

Presheaf models and process calculi

Lecture notes (draft of June 14, 2002)

Mikkel Nygaard

DISI, May–June 2002

1 Introduction

Process calculi like CCS have been motivated and studied operationally, thus from the outset lacking the abstract mathematical treatment provided by a domain theory. Consequently, concurrency has become a rather separate study; in particular, higher-order and functional features as known from sequential programming are most often treated in an ad hoc fashion, if at all.

The study of presheaf models of processes [3, 14] can be seen as an attempt to bring concurrency back within the realm of traditional denotational semantics by providing a domain theory for concurrent computation. Much of the work so far [4, 6, 7, 8, 9, 10, 27] has concentrated on developing the domain theory itself and on showing how to handle existing models and notions from process calculi within it. Meanwhile, a full operational understanding of presheaf models has still not been obtained. A sensible way to proceed would be to

- exploit the domain theory to define mathematically natural process calculi; and
- approach an operational understanding of presheaf models by investigating the operational semantics of these calculi.

These notes report on joint work with Glynn Winskel along those lines [21, 22]. Section 2 contains a brief introduction to CCS intended for readers unfamiliar with Milner’s work. To give an idea of what our approach has to offer in terms of process calculi, we give in Section 3 an operational account of a higher-order process language which can be viewed as an extension of the lambda calculus with a CCS-like prefix operation.

It is hoped that after this appetiser, the reader will be motivated for the presentation in later sections of the more abstract material concerning presheaves and the different categories they organise themselves in.

2 CCS

In the preface of [25], Robin Milner writes:

In 1972 [...] I tried to apply semantic ideas – known from work on sequential programming – to a concurrent programming language and I found them insufficient.

In particular, Milner found that viewing a program as a function over memory states, and thus identifying the programs

(1) $X := 1$ and (2) $X := 0; X := X + 1$,

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

This led Milner to search for something more fundamental – a new calculus with *interaction* or *communication* as the central idea. He and Tony Hoare [11] independently came up with the same primitive notion of *indivisible interaction*. Milner’s calculus, called “Calculus of Communicating Systems” [24], or CCS, is an attempt of providing a paradigm of concurrent computation in the same sense that the lambda calculus is a paradigm for sequential computation.

2.1 Intuition

Terms of CCS stand for computing agents, or processes, and are built using a small set of operations. Processes may perform sequences of atomic actions, among which may be actions of communication. We start with giving some intuition through the use of a series of vending machine examples.

Prefixing A “one-time” vending machine, which may acept a coin, then give coffee, after which it is the inactive process, \emptyset , unable to perform further actions:

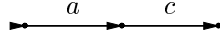


Figure 1: $a.c.\emptyset$

Relabelling A one-time tea vending machine using the description of the coffee version:

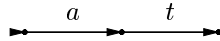


Figure 2: $(a.c.\emptyset)[t/c]$

Sum Two one-time vending machines selling both coffee and tea:

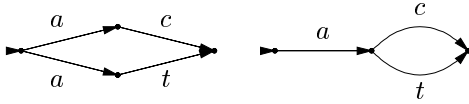


Figure 3: $a.c.\emptyset + a.t.\emptyset$ and $a.(c.\emptyset + t.\emptyset)$

Recursion A vending machine which may repeatedly accept a coin and give out coffee:

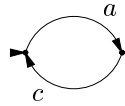


Figure 4: $rec\ x.a.c.x$

Communication Suppose the coffee vending machine consists of two subsystems, one accepting coins, the other giving out the coffee. They may communicate by a notification action n . We can describe the two systems by the terms $rec\ x.a.n.x$ and $rec\ x.\bar{n}.c.x$. Here, \bar{n} is the *complement* of the action n . Processes may synchronise on complementary actions, producing an anonymous “internal communication action”, τ :

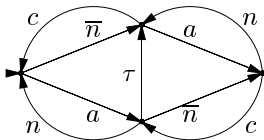


Figure 5: $rec\ x.a.n.x \mid rec\ x.\bar{n}.c.x$

Restriction But clearly we should abstract away from the internal workings of the machine. This is obtained using restriction:

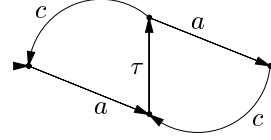


Figure 6: $(rec\ x.a.n.x \mid rec\ x.\bar{n}.c.x) \setminus \{n\}$

2.2 Theory

To formally define CCS, we assume a set of *names* A . Write \bar{A} for the set $\{\bar{a} \mid a \in A\}$ of *conames*, and L for the set $A \cup \bar{A}$ of *labels*. We let l range over L and extend complementation to the whole of L , so that $\bar{\bar{a}} = a$. Finally, we write Act for the set $L \cup \{\tau\}$ of *actions*, ranged over by α . The terms of CCS are now given by the grammar

$$t, u ::= x \mid rec\ x.t \mid \Sigma_{i \in I} t_i \mid \alpha.t \mid t|u \mid t \setminus S \mid t[f]$$

We write \emptyset for the empty sum. In the restriction term $t \setminus S$, S is a subset of A , and for relabelling $t[f]$, $f : A \rightarrow A$ is a function which we extend to Act by defining $f\bar{l} = \bar{f}l$ and $f\tau = \tau$.

The operational semantics of CCS is given as a system of rules deriving transitions of the form $t \xrightarrow{\alpha} t'$, with t, t' closed CCS terms—see Figure 7.

Example 2.1 Consider again the process of Figure 5. In clockwise order, starting from the initial state, the four states of its transition system are:

$$t|u \quad t|c.u \quad n.t|c.u \quad n.t|u.$$

Here, we have written t for $rec\ x.a.n.x$ and u for $rec\ x.\bar{n}.c.x$. \square

The calculus and its operational semantics are complemented by notions of *process equivalence*. Roughly, two processes are considered equivalent if they are indistinguishable from the point of view of a sequential observer, requesting sequences of actions, one by one. Notice that this abstracts away from the syntactic representation of agents and so transition system states. Depending on whether the observer ignores τ -actions or not, the induced equivalence is called *weak* or *strong*. We’ll concentrate on the latter where τ -actions are treated as any other actions. Here, Milner adopted David Park’s notion of bisimulation [23]:

$$\begin{array}{c}
\frac{t[\text{rec } x.t/x] \xrightarrow{\alpha} t'}{\text{rec } x.t \xrightarrow{\alpha} t'} \quad \frac{t_j \xrightarrow{\alpha} t'}{\sum_{i \in I} t_i \xrightarrow{\alpha} t'} \quad j \in I \quad \frac{}{\alpha.t \xrightarrow{\alpha} t} \\
\frac{t \xrightarrow{\alpha} t'}{t|u \xrightarrow{\alpha} t'|u} \quad \frac{t \xrightarrow{l} t' \quad u \xrightarrow{\bar{l}} u'}{t|u \xrightarrow{\tau} t'|u'} \quad \frac{u \xrightarrow{\alpha} u'}{t|u \xrightarrow{\alpha} t'|u'} \\
\frac{t \xrightarrow{\alpha} t'}{t \setminus S \xrightarrow{\alpha} t' \setminus S} \quad \alpha \notin S \cup \bar{S} \quad \frac{t \xrightarrow{\alpha} t'}{t[f] \xrightarrow{f\alpha} t'[f]}
\end{array}$$

Figure 7: CCS operational semantics

Definition 2.2 A relation R on closed CCS terms is a *bisimulation* if $t_1 R t_2$ implies

- (i) if $t_1 \xrightarrow{\alpha} t'_1$ then $t_2 \xrightarrow{\alpha} t'_2$ for some t'_2 such that $t'_1 R t'_2$.
- (ii) if $t_2 \xrightarrow{\alpha} t'_2$ then $t_1 \xrightarrow{\alpha} t'_1$ for some t'_1 such that $t'_1 R t'_2$.

We write \sim for the largest bisimulation, called *bisimilarity*.

Bisimilarity is an equivalence relation by definition, and Milner showed that it is in fact a *congruence*, i.e. it is preserved by all the operators of CCS.

Example 2.3 Consider the two processes of Figure 3. They are not bisimilar, because if R is a bisimulation relation which relates the processes $a.c.\emptyset + a.t.\emptyset$ and $a.(c.\emptyset + t.\emptyset)$, then then by (i) above, it must also relate the pair $c.\emptyset$ and $c.\emptyset + t.\emptyset$ which violates (ii) because the latter can do a t -transition which the former cannot match. \square

Example 2.4 The process of Figure 6 is bisimilar to the process $\text{rec } x.a.\tau.(c.x + a.c.\tau.c.x)$. \square

Notice that in this last example, parallel composition is reduced to nondeterminism (up to bisimilarity). A well-known result by Milner, called the *expansion law*, shows how this happens in general. Suppose that

$$t \sim \sum_{\alpha} \sum_{i \in I(\alpha)} \alpha.t_i \quad \text{and} \quad u \sim \sum_{\alpha} \sum_{j \in J(\alpha)} \alpha.u_j .$$

Then

$$\begin{aligned}
t|u \sim & \sum_{\alpha} \sum_{i \in I(\alpha)} \alpha.(t_i|u) + \\
& \sum_{\alpha} \sum_{j \in J(\alpha)} \alpha.(t|u_j) + \\
& \sum_l \sum_{i \in I(l), j \in J(\bar{l})} \tau.(t_i|u_j) .
\end{aligned}$$

2.3 Returning to semantics

CCS is now a classic theory which has inspired a school of researchers throughout the world and Milner received the Turing Award in 1991 partly in recognition of this foundational work in concurrency theory. On the down side, the very shift away from traditional semantics that led Milner to CCS has made concurrency a rather separate study. However, recent research indicates that it need not remain so.

In the handbook chapter [28], Glynn Winskel and Mogens Nielsen show how categorical notions (in particular, adjunctions) relate models of concurrency and allow for the transfer of definitions and results between them. For example, there is an adjunction between the category **Tree** of trees and the category **Tran** of transition systems,

$$\begin{array}{ccc}
& I & \\
\text{Tree} & \xrightarrow{\quad} & \text{Tran} \\
& \perp & \\
& U &
\end{array}$$

Here, I is inclusion and U the familiar operation of *unfolding* a transition system to a tree. The transition systems of Figure 3 unfolds to the trees

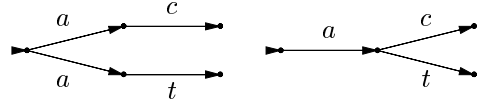


Figure 8: Unfoldings

In the paper [14], André Joyal, Nielsen, and Winskel extend the above work by defining a categorical notion of bisimulation, via *open maps* in a topos of *presheaves*. They point out that presheaves may be a fruitful abstract model for concurrency, generalising many familiar models.

Winskel and his PhD student Gian Luca Cattani go on to develop this idea in collaboration with a number of other researchers—see Cattani’s thesis [3].

The new work [21, 22] presented in these notes takes the development “full circle” back to considering process calculi, but now based on a denotational model of presheaves. This offers

- universal constructions that guide the definition of process operations away from the ad hoc;
- higher-order and functional features; and
- a type system.

In particular, in the following section, we’ll describe an extension of the lambda calculus with a CCS-like prefix operation. This new language, here called HOPLA, can be given a denotational semantics using a cartesian closed category \mathbf{Cts} of presheaf categories.

3 HOPLA

Rather than going into the categorical details straight away, we start by giving an operational account of HOPLA. The language is typed. The type of a process describes the possible computation paths the process can perform. Computation paths may consist simply of sequences of atomic actions (as for CCS) but they may also represent the input-output behaviour of a process. A typing judgement

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

means that a process t yields computation paths in \mathbb{Q} once processes with computation paths in $\mathbb{P}_1, \dots, \mathbb{P}_k$ are assigned to the variables x_1, \dots, x_k respectively. The types \mathbb{P} are built using a “prefixed sum,” products, function spaces and recursive definition. It is notable that although we can express many kinds of concurrent processes in the language, the language itself does not have many features typical of process calculi built-in, beyond that of a nondeterministic sum and prefix operations.

The operations associated with the prefix-sum type constructor are central to the expressiveness of the language. A prefix-sum type has the form $\Sigma_{\alpha \in A} \alpha. \mathbb{P}_\alpha$; it describes computation paths in which first an action $\beta \in A$ is performed before resuming as a computation path in \mathbb{P}_β . The prefix sum is associated with prefix operations taking a process t of type \mathbb{P}_β to $\beta.t$ of type

$\Sigma_{\alpha \in A} \alpha. \mathbb{P}_\alpha$, as well as a prefix-match $[u > \beta.x \Rightarrow t]$, where u has prefix-sum type, x has type \mathbb{P}_β and t generally involves the variable x . The term $[u > \beta.x \Rightarrow t]$ matches u against the pattern $\beta.x$ and passes the results of successful matches for x on to t . In more detail, we have the following isomorphisms in the denotational semantics:

$$\begin{aligned} [\alpha.u > \alpha.x \Rightarrow t] &\cong t[u/x] \\ [\beta.u > \alpha.x \Rightarrow t] &\cong \emptyset \text{ if } \alpha \neq \beta \\ [\Sigma_{i \in I} u_i > \alpha.x \Rightarrow t] &\cong \Sigma_{i \in I} [u_i > \alpha.x \Rightarrow t] \end{aligned}$$

The types of the language, interpreted as objects of \mathbf{Cts} , are given by the grammar

$$\mathbb{P}, \mathbb{Q} ::= \Sigma_{\alpha \in A} \alpha. \mathbb{P}_\alpha \mid \mathbb{P} \rightarrow \mathbb{Q} \mid \mathbb{P} \& \mathbb{Q} \mid P \mid \mu_j \vec{P}. \vec{\mathbb{P}}.$$

Here, P is drawn from a set of type variables used in defining recursive types; $\mu_j \vec{P}. \vec{\mathbb{P}}$ is interpreted as the j -component, for $1 \leq j \leq k$, of the “least” solution to the defining equations

$$P_1 = \mathbb{P}_1, \dots, P_k = \mathbb{P}_k,$$

in which the expressions $\mathbb{P}_1, \dots, \mathbb{P}_k$ may contain the P_j ’s. We shall write $\mu \vec{P}. \vec{\mathbb{P}}$ as an abbreviation for the k -tuple with j -component $\mu_j \vec{P}. \vec{\mathbb{P}}$.

