathbf{Set}}$ regarded as 2-categories, in which the 2-cells are natural transformations. We can easily extend lifting to a 2-functor $(-)_\perp : \mathbf{Aff}_{\mathbf{Set}} \to \mathbf{Lin}_{\mathbf{Set}}$; for $F : \mathbb{P} \to \mathbb{Q}$ in $\mathbf{Aff}_{\mathbf{Set}}$, the functor $F_\perp : \mathbb{P}_\perp \to \mathbb{Q}_\perp$ in $\mathbf{Lin}_{\mathbf{Set}}$ takes $Y \in \widehat{\mathbb{P}_\perp}$ with decomposition $\Sigma_{i \in Y(\perp)} \lfloor Y_i \rfloor$ to $F_\perp(Y) = \Sigma_{i \in Y(\perp)} \lfloor F(Y_i) \rfloor$. Lifting restricts to a 2-comonad on $\mathbf{Lin}_{\mathbf{Set}}$ with $\mathbf{Aff}_{\mathbf{Set}}$ as its coKleisli category.

The comonad $(-)_\perp$ has turned the model of linear logic $\mathbf{Lin}_{\mathbf{Set}}$ into a model $\mathbf{Aff}_{\mathbf{Set}}$ of affine linear logic (where the tensor unit is terminal).

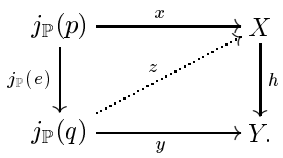
1.4.3 Bisimulation

Presheaves are being thought of as nondeterministic processes on which equivalences such as bisimulation are important in abstracting away from inessential differences of behaviour. Bisimulation between presheaves is derived from notion of open map between presheaves [18, 19].

A morphism $h : X \to Y$, between presheaves X and Y, is *open* iff for all morphisms $e : p \to q$ in $\mathbb{P}_\perp$, any commuting square

$$\begin{array}{ccc} j_{\mathbb{P}}(p) & \xrightarrow{x} & X \\ {\scriptstyle j_{\mathbb{P}}(e)}\downarrow & & \downarrow{\scriptstyle h} \\ j_{\mathbb{P}}(q) & \xrightarrow[y]{} & Y \end{array}$$

can be split into two commuting triangles



That the square commutes means that the path $h \circ x$ in Y can be extended via e to a path y in Y. That the two triangles commute means that the path x can be extended via e to a path z in X which matches y.

Open maps are a generalisation of functional bisimulations, or zig-zag morphisms, known from transition systems [19]. Presheaves in $\widehat{\mathbb{P}}$ are *bisimilar* iff there is a span of open maps between them.†

The preservation of connected colimits by a functor between presheaf categories is sufficient to ensure that it preserves open maps and bisimulation.

Theorem 1.3 *[11, 7] Let $G : \widehat{\mathbb{P}} \to \widehat{\mathbb{Q}}$ be any connected-colimit preserving functor between presheaf categories. Then G preserves open maps and open-map bisimulation.*

1.5 Constructions

We now describe the constructions which form the basis of a denotational semantics for a language for affine-linear processes. The types of the language will be interpreted as objects (in fact, path orders) and the terms, describing processes with free variables, as maps of an affine category. The constructions can be read as being in both the category of domains $\mathbf{Aff_2}$ and the category of presheaf models $\mathbf{Aff_{Set}}$, sometimes referring to their linear subcategories. By using the neutral language of categories we can describe the operations in either set-up. Below **Aff** can refer to either $\mathbf{Aff_2}$ or $\mathbf{Aff_{Set}}$, and correspondingly **Lin** to either $\mathbf{Lin_2}$ or $\mathbf{Lin_{Set}}$. $\mathbf{Aff_{Set}}$ and $\mathbf{Lin_{Set}}$ are 2-categories in which maps are related by natural transformations. There, and so in the general discussion with **Aff**, we can only characterise constructions to within isomorphism of maps. Of course, isomorphism of maps coincides with

† We have chosen here to develop the definition of open map from the strict Yoneda embedding rather than the Yoneda embedding. Maps between presheaves are open with respect to strict Yoneda iff they are surjective and open with respect to Yoneda.

equality in the categorises of domains. Although the operations will always be with respect to path orders they could all be extended easily to small categories.

1.5.1 Sums and fixed points

Each object $\mathbb{P}$ is associated with (nondeterministic) sum operations, a map $\Sigma : \&_{i\in I}\,\mathbb{P} \to \mathbb{P}$ in **Aff** taking a tuple $\{X_i \mid i \in I\}$, to the sum (coproduct) $\Sigma_{i\in I} X_i$ in $\widehat{\mathbb{P}}$. The empty sum yields $\emptyset \in \mathbb{P}$. Finite sums, of size k, are typically written as $X_1 + \cdots + X_k$.

For objects $\mathbb{P}$ and $\mathbb{Q}$, the category $\mathbf{Aff}(\mathbb{P},\mathbb{Q})$ of maps with natural transformations, being equivalent to $\widehat{(\mathbb{P}_\perp)^{op}} \times \mathbb{Q}$, has all colimits and in particular all ω-colimits. Any operation $G : \mathbf{Aff}(\mathbb{P},\mathbb{Q}) \to \mathbf{Aff}(\mathbb{P},\mathbb{Q})$ which preserves connected colimits will have a fixed point *fix* $G : \mathbb{P} \to \mathbb{Q}$, a map in **Aff**. We will build up the denotation of fixed points out of composition in **Aff**. The composition $G \circ F$ of maps F in $\mathbf{Aff}(\mathbb{P},\mathbb{Q})$ and G in $\mathbf{Aff}(\mathbb{Q},\mathbb{R})$, being got as the application $G(F(-))$, preserves connected colimits in the argument F, and colimits in G.†

1.5.2 Tensor

The tensor product $\mathbb{P}\otimes\mathbb{Q}$ of path orders $\mathbb{P}$, $\mathbb{Q}$ is given by the set $(\mathbb{P}_\perp \times \mathbb{Q}_\perp) \setminus \{(\perp,\perp)\}$, ordered coordinatewise, in other words, as the product of $\mathbb{P}_\perp$ and $\mathbb{Q}_\perp$ as partial orders but with the bottom element $(\perp,\perp)$ removed.

Let $F : \mathbb{P} \to \mathbb{P}'$ and $G : \mathbb{Q} \to \mathbb{Q}'$. We define $F \otimes G : \mathbb{P}\otimes\mathbb{Q} \to \mathbb{P}'\otimes\mathbb{Q}'$ as the extension (*cf.* Sections 1.3.2 and 1.4.2) $H^\dagger$ of a functor

$$H : (\mathbb{P}\otimes\mathbb{Q})_\perp \to \widehat{\mathbb{P}'\otimes\mathbb{Q}'} \ .$$

Notice that $(\mathbb{P}\otimes\mathbb{Q})_\perp$ is isomorphic to the product as partial orders of $\mathbb{P}_\perp \times \mathbb{Q}_\perp$ in which the bottom element is then $(\perp,\perp)$. With this realisation of $(\mathbb{P}\otimes\mathbb{Q})_\perp$ we can define $H : \mathbb{P}_\perp \times \mathbb{Q}_\perp \to \widehat{\mathbb{P}'\otimes\mathbb{Q}'}$ by taking

$$(H(p,q))(p',q') = \lfloor F(p)\rfloor(p') \times \lfloor G(q)\rfloor(q')$$

for $p \in \mathbb{P}_\perp$, $q \in \mathbb{Q}_\perp$ and $(p',q') \in \mathbb{P}'\otimes\mathbb{Q}'$—on the right we use the product (in **Set**, or **2** where it amounts to the meet).

† The story specialises to the category of domains $\mathbf{Aff_2}$. In a domain, colimits reduce to joins and connected colimits to nonempty joins. In particular, ω-colimits amount to joins of ω-chains. An operation between domains preserves (connected) colimits iff it preserves (nonempty) joins.

The unit for tensor is the empty path order $\mathbb{O}$.

Objects $X \in \widehat{\mathbb{P}}$ correspond to maps $\tilde{X} : \mathbb{O} \to \mathbb{P}$ sending $\emptyset$ to X. Given $X \in \widehat{\mathbb{P}}$ and $Y \in \widehat{\mathbb{Q}}$ we define $X \otimes Y \in \widehat{\mathbb{P} \otimes \mathbb{Q}}$ to be the element pointed to by $\tilde{X} \otimes \tilde{Y} : \mathbb{O} \to \mathbb{P} \otimes \mathbb{Q}$.

1.5.3 Function space

The function space of path orders $\mathbb{P} \multimap \mathbb{Q}$ is given by the product of partial orders $(\mathbb{P}_\perp)^{op} \times \mathbb{Q}$. Thus the elements of $\mathbb{P} \multimap \mathbb{Q}$ are pairs, which we write suggestively as $(p \mapsto q)$, with $p \in \mathbb{P}_\perp, q \in \mathbb{Q}$, ordered by

$$(p' \mapsto q') \leq (p \mapsto q) \iff p \leq p' \ \& \ q' \leq q$$

—note the switch in order on the left.

We have the following chain of isomorphisms between partial orders:

$$\mathbb{P} \otimes \mathbb{Q} \multimap \mathbb{R} = (\mathbb{P} \otimes \mathbb{Q})_\perp{}^{op} \times \mathbb{R} \ \cong \ \mathbb{P}_\perp{}^{op} \times \mathbb{Q}_\perp{}^{op} \times \mathbb{R} \ \cong \ \mathbb{P} \multimap (\mathbb{Q} \multimap \mathbb{R}) \ .$$

This gives an isomorphism between $\mathbb{P} \otimes \widehat{\mathbb{Q} \multimap \mathbb{R}}$ and $\mathbb{P} \multimap \widehat{(\mathbb{Q} \multimap \mathbb{R})}$. Thus there is a 1-1 correspondence *curry* from maps $\mathbb{P} \otimes \mathbb{Q} \to \mathbb{R}$ to maps $\mathbb{P} \to (\mathbb{Q} \multimap \mathbb{R})$ in **Aff**; its inverse is called *uncurry*. We obtain *linear application*, $app : (\mathbb{P} \multimap \mathbb{Q}) \otimes \mathbb{P} \to \mathbb{Q}$, as $uncurry(1_{\mathbb{P} \multimap \mathbb{Q}})$.

We shall write $u\ t$ for the application of u of type $\mathbb{P} \multimap \mathbb{Q}$ to t of type $\mathbb{P}$. The ability to curry justifies the formation of terms $\lambda x.u$ of type $\mathbb{P} \multimap \mathbb{Q}$ by lambda abstraction where u of type $\mathbb{Q}$ is a term with free variable x of type $\mathbb{P}$. Because of linearity constraints on the occurrence of variables, we will have that an application $(\lambda x.u)t$ will be isomorphic to a substitution $u[t/x]$—see Lemma 1.8.

1.5.4 Products

The product of path orders $\mathbb{P}\&\mathbb{Q}$ is given by the disjoint union of $\mathbb{P}$ and $\mathbb{Q}$. An object 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 $\pi_2 : \mathbb{P}\&\mathbb{Q} \to \mathbb{Q}$. More general, not just binary, products $\&_{i \in I} \mathbb{P}_i$ 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 $\langle F_i \rangle_{i \in I} : \mathbb{P} \to \&_{i \in I} \mathbb{P}_i$ 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 $\mathbb{O}$ and as the terminal object is associated with unique maps $\mathbb{P} \to \mathbb{O}$, constantly $\emptyset$, for any path order $\mathbb{P}$. Finite products, of size k, are most often written as $\mathbb{P}_1 \& \cdots \& \mathbb{P}_k$.

Because there are empty objects we can define maps in **Lin** from products to tensors of path orders. For instance, in the binary case, $\sigma : \mathbb{P}\&\mathbb{Q} \to \mathbb{P} \otimes \mathbb{Q}$ in **Lin** is specified by

$$(X, Y) \mapsto (X \otimes \emptyset) + (\emptyset \otimes Y) \ .$$

The composition of such a map with the diagonal map of the product, *viz.*

$$\delta_{\mathbb{P}} : \mathbb{P} \xrightarrow{diag} \mathbb{P}\&\mathbb{P} \xrightarrow{\sigma} \mathbb{P} \otimes \mathbb{P}$$

gives a weak form of diagonal map taking X to $(X \otimes \emptyset) + (\emptyset \otimes X)$. General weak diagonal maps

$$\delta_{\mathbb{P}k} : \mathbb{P} \to \mathbb{P} \otimes \cdots \otimes \mathbb{P}$$

in **Lin**, from $\mathbb{P}$ to k copies of $\mathbb{P}$ tensored together, are defined analogously. They will play a role later in the semantics of a general affine linear language; weak diagonal maps allow the same argument to be used in several different, though incompatible, ways.

1.5.5 Prefixed sums

The category **Aff** does not have coproducts. However, we can build a useful sum in **Aff** with the help of the coproduct of **Lin** and lifting. Let $\mathbb{P}_\alpha$, for $\alpha \in A$, be a family of path orders. As their prefixed sum, $\Sigma_{\alpha \in A} \alpha \mathbb{P}_\alpha$, we take the disjoint union of the path orders $\Sigma_{\alpha \in A} \mathbb{P}_{\alpha \perp}$, over the underlying set $\bigcup_{\alpha \in A} \{\alpha\} \times (\mathbb{P}_\alpha)_\perp$; the latter path order forms a coproduct in **Lin** with the obvious injections $in_\beta : \mathbb{P}_{\beta \perp} \to \Sigma_{\alpha \in A} \alpha \mathbb{P}_\alpha$, for $\beta \in A$. The *injections* $\beta : \mathbb{P}_\beta \to \Sigma_{\alpha \in A} \alpha \mathbb{P}_\alpha$ in **Aff**, for $\beta \in A$, are defined to be the composition $\beta = in_\beta(\lfloor - \rfloor)$. This construction is not a coproduct in **Aff**. However, it does satisfy a weaker property analogous to the universal property of a coproduct. Suppose $F_\alpha : \mathbb{P}_\alpha \to \mathbb{Q}$ are maps in **Aff** for all $\alpha \in A$. Then, there is a mediating map

$$F : \Sigma_{\alpha \in A} \alpha \mathbb{P}_\alpha \to \mathbb{Q}$$

in **Lin** determined to within isomorphism such that

$$F \circ \alpha \cong F_\alpha$$

for all $\alpha \in A$.

Suppose that the family of maps $F_\alpha : \mathbb{P}_\alpha \to \mathbb{Q}$, with $\alpha \in A$, has the property that each F_α is constantly $\emptyset$ whenever $\alpha \in A$ is different from

β and that F_β is $H : \mathbb{P}_\beta \to \mathbb{Q}$. Write $H_{@\beta} : \Sigma_{\alpha\in A}\alpha\mathbb{P}_\alpha \to \mathbb{Q}$ for a choice of mediating map in **Lin**. Then

$$H_{@\beta}(\beta Y) \cong H(Y)\ ,\ H_{@\beta}(\alpha Z) = \emptyset \text{ if } \alpha \neq \beta\ , H_{@\beta}(\Sigma_{i\in I} X_i) \cong \Sigma_{i\in I} H_{@\beta}(X_i)\ ,$$

where $Y \in \widehat{\mathbb{P}}_\beta$, $Z \in \widehat{\mathbb{P}}_\alpha$ and $X_i \in \widehat{\Sigma_{\alpha\in A}\alpha\mathbb{P}_\alpha}$ for all $i \in I$. In particular, for empty sums, $H_{@\beta}(\emptyset) = \emptyset$.

For a general family $F_\alpha : \mathbb{P}_\alpha \to \mathbb{Q}$, with $\alpha \in A$, we can describe the action of the mediating morphism, to within isomorphism, on $X \in \widehat{\Sigma_{\alpha\in A}\alpha\mathbb{P}_\alpha}$ as $F(X) = \Sigma_{\alpha\in A}(F_\alpha)_{@\alpha}(X)$.

If a term u of type $\mathbb{Q}$ with free variable x of type $\mathbb{P}$ denotes $H : \mathbb{P}_\beta \to \mathbb{Q}$ in **Aff** and t is of type $\Sigma_{\alpha\in A}\alpha\mathbb{P}_\alpha$, then we shall write

$$[t > \alpha x \Rightarrow u]$$

for $H_{@\beta}(t)$. This term $[t > \alpha x \Rightarrow u]$ tests or matches t denoting an element of a prefixed sum against the pattern αx and passes the results of successful matches for x on to u; the possibly multiple results of successful matches are then summed together.

Because prefixed sum is not a coproduct we do not have that tensor distributes over prefixed sum. However there is a map in **Aff**,

$$\mathit{dist} : \mathbb{Q} \otimes \Sigma_{\alpha\in A}\alpha\mathbb{P}_\alpha \to \Sigma_{\alpha\in A}\alpha(\mathbb{Q} \otimes \mathbb{P}_\alpha)\ ,$$

expressing a form of distributivity. The map *dist* is given as the extension $H^\dagger$ of the functor

$$H : \mathbb{Q}_\perp \times (\Sigma_{\alpha\in A}\alpha\mathbb{P}_\alpha)_\perp \to \Sigma_{\alpha\in A}\alpha(\mathbb{Q} \otimes \mathbb{P}_\alpha)$$

where

$$H(q, (\alpha, p)) = y_{\Sigma_{\alpha\in A}\alpha(\mathbb{Q}\otimes\mathbb{P}_\alpha)}(\alpha, (q, p)) \quad \text{and} \quad H(q, \perp) = \emptyset\ .$$

Unary prefixed sums in **Aff**, when the indexing set is a singleton, are an important special case as they amount to lifting.

1.5.6 Recursive type definitions

Suppose that we wish to model a process language rather like CCS but where processes are passed instead of discrete values, subject to the linearity constraint that when a process is received it can be run at most once. Assume the synchronised communication occurs along channels forming the set A. The path orders can be expected to be the "least" to satisfy the following equations:

$$\mathbb{P} = \tau\mathbb{P} + \Sigma_{a\in A} a!\mathbb{C} + \Sigma_{a\in A} a?\mathbb{F}\ , \qquad \mathbb{C} = \mathbb{P} \otimes \mathbb{P}\ , \qquad \mathbb{F} = (\mathbb{P} \multimap \mathbb{P})\ .$$

The three components of process paths $\mathbb{P}$ represent paths beginning with a silent (τ) action, an output on a channel ($a!$), resuming as a concretion path (in $\mathbb{C}$), and an input from a channel ($a?$), resuming as an abstraction path (in $\mathbb{F}$). It is our choice of path for abstractions which narrows us to an *affine-linear* process-passing language, one where the input process can be run at most once to yield a single (computation) path.

We can solve such recursive equations for path orders by several techniques, ranging from sophisticated methods providing inductive and coinductive characterisations [8], to simple methods essentially based on inductive definitions. Paralleling techniques on information systems [20], path orders under the order

$$\mathbb{P} \trianglelefteq \mathbb{Q} \iff \mathbb{P} \subseteq \mathbb{Q} \;\&\; (\forall p, p' \in \mathbb{P}.\; p \leq_{\mathbb{P}} p' \iff p \leq_{\mathbb{Q}} p')$$

form a (large) complete partial order with respect to which all the constructions on path orders we have just seen can be made Scott continuous. Solutions to equations like those above are then obtained as (simultaneous) least fixed points.

1.6 An affine-linear language for processes

Assume that path orders are presented using the constructions with the following syntax:

$$\begin{aligned}\mathbb{T} ::= & \mathbb{O} \mid \mathbb{T}_1 \otimes \mathbb{T}_2 \mid \mathbb{T}_1 \multimap \mathbb{T}_2 \mid \Sigma_{\alpha \in A} \alpha \mathbb{T}_\alpha \mid \mathbb{T}_1 \& \mathbb{T}_2 \\ & \mid P \mid \mu_j P_1, \cdots, P_k.(\mathbb{T}_1, \cdots, \mathbb{T}_k)\end{aligned}$$

All the construction names have been met earlier with the exception of the notation for recursively defined path orders. Above P is drawn from a set of variables used in the recursive definition of path orders; $\mu_j P_1, \cdots, P_k.(\mathbb{T}_1, \cdots, \mathbb{T}_k)$ stands for the j-component (so $1 \leq j \leq k$) of the least solution to the defining equations

$$P_1 = \mathbb{T}_1,\ \cdots,\ P_k = \mathbb{T}_k\ ,$$

in which the expressions $\mathbb{T}_1, \cdots, \mathbb{T}_k$ may contain $P_1, \cdots, P_k$. We shall write $\mu P_1, \cdots, P_k.(\mathbb{T}_1, \cdots, \mathbb{T}_k)$ as an abbreviation for

$$(\mu_1 P_1, \cdots, P_k.(\mathbb{T}_1, \cdots, \mathbb{T}_k), \cdots, \mu_k P_1, \cdots, P_k.(\mathbb{T}_1, \cdots, \mathbb{T}_k))\ .$$

In future we will often use vector notation and, for example, write $\mu \overrightarrow{P}.\overrightarrow{\mathbb{T}}$ for the expression above, and confuse a closed expression for a path order with the path order itself.