The raw syntax is given by the grammar

$$\begin{aligned} t, u ::= &x \mid \text{rec } x.t \mid \Sigma_{i \in I} t_i \mid \alpha.t \mid [u > \alpha.x \Rightarrow t] \mid \\ &\lambda x.t \mid t \ u \mid (t, u) \mid \text{fst } t \mid \text{snd } t \end{aligned}$$

The variable x in the match term $[u > \alpha.x \Rightarrow t]$ is a binding occurrence and so binds later occurrences of the variable in the body t . We shall take for granted an understanding of free and bound variables, and substitution on raw terms.

Let $\mathbb{P}_1, \dots, \mathbb{P}_k, \mathbb{Q}$ be closed type expressions and assume that the variables x_1, \dots, x_k are distinct. A syntactic judgement $x_1 : \mathbb{P}_1, \dots, x_k : \mathbb{P}_k \vdash t : \mathbb{Q}$ can be interpreted as a map $\mathbb{P}_1 \& \dots \& \mathbb{P}_k \rightarrow \mathbb{Q}$ in \mathbf{Cts} . We let Γ range over environment lists $x_1 : \mathbb{P}_1, \dots, x_k : \mathbb{P}_k$, which we may treat as finite maps from variables to closed type expressions. The term formation rules are given in Figure 9.

As is to be expected, we have a syntactic substitution lemma:

Lemma 3.1 *Suppose $\Gamma, x : \mathbb{P} \vdash t : \mathbb{Q}$ and $\Gamma \vdash u : \mathbb{P}$. Then $\Gamma \vdash t[u/x] : \mathbb{Q}$.*

The semantic counterpart essentially says that the denotation of $t[u/x]$ is the functorial composition of the denotations of t and u .

$$\begin{array}{c}
\frac{\Gamma(x) = \mathbb{P}}{\Gamma \vdash x : \mathbb{P}} \quad \frac{\Gamma, x : \mathbb{P} \vdash t : \mathbb{P}}{\Gamma \vdash \text{rec } x.t : \mathbb{P}} \quad \frac{\Gamma \vdash t_j : \mathbb{P} \quad \text{all } j \in I}{\Gamma \vdash \sum_{i \in I} t_i : \mathbb{P}} \\
\frac{\Gamma \vdash t : \mathbb{P}_\beta \quad \text{where } \beta \in A}{\Gamma \vdash \beta.t : \sum_{\alpha \in A} \alpha.\mathbb{P}_\alpha} \quad \frac{\Gamma \vdash u : \sum_{\alpha \in A} \alpha.\mathbb{P}_\alpha \quad \Gamma, x : \mathbb{P}_\beta \vdash t : \mathbb{Q} \quad \text{where } \beta \in A}{\Gamma \vdash [u > \beta.x \Rightarrow t] : \mathbb{Q}} \\
\frac{\Gamma, x : \mathbb{P} \vdash t : \mathbb{Q}}{\Gamma \vdash \lambda x.t : \mathbb{P} \rightarrow \mathbb{Q}} \quad \frac{\Gamma \vdash t : \mathbb{P} \rightarrow \mathbb{Q} \quad \Gamma \vdash u : \mathbb{P}}{\Gamma \vdash t u : \mathbb{Q}} \\
\frac{\Gamma \vdash t : \mathbb{P} \quad \Gamma \vdash u : \mathbb{Q}}{\Gamma \vdash (t, u) : \mathbb{P} \& \mathbb{Q}} \quad \frac{\Gamma \vdash t : \mathbb{P} \& \mathbb{Q}}{\Gamma \vdash \text{fst } t : \mathbb{P}} \quad \frac{\Gamma \vdash t : \mathbb{P} \& \mathbb{Q}}{\Gamma \vdash \text{snd } t : \mathbb{Q}} \\
\frac{\Gamma \vdash t : \mathbb{P}_j[\mu \vec{P}. \vec{P} / \vec{P}]}{\Gamma \vdash t : \mu_j \vec{P}. \vec{P}} \quad \frac{\Gamma \vdash t : \mu_j \vec{P}. \vec{P}}{\Gamma \vdash t : \mathbb{P}_j[\mu \vec{P}. \vec{P} / \vec{P}]}
\end{array}$$

Figure 9: Typing rules for terms

$$\frac{\beta \in A}{\sum_{\alpha \in A} \alpha.\mathbb{P}_\alpha : \beta : \mathbb{P}_\beta} \quad \frac{\vdash u : \mathbb{P} \quad \mathbb{Q} : a : \mathbb{P}'}{\mathbb{P} \rightarrow \mathbb{Q} : u \mapsto a : \mathbb{P}'} \quad \frac{\mathbb{P} : a : \mathbb{P}'}{\mathbb{P} \& \mathbb{Q} : (a, -) : \mathbb{P}'} \quad \frac{\mathbb{P}_j[\mu \vec{P}. \vec{P} / \vec{P}] : a : \mathbb{P}'}{\mu_j \vec{P}. \vec{P} : a : \mathbb{P}'}$$

Figure 10: Typing rules for actions

$$\begin{array}{c}
\frac{t[\text{rec } x.t/x] \xrightarrow{a} t'}{\text{rec } x.t \xrightarrow{a} t'} \quad \frac{t_j \xrightarrow{a} t' \quad j \in I}{\sum_{i \in I} t_i \xrightarrow{a} t'} \quad \frac{}{\alpha.t \xrightarrow{\alpha} t} \quad \frac{u \xrightarrow{\alpha} u' \quad t[u'/x] \xrightarrow{a} t'}{[u > \alpha.x \Rightarrow t] \xrightarrow{a} t'} \\
\frac{t[u/x] \xrightarrow{a} t'}{\lambda x.t \xrightarrow{u \mapsto a} t'} \quad \frac{t \xrightarrow{u \mapsto a} t'}{t u \xrightarrow{a} t'} \\
\frac{t \xrightarrow{a} t'}{(t, u) \xrightarrow{(a, -)} t'} \quad \frac{u \xrightarrow{a} u'}{(t, u) \xrightarrow{(-, a)} u'} \quad \frac{t \xrightarrow{(a, -)} t'}{\text{fst } t \xrightarrow{a} t'} \quad \frac{t \xrightarrow{(-, a)} t'}{\text{snd } t \xrightarrow{a} t'}
\end{array}$$

Figure 11: HOPLA operational semantics

$$\begin{array}{c}
\frac{t[\text{rec } x.t/x] \Downarrow v}{\text{rec } x.t \Downarrow v} \quad \frac{t_j \Downarrow v \quad j \in I}{\sum_{i \in I} t_i \Downarrow v} \quad \frac{}{\alpha.t \Downarrow \alpha.t} \quad \frac{u \Downarrow \alpha.u' \quad t[u'/x] \Downarrow v}{[u > \alpha.x \Rightarrow t] \Downarrow v} \\
\frac{}{\lambda x.t \Downarrow \lambda x.t} \quad \frac{t \Downarrow \lambda x.t' \quad t'[u/x] \Downarrow v}{t u \Downarrow v} \\
\frac{}{(t, u) \Downarrow (t, u)} \quad \frac{t \Downarrow (t', t') \quad t' \Downarrow v}{\text{fst } t \Downarrow v} \quad \frac{t \Downarrow (t', t') \quad t'' \Downarrow v}{\text{snd } t \Downarrow v}
\end{array}$$

Figure 12: HOPLA evaluation

3.1 Operational semantics

With actions given by the grammar

$$a ::= \alpha \mid u \mapsto a \mid (a, -) \mid (-, a),$$

the operational rules in Figure 11 define a transition semantics for the language. In the rule for lambda-abstraction, we must have $x : \mathbb{P} \vdash t : \mathbb{Q}$ and $\vdash u : \mathbb{P}$ for some \mathbb{P}, \mathbb{Q} .

Example 3.2 Suppose $x : \mathbb{P} \vdash t : \mathbb{Q}$ and $\vdash u : \mathbb{P}$. The derivation

$$\frac{\frac{\vdots}{t[u/x] \xrightarrow{a} t'}}{\lambda x.t \xrightarrow{u \mapsto a} t'} \xrightarrow{a} t'$$

shows how the rules for abstraction and application cooperate. \square

To show that the rules are type-correct, we assign types to actions a using a judgement of the form $\mathbb{P} : a : \mathbb{P}'$. Intuitively, after performing the action a , what remains of a computation path in \mathbb{P} is a computation path in \mathbb{P}' . The typing rules for actions are given in Figure 10 (the rule for $(-, a)$ is omitted). Notice that in any judgement $\mathbb{P} : a : \mathbb{P}'$, the type \mathbb{P}' is unique given \mathbb{P} and a .

Proposition 3.3 *Suppose $\vdash t : \mathbb{P}$. If $t \xrightarrow{a} t'$ then $\mathbb{P} : a : \mathbb{P}'$ and $\vdash t' : \mathbb{P}'$.*

The rules are sound and complete with respect to the denotational semantics: derivations $t \xrightarrow{a} t'$ are in 1-1 correspondence with “rooted components” of $\llbracket a \rrbracket(\llbracket t \rrbracket)$, with $\llbracket t' \rrbracket$ corresponding to such a rooted component. What this means will be made clear later.

As a side-remark, the operational rules incorporate evaluation in the following sense: Let values v be given by the grammar

$$v ::= \alpha.t \mid \lambda x.t \mid (t, u).$$

Then we can define a nondeterministic evaluation relation \Downarrow , see Figure 12, such that

Proposition 3.4 *If d is a derivation of $t \xrightarrow{a} t'$ then there is a value v such that $t \Downarrow v$ and $v \xrightarrow{a} t'$ by a subderivation of d .*

The evaluation relation respects types, but beware that because of the rule for sum it preserves neither meaning nor bisimilarity.

$$\begin{aligned} (i) \quad & rec\ x.t \sim t[rec\ x.t/x] \\ (ii) \quad & [\alpha.u > \alpha.x \Rightarrow t] \sim t[u/x] \\ (iii) \quad & [\beta.u > \alpha.x \Rightarrow t] \sim \emptyset \quad \text{if } \alpha \neq \beta \\ (iv) \quad & [\sum_{i \in I} u_i > \alpha.x \Rightarrow t] \sim \sum_{i \in I} [u_i > \alpha.x \Rightarrow t] \\ (v) \quad & [u > \alpha.x \Rightarrow \sum_{i \in I} t_i] \sim \sum_{i \in I} [u > \alpha.x \Rightarrow t_i] \\ (vi) \quad & (\lambda x.t)\ u \sim t[u/x] \\ (vii) \quad & \lambda x.(t\ x) \sim t \\ (viii) \quad & \lambda x.(\sum_{i \in I} t_i) \sim \sum_{i \in I} (\lambda x.t_i) \\ (ix) \quad & (\sum_{i \in I} t_i)\ u \sim \sum_{i \in I} (t_i\ u) \\ (x) \quad & fst(t, u) \sim t \\ (xi) \quad & snd(t, u) \sim u \\ (xii) \quad & (fst\ t, snd\ t) \sim t \\ (xiii) \quad & \sum_{i \in I} (t_i, u_i) \sim (\sum_{i \in I} t_i, \sum_{i \in I} u_i) \\ (xiv) \quad & fst(\sum_{i \in I} t_i) \sim \sum_{i \in I} (fst\ t_i) \\ (xv) \quad & snd(\sum_{i \in I} t_i) \sim \sum_{i \in I} (snd\ t_i) \end{aligned}$$

Figure 13: Bisimilar terms

3.2 Bisimulation

We extend Definition 2.2 to the operational semantics of HOPLA by demanding that a bisimulation R respects types, and by quantifying over general actions a .

Theorem 3.5 *Bisimilarity is a congruence.*

PROOF: Using Howe’s method [12]. \square

Figure 13 shows a number of (closed, well-formed) terms which can all be shown bisimilar. Items (i), (ii), (vi), (x), and (xi) are “ β -rules”, while (vii) and (xii) are “ η -rules”. In each case $t_1 \sim t_2$, we can prove that the identity relation extended by the pair (t_1, t_2) is a bisimulation, so the correspondence is very tight, as is to be expected since in the denotational semantics, we have $\llbracket t_1 \rrbracket \cong \llbracket t_2 \rrbracket$.

The following results characterises bisimilarity at arrow type:

Proposition 3.6 *Let t_1, t_2 be closed terms of function type $\mathbb{P} \rightarrow \mathbb{Q}$. The following are equivalent:*

- (i) t_1 and t_2 are bisimilar;
- (ii) $t_1\ u$ and $t_2\ u$ are bisimilar for all closed terms u of type \mathbb{P} ; and
- (iii) $t_1\ u_1$ and $t_2\ u_2$ are bisimilar for all pairs of bisimilar closed terms u_1, u_2 of type \mathbb{P} .

3.3 Encoding CCS

We specify the type \mathbb{P} of CCS computation paths by

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

The terms of CCS can then be expressed in HOPLA as the following terms of type \mathbb{P} :

$$\begin{aligned} \llbracket x \rrbracket &\equiv x & \llbracket \text{rec } x.t \rrbracket &\equiv \text{rec } x. \llbracket t \rrbracket \\ \llbracket \alpha.t \rrbracket &\equiv \alpha. \llbracket t \rrbracket & \llbracket \sum_{i \in I} t_i \rrbracket &\equiv \sum_{i \in I} \llbracket t_i \rrbracket \\ \llbracket t|u \rrbracket &\equiv P \llbracket t \rrbracket \llbracket u \rrbracket & \llbracket t \setminus S \rrbracket &\equiv R_S \llbracket t \rrbracket \\ \llbracket t[f] \rrbracket &\equiv N_f \llbracket t \rrbracket \end{aligned}$$

Here, P , R_S , and N_f are abbreviations for the following recursively defined processes:

$$\begin{aligned} P &\equiv \text{rec } p.\lambda x.\lambda y. \\ &\quad \sum_{\alpha} [x > \alpha.x' \Rightarrow \alpha.(p \ x' \ y)] + \\ &\quad \sum_{\alpha} [y > \alpha.y' \Rightarrow \alpha.(p \ x \ y')] + \\ &\quad \sum_i [x > l.x' \Rightarrow [y > \bar{l}.y' \Rightarrow \tau.(p \ x' \ y')]] \\ R_S &\equiv \text{rec } r.\lambda x.\sum_{\alpha \notin (S \cup \bar{S})} [x > \alpha.x' \Rightarrow \alpha.(r \ x')] \\ N_f &\equiv \text{rec } n.\lambda x.\sum_{\alpha} [x > \alpha.x' \Rightarrow f(\alpha).(n \ x')] \end{aligned}$$

Proposition 3.7 *If $t \xrightarrow{\alpha} t'$ is derivable in CCS then $\llbracket t \rrbracket \xrightarrow{\alpha} \llbracket t' \rrbracket$ according to HOPLA. And conversely, if $\llbracket t \rrbracket \xrightarrow{a} u$ is derivable in HOPLA, then for some α, t' we have $a = \alpha$ and $u \equiv \llbracket t' \rrbracket$ and $t \xrightarrow{\alpha} t'$ according to CCS.*

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

We can recover the expansion law for general reasons. Write $t|u$ for the application $P \ t \ u$, where t and u are terms of type \mathbb{P} . Suppose

$$t \sim \sum_{\alpha} \sum_{i \in I(\alpha)} \alpha.t_i \quad \text{and} \quad u \sim \sum_{\alpha} \sum_{j \in J(\alpha)} \alpha.u_j.$$

Using Figure 13 item (i), (vi), then items (ii-iv), we get

$$\begin{aligned} t|u &\sim \sum_{\alpha} [t > \alpha.x' \Rightarrow \alpha.(x'|u)] + \\ &\quad \sum_{\alpha} [u > \alpha.y' \Rightarrow \alpha.(t|y')] + \\ &\quad \sum_i [t > l.x' \Rightarrow [u > \bar{l}.y' \Rightarrow \tau.(x'|y')]] \\ &\sim \sum_{\alpha} \sum_{i \in I(\alpha)} \alpha.(t_i|u) + \\ &\quad \sum_{\alpha} \sum_{j \in J(\alpha)} \alpha.(t|u_j) + \\ &\quad \sum_i \sum_{i \in I(i), j \in J(\bar{i})} \tau.(t_i|u_j). \end{aligned}$$

Finally, note that although HOPLA can encode CCS in a simple way, the language has no distinguished invisible action τ , so there is the issue of how to support more abstract operational equivalences such as weak bisimulation, perhaps starting from [7].

4 Processes as presheaves

We now begin an abstract mathematical study of processes, starting from the idea of representing a process as a kind of collection of its possible computation paths.

4.1 Path-based models of processes

Consider processes which, like those of CCS, can perform simple atomic actions, one at a time, among which may be actions of synchronisation. An old idea is to represent the nondeterministic behaviour of such a process as a “collection” of the computation paths it can perform. The trace model [11] and tree model [24] of processes are based on different ideas of what this means. A trace set of a process simply expresses whether or not a finite sequence of actions, a trace, is possible for the process. A tree expresses not only what paths are present but also how paths are subpaths, or restrictions, of others, thus keeping track of nondeterministic branching.

Example 4.1 Consider the two processes of Figure 3. Their unfoldings as trees (see Figure 8) are clearly different, but as trace sets they are both described by

$$\{\epsilon, a, ac, at\}.$$

As a result, trace sets cannot explain the possible deadlock encountered by a coffee drinker process $\bar{a}.\bar{c}.\emptyset$ interacting with the version on the left of the figure. \square

4.2 Presheaves

The data given by the tree representation, what paths are present and how they restrict to smaller paths, is precisely that caught in a presheaf over 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”.

To illustrate the idea, suppose that actions are drawn from the alphabet $A = \{a, c, t\}$, and consider processes whose computation paths have the shape of strings of actions, so members of A^* . The substring ordering \leq makes A^* a poset, and so a category with an arrow from s to s' precisely when $s \leq s'$, see Figure 14.

A presheaf over A^* is a functor from the opposite category $(A^*)^{\text{op}}$, where all the arrows are reversed, to the category of sets and functions Set . When thinking

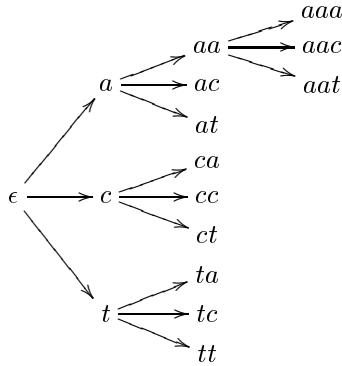


Figure 14: Part of the poset/category A^*

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 $s \leq s'$, the function $X(s, s') : X(s') \rightarrow X(s)$ tells how paths of shape s' restrict to subpaths of shape s . For example, a tree whose branches consists of strings in A^* is easily viewed as a presheaf X over A^* . The set $X(s)$ consists of all branches of shape s , while the function $X(s, s') : X(s') \rightarrow X(s)$ restricts a branch of shape s' to its sub-branch of shape s .

Example 4.2 Let's write $\mathbf{0}$ for the empty set, $\mathbf{1}$ for $\{\mathbf{0}\}$, and $\mathbf{2}$ for $\{\mathbf{0}, \mathbf{1}\}$. The vending machines on the left of Figure 3 is represented by the presheaf

$$X(s) = \begin{cases} \mathbf{2} & \text{if } s = a \\ \mathbf{1} & \text{if } s \in \{\epsilon, ac, at\} \\ \mathbf{0} & \text{otherwise} \end{cases}$$

where $X(a, ac)$ maps $\mathbf{0} \in X(ac)$ to $\mathbf{0} \in X(a)$ and $X(a, at)$ maps $\mathbf{0} \in X(at)$ to $\mathbf{1} \in X(a)$. On other morphisms, the action of X is uniquely determined.

Similarly, the presheaf

$$Y(s) = \begin{cases} \mathbf{1} & \text{if } s \in \{\epsilon, a, ac, at\} \\ \mathbf{0} & \text{otherwise} \end{cases}$$

over A^* represents the vending machine on the right. \square

A note on presheaves like Y : Suppose that we replace the category of sets used in the definition of presheaves by the subcategory $\mathbf{2}$ with the inclusion $\mathbf{0} \hookrightarrow \mathbf{1}$ as the only non-identity arrow. A functor Y from $(A^*)^{\text{op}}$ to $\mathbf{2}$

is the same as a monotonic function from the reverse order $(A^*)^{\text{op}}$ to the order $\mathbf{2}$, so that if $s \leq s'$ then $Y(s') \leq Y(s)$. When thinking of Y as representing a process, $Y(s) = \mathbf{1}$ means that the process can perform a path of shape s while $Y(s) = \mathbf{0}$ means that it can't. If $Y(s') = \mathbf{1}$ and $s \leq s'$, then $Y(s) = \mathbf{1}$. In other words, the functor Y 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.

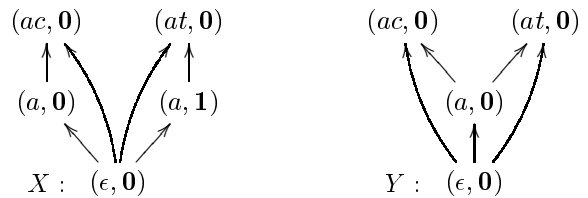
In what follows, we want to broaden computation paths to have more general shapes than sequences of atomic actions, to allow individual actions to have a more complicated structure as they have in HOPLA. So we will consider presheaves over an arbitrary small category \mathbb{P} , that is, functors $X : \mathbb{P}^{\text{op}} \rightarrow \mathbf{Set}$. The category \mathbb{P} is to be thought of as consisting of computation-path shapes where a map $e : p \rightarrow p'$ expresses how the path p is extended to the path p' .

We'll investigate what elementary category theory has to say about this data, thereby explaining a variety of process concepts in terms of "abstract nonsense".

4.3 Category of elements

Definition 4.3 Consider a presheaf X over \mathbb{P} . The *category of elements* of X , written $\mathcal{E}(X)$, has as objects pairs (p, x) with $p \in \mathbb{P}$ and $x \in X(p)$ and morphisms of the form $e : (p, x) \rightarrow (p', x')$ where $e : p \rightarrow p'$ is an arrow of \mathbb{P} such that $(Xe)x' = x$.

Example 4.4 The categories of elements of the presheaves X and Y of Example 4.2 look as follows (identity arrows are omitted):



Notice that the arrows of these categories may be seen as the reflexive, transitive closure of the transition relations of the trees of Figure 8. \square

So the category of elements can be understood intuitively as the tree or transition system of a process represented by a presheaf. But beware that the system may

not have an initial state as it does for X and Y of the example; they are both *rooted* according to

Definition 4.5 A presheaf X over \mathbb{P} is *rooted* when \mathbb{P} has an initial element, \perp , and $X(\perp)$ is a singleton.

A rooted presheaf over A^* corresponds directly to a tree whose branches are in A^* .

In general, a presheaf X over A^* determines a set of such trees, the set of roots given by $X(\epsilon)$. For example, we may consider the diagram above as *one* presheaf Z over A^* , in which $Z(s)$ is the disjoint union of $X(s)$ and $Y(s)$ for s in A^* . The corresponding transition system has two starting states, contrary to the standard definition of transition systems—and so it may seem reasonable to restrict attention to rooted presheaves. In terms of category theory, that would mean working with *sheaves* instead of presheaves. Alternatively, we can simply view X and Y as presheaves over A^+ ; such presheaves are easily seen to be in bijective correspondence with rooted presheaves over A^* .

4.4 Natural transformations

The presheaves over a small category \mathbb{P} are the objects of the functor category $\mathbb{P} = [\mathbb{P}^{\text{op}}, \mathbf{Set}]$, with arrows being natural transformations. Spelled out, a natural transformation $\alpha : X \rightarrow Y$ between presheaves X and Y over \mathbb{P} is a family $(\alpha_p)_{p \in \mathbb{P}}$ of functions

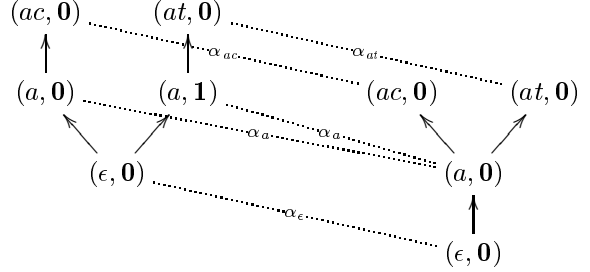
$$\alpha_p : X(p) \rightarrow Y(p),$$

satisfying that for any arrow $e : p \rightarrow p'$ of \mathbb{P} , the naturality square

$$\begin{array}{ccc} p' & & X(p') \xrightarrow{\alpha_{p'}} Y(p') \\ e \uparrow & & \downarrow X(e) \quad \downarrow Y(e) \\ p & & X(p) \xrightarrow{\alpha_p} Y(p) \end{array}$$

commutes. It is perhaps not entirely obvious what process concept this amounts to, so we replay it in terms of categories of elements. Since each α_p is a function $X(p) \rightarrow Y(p)$, α induces a map $\mathcal{E}(\alpha)$ between the objects of $\mathcal{E}(X)$ and $\mathcal{E}(Y)$, sending (p, x) to $(p, \alpha_p x)$. Now, naturality of α is simply the same as functoriality of $\mathcal{E}(\alpha)$: that it sends morphisms $(p, x) \xrightarrow{e} (p', x')$ of $\mathcal{E}(X)$ to morphisms $(p, \alpha_p x) \xrightarrow{e} (p', \alpha_{p'} x')$ of $\mathcal{E}(Y)$. So it seems that a natural transformation from X to Y shows how Y may “simulate” X .

Example 4.6 We can define a natural transformation $\alpha : X \rightarrow Y$ between the presheaves of Example 4.2:



In the diagram $(p, x) \xrightarrow{\alpha_p} (p, y)$ means that $\alpha_p x = y$, or equivalently that $\mathcal{E}(\alpha)$ maps (p, x) to (p, y) . \square

Since α preserves transitions in the categories of elements, we can think of it as a *simulation relation* on transition systems, defined analogously to item (i) of Definition 2.2. Moreover, its graph is a function; such relations are called *functional simulations*. But to obtain a categorical definition of bisimulation, we need a way of *reflecting* as well as preserving behaviour.