The operations of Sections 1.3 and 1.5 form the basis of a syntax of terms which will be subject to typing and linearity constraints:

$$
\begin{array}{lll}
t, u, v, \cdots ::= & x, y, z, \cdots & \text{(Variables)} \\
& \emptyset \ \mid \ \Sigma_{i\in I} t_i \ \mid & \text{(Sums)} \\
& \mathit{rec}\ x.t \ \mid & \text{(Recursive definitions)} \\
& \lambda x.t \ \mid \ u\ v \ \mid & \text{(Abstraction, application)} \\
& \alpha t \ \mid \ [t > \alpha x \Rightarrow u] \ \mid & \text{(Injections and match)} \\
& (t, u) \ \mid \ [t > (x, -) \Rightarrow u] \ \mid & \\
& \qquad [t > (-, x) \Rightarrow u] \ \mid & \text{(Pairing and match)} \\
& t \otimes u \ \mid \ [t > x \otimes y \Rightarrow u] & \text{(Tensor and match)}
\end{array}
$$

The language, first introduced in [30], is similar to that in [1], being based on a form of pattern matching. Accordingly, variables in the pattern, like x in the pattern of $[t > \alpha x \Rightarrow u]$, are binding occurrences and bind later occurrences of the variable in the body, u in this case. We shall take for granted an understanding of free and bound variables, and substitution on raw terms. In examples we will allow ourselves to use $+$ both in writing sums of terms and prefixed sums of path orders.

Let $\mathbb{P}_1, \cdots, \mathbb{P}_k$ be closed expressions for path orders and assume that the variables $x_1, \cdots, x_k$ are distinct. A syntactic judgement

$$x_1 : \mathbb{P}_1, \cdots, x_k : \mathbb{P}_k \vdash t : \mathbb{Q}$$

stands for a map

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

in **Aff**. We shall typically write Γ, or Δ, for an environment list $x_1 : \mathbb{P}_1, \cdots, x_k : \mathbb{P}_k$. We shall most often abbreviate the denotation map to

$$\mathbb{P}_1 \otimes \cdots \otimes \mathbb{P}_k \xrightarrow{t} \mathbb{Q} \ , \ \text{or even } \Gamma \xrightarrow{t} \mathbb{Q} \ .$$

Here k may be 0 so the environment list in the syntactic judgement is empty and the corresponding tensor product the empty path order $\mathbb{0}$.

An affine-linear language will restrict copying and so substitutions of a common term into distinct variables. The counterpart in the models is the absence of a suitable diagonal map from objects $\mathbb{P}$ to $\mathbb{P} \otimes \mathbb{P}$. For example the function $X \mapsto X \otimes X$ from $\widehat{\mathbb{P}}$ to $\widehat{\mathbb{P} \otimes \mathbb{P}}$ is not in general a map in **Aff**.† Consider a term $t(x, y)$, with its free variables x and y

† To see this, for example in $\mathbf{Aff_2}$, assume that $\mathbb{P}$ is the discrete order on the set $\{a, b\}$. Then the nonempty sum $x = y_{\mathbb{P}}(a) + y_{\mathbb{P}}(b)$ is not sent to

$$(y_{\mathbb{P}}(a) \otimes \ y_{\mathbb{P}}(a)) + (y_{\mathbb{P}}(b) \otimes \ y_{\mathbb{P}}(b)) = y_{\mathbb{P}\otimes\mathbb{P}}(a, a) + y_{\mathbb{P}\otimes\mathbb{P}}(b, b)$$

shown explicitly, for which

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

corresponding to a map $\mathbb{P} \otimes \mathbb{P} \xrightarrow{t(x,y)} \mathbb{Q}$ 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 $t(x, y) = x \otimes y$. There is however a condition on how the variables x and y occur in t which ensures that the judgement $x : \mathbb{P} \vdash t(x, x) : \mathbb{Q}$ holds and that it denotes the map in **Aff** obtained as the composition

$$\mathbb{P} \xrightarrow{\delta} \mathbb{P} \otimes \mathbb{P} \xrightarrow{t(x,y)} \mathbb{Q}$$

—using the weak diagonal map seen earlier in Section 1.5.4. (For example, in the term $x + y$, where x and y have the same type $\mathbb{P}$, only computation at one of the arguments x and y is possible, and it is legitimate to diagonalise to $x + x$ to obtain an affine-linear map.) Syntactically, this is assured if the variables x and y are *not crossed* in t according to the following definition:

Definition 1.4 Let t be a raw term. Say a set of variables V is *crossed* in t iff there are subterms of t of the form

$$\text{a tensor } s \otimes u, \text{ an application } (s\ u), \text{ or a match } [v > u \Rightarrow s]$$

for which t has free occurrences of variables from V appearing in both s and u.

For example, variables x and y are crossed in $x \otimes y$, but variables x and y are not crossed in $(x + y) \otimes z$. Note that a set of variables V is crossed in a term t if V contains variables x, y, not necessarily distinct, so that $\{x, y\}$ is crossed in t. We are mainly interested in when sets of variables are *not crossed* in a term. A set of variables $\{x_1, \cdots, x_k\}$ not being crossed in a term t ensures that computation paths at arguments $x_1, \cdots, x_k$ are in conflict—at most one can contribute to the computation path of t. Sets of variables of the same type which are not crossed in a term will behave like single variables with regard to substitutions.

as would be needed to preserve non-empty sums, but instead to

$$x \otimes x = y_{\mathbb{P}\otimes\mathbb{P}}(a, a) + y_{\mathbb{P}\otimes\mathbb{P}}(b, b) + y_{\mathbb{P}\otimes\mathbb{P}}(a, b) + y_{\mathbb{P}\otimes\mathbb{P}}(b, a)$$

with extra "cross terms."

The term-formation rules are listed below alongside their interpretations as constructors on morphisms, taking the morphisms denoted by the premises to that denoted by the conclusion (along the lines of [2]). We assume that the variables in any enviroment list which appears are distinct.

Structural rules:

$$\frac{}{x:\mathbb{P}\vdash x:\mathbb{P}} \quad \text{, interpreted as} \quad \frac{}{\mathbb{P}\xrightarrow{1_{\mathbb{P}}}\mathbb{P}}\ .$$

$$\frac{\Delta\vdash t:\mathbb{P}}{\Gamma,\Delta\vdash t:\mathbb{P}} \quad \text{, interpreted as} \quad \frac{\Delta\xrightarrow{t}\mathbb{P}}{\Gamma\otimes\Delta\xrightarrow{\emptyset\otimes 1_\Delta}\mathbb{0}\otimes\Delta\cong\Delta\xrightarrow{t}\mathbb{P}}\ .$$

$\dfrac{\Gamma,x:\mathbb{P},y:\mathbb{Q},\Delta\vdash t:\mathbb{R}}{\Gamma,y:\mathbb{Q},x:\mathbb{P},\Delta\vdash t:\mathbb{R}}$, interpreted via $s:\mathbb{Q}\otimes\mathbb{P}\cong\mathbb{P}\otimes\mathbb{Q}$ as

$$\frac{\Gamma\otimes\mathbb{P}\otimes\mathbb{Q}\otimes\Delta\xrightarrow{t}\mathbb{R}}{\Gamma\otimes\mathbb{Q}\otimes\mathbb{P}\otimes\Delta\xrightarrow{1_\Gamma\otimes s\otimes 1_\Delta}\Gamma\otimes\mathbb{P}\otimes\mathbb{Q}\otimes\Delta\xrightarrow{t}\mathbb{R}}\ .$$

Recursive path orders:

$$\frac{\Gamma\vdash t:\mathbb{T}_j[\mu\overrightarrow{P}.\overrightarrow{\mathbb{T}}/\overrightarrow{P}]}{\Gamma\vdash t:\mu_j\overrightarrow{P}.\overrightarrow{\mathbb{T}}}\ , \qquad \frac{\Gamma\vdash t:\mu_j\overrightarrow{P}.\overrightarrow{\mathbb{T}}}{\Gamma\vdash t:\mathbb{T}_j[\mu\overrightarrow{P}.\overrightarrow{\mathbb{T}}/\overrightarrow{P}]}\ .$$

where the premise and conclusion of each rule are interpreted as the same map because $\mu_j\overrightarrow{P}.\overrightarrow{\mathbb{T}}$ and $\mathbb{T}_j[\mu\overrightarrow{P}.\overrightarrow{\mathbb{T}}/\overrightarrow{P}]$ denote equal path orders.

Sums of terms:

$$\frac{}{\Gamma\vdash\emptyset:\mathbb{P}} \quad \text{, interpreted as} \quad \frac{}{\Gamma\xrightarrow{\emptyset}\mathbb{P}} \text{ , the constantly } \emptyset \text{ map.}$$

$$\frac{\Gamma\vdash t_i:\mathbb{P}\text{ for all }i\in I}{\Gamma\vdash\Sigma_{i\in I}t_i:\mathbb{P}} \quad \text{, interpreted as} \quad \frac{\Gamma\xrightarrow{t_i}\mathbb{P}\text{ for all }i\in I}{\Gamma\xrightarrow{\langle t_i\rangle_{i\in I}}\&_{i\in I}\,\mathbb{P}\xrightarrow{\Sigma}\mathbb{P}}\ .$$

Recursive definitions:

$$\frac{\Gamma,x:\mathbb{P}\vdash t:\mathbb{P}\quad\{y,x\}\text{ not crossed in }t\text{ for all }y\text{ in }\Gamma}{\Gamma\vdash rec\ x.t:\mathbb{P}}\ ,$$

$$\text{interpreted as} \quad \frac{\Gamma\otimes\mathbb{P}\xrightarrow{t}\mathbb{P}}{\Gamma\xrightarrow{fix\ F}\mathbb{P}}\ .$$

—see Section 1.5.1, where for $g:\Gamma\to\mathbb{P}$ the map $F(g):\Gamma\to\mathbb{P}$ is the composition

$$\Gamma\xrightarrow{\delta}\Gamma\otimes\Gamma\xrightarrow{1_\Gamma\otimes g}\Gamma\otimes\mathbb{P}\xrightarrow{t}\mathbb{P}\ .$$

Abstraction:

$$\frac{\Gamma,x:\mathbb{P}\vdash t:\mathbb{Q}}{\Gamma\vdash\lambda x.t:\mathbb{P}\multimap\mathbb{Q}} \quad \text{, interpreted as} \quad \frac{\Gamma\otimes\mathbb{P}\xrightarrow{t}\mathbb{Q}}{\Gamma\xrightarrow{curry\ t}(\mathbb{P}\multimap\mathbb{Q})}\ .$$

Application:

$$\frac{\Gamma \vdash u : \mathbb{P} \multimap \mathbb{Q} \quad \Delta \vdash v : \mathbb{P}}{\Gamma, \Delta \vdash u\ v : \mathbb{Q}}\ ,$$

$$\text{interpreted as} \quad \frac{\Gamma \xrightarrow{u} (\mathbb{P} \multimap \mathbb{Q}) \quad \Delta \xrightarrow{v} \mathbb{P}}{\Gamma \otimes \Delta \xrightarrow{u \otimes v} (\mathbb{P} \multimap \mathbb{Q}) \otimes \mathbb{P} \xrightarrow{app} \mathbb{Q}}\ .$$

Injections and match for prefixed sums:

$$\frac{\Gamma \vdash t : \mathbb{P}_\beta,\ \text{where } \beta \in A}{\Gamma \vdash \beta t : \Sigma_{\alpha \in A} \alpha \mathbb{P}_\alpha}\ \text{, interpreted as}\ \frac{\Gamma \xrightarrow{t} \mathbb{P}_\beta\ ,\ \text{where } \beta \in A}{\Gamma \xrightarrow{t} \mathbb{P}_\beta \xrightarrow{\beta} \Sigma_{\alpha \in A} \alpha \mathbb{P}_\alpha}\ .$$

$$\frac{\Gamma, x : \mathbb{P}_\beta \vdash u : \mathbb{Q}\ ,\ \text{where } \beta \in A. \quad \Delta \vdash t : \Sigma_{\alpha \in A} \alpha \mathbb{P}_\alpha}{\Gamma, \Delta \vdash [t > \beta x \Rightarrow u] : \mathbb{Q}}\ \text{, interpreted as}$$

$$\frac{\Gamma \otimes \mathbb{P}_\beta \xrightarrow{u} \mathbb{Q} \quad \Delta \xrightarrow{t} \sum_{\alpha \in A} \alpha \mathbb{P}_\alpha}{\Gamma \otimes \Delta \xrightarrow{1_\Gamma \otimes t} \Gamma \otimes \sum_{\alpha \in A} \alpha \mathbb{P}_\alpha \xrightarrow{dist} \Sigma_{\alpha \in A} \alpha(\Gamma \otimes \mathbb{P}_\alpha) \xrightarrow{u_{@\beta}} \mathbb{Q}}\ .$$

Pairing and matches for products:

$$\frac{\Gamma \vdash t : \mathbb{P} \quad \Gamma \vdash u : \mathbb{Q}}{\Gamma \vdash (t, u) : \mathbb{P}\&\mathbb{Q}}\ \text{, interpreted as}\ \frac{\Gamma \xrightarrow{t} \mathbb{P} \quad \Gamma \xrightarrow{u} \mathbb{Q}}{\Gamma \xrightarrow{\langle t, u \rangle} \mathbb{P}\&\mathbb{Q}}\ .$$

$$\frac{\Gamma, x : \mathbb{P} \vdash u : \mathbb{R} \quad \Delta \vdash t : \mathbb{P}\&\mathbb{Q}}{\Gamma, \Delta \vdash [t > (x, -) \Rightarrow u] : \mathbb{R}}\text{, interpreted as}$$

$$\frac{\Gamma \otimes \mathbb{P} \xrightarrow{u} \mathbb{R} \quad \Delta \xrightarrow{t} \mathbb{P}\&\mathbb{Q}}{\Gamma \otimes \Delta \xrightarrow{1_\Gamma \otimes (\pi_1 \circ t)} \Gamma \otimes \mathbb{P} \xrightarrow{u} \mathbb{R}}\ .$$

$$\frac{\Gamma, x : \mathbb{Q} \vdash u : \mathbb{R} \quad \Delta \vdash t : \mathbb{P}\&\mathbb{Q}}{\Gamma, \Delta \vdash [t > (-, x) \Rightarrow u] : \mathbb{R}}\text{, interpreted as}$$

$$\frac{\Gamma \otimes \mathbb{Q} \xrightarrow{u} \mathbb{R} \quad \Delta \xrightarrow{t} \mathbb{P}\&\mathbb{Q}}{\Gamma \otimes \Delta \xrightarrow{1_\Gamma \otimes (\pi_2 \circ t)} \Gamma \otimes \mathbb{Q} \xrightarrow{u} \mathbb{R}}\ .$$

Tensor operation and match for tensor:

$$\frac{\Gamma \vdash t : \mathbb{P} \quad \Delta \vdash u : \mathbb{Q}}{\Gamma, \Delta \vdash t \otimes u : \mathbb{P} \otimes \mathbb{Q}}\ \text{, interpreted as}\ \frac{\Gamma \xrightarrow{t} \mathbb{P} \quad \Delta \xrightarrow{u} \mathbb{Q}}{\Gamma \otimes \Delta \xrightarrow{t \otimes u} \mathbb{P} \otimes \mathbb{Q}}\ .$$

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

$$\frac{\Gamma \otimes \mathbb{P} \otimes \mathbb{Q} \xrightarrow{u} \mathbb{R} \quad \Delta \xrightarrow{t} \mathbb{P} \otimes \mathbb{Q}}{\Gamma \otimes \Delta \xrightarrow{1_\Gamma \otimes t} \Gamma \otimes \mathbb{P} \otimes \mathbb{Q} \xrightarrow{u} \mathbb{R}}\ .$$

By a straightforward induction on the derivation of the typing judgement we obtain:

Proposition 1.5 *Suppose* $\Gamma, x : \mathbb{P} \vdash t : \mathbb{Q}$. *The set* $\{x\}$ *is not crossed in* t.

Exploiting the naturality of the various operations used in the semantic definitions, we can prove a general substitution lemma. It involves the weak diagonal maps $\delta_k : \mathbb{P} \to \mathbb{P} \otimes \cdots \otimes \mathbb{P}$ of Section 1.5.4.

Lemma 1.6 *(Substitution Lemma) Suppose*

$$\Gamma, x_1 : \mathbb{P}, \cdots, x_k : \mathbb{P} \vdash t : \mathbb{Q}$$

and that the set of variables $\{x_1, \cdots, x_k\}$ *is not crossed in* t. *Suppose* $\Delta \vdash u : \mathbb{P}$ *where the variables of* Γ *and* Δ *are disjoint. Then,*

$$\Gamma, \Delta \vdash t[u/x_1, \cdots, u/x_k] : \mathbb{Q}$$

and

$$\begin{aligned}&[\![\Gamma, \Delta \vdash t[u/x_1, \cdots, u/x_k] : \mathbb{Q}]\!] \cong \\ &\quad [\![\Gamma, x_1 : \mathbb{P}, \cdots, x_k : \mathbb{P} \vdash t : \mathbb{Q}]\!] \quad \circ \quad (1_\Gamma \otimes (\delta_k \circ [\![\Delta \vdash u : \mathbb{P}]\!])) \ .\end{aligned}$$

In particular, as singleton sets of variables are not crossed in well-formed terms, we can specialise the Substitution Lemma to the following:

Corollary 1.7 *If* $\Gamma, x : \mathbb{P} \vdash t : \mathbb{Q}$ *and* $\Delta \vdash u : \mathbb{P}$, *where the variables of* Γ *and* Δ *are disjoint, then* $\Gamma, \Delta \vdash t[u/x] : \mathbb{Q}$ *and*

$$[\![\Gamma, \Delta \vdash t[u/x] : \mathbb{Q}]\!] \cong [\![\Gamma, x : \mathbb{P} \vdash t : \mathbb{Q}]\!] \circ (1_\Gamma \otimes [\![\Delta \vdash u : \mathbb{P}]\!]) \ .$$

As consequences of Corollary 1.7, linear application amounts to substitution, and recursions unfold in the expected way:

Lemma 1.8 *Suppose* $\Gamma \vdash (\lambda x.t)\ u : \mathbb{Q}$. *Then,* $\Gamma \vdash t[u/x] : \mathbb{Q}$ *and*

$$[\![\Gamma \vdash (\lambda x.t)\ u : \mathbb{Q}]\!] \cong [\![\Gamma \vdash t[u/x] : \mathbb{Q}]\!] \ .$$

Lemma 1.9 *Suppose* $\Gamma \vdash rec\ x.t : \mathbb{P}$. *Then* $\Gamma \vdash t[rec\ x.t/x] : \mathbb{P}$ *and*

$$[\![\Gamma \vdash rec\ x.t : \mathbb{P}]\!] \cong [\![\Gamma \vdash t[rec\ x.t/x] : \mathbb{P}]\!] \ .$$

The next lemma follows directly from the universal properties of prefixed sum (the last property because the mediating map is in **Lin**, so preserves sums):