4.5 Open maps

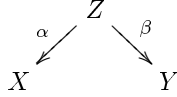
From a process viewpoint, it would be reasonable to make the extra requirement on a natural transformation $\alpha : X \rightarrow Y$ that whenever we have the situation on the left below, we may “complete the square” as on the right:

$$\begin{array}{ccc} (p', y') & & (p', x') \xrightarrow{\alpha_{p'}} (p', y') \\ e \uparrow & & e \uparrow \quad \uparrow e \\ (p, x) \xrightarrow{\alpha_p} (p, y) & & (p, x) \xrightarrow{\alpha_p} (p, y) \end{array}$$

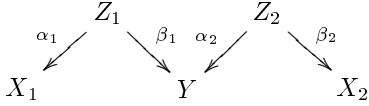
This can be formulated as the requirement that each naturality square of α (see opposite column), is a *quasi-pullback*, so that whenever we have $x \in X(p)$ and $y' \in Y(p')$ with $\alpha_p x = (Y e) y'$, then there exists an $x' \in X(p')$ such that $(X e) x' = x$ and $\alpha_{p'} x' = y'$. In this case, we call α an *open map* of presheaves, following Joyal and Moerdijk. In [15] their notion of open map in a topos is defined axiomatically, as being a class \mathcal{O} of morphisms satisfying, among other things

- (A1) any iso belongs to \mathcal{O}
- (A2) \mathcal{O} is closed under composition
- (A3) \mathcal{O} is stable under pullbacks

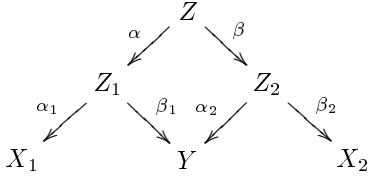
Open maps between rooted presheaves, so trees, over A^* are functional bisimulations. Since bisimilarity is an equivalence relation, the trees X and Y over A^* are bisimilar iff there exists a *span* of open maps between rooted presheaves:



More generally, the axioms of open maps above make sure that “being related by a span of open maps” is an equivalence relation: Reflexivity is obtained using (A1) and the trivial span $X \leftarrow X \rightarrow X$, while symmetry is already implicit in the notion of a span. As for transitivity, consider the two spans



Since $\widehat{\mathbb{P}}$ has all limits, we can adjoin a pullback,



Now, since β_1 and α_2 are open, so are α and β by (A3). Using (A2), $\alpha_1 \circ \alpha$ and $\beta_2 \circ \beta$ are open, so that X_1, X_2 are now related by a span of open maps as wanted.

We’ve been cheating a little bit here, since in general, the intuition that open maps are functional bisimulations is incorrect. There is a trivial span relating any pair of presheaves X, Y since $\widehat{\mathbb{P}}$ has an initial object (the empty presheaf, with empty contribution at each $p \in \mathbb{P}$). Again, we could take this as an indication to work only with rooted presheaves, or we could look for a requirement on the spans. As for the latter, demanding that the legs of the span are *epi* seems reasonable: open maps between rooted presheaves over A^* are automatically epi, and viewing trees over A^* as presheaves over A^+ , this extra requirement is exactly what is needed to show that bisimilarity as defined by open maps coincide with bisimilarity as defined by Park and Milner. To avoid restricting the class of presheaves, we take

Definition 4.7 Presheaves X, Y over \mathbb{P} are *open-map bisimilar* if they are related by a span of epi open maps.

4.6 Yoneda

To each presheaf category $\widehat{\mathbb{P}}$ is associated a canonical functor $\mathbb{P} \rightarrow \widehat{\mathbb{P}}$, saying how to view paths as processes:

Definition 4.8 Let \mathbb{P} be a small category. The *Yoneda functor* $y_{\mathbb{P}} : \mathbb{P} \rightarrow \widehat{\mathbb{P}}$ maps $p \in \mathbb{P}$ to the presheaf $\mathbb{P}(-, p)$ and maps $e : p \rightarrow p'$ of \mathbb{P} to the natural transformation $\mathbb{P}(-, e)$ obtained by postcomposition with e .

Example 4.9 The presheaf $A^*(-, s)$ is given by

$$A^*(-, s)(s') = A^*(s', s) = \begin{cases} \mathbf{1} & \text{if } s' \leq s \\ \mathbf{0} & \text{otherwise} \end{cases}$$

So $A^*(-, s)$ is a single branch with shape s , whereas $A^*(-, s \leq s')$ shows how it is included in a longer branch, with shape s' . \square

Intuitively, $y_{\mathbb{P}}$ maps a computation path shape p to a process that may do a single computation of shape p , and a path extension $e : p \rightarrow p'$ to a simulation of this computation by a longer one.

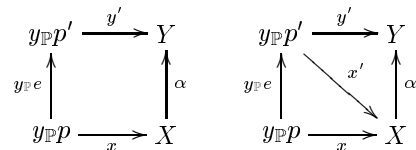
Following this intuition, a natural transformation $y_{\mathbb{P}}p \rightarrow X$ shows how the process X may simulate a process capable of performing just the path p . But then the set $X(p)$ of computation paths of X of shape p should be the same as the set of such natural transformations. This is the content of

Lemma 4.10 (Yoneda) For any $p \in \mathbb{P}$ and $X \in \widehat{\mathbb{P}}$ there is a bijection $\widehat{\mathbb{P}}(y_{\mathbb{P}}p, X) \cong X(p)$.

Example 4.11 For the presheaf X of Example 4.2, there are two natural transformations $A^*(-, a) \rightarrow X$, corresponding to the two elements $\mathbf{0}$ and $\mathbf{1}$ of $X(a)$. \square

Justified by the Yoneda lemma, if $(p, x) \in \mathcal{E}(X)$ we write x also for the corresponding natural transformation $y_{\mathbb{P}}p \rightarrow X$. This gives the following alternative characterisation of open maps:

Proposition 4.12 A map $\alpha : X \rightarrow Y$ in $\widehat{\mathbb{P}}$ is open iff for every arrow $e : p \rightarrow p'$ of \mathbb{P} , a commuting square (on the left below) can be split into two commuting triangles (on the right):



4.7 Completeness

A presheaf category has all limits and colimits, given pointwise from the limits and colimits in **Set**. In particular, a presheaf category has all sums (coproducts) of presheaves; the sum $\sum_{i \in I} X_i$ of presheaves X_i over \mathbb{P} has a contribution $\sum_{i \in I} X_i(p)$, the disjoint union of sets, at $p \in \mathbb{P}$. 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.

Example 4.13 Consider once again the two processes of Figure 3. If we view them as presheaves over A^+ , their nondeterministic sum, according to the above, corresponds to their sum as CCS terms. On the other hand, if we form their sum as rooted presheaves over A^* , we get the presheaf Z described in Section 4.3, which is not rooted, and cannot be expressed in CCS.

Similarly, the presheaf $\emptyset \in A^+$ corresponds to the inactive CCS process, also written \emptyset , while this is not so for $\emptyset \in A^*$ which intuitively cannot even do the empty computation path. \square

As promised in Section 4.2, we can exhibit a process as a colimit of its paths:

Proposition 4.14 (Density formula) For any $X \in \widehat{\mathbb{P}}$ we have $X \cong \int^{(p,x) \in \mathcal{E}(X)} y_p p$.

The role of the colimit is to “glue together” the paths of X as dictated by the category of elements of X .

At the start of Section 4.6, we said that the Yoneda functor is “canonical”. We now make this precise: The functor $y_p : \mathbb{P} \rightarrow \widehat{\mathbb{P}}$ satisfies the universal property that for any functor $F : \mathbb{P} \rightarrow \mathcal{C}$, where \mathcal{C} is a category with all colimits, there is a colimit-preserving functor $G : \widehat{\mathbb{P}} \rightarrow \mathcal{C}$, determined to within isomorphism, such that $F \cong G \circ y_p$ —see e.g. [19], page 43:

$$\begin{array}{ccc} \mathbb{P} & \xrightarrow{y_p} & \widehat{\mathbb{P}} \\ & \searrow F & \downarrow G \\ & & \mathcal{C} \end{array}$$

The proof uses the density formula. For $X \in \widehat{\mathbb{P}}$, we take $G(X)$ to be the colimit $\int^{(p,x) \in \mathcal{E}(X)} Fp$ in \mathcal{C} .

By taking \mathcal{C} to be a presheaf category $\widehat{\mathbb{Q}}$, this tells us that colimit-preserving functors between presheaf categories may be useful. We’ll now consider them as candidates for interpreting operations on processes.

5 Linearity

In distributed computation it can be hard or impossible for a process to copy a process, while it is generally easy for a process to ignore another process. For this reason an operation on processes associated with distributed computation often has the following property:

A computation path of the process arising from the application of the operation to an input process has resulted from a single (possibly empty) computation path of the input process.

As we will see, this property expresses that the operation is a kind of *linear map*, in that it is determined by its action on single, possibly empty, computation paths.

Example 5.1 The basic operations of CCS are all linear in the above sense. For example, let t be a CCS process, and consider the operations of prefixing t with an action a , composing t with another process u , and restricting t by label set S . A non-empty computation path of the process $a.t$ either has the shape a , involving the empty computation path of t , or is of the shape as , where s is the shape of a single computation path of t . Likewise, a computation path of the parallel composition $t|u$ may be obtained solely from u , so by letting t perform the empty computation path, while non-empty computation paths of $t \setminus S$ involve a single, non-empty computation path of t . \square

Consider again the diagram ending the preceding section. By the universal property, colimit-preserving functors $\widehat{\mathbb{P}} \rightarrow \widehat{\mathbb{Q}}$ are determined (up to isomorphism) by functors $\mathbb{P} \rightarrow \widehat{\mathbb{Q}}$, so by their actions on paths! This seems to indicate that colimit-preserving functors fit the intuition above. Let us see how to make this precise.

5.1 A model of linear logic

Define the category **Lin** to consist of small categories $\mathbb{P}, \mathbb{Q}, \dots$ with maps $G : \mathbb{P} \rightarrow \mathbb{Q}$ the colimit-preserving functors $\widehat{\mathbb{P}} \rightarrow \widehat{\mathbb{Q}}$. By the universal property, colimit-preserving functors $G : \widehat{\mathbb{P}} \rightarrow \widehat{\mathbb{Q}}$ correspond to within isomorphism to functors $F : \mathbb{P} \rightarrow \widehat{\mathbb{Q}}$, and such functors are in bijective correspondence with *profunctors* $\overline{F} : \mathbb{P} \dashrightarrow \mathbb{Q}$. Recall that the category of profunctors from \mathbb{P} to \mathbb{Q} , written **Prof**(\mathbb{P}, \mathbb{Q}), is the functor category $[\mathbb{P} \times \mathbb{Q}^{\text{op}}, \mathbf{Set}]$, which clearly equals the category

of presheaves $\widehat{\mathbb{P}^{\text{op}} \times \mathbb{Q}}$, and is isomorphic to the functor category $[\mathbb{P}, \widehat{\mathbb{Q}}]$. We thus have the chain of equivalences:

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

The more symmetric, relational presentation via pro-functors exposes an involution central in understanding \mathbf{Lin} as a categorical model of classical linear logic. The involution of linear logic, \mathbb{P}^\perp , on an object \mathbb{P} , is given by \mathbb{P}^{op} ; clearly presheaves over $\mathbb{P}^{\text{op}} \times \mathbb{Q}$ correspond to presheaves over $(\mathbb{Q}^{\text{op}})^{\text{op}} \times \mathbb{P}^{\text{op}}$, showing how maps $G : \mathbb{P} \rightarrow \mathbb{Q}$ correspond to maps $G^\perp : \mathbb{Q}^{\text{op}} \rightarrow \mathbb{P}^{\text{op}}$ in \mathbf{Lin} .

The model is somewhat degenerate: The “par” $\mathbb{P} \wp \mathbb{Q}$ and tensor product $\mathbb{P} \otimes \mathbb{Q}$ of \mathbb{P} and \mathbb{Q} are both given by the product of small categories $\mathbb{P} \times \mathbb{Q}$ (so the function space from \mathbb{P} to \mathbb{Q} is given by $\mathbb{P}^{\text{op}} \times \mathbb{Q}$). Likewise, on objects \mathbb{P} and \mathbb{Q} , products $\mathbb{P} \& \mathbb{Q}$ and coproducts $\mathbb{P} \oplus \mathbb{Q}$ are both given by $\mathbb{P} + \mathbb{Q}$, the disjoint juxtaposition of \mathbb{P} and \mathbb{Q} . The neutral elements are the terminal category $\mathbb{1}$ with one object and one arrow, and the initial category $\mathbb{0}$ with no objects, respectively.

As for the exponential $!$ of linear logic, there is room for choice—more about this later.

5.2 A linear process language?

Consider interpreting a process language in the category \mathbf{Lin} . The rich type discipline seems promising, and furthermore, we have a congruence result “for free” in that

Proposition 5.2 [5] *Let $G : \widehat{\mathbb{P}} \rightarrow \widehat{\mathbb{Q}}$ be any colimit-preserving functor between presheaf categories. Then G preserves open maps and open-map bisimulation.*

So any process operation interpreted as a map of \mathbf{Lin} will automatically respect open-map bisimulation.

Unfortunately, because the colimit of the empty diagram is \emptyset , the maps of \mathbf{Lin} , unlike many process operations, always send the inactive process to the inactive process! In particular, we cannot interpret prefixing or parallel composition a la CCS in \mathbf{Lin} . So colimit-preserving maps do not quite fit the intuition from above—they never ignore their arguments.

We could extend to maps from $!\mathbb{P}$ to \mathbb{Q} , for objects \mathbb{P} and \mathbb{Q} in \mathbf{Lin} , but by the properties of the exponential, this would allow arbitrary copying of the argument process. At least for CCS, we only need maps to ignore their arguments and this can be got much more cheaply, by moving to a model of *affine-linear* logic.

6 Lifting

To obtain a model of affine-linear logic, we’ll consider the operation of *lifting*, which intuitively extends path categories with the empty path and presheaves with a single computation of this shape.