Lemma 1.10 *Properties of prefix match:*

$$\begin{aligned}
&[\![\Gamma \vdash [\beta t > \beta x \Rightarrow u] : \mathbb{Q}]\!] \cong [\![\Gamma \vdash u[t/x] : \mathbb{Q}]\!] \ , \\
&[\![\Gamma \vdash [\alpha t > \beta x \Rightarrow u] : \mathbb{Q}]\!] = \emptyset \quad \textit{if } \alpha \neq \beta \ , \\
&[\![\Gamma \vdash [\Sigma_{i\in I} t_i > \beta x \Rightarrow u] : \mathbb{Q}]\!] \cong \Sigma_{i\in I} [\![\Gamma \vdash [t_i > \beta x \Rightarrow u] : \mathbb{Q}]\!] \ .
\end{aligned}$$

General patterns

We can write terms more compactly by generalising the patterns in matches. General patterns are built up according to

$$p ::= x \mid \emptyset \mid \alpha p \mid p \otimes q \mid (p, -) \mid (-, p) \ .$$

A match on a pattern $[u > p \Rightarrow t]$ binds the free variables of the pattern p to the resumptions after following the path specified by the pattern in u; because the term t may contain these variables freely the resumptions may influence the computation of t. Such a match is understood inductively as an abbreviation for a term in the metalanguage:

$$\begin{aligned}
&[u > x \Rightarrow t] \equiv (\lambda x.t)\ u \ , \qquad [u > \emptyset \Rightarrow t] \equiv t \ , \\
&[u > \alpha p \Rightarrow t] \equiv [u > \alpha x \Rightarrow [x > p \Rightarrow t]] \qquad \text{for a fresh variable } x, \\
&[u > (p, -) \Rightarrow t] \equiv [u > (x, -) \Rightarrow [x > p \Rightarrow t]] \qquad \text{for a fresh variable } x, \\
&[u > (-, p) \Rightarrow t] \equiv [u > (-, x) \Rightarrow [x > p \Rightarrow t]] \qquad \text{for a fresh variable } x, \\
&[u > p \otimes q \Rightarrow t] \equiv [u > x \otimes y \Rightarrow [x > p \Rightarrow [y > q \Rightarrow t]]] \text{ for fresh } x, y.
\end{aligned}$$

Let $\lambda x \otimes y.t$ stand for $\lambda w.[w > x \otimes y \Rightarrow t]$, where w is a fresh variable, and write $[u_1 > p_1, \cdots, u_k > p_k \Rightarrow t]$ to abbreviate $[u_1 > p_1 \Rightarrow [\cdots [u_k > p_k \Rightarrow t] \cdots]$.

1.7 Examples

The affine-linear language is remarkably expressive, as the following examples show. Through having denotations in $\mathbf{Aff}_{\mathbf{Set}}$, all operations expressible in the language will automatically preserve open-map bisimulation.

1.7.1 CCS

As in CCS, assume a set of labels A, a complementation operation producing $\bar{a}$ from a label a, with $\bar{\bar{a}} = a$, and a distinct label τ. In the metalanguage we can specify the path order $\mathbb{P}$ as the solution to†

$$\mathbb{P} = \tau\mathbb{P} + \Sigma_{a\in A} a\mathbb{P} + \Sigma_{a\in A}\bar{a}\mathbb{P} \ .$$

So $\mathbb{P}$ is given as $\mu P.\tau P + \Sigma_{a\in A} aP + \Sigma_{a\in A}\bar{a}P$. There are injections from $\mathbb{P}$ into its expression as a prefixed sum given as τt, at and $\bar{a}t$ for $a \in A$ and a term t of type $\mathbb{P}$. The CCS parallel composition can be defined as the following term of type $\mathbb{P} \otimes \mathbb{P} \multimap \mathbb{P}$ in the metalanguage:

$$\begin{aligned} Par = rec\ P.\ \lambda x \otimes y.\ & \Sigma_{\alpha\in A\cup\{\tau\}}[x > \alpha x' \Rightarrow \alpha(P(x' \otimes y))] + \\ & \Sigma_{\alpha\in A\cup\{\tau\}}[y > \alpha y' \Rightarrow \alpha(P(x \otimes y'))] + \\ & \Sigma_{a\in A}[x > ax',\ y > \bar{a}y' \Rightarrow \tau(P(x' \otimes y'))] \ . \end{aligned}$$

The other CCS operations are easy to encode, though recursive definitions in CCS have to be restricted to fit within the affine language. Interpreted in $\mathbf{Aff_2}$ two CCS terms will have the same denotation iff they have same traces (or execution sequences). By virtue of having been written down in the metalanguage the operation of parallel composition will preserve open-map bisimulation when interpreted in $\mathbf{Aff_{Set}}$; for this specific $\mathbb{P}$, open-map bisimulation coincides with strong bisimulation. In $\mathbf{Aff_{Set}}$ we can recover the expansion law by the properties of prefix match—Lemma 1.10. In detail, write $X|Y$ for $Par\ X \otimes Y$, where X and Y are terms of type $\mathbb{P}$. Suppose

$$X = \Sigma_{\alpha\in A\cup\{\tau\}}\Sigma_{i\in I(\alpha)}\alpha X_i \ , \qquad Y = \Sigma_{\alpha\in A\cup\{\tau\}}\Sigma_{j\in J(\alpha)}\alpha Y_j \ .$$

Using Lemma 1.8, and then that the matches distribute over nondeterministic sums,

$$\begin{aligned} X|Y \cong & \Sigma_{\alpha\in A\cup\{\tau\}}[X > \alpha x' \Rightarrow \alpha(x'|Y)] + \Sigma_{\alpha\in A\cup\{\tau\}}[Y > \alpha y' \Rightarrow \alpha(X|y')] \\ & + \Sigma_{a\in A}[X > ax',\ Y > \bar{a}y' \Rightarrow \tau(x'|y')] \\ \cong & \Sigma_{\alpha\in A\cup\{\tau\}}\Sigma_{i\in I(\alpha)}\alpha(X_i|Y) + \Sigma_{\alpha\in A\cup\{\tau\}}\Sigma_{j\in J(\alpha)}\alpha(X|Y_j) \\ & + \Sigma_{a\in A}\Sigma_{i\in I(a),j\in J(\bar{a})}\tau(X_i|Y_j) \ . \end{aligned}$$

In similar ways it is easy to express CSP in the affine-linear language along the lines of [4], and any parallel composition given by a synchronisation algebra [31].

† In examples, for readablility, we will generally write recursive definitions of types and processes as equations.

1.7.2 A linear higher-order process language

Recall the path orders for processes, concretions and abstractions for a higher-order language in Section 1.5.6. We are chiefly interested in the parallel composition of processes, $Par_{\mathbb{P},\mathbb{P}}$ of type $\mathbb{P} \otimes \mathbb{P} \multimap \mathbb{P}$. But parallel composition is really a family of mutually dependent operations also including components such as $Par_{\mathbb{F},\mathbb{C}}$ of type $\mathbb{F} \otimes \mathbb{C} \multimap \mathbb{P}$ to say how abstractions compose in parallel with concretions *etc.* All these components can be tupled together in a product using &, and parallel composition defined as a simultaneous recursive definition whose component at $\mathbb{P} \otimes \mathbb{P} \multimap \mathbb{P}$ satisfies

$$\begin{aligned} P|Q = &\Sigma_\alpha[P > \alpha x \Rightarrow \alpha(x|Q)] + \\ &\Sigma_\alpha[Q > \alpha y \Rightarrow \alpha(P|y)] + \\ &\Sigma_a[P > a?f,\ Q > a!(s \otimes r) \Rightarrow \tau((f\ s)|r)] + \\ &\Sigma_a[P > a!(s \otimes r),\ Q > a?f \Rightarrow \tau(r|(f\ s))]\ , \end{aligned}$$

where we have chosen suggestive names for the injections and, for instance, $P|Q$ abbreviates $Par_{\mathbb{P},\mathbb{P}}(P \otimes Q)$. In the summations $a \in A$ and α ranges over $a!, a?, \tau$ for $a \in A$.

1.7.3 Mobile ambients with public names

We can translate the Ambient Calculus with public names [6] into the affine-linear language, following similar lines to the linear process-passing language above. Assume a fixed set of ambient names $n, m, \cdots \in N$. The syntax of ambients is extended beyond just processes (P) to include concretions (C) and abstractions (F), following [5]:

$$\begin{aligned} P ::= &\emptyset \mid P|P \mid repP \mid n[P] \mid in\ n\ P \mid out\ n\ P \mid open\ n!\ P \mid \\ &\tau\ P \mid mvin\ n!\ C \mid mvout\ n!\ C \mid open\ n?\ P \mid mvin\ n?\ F \mid x \\ C ::= &P \otimes P \\ F ::= &\lambda x.P \end{aligned}$$

The notation for actions departs a little from that of [5]. Here some actions are marked with ! and others with ?—active (or inceptive) actions are marked by ! and passive (or receptive) actions by ?. We say actions α and β are *complementary* iff one has the form *open* $n!$ or $mvin\ n!$ while the other is *open* $n?$ or $mvin\ n?$ respectively. Complementary actions can synchronise together to form a τ-action. We adopt a slightly different notation for concretions ($P \otimes R$ instead of $\langle P \rangle R$) and abstractions ($\lambda x.P$

instead of $(x)P$) to make their translation into the affine-linear language clear.

The usual conventions are adopted for variables. Terms are assumed to be *linear*, in that a variable appears on at most one side of any parallel compositions within the term, and subterms of the form $repP$ have no free variables. A replication $repP$ is intended to behave as $P \mid repP$ so readily possesses a recursive definition in the affine-linear language.

Suitable path orders for ambients are given recursively by:

$$\mathbb{P} = \tau\mathbb{P} + \Sigma_{n\in N} in\ n\ \mathbb{P} + \Sigma_{n\in N} out\ n\ \mathbb{P} + \Sigma_{n\in N} open\ n!\mathbb{P} +$$
$$\Sigma_{n\in N} mvin\ n!\mathbb{C} + \Sigma_{n\in N} mvout\ n!\mathbb{C} + \Sigma_{n\in N} open\ n?\mathbb{P} + \Sigma_{n\in N} mvin\ n?\mathbb{F}$$
$$\mathbb{C} = \mathbb{P} \otimes \mathbb{P}$$
$$\mathbb{F} = \mathbb{P} \multimap \mathbb{P}$$

The eight components of the prefixed sum in the equation for $\mathbb{P}$ correspond to eight forms of ambient actions: τ, $in\ n$, $out\ n$, $open\ n!$, $mvin\ n!$, $mvout\ n!$, $open\ n?$ and $mvin\ n?$. We obtain the prefixing operations as injections into the appropriate component of $\mathbb{P}$ as a prefixed sum.

Parallel composition is really a family of operations, one of which is a binary operation between processes but where in addition there are parallel compositions of abstractions with concretions, and even abstractions with processes and concretions with processes. The family of operations

$$\begin{array}{ll} (-|-) : \mathbb{F} \otimes \mathbb{C} \multimap \mathbb{P}, & (-|-) : \mathbb{C} \otimes \mathbb{F} \multimap \mathbb{P}, \\ (-|-) : \mathbb{F} \otimes \mathbb{P} \multimap \mathbb{F}, & (-|-) : \mathbb{P} \otimes \mathbb{F} \multimap \mathbb{F}, \\ (-|-) : \mathbb{C} \otimes \mathbb{P} \multimap \mathbb{C}, & (-|-) : \mathbb{P} \otimes \mathbb{C} \multimap \mathbb{C} \end{array}$$

are defined in a simultaneous recursive definition as follows:
Processes in parallel with processes:

$$\begin{aligned} P|Q = {} & \Sigma_\alpha[P > \alpha x \Rightarrow \alpha(x|Q)] + \\ & \Sigma_\alpha[Q > \alpha y \Rightarrow \alpha(P|y)] + \\ & \Sigma_n[P > open\ n!x,\ Q > open\ n?y \Rightarrow \tau(x|y)] + \\ & \Sigma_n[P > open\ n?x,\ Q > open\ n!y \Rightarrow \tau(x|y)] + \\ & \Sigma_n[P > mvin\ n?f,\ Q > mvin\ n!(s \otimes r) \Rightarrow \tau((f\ s)|r)] + \\ & \Sigma_n[P > mvin\ n!(s \otimes r),\ Q > mvin\ n?f \Rightarrow \tau(r|(f\ s))]\ . \end{aligned}$$

Abstractions in parallel with concretions:

$$F|C = [C > s \otimes r \Rightarrow (F\ s)|r]\ .$$

Abstractions in parallel with processes:

$$F|P = \lambda x.((F\ x)|P)\ .$$

Concretions in parallel with processes:

$$C|P = [C > s \otimes r \Rightarrow s \otimes (r|P)]\ .$$

The remaining cases are given symmetrically.

Presheaves X, Y over $\mathbb{P}$ will have decompositions into rooted components:

$$X \cong \Sigma_\alpha \Sigma_{i \in X(\alpha)} \alpha X_i\ , \qquad Y \cong \Sigma_\alpha \Sigma_{j \in X(\alpha)} \alpha Y_j$$

—here α ranges over ambient actions. By the properties of prefix-match (Lemma 1.10), their parallel composition satisfies the expansion law

$$\begin{aligned}
&X|Y \cong \Sigma_\alpha \Sigma_{i \in X(\alpha)} \alpha(X_i|Y) + \Sigma_\alpha \Sigma_{j \in Y(\alpha)} \alpha(X|Y_j) + \\
&\Sigma_n \Sigma_{i \in X(open\ n!), j \in Y(open\ n?)} \tau(X_i|Y_j) + \Sigma_n \Sigma_{i \in X(open\ n?), j \in Y(open\ n!)} \tau(X_i|Y_j) + \\
&\Sigma_n \Sigma_{i \in X(mvin\ n!), j \in Y(mvin\ n?)} \tau(X_i|Y_j) + \Sigma_n \Sigma_{i \in X(mvin\ n?), j \in Y(mvin\ n!)} \tau(X_i|Y_j)\ .
\end{aligned}$$

Ambient creation can be defined recursively in the affine-linear language:

$$\begin{aligned}
m[P] = &[P > \tau x \Rightarrow \tau m[x]] + \\
&\Sigma_n [P > in\ n\ x \Rightarrow mvin\ n!(m[x] \otimes \emptyset)] + \\
&\Sigma_n [P > out\ n\ x \Rightarrow mvout\ n!(m[x] \otimes \emptyset)] + \\
&[P > mvout\ m!(s \otimes r) \Rightarrow \tau(s|m[r])] + \\
&open\ m?P\ +\ mvin\ m?\ \lambda y.m[P|y]\ .
\end{aligned}$$

The denotations of ambients are determined by their capabilities: an ambient $m[P]$ can perform the internal (τ) actions of P, enter a parallel ambient ($mvin\ n!$) if called upon to do so by an $in\ n$-action of P, exit an ambient n ($mvout\ n!$) if P so requests through an $out\ n$-action, be exited if P 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.

By the properties of prefix-match, there is an expansion theorem for ambient creation. For X with decomposition

$$X = \Sigma_\alpha \Sigma_{i \in X(\alpha)} \alpha X_i \ ,$$

where α ranges over atomic actions of ambients,

$$\begin{aligned} m[X] \cong & \Sigma_{i \in X(\tau)} \tau \ m[X_i] \\ & \Sigma_n \Sigma_{j \in X(in\ n)} mvin\ n!(m[X_j] \otimes \emptyset) + \\ & \Sigma_n \Sigma_{k \in X(out\ n)} mvout\ n!(m[X_k] \otimes \emptyset) \\ & \Sigma_{s \in X(mvout\ m!)} [X_s > s \otimes r \Rightarrow \tau(s | m[r])] + \\ & open\ m?X + mvin\ m?(\lambda y.m[X|y]) \ . \end{aligned}$$

1.7.4 Nondeterministic dataflow

The affine linear language allows us to define processes of the kind encountered in treatments of nondeterministic dataflow.

Define $\mathbb{P}$ recursively so that

$$\mathbb{P} = a\mathbb{P} + b\mathbb{P} \ .$$

$\mathbb{P}$ consists of finite streams (or sequences) of a's and b's.

The recursively defined process $A : \mathbb{P} \multimap \mathbb{P}$ selects and outputs a's from a stream of a's and while ignoring all b's:

$$A = \lambda x. \ [x > ax' \Rightarrow a(A\ x')] + [x > bx' \Rightarrow z_1 \otimes (A\ x')]$$

The recursively defined process $F : \mathbb{P} \otimes \mathbb{P}$ produces two parallel streams of a's and b's as output such that it outputs the same number of a's and b's to both streams:

$$F = [F > z_1 \otimes z_2 \Rightarrow (az_1) \otimes (az_2) + (bz_1) \otimes (bz_2)]$$

The recursively defined process $S : \mathbb{P} \multimap (\mathbb{P} \otimes \mathbb{P})$ separates a stream of a's and b's into two streams, the first consisting solely of a's and the second solely of b's:

$$\begin{aligned} S = \lambda x. \quad & [x > ax', (S\ x') > z_1 \otimes z_2 \Rightarrow (az_1) \otimes z_2] + \\ & [x > bx', (S\ x') > z_1 \otimes z_2 \Rightarrow z_1 \otimes (bz_2)] \end{aligned}$$

A subcategory of $\mathbf{Aff_{Set}}$ supports a "trace operation" to represent processes with feedback loops (see [13]). The trace operation is, however, not definable in the present affine-linear language. It can be shown by induction on the typing derivation of a term that:

Proposition 1.11 *Suppose* $\Gamma \vdash t : \mathbb{Q}$ *where* $\Gamma \equiv x_1 : \mathbb{P}_1, \cdots, x_k : \mathbb{P}_k$. *Let* p *be path in* $(\mathbb{P}_1 \otimes \cdots \otimes \mathbb{P}_k)_\perp$ *and* q *be a path in* $\mathbb{Q}$. *The presheaf denotation of a term* $\Gamma \vdash t : \mathbb{Q}$, *applied to* $j_{\mathbb{P}_1 \otimes \cdots \otimes \mathbb{P}_k}(p)$ *as presheaf, has a nonempty contribution at* q *iff the trace-set denotation of* $\Gamma \vdash t : \mathbb{Q}$, *applied to* $j_{\mathbb{P}_1 \otimes \cdots \otimes \mathbb{P}_k}(p)$ *as a trace-set, contains* q.

Thus, supposing that the trace operation of [13] were definable in the presheaf semantics, we would obtain a compositional relational semantics of nondeterministic dataflow with feedback, shown impossible by the Brock-Ackerman anomaly [3].

1.8 Nonlinearity

Of course code can be copied, and this may lead to maps which are not linear. According to the discipline of linear logic, nonlinear maps from $\mathbb{P}$ to $\mathbb{Q}$ are introduced as linear maps from $!\mathbb{P}$ to $\mathbb{Q}$—the exponential $!$ applied to $\mathbb{P}$ allows arguments from $\mathbb{P}$ to be copied or discarded freely.

In the domain model of linear logic $!\mathbb{P}$ can be taken to be the finite-join completion of $\mathbb{P}$. Then, the nonlinear maps, maps $!\mathbb{P} \to \mathbb{Q}$ in $\mathbf{Lin_2}$, correspond to Scott continuous functions $\widehat{\mathbb{P}} \to \widehat{\mathbb{Q}}$. A close analogue for presheaf models is to interpret $!\mathbb{P}$ as the finite-colimit completion of $\mathbb{P}$. Note that now $!\mathbb{P}$ is a category, and no longer just a partial order. With this understanding of $!\mathbb{P}$, it can be shown that $\widehat{!\mathbb{P}}$ with the inclusion functor $!\mathbb{P} \to \widehat{!\mathbb{P}}$ is the free filtered colimit completion of $!\mathbb{P}$—see [21]. It follows that maps $!\mathbb{P} \to \mathbb{Q}$ in $\mathbf{Lin_{Set}}$ correspond, to within isomorphism, to continuous (*i.e.*, filtered colimit preserving) functors $\widehat{\mathbb{P}} \to \widehat{\mathbb{Q}}$. But, unfortunately, continuous functors from $\widehat{\mathbb{P}}$ to $\widehat{\mathbb{Q}}$ need not send open maps to open maps. This raises the question of whether other choices of exponential fit in better with bisimulation.