6.1 Rooted presheaves

We’ll write \mathbb{P}_\perp for \mathbb{P} with a new initial object \perp freely adjoined. Spelled out, \mathbb{P}_\perp has objects $p \in \mathbb{P}$ together with \perp , and arrows given by those of \mathbb{P} together with a unique arrow $!_p : \perp \rightarrow p$ for all $p \in \mathbb{P}$.

Example 6.1 It is easy to see that $A_\perp^+ \cong A^*$. \square

Given a presheaf $X \in \mathbb{P}$, we’ll write $\lfloor X \rfloor$ for the presheaf over \mathbb{P}_\perp defined by taking $\lfloor X \rfloor(p)$ to be a copy of $X(p)$ for any $p \in \mathbb{P}$, and $\lfloor X \rfloor(\perp) = \mathbb{1}$. On arrows of \mathbb{P} , $\lfloor X \rfloor$ acts like X , while $\lfloor X \rfloor(!_p)$ is forced to be the constantly $\mathbb{0}$ map, any $p \in \mathbb{P}$.

In terms of categories of elements, $\mathcal{E}[\lfloor X \rfloor]$ has the same objects as $\mathcal{E}(X)$ in addition to $(\perp, \mathbb{0})$, from which there is a unique transition to any other object. So $\mathcal{E}[\lfloor X \rfloor]$ is $\mathcal{E}(X)$ with a freely adjoined initial object.

Example 6.2 For any presheaf X over A^+ , the presheaf $\lfloor X \rfloor$ over A^* is the tree obtained by adding a root to X . \square

In fact, lifting extends to a functor

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

A natural transformation from X to Y in $\widehat{\mathbb{P}}$ is sent to its obvious extension from $\lfloor X \rfloor$ to $\lfloor Y \rfloor$ in $\widehat{\mathbb{P}_\perp}$. Since the \perp -component of any natural transformation between lifted presheaves is uniquely determined, the image of lifting is the full subcategory of $\widehat{\mathbb{P}_\perp}$ of rooted presheaves.

The following result will be important later:

Proposition 6.3 *Any presheaf X in $\widehat{\mathbb{P}_\perp}$ has a decomposition as a sum of rooted presheaves*

$$X \cong \Sigma_{i \in X(\perp)} \lfloor X_i \rfloor,$$

where, for $i \in X(\perp)$, the presheaf X_i in $\widehat{\mathbb{P}}$ is, to within isomorphism, given as

$$X_i(p) = \{x \in X(p) \mid (X!_p)x = i\}.$$

Example 6.4 For a presheaf X over A^* , the rooted component decomposition exhibits the forest X as a sum of trees. \square

6.2 Strict Yoneda

It turns out that \mathbb{P}_\perp relates to $\widehat{\mathbb{P}}$ in much the same way as \mathbb{P} relates to $\widehat{\mathbb{P}}$.

Definition 6.5 The *strict Yoneda functor* $j_{\mathbb{P}} : \mathbb{P}_\perp \rightarrow \widehat{\mathbb{P}}$, sends \perp to \emptyset and elsewhere acts like $y_{\mathbb{P}}$. Formally, it maps $p \in \mathbb{P}_\perp$ to $\mathbb{P}_\perp(i-, p)$ where i is the inclusion $\mathbb{P} \hookrightarrow \mathbb{P}_\perp$.

Intuitively, $j_{\mathbb{P}}$ maps a possibly empty computation path shape to the process that may do a single computation of this shape—where it is understood that the empty computation path gives rise to the inactive process.

Example 6.6 j_{A^+} maps a string $a_1 a_2 \cdots a_k$ in A^* to the presheaf over A^+ represented by the CCS term $a_1 . a_2 . \dots . a_k . \emptyset$. \square

Since natural transformations $\alpha : j_{\mathbb{P}} p \rightarrow X$ represent the ways X may simulate $j_{\mathbb{P}} p$, and since there is just one way to simulate the inactive process, we expect

Lemma 6.7 (Strict Yoneda) For any $p \in \mathbb{P}_\perp$ and $X \in \widehat{\mathbb{P}}$ there is a bijection $\widehat{\mathbb{P}}(j_{\mathbb{P}} p, X) \cong [X](p)$.

Again, if $(p, x) \in \mathcal{E}[X]$ we are justified to write x also for the corresponding natural transformation $j_{\mathbb{P}} p \rightarrow X$.

Proposition 6.8 A map $\alpha : X \rightarrow Y$ in $\widehat{\mathbb{P}}$ is open and epi iff for every arrow $e : p \rightarrow p'$ of \mathbb{P}_\perp , a commuting square (on the left below) can be split into two commuting triangles (on the right):

$$\begin{array}{ccc} j_{\mathbb{P}} p' & \xrightarrow{y'} & Y \\ j_{\mathbb{P}} e \uparrow & & \uparrow \alpha \\ j_{\mathbb{P}} p & \xrightarrow{x} & X \end{array} \quad \begin{array}{ccc} j_{\mathbb{P}} p' & \xrightarrow{y'} & Y \\ & \searrow x' & \uparrow \alpha \\ j_{\mathbb{P}} p & \xrightarrow{x} & X \end{array}$$

We may formulate this result as saying that presheaves $X, Y \in \widehat{\mathbb{P}}$ are open-map bisimilar iff they are related by a span of “ $j_{\mathbb{P}}$ -open maps”; Proposition 4.12 shows how open maps may be seen as “ $y_{\mathbb{P}}$ -open maps”. Indeed, there may well be other notions of paths and other inclusions of those paths into $\widehat{\mathbb{P}}$ that give sensible definitions of open-map bisimulation.

Example 6.9 The j_{A^+} -open maps are exactly the functional bisimulations in the sense of Park/Milner. \square

6.3 Connected colimits

The density formula Proposition 4.14 shows how, via Yoneda, any presheaf X over \mathbb{P} can be regarded as a colimit of its “nonempty paths”. Using strict Yoneda instead, we may view X as a *connected colimit* of all its paths, including the empty one. A connected colimit is a colimit of a nonempty, connected diagram.

Proposition 6.10 (Strict density formula) For $X \in \widehat{\mathbb{P}}$ we have $X \cong \int^{(p,x) \in \mathcal{E}[X]} j_{\mathbb{P}} p$.

Here, $\mathcal{E}[X]$ is a nonempty, connected category, because it has an initial element, (\perp, \emptyset) .

We therefore have that the presheaf category $\widehat{\mathbb{P}}$ with $j_{\mathbb{P}}$ is a free connected-colimit completion of \mathbb{P}_\perp . Together they satisfy the universal property that for any functor $F : \mathbb{P}_\perp \rightarrow \mathcal{C}$, where \mathcal{C} is a category with all connected colimits, there is a connected-colimit preserving functor $G : \widehat{\mathbb{P}} \rightarrow \mathcal{C}$, determined to within isomorphism, such that $F \cong G \circ j_{\mathbb{P}}$:

$$\begin{array}{ccc} \mathbb{P}_\perp & \xrightarrow{j_{\mathbb{P}}} & \widehat{\mathbb{P}} \\ & \searrow F & \downarrow G \\ & & \mathcal{C} \end{array}$$

We’ll sometimes write F^\dagger for G . The universal property suggests the importance of connected-colimit preserving functors. Moreover, it says that these functors match the intuition of Section 5: they are determined by their action on single, possibly empty, computation paths.

6.4 A model of affine-linear logic

Define **Aff** to be the category consisting of small categories $\mathbb{P}, \mathbb{Q}, \dots$, with maps $G : \mathbb{P} \rightarrow \mathbb{Q}$ the connected-colimit preserving functors $\widehat{\mathbb{P}} \rightarrow \widehat{\mathbb{Q}}$. It has **Lin** in which the maps preserve arbitrary colimits as a subcategory, one which shares the same objects. We can easily characterise those maps in **Aff** which are in **Lin**:

Proposition 6.11 Suppose $G : \widehat{\mathbb{P}} \rightarrow \widehat{\mathbb{Q}}$ preserves connected colimits. The following properties are equivalent:

- (i) G preserves all colimits;
- (ii) G preserves all coproducts (sums);
- (iii) G is strict, i.e., $G(\emptyset) = \emptyset$.

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

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

and consequently the equivalence

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

The equivalence is part of an adjunction (the inclusion $\mathbf{Lin} \hookrightarrow \mathbf{Aff}$ has $(-)_\perp$ defined below as left adjoint; the unit has components $[-] : \mathbb{P} \rightarrow \widehat{\mathbb{P}}$ in \mathbf{Aff}) between \mathbf{Aff} and \mathbf{Lin} regarded as 2-categories, in which the 2-cells are natural transformations. We can easily extend lifting to a 2-functor

$$(-)_\perp : \mathbf{Aff} \rightarrow \mathbf{Lin};$$

for $G : \mathbb{P} \rightarrow \mathbb{Q}$ in \mathbf{Aff} , the functor $G_\perp : \mathbb{P}_\perp \rightarrow \mathbb{Q}_\perp$ in \mathbf{Lin} takes $X \in \widehat{\mathbb{P}_\perp}$ with decomposition $\sum_{i \in X(\perp)} [X_i]$ to

$$G_\perp(X) = \sum_{i \in X(\perp)} [G(X_i)].$$

Lifting restricts to a 2-comonad on \mathbf{Lin} with \mathbf{Aff} as its coKleisli category. The comonad $(-)_\perp$ has turned the model of linear logic \mathbf{Lin} into a model \mathbf{Aff} of affine linear logic (that is, 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, see [13]).

6.5 An affine-linear process language

The constructions in \mathbf{Aff} presented below form the basis of a denotational semantics of an affine-linear process language. The types and open terms of that language will be interpreted, respectively, as objects and arrows of \mathbf{Aff} . Actually, only the full subcategory of *path orders*, small partial order categories, is needed, and we'll simplify the discussion accordingly, treating denotations of types as though they were just partial orders.

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

Proposition 6.12 [5] *Let $G : \widehat{\mathbb{P}} \rightarrow \widehat{\mathbb{Q}}$ be any connected-colimit preserving functor between presheaf categories. Then G preserves open maps and open-map bisimulation.*

Hence, open-map bisimulation is already a congruence for our language.

6.6 Constructions in \mathbf{Aff}

Prefixesum The category \mathbf{Aff} does not have coproducts (since all constant functors are maps of \mathbf{Aff} there can be no initial object, so empty coproduct). However, we can build a useful sum in \mathbf{Aff} with the help of the coproduct of \mathbf{Lin} and lifting. Let \mathbb{P}_α , for $\alpha \in A$, be a family of path orders. As their prefixed sum, $\sum_{\alpha \in A} \alpha.\mathbb{P}_\alpha$, we take the disjoint union of the path orders $\mathbb{P}_{\alpha_\perp}$, over the underlying set $\bigcup_{\alpha \in A} \{\alpha\} \times \mathbb{P}_{\alpha_\perp}$; the latter path order forms a coproduct in \mathbf{Lin} with the obvious injections $in_\beta : \mathbb{P}_{\beta_\perp} \rightarrow \sum_{\alpha \in A} \alpha.\mathbb{P}_\alpha$, for $\beta \in A$. The injections $\beta.(-) : \mathbb{P}_\beta \rightarrow \sum_{\alpha \in A} \alpha.\mathbb{P}_\alpha$ in \mathbf{Aff} , for $\beta \in A$, are defined to be the compositions $\beta.(-) = in_\beta [-]$. Finite prefixed sums are written $\alpha_1.\mathbb{P}_1 + \dots + \alpha_k.\mathbb{P}_k$.

This construction is not a coproduct in \mathbf{Aff} . However, it does satisfy a weaker property analogous to the universal property of a coproduct. Suppose $F_\alpha : \mathbb{P}_\alpha \rightarrow \mathbb{Q}$ are maps in \mathbf{Aff} for all $\alpha \in A$. Then, there is a mediating map $F : \sum_{\alpha \in A} \alpha.\mathbb{P}_\alpha \rightarrow \mathbb{Q}$ in \mathbf{Lin} determined to within isomorphism such that $F \circ \alpha.(-) = F_\alpha$ for all $\alpha \in A$.

Suppose that the family of maps $F_\alpha : \mathbb{P}_\alpha \rightarrow \mathbb{Q}$, 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 $H : \mathbb{P}_\beta \rightarrow \mathbb{Q}$. Write $H_{\otimes\beta} : \sum_{\alpha \in A} \alpha.\mathbb{P}_\alpha \rightarrow \mathbb{Q}$ for a choice of mediating map in \mathbf{Lin} .

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

$$[u > \beta.x \Rightarrow t]$$

for $H_{\otimes\beta}(u)$. As in HOPLA, this construction “tests” or matches u against the pattern $\beta.x$ and passes the results of successful matches for x on to t ; the possibly multiple results of successful matches are then summed together—cf. Lemma 7.7 below.

Product 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 $\pi_1 : \mathbb{P} \& \mathbb{Q} \rightarrow \mathbb{P}$ and $\pi_2 : \mathbb{P} \& \mathbb{Q} \rightarrow \mathbb{Q}$. 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 $F_i : \mathbb{P} \rightarrow \mathbb{P}_i$, for $i \in I$, can be tupled together to form a unique map $\langle F_i \rangle_{i \in I} : \mathbb{P} \rightarrow \&_{i \in I} \mathbb{P}_i$ 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 $\mathbb{0}$ and as the terminal object is associated with unique maps $!_{\mathbb{P}} : \mathbb{P} \rightarrow \mathbb{0}$, constantly \emptyset , for any path order \mathbb{P} .