Bear in mind the intuition that objects of $\mathbb{P}$ correspond to the shapes of computation path a process, represented as a presheaf in $\widehat{\mathbb{P}}$, might perform. An object of $!\mathbb{P}$ should represent a computation path of an assembly of processes each with computation-path shapes in $\mathbb{P}$—the assembly of processes can then be the collection of copies of a process, possibly at different states. If we take $!\mathbb{P}$ to be the finite colimit completion of $\mathbb{P}$, an object of $!\mathbb{P}$ as a finite colimit would express how paths coincide initially and then branch. One way to understand this object as a computation path of an assembly of processes, is that the assembly of processes is not fixed once and for all. Rather the assembly grows as further copies are invoked, and that these copies can be made of a

processes *after* they have run for a while. The copies can then themselves be run and the resulting processes copied. In this way, by keeping track of the origins of copies, we can account for the identifications of sub-paths.

This intuition suggests exploring other less liberal ways of copying, without, for example, being able to copy after some initial run. We will discover candidates for exponentials $!\mathbb{P}$ based on computation-path shapes of simple assemblies of processes, ones built out of indexed families. We start with an example.

1.8.1 An example

First observe the hopeful sign that maps which are not linear may still preserve bisimulation. For example, a functor yielding a presheaf $H(X,Y)$, for presheaves X and Y over $\mathbb{P}$, which is "bilinear" or "affine bilinear," in the sense that it is linear (*i.e.*, colimit preserving) or affine linear (*i.e.*, connected-colimit preserving) in each argument separately, when diagonalised to the functor giving $H(X,X)$ for X in $\widehat{\mathbb{P}}$, will still preserve open maps and bisimulation. A well-known example of a bilinear functor is the product operation on presheaves [18]; with one argument fixed, the product is left adjoint to the exponentiation in the presheaf category, and so product preserves colimits, and thus open maps, in each argument. On similar lines, it can be shown that the tensor operations in $\mathbf{Lin}_{\mathbf{Set}}$ and $\mathbf{Aff}_{\mathbf{Set}}$ are bilinear and affine bilinear, respectively. With this encouragement we look for alternative interpretations of the exponential $!$, where the nonlinear maps $!\mathbb{P} \to \mathbb{Q}$ in $\mathbf{Lin}_{\mathbf{Set}}$ preserve open maps.

Because sum preserves open maps, by the remarks above, the functor *copy* taking a presheaf X over $\mathbb{P}$ to the presheaf

$$copy(X) = 1 + X + X^2 + X^3 + \cdots + X^k + \cdots$$

over

$$!\mathbb{P} = \mathbb{1} + \mathbb{P} + \mathbb{P}^2 + \mathbb{P}^3 + \cdots + \mathbb{P}^k + \cdots$$

will preserve open maps. Here the superscripts abbreviate repeated applications of tensor in $\mathbf{Lin}_{\mathbf{Set}}$. So $\mathbb{P}^k$ is the product of k copies of the partial order $\mathbb{P}$, in which the objects are k-tuples of objects of $\mathbb{P}$—in particular, $\mathbb{1}$ is the partial order consisting solely of the empty tuple called 1 above. The presheaf X^k comprises k copies of X tensored together, so that $X^k\langle p_1, \cdots, p_k\rangle = X(p_1) \times \cdots \times X(p_k)$.

By supplying "coefficients" we can obtain various nonlinear maps. An appropriate form of polynomial is given by a functor

$$F :!\mathbb{P} \to \widehat{\mathbb{Q}} \ ,$$

which splits up into a family of functors

$$F_k : \mathbb{P}^k \to \widehat{\mathbb{Q}} \ , \text{ for } k \in \omega \ .$$

We can extend F to a functor $F[-] = F \cdot \mathit{copy}(-) : \widehat{\mathbb{P}} \to \widehat{\mathbb{Q}}$. For $X \in \widehat{\mathbb{P}}$,

$$F[X] = F_0 + F_1 \cdot X + F_2 \cdot X^2 + F_3 \cdot X^3 + \cdots + F_k \cdot X^k + \cdots$$

Because $F \cdot -$ is colimit-preserving it preserves open maps. So does *copy*. Hence $F[-]$ preserves open maps.

Note, that the original polynomial F is not determined to within isomorphism by the functor $F[-]$ it induces. (We can only hope for such uniqueness if we restrict to polynomials which are symmetric, *i.e.*, such that $F_k \cong F_k \circ \pi$ for all permutations π of the k arguments.)

We write $\mathbf{Poly}(\mathbb{P}, \mathbb{Q})$ for the functor category $[!\mathbb{P}, \widehat{\mathbb{Q}}]$ of polynomials from $\mathbb{P}$ to $\mathbb{Q}$. In order to compose polynomials, $F \in \mathbf{Poly}(\mathbb{P}, \mathbb{Q})$ and $G \in \mathbf{Poly}(\mathbb{Q}, \mathbb{R})$, we first define $F^! \in \mathbf{Poly}(\mathbb{P}, !\mathbb{Q})$ by taking

$$F^!\langle p_1, \cdots, p_n\rangle\langle q_1, \cdots, q_k\rangle = \Sigma_{\mu\langle s_1, \cdots, s_k\rangle = \langle p_1, \cdots, p_n\rangle} Fs_1q_1 \times \cdots \times Fs_kq_k \ ,$$

when $\langle p_1, \cdots, p_n\rangle \in !\mathbb{P}$ and $\langle q_1, \cdots, q_k\rangle \in !\mathbb{Q}$. The operation $\mu :!!\mathbb{P} \to \mathbb{P}$ flattens, by concatenation, a tuple $\langle s_1, \cdots, s_k\rangle$ of tuples $s_r = \langle s_{r1}, \cdots, s_{rm_r}\rangle$, for $1 \leq r \leq k$, down to a tuple

$$\mu\langle s_1, \cdots, s_k\rangle = \langle s_{11}, \cdots, s_{1m_1}, s_{21}, \cdots s_{k1}, \cdots, s_{km_k}\rangle \ .$$

So, the sum is indexed by all ways to partition $\langle p_1, \cdots, p_n\rangle$ into tuples $\langle s_1, \cdots, s_k\rangle$. Now, we can define the composition of polynomials to be

$$G \circ F = G \cdot (F^!-) \in \mathbf{Poly}(\mathbb{P}, \mathbb{R}) \ .$$

At $\langle p_1, \cdots, p_n\rangle$ in $!\mathbb{P}$,

$$G \circ F\langle p_1, \cdots, p_n\rangle = \Sigma_{\mu\langle s_1, \cdots, s_k\rangle = \langle p_1, \cdots, p_n\rangle} G_k \cdot Fs_1 \times \cdots \times Fs_k \ ,$$

where $Fs_1 \times \cdots Fs_k$, built using the tensor of **Lin**, is such that

$$Fs_1 \times \cdots Fs_k\langle q_1, \cdots, q_k\rangle = Fs_1q_1 \times \cdots Fs_kq_k$$

for $\langle q_1, \cdots, q_k\rangle$ in $!\mathbb{Q}$. The composition of polynomials is only defined to within isomorphism; they form a bicategory **Poly**, rather than a category.

Note that $!\mathbb{O} = \mathbb{1}$. In the special case where $F :!\mathbb{O} \to \widehat{\mathbb{Q}}$, so that F

merely points to a presheaf X in $\widehat{\mathbb{Q}}$, the composition $G \circ F$ of a polynomial $G : !\mathbb{Q} \to \widehat{\mathbb{R}}$ with the polynomial F is isomorphic to $G[X]$. So certainly compositions of this form preserve open maps and bisimulation.

More generally, polynomials in $\mathbf{Poly}(\mathbb{P}, \mathbb{Q})$ and $\mathbf{Poly}(\mathbb{Q}, \mathbb{R})$ correspond to presheaves in $\widehat{(!\mathbb{P})^{op} \times \mathbb{Q}}$ and $\widehat{(!\mathbb{Q})^{op} \times \mathbb{R}}$, respectively. So under this correspondence polynomials are related by open maps and bisimulation. It can be shown that the composition of polynomials in general preserves open maps, so bisimulation, between polynomials.

However, the present interpretation of ! fails as a candidate for the exponential of linear logic. This is because **Poly** is not cartesian-closed in any reasonable sense. It easy to see that there is an isomorphism of categories

$$\mathbf{Poly}(\mathbb{R}, \mathbb{P}\&\mathbb{Q}) \cong \mathbf{Poly}(\mathbb{R}, \mathbb{P}) \times \mathbf{Poly}(\mathbb{R}, \mathbb{Q}) \ ,$$

natural in $\mathbb{R}$ in $\mathbf{Lin}_{\mathbf{Set}}$, showing the sense in which $\mathbb{P}\&\mathbb{Q}$, given by juxtaposition, remains a product in the bicategory of polynomials. There is also clearly an isomorphism of functor categories

$$[!\mathbb{P} \times !\mathbb{Q}, \widehat{\mathbb{R}}] \cong [!\mathbb{P}, \widehat{(!\mathbb{Q})^{op} \times \mathbb{R}})] \ .$$

But, in general, $!(\mathbb{P}\&\mathbb{Q})$ and $!\mathbb{P} \times !\mathbb{Q}$ are not isomorphic, so that $(!\mathbb{Q})^{op} \times \mathbb{R}$ is not a function space for the polynomials with respect to $-\&-$. The difficulty boils down to a lack of symmetry in the current definition of $!\mathbb{P}$, where tuples like $\langle p_1, \cdots, p_k \rangle$ and its permutations $\langle p_{\pi(1)}, \cdots, p_{\pi(k)} \rangle$ are not necessarily related by any maps. Nor for that matter, are there any maps from a tuple $\langle p_1, \cdots, p_k \rangle$ to a larger tuple $\langle p_1, \cdots, p_k, \cdots, p_m \rangle$, even though intuitively the larger tuple would be a path of a larger assembly of processes, so arguably an extension of the smaller tuple in which further copies have been invoked.

To allow different kinds of polynomial, polynomials which can take account of the symmetry there exists between different copies and also permit further copies to be invoked as needed, we broaden the picture.

1.8.2 General polynomials

The example suggests that we take assemblies of processes to be families where we can reindex copies, precisely how being prescribed in $\mathbb{U}$, a subcategory of sets in which the maps are the possible reindexings. A $\mathbb{U}$-family of a category $\mathcal{A}$ comprises $\langle A_i \rangle_{i \in I}$ where $i \in I$, with I an object of $\mathbb{U}$, index objects A_i in $\mathcal{A}$. A map of families $(f, e) : \langle A_i \rangle_{i \in I} \to \langle A'_j \rangle_{j \in J}$ consists of a reindexing function $f : I \to J$ in $\mathbb{U}$ and $e = \langle e_i \rangle_{i \in I}$, a family

of maps $e_i : A_i \to A'_{f(i)}$ in $\mathcal{A}$. With the obvious composition we obtain $\mathcal{F}_{\mathbb{U}}(\mathcal{A})$, the category of $\mathbb{U}$-families.

Imitating the example, we define the category of polynomials

$$\mathbf{Poly}_{\mathbb{U}}(\mathbb{P}, \mathbb{Q})$$

from $\mathbb{P}$ to $\mathbb{Q}$, to be the functor category $[\mathcal{F}_{\mathbb{U}}(\mathbb{P}), \mathbb{Q}]$. Under sufficient conditions, that $\mathbb{U}$ is small, has a singleton and dependent sums (a functor $\Sigma : \mathcal{F}_{\mathbb{U}}(\mathbb{U}) \to \mathbb{U}$ collapsing any family of sets in $\mathbb{U}$ to a set in $\mathbb{U}$), we can compose polynomials in the manner of the coKleisli construction. For this we need to turn $\mathcal{F}_{\mathbb{U}}$ into a functor on polynomials for which we need a "distributive law" converting a family of presheaves into a presheaf over families of paths. It can be shown that provided all the maps in $\mathbb{U}$ (the possible reindexings) are injective, composition of polynomials preserves open maps and bisimulation. Provided $\mathbb{U}$ contains the empty set, we can specialise composition, as in the example, to obtain a functor $F[-] : \widehat{\mathbb{P}} \to \widehat{\mathbb{Q}}$ from a polynomial F in $\mathbf{Poly}_{\mathbb{U}}(\mathbb{P}, \mathbb{Q})$.

The example is now seen as the special case in which $\mathbb{U}$ consists of subsets, possibly empty, of positive natural numbers $\{1, \cdots, n\}$ with identities as the only maps. In the special case in which $\mathbb{U}$ is the full subcategory of **Set** consisting of the empty set and a singleton, polynomials amount to functors $\mathbb{P}_\perp \to \widehat{\mathbb{Q}}$ so to maps in $\mathbf{Aff}_{\mathbf{Set}}$. If we take $\mathbb{U}$ to be $\mathbb{I}$ (finite sets with injections) or $\mathbb{B}$ (finite sets with bijections), we can repair an inadequacy in the example; then, $\mathcal{F}_{\mathbb{U}}(\mathbb{P}\&\mathbb{Q})$ and $\mathcal{F}_{\mathbb{U}}(\mathbb{P}) \times \mathcal{F}_{\mathbb{U}}(\mathbb{Q})$ are isomorphic, so that we obtain a function space for the polynomials with respect to the product $-\&-$. Both $\mathcal{F}_{\mathbb{I}}$ and $\mathcal{F}_{\mathbb{B}}$ are good candidates for the exponential !—they also behave well with respect to bisimulation.

There is a fly in the ointment however. The complete mathematical story, in which one would see the polynomials as maps in a coKleisli construction, uses bicategories and at least pseudo (co)monads on biequivalent 2-categories. At the time of writing (December 2001) the theory of pseudo monads, even the definitions, is not sufficiently developed.

1.9 Related work

This article presents two domain theories for concurrent computation. One uses domains of a traditional kind, though in a non-traditional way through being based on computation paths (though the path-based domain theory here was anticipated in Matthew Hennessy's work on domain models of concurrency [14]). In the other domain theory, domains

are understood as presheaf categories, accompanied by the bisimulation equivalence got from open maps.

Just as there are alternatives to the domain theory of Dana Scott, in particular, the stable domain theory of Gérard Berry, so are there alternative denotational semantics of the affine-linear language. Mikkel Nygaard and I have shown how to give an event-structure semantics to the affine-linear language; both types and terms denote event structures. (Event structures are a key "independence model" for computation, one in which the concurrency of events is represented by their causal independence.) The domain theory can be seen as analogous to the stable domain theory of Berry. The presheaf semantics here and that based on event structures differ at function spaces. In the fragment of the affine-linear language without function spaces, the event-structure semantics gives an informative representation of the definable presheaves; elements of a definable presheaf correspond to finite configurations of an event structure, with restriction in the presheaf matched by restriction to a subconfiguration in the event structure. Tensor corresponds to a simple parallel composition of event structures got by disjoint juxtaposition. Unfortunately the event structures definable in the affine-linear language can be shown too impoverished to coincide with those of the event-structure semantics of CCS, given for example in [31].

Mikkel Nygaard and I are developing an operational semantics for the affine linear language [26]. This work has also led us to an expressive nonlinear language with a simple operational semantics; its denotational semantics is based on a choice of exponential from Section 1.8. One aim is to give an operational account of open-map bisimulation on higher-order processes.

The semantics here does not cover name generation as in Milner's π-Calculus. Although one can give a presheaf semantics to the pi-Calculus [9], we do not presently know how to extend this to also include higher-order processes.

This article has demonstrated that linearity as formalised in linear logic can play a central role in developing a domain theory suitable for concurrent computation. At the same time, the mathematical neutrality of the domain theories here begins to show how concurrency need not remain the rather separate study it has become. There are unresolved issues in extending the work of this article towards a fully-fledged domain theory, one able to cope more completely with the range of models for concurrency. Among them is the question of how to extend this work

to include name generation, its relation with operational semantics, and the place of independence models such as event structures.

Acknowledgements

A good deal of the background for this work was developed with Gian Luca Cattani for his PhD [7]. Discussions with Martin Hyland and John Power have played a crucial role in the ongoing work on nonlinearity. I am grateful for discussions with my PhD student Mikkel Nygaard.

References

[1] S. Abramsky. Computational interpretations of linear logic. *Theoretical Computer Science*, 111, 1-2, 3–57, 1993.

[2] T. Braüner. An Axiomatic Approach to Adequacy. BRICS Dissertation Series DS-96-4, 1996.

[3] J. Brock & W. Ackerman. Scenarios: A model of non-determinate computation. In *Proc. of Formalization of Programming Concepts*, LNCS 107, 1981.

[4] S.D. Brookes. On the relationship of CCS and CSP. In *Proc. of ICALP'83*, LNCS 154, 1983.

[5] L. Cardelli & A. Gordon. A commitment relation for the ambient calculus. Note ambient-commitment.pdf at http://research.microsoft.com/ adg/Publications/, 2000.

[6] L. Cardelli & A. Gordon. Anytime, Anywhere. Modal logics for mobile ambients. In *Proc. of POPL'00,* 2000.

[7] G. L. Cattani. PhD thesis, CS Dept., University of Aarhus, BRICS-DS-99-1, 1999.

[8] G. L. Cattani, M. Fiore & G. Winskel. A Theory of Recursive Domains with Applications to Concurrency. In *Proc. of LICS '98.*

[9] G. L. Cattani, I. Stark, & G. Winskel. Presheaf Models for the π-Calculus. In *Proc. of CTCS '97*, LNCS 1290, 1997.

[10] G. L. Cattani & G. Winskel. Presheaf Models for Concurrency. In *Proc. of CSL' 96*, LNCS 1258, 1997.

[11] G. L. Cattani & G. Winskel. Profunctors, open maps and bisimulation. Manuscript, 2000.

[12] G. L. Cattani, A. J. Power & G. Winskel. A categorical axiomatics for bisimulation. In *Proc. of CONCUR'98*, LNCS 1466, 1998.

[13] T. Hildebrandt, P. Panangaden & G. Winskel. Relational semantics of nondeterministic dataflow. In *Proc. of CONCUR'98*, LNCS 1466, 1998.

[14] M. Hennessy. A Fully Abstract Denotational Model for Higher-Order Processes. *Information and Computation*, 112:55–95, 1994.

[15] M. Hennessy & G.D. Plotkin. Full abstraction for a simple parallel programming language. In *Proc. of MFCS'79*, LNCS 74, 1979.

[16] C.A.R. Hoare. A model for communicating sequential processes. *Tech. Report PRG-22, University of Oxford Computing Lab.*, 1981.

[17] B. Jacobs. Semantics of weakening and contraction. *Annals of Pure and Applied Logic*, 69:73–106, 1994.

[18] A. Joyal & I. Moerdijk. A completeness theorem for open maps. *Annals of Pure and Applied Logic*, 70:51–86, 1994.
[19] A. Joyal, M. Nielsen & G. Winskel. Bisimulation from open maps. *Information and Computation*, 127:164–185, 1996.
[20] G.Winskel & K.Larsen. Using information systems to solve recursive domain equations effectively. LNCS 173, 1984.
[21] G.M. Kelly. *Basic concepts of enriched category theory.* London Math. Soc. Lecture Note Series 64, CUP, 1982.
[22] S. Mac Lane & I. Moerdijk. *Sheaves in Geometry and Logic.* Springer-Verlag, 1992..
[23] R. Milner. *A Calculus of Communicating Systems.* LNCS 92, 1980.
[24] R. Milner. *Communication and concurrency.* Prentice Hall, 1989.
[25] M. Nielsen, G.D. Plotkin & G. Winskel. Petri nets, Event structures and Domains, part 1. Theoretical Computer Science, vol. 13, 1981.
[26] M. Nygaard & G. Winskel. Linearity in distributed computation. In *Proc. of LICS '02.*
[27] G. Winskel. A representation of completely distributive algebraic lattices. Report of the Computer Science Dept., Carnegie-Mellon University, 1983.
[28] G. Winskel. An introduction to event structures. In *Proc. of REX summerschool in temporal logic, 'May 88*, LNCS 354, 1988.
[29] G. Winskel. A presheaf semantics of value-passing processes. In *Proceedings of CONCUR'96*, LNCS 1119, 1996.
[30] G. Winskel. A linear metalanguage for concurrency. In *Proceedings of AMAST'98*, LNCS 1548, 1999.
[31] G. Winskel & M. Nielsen. Models for concurrency. In *Handbook of Logic in Computer Science, Vol.4,* OUP, 1995.