$t, u ::=$	$x \mid \text{rec } x.t \mid \sum_{i \in I} t_i \mid$	(Variables, recursion and sum)
	$\alpha.t \mid [u > \alpha.x \Rightarrow t] \mid$	(Prefixing and prefix match)
	$(t, u) \mid [u > (x, -) \Rightarrow t] \mid [u > (-, x) \Rightarrow t] \mid$	(Pairing and product match)
	$t \otimes u \mid [u > x \otimes y \Rightarrow t] \mid$	(Tensor and tensor match)
	$\lambda x.t \mid t u$	(Abstraction and application)

Figure 15: Raw syntax

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. Intuitively, a computation path in $\mathbb{P} \otimes \mathbb{Q}$ is a pair $p \otimes q$ of computation paths of \mathbb{P} and \mathbb{Q} , respectively, one of which may be empty.

Let $F : \mathbb{P} \rightarrow \mathbb{P}'$ and $G : \mathbb{Q} \rightarrow \mathbb{Q}'$. We define

$$F \otimes G : \mathbb{P} \otimes \mathbb{Q} \rightarrow \mathbb{P}' \otimes \mathbb{Q}'$$

as the extension H^\dagger of a functor $H : (\mathbb{P} \otimes \mathbb{Q})_\perp \rightarrow \widehat{\mathbb{P}' \otimes \mathbb{Q}'}$. Notice that $(\mathbb{P} \otimes \mathbb{Q})_\perp \cong \mathbb{P}_\perp \times \mathbb{Q}_\perp$ and so we can define $H : \mathbb{P}_\perp \times \mathbb{Q}_\perp \rightarrow \widehat{\mathbb{P}' \otimes \mathbb{Q}'}$ by taking

$$(H(p, q))(p' \otimes q') = [F(j_{\mathbb{P}}p)](p') \times [G(j_{\mathbb{Q}}q)](q')$$

for $p \in \mathbb{P}_\perp, q \in \mathbb{Q}_\perp$ and $p' \otimes q' \in \mathbb{P}' \otimes \mathbb{Q}'$.

The unit for tensor is the empty path order \mathbb{O} . Objects $X \in \widehat{\mathbb{P}}$ correspond to maps $\tilde{X} : \mathbb{O} \rightarrow \mathbb{P}$ 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 $\tilde{X} \otimes \tilde{Y} : \mathbb{O} \rightarrow \mathbb{P} \otimes \mathbb{Q}$.

Function space The function space of path orders $\mathbb{P} \multimap \mathbb{Q}$ is given by the product of partial orders $(\mathbb{P}_\perp)^{\text{op}} \times \mathbb{Q}$.

We have the following chain of isomorphisms:

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

So that $\mathbb{P} \otimes \mathbb{Q} \multimap \mathbb{R} \cong \mathbb{P} \multimap (\mathbb{Q} \multimap \mathbb{R})$. Thus there is a 1-1 correspondence *curry* from maps $\mathbb{P} \otimes \mathbb{Q} \rightarrow \mathbb{R}$ to maps $\mathbb{P} \rightarrow (\mathbb{Q} \multimap \mathbb{R})$ in **Aff**; its inverse is called *uncurry*. We obtain linear application,

$$\text{app} : (\mathbb{P} \multimap \mathbb{Q}) \otimes \mathbb{P} \rightarrow \mathbb{Q},$$

as *uncurry*($1_{\mathbb{P} \multimap \mathbb{Q}}$).

We shall write $t u$ for the application of t of type $\mathbb{P} \multimap \mathbb{Q}$ to u of type \mathbb{P} . The ability to curry justifies the formation of terms $\lambda x.t$ of type $\mathbb{P} \multimap \mathbb{Q}$ by lambda abstraction where t of type \mathbb{Q} is a term with free variable x of type \mathbb{P} .

7 ALLAN

The type system of the affine-linear language, here called ALLAN for brevity, extends that of HOPLA with the tensor constructor:

$$\begin{aligned} \mathbb{P}, \mathbb{Q} ::= & \sum_{\alpha \in A} \alpha. \mathbb{P}_\alpha \mid \mathbb{P} \& \mathbb{Q} \mid \mathbb{P} \otimes \mathbb{Q} \mid \mathbb{P} \multimap \mathbb{Q} \mid \\ & P \mid \mu_j \vec{P}. \vec{P} \end{aligned}$$

The interpretation of closed type expressions as objects of **Aff** should be clear from Section 6.6 and the operations there form the basis of a syntax of terms, see Figure 15.

The raw syntax is subject to typing and linearity constraints. Let $\mathbb{P}_1, \dots, \mathbb{P}_k, \mathbb{Q}$ be closed expressions for path orders and let the variables x_1, \dots, x_k be distinct. A syntactic judgement

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

stands for a map

$$[[x_1 : \mathbb{P}_1, \dots, x_k : \mathbb{P}_k \vdash t : \mathbb{Q}]] : \mathbb{P}_1 \otimes \dots \otimes \mathbb{P}_k \rightarrow \mathbb{Q}$$

in **Aff**. We shall typically write Γ , or Δ , for an environment list $x_1 : \mathbb{P}_1, \dots, x_k : \mathbb{P}_k$ and most often abbreviate the denotation to $\mathbb{P}_1 \otimes \dots \otimes \mathbb{P}_k \xrightarrow{t} \mathbb{Q}$, or even $\Gamma \xrightarrow{t} \mathbb{Q}$. When the environment list is empty, the corresponding tensor product is the empty path order \mathbb{O} .

7.1 Substitution

Consider the expected isomorphism

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

and suppose that $t \equiv x \otimes x$ and $u \equiv a.\emptyset + b.\emptyset$. Now, one computation of $u \otimes u$ has shape $a \otimes b$ and this gives us a mismatch with the intuition that any computation path of the output should result from a single computation path of the input! The counterpart in the model is the absence of a suitable diagonal map from object

$$\begin{array}{c}
\frac{}{x : \mathbb{P} \vdash x : \mathbb{P}} \quad \frac{\Delta \vdash t : \mathbb{P}}{\Gamma, \Delta \vdash t : \mathbb{P}} \quad \frac{\Gamma, x : \mathbb{P}, y : \mathbb{Q}, \Delta \vdash t : \mathbb{R}}{\Gamma, y : \mathbb{Q}, x : \mathbb{P}, \Delta \vdash t : \mathbb{R}} \\
\\
\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 \text{rec } x.t : \mathbb{P}} \quad \frac{\Gamma \vdash t_i : \mathbb{P} \text{ for all } i \in I}{\Gamma \vdash \sum_{i \in I} t_i : \mathbb{P}} \\
\\
\frac{\Gamma \vdash t : \mathbb{P}_\beta \text{ where } \beta \in A}{\Gamma \vdash \beta.t : \sum_{\alpha \in A} \mathbb{P}_\alpha} \quad \frac{\Gamma, x : \mathbb{P}_\beta \vdash t : \mathbb{Q} \text{ where } \beta \in A \quad \Delta \vdash u : \sum_{\alpha \in A} \mathbb{P}_\alpha}{\Gamma, \Delta \vdash [u > \beta.x \Rightarrow t] : \mathbb{Q}} \\
\\
\frac{\Gamma \vdash t : \mathbb{P} \quad \Gamma \vdash u : \mathbb{Q}}{\Gamma \vdash (t, u) : \mathbb{P} \& \mathbb{Q}} \quad \frac{\Gamma, x : \mathbb{P} \vdash t : \mathbb{R} \quad \Delta \vdash u : \mathbb{P} \& \mathbb{Q}}{\Gamma, \Delta \vdash [u > (x, -) \Rightarrow t] : \mathbb{R}} \quad \frac{\Gamma, x : \mathbb{Q} \vdash t : \mathbb{R} \quad \Delta \vdash u : \mathbb{P} \& \mathbb{Q}}{\Gamma, \Delta \vdash [u > (-, x) \Rightarrow t] : \mathbb{R}} \\
\\
\frac{\Gamma \vdash t : \mathbb{P} \quad \Delta \vdash u : \mathbb{Q}}{\Gamma, \Delta \vdash t \otimes u : \mathbb{P} \otimes \mathbb{Q}} \quad \frac{\Gamma, x : \mathbb{P}, y : \mathbb{Q} \vdash t : \mathbb{R} \quad \Delta \vdash u : \mathbb{P} \otimes \mathbb{Q}}{\Gamma, \Delta \vdash [u > x \otimes y \Rightarrow t] : \mathbb{R}} \\
\\
\frac{\Gamma, x : \mathbb{P} \vdash t : \mathbb{Q}}{\Gamma \vdash \lambda x.t : \mathbb{P} \multimap \mathbb{Q}} \quad \frac{\Gamma \vdash t : \mathbb{P} \multimap \mathbb{Q} \quad \Delta \vdash u : \mathbb{P}}{\Gamma, \Delta \vdash t u : \mathbb{Q}} \\
\\
\frac{\Gamma \vdash t : \mathbb{P}_j[\mu \vec{P}. \vec{P} / \vec{P}]}{\Gamma \vdash t : \mu_j \vec{P}. \vec{P}} \quad \frac{\Gamma \vdash t : \mu_j \vec{P}. \vec{P}}{\Gamma \vdash t : \mathbb{P}_j[\mu \vec{P}. \vec{P} / \vec{P}]}
\end{array}$$

Figure 16: ALLAN term formation rules

\mathbb{P} to $\mathbb{P} \otimes \mathbb{P}$ to interpret $x \otimes x$. On the other hand, raw terms like $x + x$ or (x, x) are fine intuitively and may be interpreted so as to validate the isomorphism above. So an appropriate syntactic restriction is to disallow any variable to occur freely in on both sides of a \otimes , and accordingly, we will have term formation rules like that for tensor in Figure 16. Similar restrictions apply to application and match terms.

Now consider the expected isomorphism

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

With $t \equiv x \otimes y$, unfolding immediately yields a term with y occurring freely on both sides of a tensor. We therefore make the following definition (p ranges over patterns $\alpha.x, (x, -), (-, x), x \otimes y$):

Definition 7.1 Let v be a raw term. Say a set of variables V is *crossed* in v iff there are subterms of v of the form tensor $t \otimes u$, application $t u$, or match $[t > p \Rightarrow u]$, for which v has free occurrences of variables from V appearing in both t and u .

The rule for recursion in Figure 16 is restricted by requiring that $\{y, x\}$ is not crossed in t for all y in Γ .

These linearity constraints are strong enough to show the following syntactic results:

Proposition 7.2 Suppose $\Gamma, x : \mathbb{P} \vdash t : \mathbb{Q}$. The set $\{x\}$ is not crossed in t .

Lemma 7.3 (Well-formed substitution) Suppose

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

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

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

We end this section by discussing the semantic counterpart of “variables not being crossed”. Consider a term

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

with x and y not crossed in t . Intuitively, this means that in any computation of $t[u/x, v/y]$, only one of the arguments u and v may be used. In terms of **Aff**, the interpretation $F : \mathbb{P} \otimes \mathbb{Q} \rightarrow \mathbb{R}$ of t satisfies

$$F(X \otimes Y) \cong F(X \otimes \emptyset + \emptyset \otimes Y).$$

So F may be seen as essentially a map $\mathbb{P} \& \mathbb{Q} \rightarrow \mathbb{R}$ via precomposition with

$$\sigma : \mathbb{P} \& \mathbb{Q} \rightarrow \mathbb{P} \otimes \mathbb{Q},$$

$$\begin{array}{c}
\frac{}{\mathbb{P} \xrightarrow{1_{\mathbb{P}}} \mathbb{P}} \quad \frac{\Delta \xrightarrow{t} \mathbb{P}}{\Gamma \otimes \Delta \xrightarrow{1_{\Gamma} \otimes t} \mathbb{O} \otimes \mathbb{P} \cong \mathbb{P}} \quad \frac{\Gamma \otimes \mathbb{P} \otimes \mathbb{Q} \otimes \Delta \xrightarrow{t} \mathbb{R}}{\Gamma \otimes \mathbb{Q} \otimes \mathbb{P} \otimes \Delta \cong \Gamma \otimes \mathbb{P} \otimes \mathbb{Q} \otimes \Delta \xrightarrow{t} \mathbb{R}} \\
\frac{\Gamma \otimes \mathbb{P} \xrightarrow{t} \mathbb{P}}{\Gamma \xrightarrow{fix\ G} \mathbb{P}} \quad \frac{\Gamma \xrightarrow{t_i} \mathbb{P} \text{ for all } i \in I}{\Gamma \xrightarrow{\langle t_i \rangle_{i \in I}} \mathbf{\&}_{i \in I} \mathbb{P} \xrightarrow{\Sigma} \mathbb{P}} \\
\frac{\Gamma \xrightarrow{t} \mathbb{P}_{\beta}}{\Gamma \xrightarrow{t} \mathbb{P}_{\beta} \xrightarrow{\beta.(-)} \Sigma_{\alpha \in A} \alpha. \mathbb{P}_{\alpha}} \quad \frac{\Gamma \otimes \mathbb{P}_{\beta} \xrightarrow{t} \mathbb{Q} \quad \Delta \xrightarrow{u} \Sigma_{\alpha \in A} \alpha. \mathbb{P}_{\alpha}}{\Gamma \otimes \Delta \xrightarrow{1_{\Gamma} \otimes u} \Gamma \otimes \Sigma_{\alpha \in A} \alpha. \mathbb{P}_{\alpha} \xrightarrow{t_{\mathbb{O}\beta} \circ dist} \mathbb{Q}} \\
\frac{\Gamma \xrightarrow{t} \mathbb{P} \quad \Gamma \xrightarrow{u} \mathbb{Q}}{\Gamma \xrightarrow{\langle t, u \rangle} \mathbb{P} \& \mathbb{Q}} \quad \frac{\Gamma \otimes \mathbb{P} \xrightarrow{t} \mathbb{R} \quad \Delta \xrightarrow{u} \mathbb{P} \& \mathbb{Q}}{\Gamma \otimes \Delta \xrightarrow{1_{\Gamma} \otimes (\pi_1 \circ u)} \Gamma \otimes \mathbb{P} \xrightarrow{t} \mathbb{R}} \quad \frac{\Gamma \otimes \mathbb{Q} \xrightarrow{t} \mathbb{R} \quad \Delta \xrightarrow{u} \mathbb{P} \& \mathbb{Q}}{\Gamma \otimes \Delta \xrightarrow{1_{\Gamma} \otimes (\pi_2 \circ u)} \Gamma \otimes \mathbb{Q} \xrightarrow{t} \mathbb{R}} \\
\frac{\Gamma \xrightarrow{t} \mathbb{P} \quad \Delta \xrightarrow{u} \mathbb{Q}}{\Gamma \otimes \Delta \xrightarrow{t \otimes u} \mathbb{P} \otimes \mathbb{Q}} \quad \frac{\Gamma \otimes \mathbb{P} \otimes \mathbb{Q} \xrightarrow{t} \mathbb{R} \quad \Delta \xrightarrow{u} \mathbb{P} \otimes \mathbb{Q}}{\Gamma \otimes \Delta \xrightarrow{1_{\Gamma} \otimes u} \Gamma \otimes \mathbb{P} \otimes \mathbb{Q} \xrightarrow{t} \mathbb{R}} \\
\frac{\Gamma \otimes \mathbb{P} \xrightarrow{t} \mathbb{Q}}{\Gamma \xrightarrow{curry\ t} \mathbb{P} \multimap \mathbb{Q}} \quad \frac{\Gamma \xrightarrow{t} \mathbb{P} \multimap \mathbb{Q} \quad \Delta \xrightarrow{u} \mathbb{P}}{\Gamma \otimes \Delta \xrightarrow{t \otimes u} (\mathbb{P} \multimap \mathbb{Q}) \otimes \mathbb{P} \xrightarrow{app} \mathbb{Q}}
\end{array}$$

$\mu_j \vec{P}. \vec{P}$ and $\mathbb{P}_j[\mu \vec{P}. \vec{P} / \vec{P}]$ denote equal path orders, so the premise and conclusion are interpreted as the same map.

Figure 17: ALLAN interpretations

specified by $(X, Y) \mapsto X \otimes \mathbb{O} + \mathbb{O} \otimes Y$. The composition of such a map with the diagonal map of the product, *viz.*

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

takes X to $X \otimes \mathbb{O} + \mathbb{O} \otimes X$ and gives a weak form of diagonal map. Analogously, one can define general weak diagonal maps $\delta_{\mathbb{P}^k} : \mathbb{P} \rightarrow \mathbb{P} \otimes \cdots \otimes \mathbb{P}$ in \mathbf{Lin} from \mathbb{P} to k copies of \mathbb{P} tensored together. Weak diagonal maps allow the same argument to be used in several different, though incompatible, ways.

7.2 Denotational semantics

The term formation rules of Figure 16 are interpreted as constructors on morphisms, taking the morphisms denoted by the premises to that denoted by the conclusion (*cf.* [2]). The rules are shown in Figure 17, to be read on a par with those of Figure 16. Some comments are in order:

Recursion In the rule for recursion, for $F : \Gamma \rightarrow \mathbb{P}$ the map $G(F) : \Gamma \rightarrow \mathbb{P}$ is the composition

$$\Gamma \xrightarrow{\delta_{\Gamma}} \Gamma \otimes \Gamma \xrightarrow{1_{\Gamma} \otimes F} \Gamma \otimes \mathbb{P} \xrightarrow{t} \mathbb{P}.$$

Any operation $\mathbf{Aff}(\Gamma, \mathbb{P}) \rightarrow \mathbf{Aff}(\Gamma, \mathbb{P})$ which preserves connected colimits will have a fixpoint $fix\ G : \Gamma \rightarrow \mathbb{P}$, a map in \mathbf{Aff} . This is so because the category $\mathbf{Aff}(\Gamma, \mathbb{P})$, being equivalent to $(\Gamma_{\perp})^{\text{op}} \times \mathbb{P}$, has all colimits and in particular all ω -colimits.

Sum As already indicated in Section 4.7, each path order \mathbb{P} is associated with (nondeterministic) sum operations, a map $\Sigma : \mathbf{\&}_{i \in I} \mathbb{P} \rightarrow \mathbb{P}$ in \mathbf{Aff} taking a tuple $\langle X_i \rangle_{i \in I}$ to the sum (coproduct) $\Sigma_{i \in I} X_i$ in $\widehat{\mathbb{P}}$.

Prefix match Because prefixed sum is not a coproduct we do not have that tensor distributes over prefixed sum. However there is a map

$$dist : \Gamma \otimes \Sigma_{\alpha \in A} \alpha. \mathbb{P}_{\alpha} \rightarrow \Sigma_{\alpha \in A} \alpha. (\Gamma \otimes \mathbb{P}_{\alpha})$$

in \mathbf{Aff} , expressing a form of distributivity, given as the extension H^{\dagger} of the functor

$$H : \Gamma_{\perp} \times (\Sigma_{\alpha \in A} \alpha. \mathbb{P}_{\alpha})_{\perp} \rightarrow \Sigma_{\alpha \in A} \widehat{\alpha. (\Gamma \otimes \mathbb{P}_{\alpha})},$$

defined by taking $H(q, \perp) = \mathbb{O}$ and

$$H(q, (\alpha, p)) = y_{\Sigma_{\alpha \in A} \alpha. (\Gamma \otimes \mathbb{P}_{\alpha})}(\alpha, (q, p)).$$

Exploiting the naturality of the various operations used in the semantic definitions, we can prove a general substitution lemma.

Lemma 7.4 (Substitution) *Suppose*

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

and that the set of variables $\{x_1, \dots, x_k\}$ is not crossed in t . Suppose $\Delta \vdash u : \mathbb{P}$ where the variables of Γ and Δ are disjoint. Then, $\Gamma, \Delta \vdash t[u/x_1, \dots, u/x_k] : \mathbb{Q}$ and, (suppressing the types for brevity)

$$\llbracket t[u/x_1, \dots, u/x_k] \rrbracket \cong \llbracket t \rrbracket \circ (1_\Gamma \otimes (\delta_{\mathbb{P}^k} \circ \llbracket u \rrbracket)) .$$

Note that in the case where $k = 1$, the lemma specialises to $\llbracket t[u/x] \rrbracket \cong \llbracket t \rrbracket \circ (1_\Gamma \otimes \llbracket u \rrbracket)$. A particular consequence is that linear application amounts to substitution:

Lemma 7.5 *If $\Gamma \vdash (\lambda x.t) u : \mathbb{Q}$, then $\Gamma \vdash t[u/x] : \mathbb{Q}$ and $\llbracket \Gamma \vdash (\lambda x.t) u : \mathbb{Q} \rrbracket \cong \llbracket \Gamma \vdash t[u/x] : \mathbb{Q} \rrbracket$.*

Similarly, we have the expected result for recursion:

Lemma 7.6 *If $\Gamma \vdash \text{rec } x.t : \mathbb{P}$, then $\Gamma \vdash t[\text{rec } x.t/x] : \mathbb{P}$ and $\llbracket \Gamma \vdash \text{rec } x.t : \mathbb{P} \rrbracket \cong \llbracket \Gamma \vdash t[\text{rec } x.t/x] : \mathbb{P} \rrbracket$.*

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

Lemma 7.7 *Properties of prefix match:*

- (i) $\llbracket \Gamma \vdash [\alpha.u > \alpha.x \Rightarrow t] : \mathbb{Q} \rrbracket \cong \llbracket \Gamma \vdash t[u/x] : \mathbb{Q} \rrbracket$
- (ii) $\llbracket \Gamma \vdash [\alpha.u > \beta.x \Rightarrow t] : \mathbb{Q} \rrbracket \cong \emptyset$ if $\alpha \neq \beta$
- (iii) $\llbracket \Gamma \vdash [\sum_{i \in I} u_i > \alpha.x \Rightarrow t] : \mathbb{Q} \rrbracket \cong \llbracket \Gamma \vdash \sum_{i \in I} [u_i > \alpha.x \Rightarrow t] : \mathbb{Q} \rrbracket$

7.3 Expressive power

The linearity constraints imply that ALLAN cannot encode CCS in full generality. In particular, the translation of Section 3.3 of a CCS term like

$$\text{rec } x.(a.x|\bar{a}.x)$$

is not well-formed, since x is crossed with itself in

$$P \llbracket a.x \rrbracket \llbracket \bar{a}.x \rrbracket .$$

But with the minor restriction that no variable occurs freely on both sides of a parallel composition, the translation into ALLAN provides the same presheaf semantics of CCS as the translation into HOPLA.

On the other hand, the tensor type of ALLAN allows us to define processes of the kind encountered in treatments of *nondeterministic dataflow* [16]. Define \mathbb{P} recursively so that

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

consisting of streams (or sequences) of a 's and b 's.

Example 7.8 A process A of type $\mathbb{P} \multimap \mathbb{P}$ which selects and outputs a 's while ignoring all b 's:

$$\text{rec } f.\lambda x.[x > a.x' \Rightarrow a.(f x')] + [x > b.x' \Rightarrow f x']$$

We have e.g. $\llbracket A a.a.b.\emptyset \rrbracket \cong \llbracket a.a.\emptyset \rrbracket$. \square

Example 7.9 A process B of type $\mathbb{P} \otimes \mathbb{P}$ which produces two identical, parallel streams of a 's and b 's as output:

$$\text{rec } p.[p > x \otimes y \Rightarrow (a.x \otimes a.y) + (b.x \otimes b.y)]$$

The category of elements of $\llbracket B \rrbracket$ has objects $(s \otimes s', \mathbf{0})$ where one of the strings s, s' over $\{a, b\}$ is a substring of the other. \square

Notice the ‘‘entanglement’’ of the two sides of the tensor above: the capabilities of the process on one side is restricted by what the process on the other side has done. Such an effect cannot be obtained using HOPLA's products because nondeterministic sum distributes over pairing (Figure 13, item (xiii)). In other words, choices on one side cannot effect the other.

Example 7.10 A process C of type $\mathbb{P} \multimap (\mathbb{P} \otimes \mathbb{P})$ which 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:

$$\text{rec } f.\lambda z.[z > a.z' \Rightarrow [(f z') > x \otimes y \Rightarrow a.x \otimes y] + [z > b.z' \Rightarrow [(f z') > x \otimes y \Rightarrow x \otimes b.y]]$$

We have e.g. $\llbracket C a.a.b.\emptyset \rrbracket \cong \llbracket a.a.\emptyset \otimes b.\emptyset \rrbracket$. \square

Of course, what we would like at this stage is to define an operational semantics that allows us to ‘‘execute’’ these examples, and prove it sound and complete with respect to the denotational semantics (in some sense). Unfortunately, as we shall see in the next section, this has turned out to be difficult.

7.4 Operational semantics

Consider a closed term t of ALLAN of type \mathbb{P} , and write t also for the presheaf interpreting it. Sections 4.3 and 6.1 suggest that we should look to $\mathcal{E}[t]$ for an operational understanding of t in terms of a transition system. We would then hope that for any morphism

$$(\perp, \mathbf{0}) \xrightarrow{!p} (p, x)$$

of $\mathcal{E}[t]$, the presheaf rooted at (p, x) is again the denotation of a term t' , so that we would have a corresponding transition $t \xrightarrow{p} t'$ in the operational semantics. This turns out to be the case for ground type, but it does not hold in general at higher order. With the hope of being able to extend to higher order later, we shall restrict ourselves to ground types.

At ground type we expect a correspondence between the labels used in transitions $t \xrightarrow{p} t'$ and the patterns of pattern match terms, so making it possible to give operational semantics to pattern matching. We therefore start by investigating patterns.

7.4.1 General patterns

An obvious extension of ALLAN is to allow general patterns according to

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

—with p being well-formed if all its variables are distinct. A match $[u > p \Rightarrow t]$ can be understood inductively as an abbreviation for a term in ALLAN, according to Figure 18. However, we can get useful insight by giving denotational semantics to patterns directly. For this we use judgements of the form

$$x_1 : \mathbb{P}_1, \dots, x_k : \mathbb{P}_k \Vdash p : \mathbb{P}$$

with the x_i distinct, interpreted as functors

$$(\mathbb{P}_1 \otimes \dots \otimes \mathbb{P}_k)_\perp \xrightarrow{a} \mathbb{P}_\perp$$

according to Figure 19. Since there are no rules for weakening or exchange, we have

Lemma 7.11 *The environment Π for which $\Pi \Vdash p : \mathbb{P}$ is given uniquely by p and \mathbb{P} .*

The four pattern matches of ALLAN can be replaced by a single one, given by the typing rule

$$\frac{\Delta \vdash u : \mathbb{P} \quad \Pi \Vdash p : \mathbb{P} \quad \Gamma, \Pi \vdash t : \mathbb{Q}}{\Delta, \Gamma \vdash [u > p \Rightarrow t] : \mathbb{Q}}$$

Confusing syntax and semantics, we'll write p^* for the map $\mathbb{P}_\perp \rightarrow \Pi_\perp$ of **Lin** obtained by composition as $X \mapsto X(p-)$. With Γ and Δ empty, the interpretation of the above rule is obtained as $\Sigma_{i \in I} t(u_i)$ where

$$\Sigma_{i \in I} [u_i] \quad \text{with} \quad I = [u](p_\perp)$$

is the rooted component decomposition of $p^*[u] \in \widehat{\Pi}_\perp$. Intuitively, I is the set of ways that u may perform computations matching p , and the u_i 's are the processes that u may become afterwards. This is the key to an operational semantics for the first-order fragment of ALLAN.

7.4.2 The tensor fragment

We now consider the *tensor fragment* of the affine-linear language, the first order fragment obtained by leaving out products (for brevity). We'll only need a sub-language of patterns,

$$p ::= \alpha.x \mid p \otimes x \mid x \otimes p.$$

We obtain *actions* as images $\llbracket \Pi \Vdash p : \mathbb{P} \rrbracket(\perp) \in \mathbb{P}_\perp$ for which we'll write

$$a ::= \alpha \mid a \otimes \perp \mid \perp \otimes a.$$

Conversely, for each a we may recover a pattern p_a (unique up to variable names) by replacing all \perp 's of a with distinct variables. We let actions stand for patterns and write

$$\mathbb{P}_1 \otimes \dots \otimes \mathbb{P}_k \Vdash a : \mathbb{P}$$

whenever $x_1 : \mathbb{P}_1, \dots, x_k : \mathbb{P}_k \Vdash p_a : \mathbb{P}$. Because of Lemma 7.11, $(\mathbb{P}_1 \otimes \dots \otimes \mathbb{P}_k)_\perp$ is uniquely given by a and \mathbb{P} ; in fact, it is isomorphic to the path order above $a \in \mathbb{P}_\perp$ with p_a its inclusion into \mathbb{P}_\perp .

If $\Pi \Vdash a : \mathbb{P}$ and $\Gamma \vdash t : \mathbb{P}$ we write a^*t for the map

$$\Gamma \xrightarrow{[t]} \mathbb{P}_\perp \xrightarrow{p_a^*} \Pi_\perp$$

of **Aff**. With t closed, a^*t is a presheaf over a lifted path order and therefore has a decomposition as a sum of rooted presheaves by Proposition 6.3. It turns out that each rooted component has the form $[t']$ for a denotation t' of a term of type Π . Judgements $t \xrightarrow{a} t'$ in the operational semantics will express that $[t']$ is a rooted component of a^*t . In fact, derivations of transitions $t \xrightarrow{a} t'$ will be in 1-1 correspondence with components of the decomposition of a^*t .

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

Figure 18: General patterns as abbreviations

Typing	Interpretation
$\frac{}{x : \mathbb{P} \Vdash x : \mathbb{P}}$	$\frac{}{\mathbb{P}_\perp \xrightarrow{1} \mathbb{P}_\perp}$
$\frac{\Pi \Vdash p : \mathbb{P}_\alpha \quad \alpha \in A}{\Pi \Vdash \alpha.p : \Sigma_{\alpha \in A} \alpha. \mathbb{P}_\alpha}$	$\frac{\Pi_\perp \xrightarrow{p} \mathbb{P}_{\alpha_\perp} \quad \alpha \in A}{\Pi_\perp \xrightarrow{[in_\alpha p]} (\Sigma_{\alpha \in A} \alpha. \mathbb{P}_\alpha)_\perp}$
$\frac{\Pi \Vdash p : \mathbb{P}}{\Pi \Vdash (p, -) : \mathbb{P} \& \mathbb{Q}}$	$\frac{\Pi_\perp \xrightarrow{p} \mathbb{P}_\perp}{\Pi_\perp \xrightarrow{p} \mathbb{P}_\perp \xrightarrow{in_1} (\mathbb{P} \& \mathbb{Q})_\perp}$
$\frac{\Pi \Vdash q : \mathbb{Q}}{\Pi \Vdash (-, q) : \mathbb{P} \& \mathbb{Q}}$	$\frac{\Pi_\perp \xrightarrow{q} \mathbb{Q}_\perp}{\Pi_\perp \xrightarrow{q} \mathbb{Q}_\perp \xrightarrow{in_2} (\mathbb{P} \& \mathbb{Q})_\perp}$
$\frac{\Pi \Vdash p : \mathbb{P} \quad \Lambda \Vdash q : \mathbb{Q}}{\Pi, \Lambda \Vdash p \otimes q : \mathbb{P} \otimes \mathbb{Q}}$	$\frac{\Pi_\perp \xrightarrow{p} \mathbb{P}_\perp \quad \Lambda_\perp \xrightarrow{q} \mathbb{Q}_\perp}{(\Pi \otimes \Lambda)_\perp \xrightarrow{p \times q} (\mathbb{P} \otimes \mathbb{Q})_\perp}$
$\frac{\Pi \Vdash p : \mathbb{P}_j[\mu \vec{P}. \vec{P}/\vec{P}]}{\Pi \Vdash p : \mu_j \vec{P}. \vec{P}} \quad \frac{\Pi \Vdash p : \mu_j \vec{P}. \vec{P}}{\Pi \Vdash p : \mathbb{P}_j[\mu \vec{P}. \vec{P}/\vec{P}]}$	$\mu_j \vec{P}. \vec{P}$ and $\mathbb{P}_j[\mu \vec{P}. \vec{P}/\vec{P}]$ denote equal path orders, so the premise and conclusion are interpreted as the same map.

Figure 19: Semantics of patterns

$$\begin{aligned}
& \text{(vi)} \frac{e_1 \Vdash u \xrightarrow{a \otimes \perp} e'_1 \Vdash u'}{e_1, u > x \otimes y, e_2 \Vdash x \xrightarrow{a} e'_1, u' > x \otimes y, e_2 \Vdash x} \\
& \text{(r)} \frac{e \Vdash t[rec\ x.t/x] \xrightarrow{a} e' \Vdash t'}{e \Vdash rec\ x.t \xrightarrow{a} e' \Vdash t'} \quad \text{(s)} \frac{e \Vdash t_j \xrightarrow{a} e' \Vdash t'}{e \Vdash \Sigma_{i \in I} t_i \xrightarrow{a} e' \Vdash t'} \quad j \in I \\
& \text{(p)} \frac{}{e \Vdash \alpha.t \xrightarrow{\alpha} e \Vdash t} \quad \text{(pm)} \frac{e \Vdash u \xrightarrow{\alpha} e' \Vdash u' \quad e' \Vdash t[u'/x] \xrightarrow{a} e'' \Vdash t'}{e \Vdash [u > \alpha.x \Rightarrow t] \xrightarrow{a} e'' \Vdash t'} \\
& \text{(tl)} \frac{e \Vdash t \xrightarrow{a} e' \Vdash t'}{e \Vdash t \otimes u \xrightarrow{a \otimes \perp} e' \Vdash t' \otimes u} \quad \text{(tm)} \frac{e, u > x \otimes y \Vdash t \xrightarrow{a} e', u' > x \otimes y \Vdash t'}{e \Vdash [u > x \otimes y \Rightarrow t] \xrightarrow{a} e' \Vdash [u' > x \otimes y \Rightarrow t']}
\end{aligned}$$

Figure 20: ALLAN operational semantics

The operational semantics is informed by isomorphisms saying how to find the rooted components of a^*t . As an example we have the isomorphisms on the left below, suggesting the rules on the right:

$$\beta^*(\alpha.t) \cong \begin{cases} [t] & \text{if } \alpha = \beta \\ \emptyset & \text{if } \alpha \neq \beta \end{cases} \quad \frac{}{\alpha.t \xrightarrow{\alpha} t}$$

$$a^*\Sigma_{i \in I} t_i \cong \Sigma_{i \in I} a^*t_i \quad \frac{t_j \xrightarrow{a} t' \quad j \in I}{\Sigma_{i \in I} t_i \xrightarrow{a} t'}$$

These rules are for closed terms. In the semantics

$$a^*[u > x \otimes y \Rightarrow t] \cong [u > x \otimes y \Rightarrow a^*t],$$

which suggests that we let t (an open term) take an a -transition in the “environment” $u > x \otimes y$. Syntactically, environments e are lists of such matches. An environment “exports” a set of variables; the empty environment exports the empty set, while $e, u > x \otimes y$ exports what e exports, except the free variables of u , plus x and y . We may formalise this using a judgement

$$e \vdash x_1 : \mathbb{P}_1, \dots, x_k : \mathbb{P}_k$$

(with the x_i distinct) which denotes the same presheaf over $\mathbb{P}_1 \otimes \dots \otimes \mathbb{P}_k$ as

$$[e \Rightarrow x_1 \otimes \dots \otimes x_k].$$

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

$$\vdash e \Rightarrow t : \mathbb{P}; \Delta$$

the same denotation as the term

$$[e \Rightarrow t \otimes x_1 \otimes \dots \otimes x_k]$$

where the x_i are now the variables exported by e but not free in t .

The operational rules are shown in Figure 20. The left rules (v_l) and (t_l) have the obvious right counterparts. In (tm) x and y are implicitly renamed to avoid overshadowing of exported variables in the environment $e, u > x \otimes y$. The rules are well-typed:

Lemma 7.12 Assume $\vdash e \Rightarrow t : \mathbb{P}; \Delta$. If

$$e \Rightarrow t \xrightarrow{a} e' \Rightarrow t',$$

then $\mathbb{P}' \Vdash a : \mathbb{P}$ and $\vdash e' \Rightarrow t' : \mathbb{P}'; \Delta$.

$$\begin{aligned} |x| &= 1 \\ |rec^0 x.t| &= 1 \\ |rec^{n+1} x.t| &= |t[rec^n x.t/x]| \oplus 1 \\ |\Sigma_{i \in I} t_i| &= (\sup_{i \in I} |t_i|) \oplus 1 \\ |\alpha.t| &= |t| \oplus 1 \\ |(t, u)| &= \max\{|t|, |u|\} \oplus 1 \\ |t \otimes u| &= |t| \oplus |u| \\ |[u > p \Rightarrow t]| &= |t| \oplus |u| \\ |e| &= 0 \\ |e, u > p| &= |e| \oplus |t| \\ |e \Rightarrow t| &= |e| \oplus |t| \end{aligned}$$

Figure 21: Size measure for terms, environments, and terms in environments

If we replace (r) by the rule

$$\frac{e \Rightarrow t[rec^n x.t/x] \xrightarrow{a} e' \Rightarrow t'}{e \Rightarrow rec^{n+1} x.t \xrightarrow{a} e' \Rightarrow t'}$$

for *tagged recursion*, we expect all transition sequences to be finite. This can be proved by defining a size measure $|\cdot|$ on terms in environments, see Figure 21. Sizes are measured using ordinals because of the possibly infinite sums and \oplus is the “natural addition” of ordinals [18], which is commutative, associative and strictly monotone in each argument.

Lemma 7.13 Assume $\vdash e \Rightarrow t : \mathbb{P}; \Delta$ with *tagged recursion*. If

$$e \Rightarrow t \xrightarrow{a} e' \Rightarrow t',$$

then $|e \Rightarrow t| > |e' \Rightarrow t'|$.

Building on this fact, we can prove the main result below which says that rooted components correspond to derivations. It is proved by well-founded induction using an order based on the size measure. The induction hypothesis says that for $\vdash e \Rightarrow t : \mathbb{P}; \Delta$ and $\mathbb{P}' \Vdash a : \mathbb{P}$ with $\vec{x} \equiv x_1 \otimes \dots \otimes x_k$ any subset of the variables in Δ , we have $(a \otimes \perp)^*[e \Rightarrow t \otimes \vec{x}] \cong \Sigma_d [[e' \Rightarrow t' \otimes \vec{x}]]$ where d ranges over derivations with conclusion of the form $e \Rightarrow t \xrightarrow{a} e' \Rightarrow t'$.

Theorem 7.14 Assume $\vdash t : \mathbb{P}$ and $\mathbb{P}' \Vdash a : \mathbb{P}$. Then $a^*t \cong \Sigma_d [t']$ where the sum is over all derivations d with conclusion $e \Rightarrow t \xrightarrow{a} e' \Rightarrow t'$.

$$\begin{array}{c}
\frac{B > x \otimes y \Rightarrow a.x \xrightarrow{a} B > x \otimes y \Rightarrow x}{B > x \otimes y \Rightarrow a.x \otimes a.y \xrightarrow{a \otimes \perp} B > x \otimes y \Rightarrow x \otimes a.y} \\
\frac{B > x \otimes y \Rightarrow (a.x \otimes a.y) + (b.x \otimes b.y) \xrightarrow{a \otimes \perp} B > x \otimes y \Rightarrow x \otimes a.y}{\Rightarrow [B > x \otimes y \Rightarrow (a.x \otimes a.y) + (b.x \otimes b.y)] \xrightarrow{a \otimes \perp} \Rightarrow [B > x \otimes y \Rightarrow x \otimes a.y]} \\
\Rightarrow B \xrightarrow{a \otimes \perp} \Rightarrow [B > x \otimes y \Rightarrow x \otimes a.y]
\end{array}$$

Figure 22: An example derivation

Example 7.15 Consider the process B of Example 7.9. In the operational semantics we can derive e.g.

$$\Rightarrow B \xrightarrow{a \otimes \perp} \Rightarrow [B > x \otimes y \Rightarrow x \otimes a.y]$$

—see Figure 22. Notice how the term on the right “remembers” the choice of doing an a action first. \square

Extending the operational semantics and Theorem 7.14 to higher order is proving difficult. The HOPLA-like rules

$$\frac{e \Rightarrow t[u/x] \xrightarrow{a} e' \Rightarrow t'}{e \Rightarrow \lambda x.t \xrightarrow{u \mapsto a} e' \Rightarrow t'}$$

$$\frac{e \Rightarrow t \xrightarrow{u \mapsto a} e' \Rightarrow t'}{e \Rightarrow t u \xrightarrow{a} e' \Rightarrow t'}$$

would contaminate actions with exported variables and this does not always make sense. As an example, consider the “derivation”

$$\begin{array}{c}
\vdots \\
\frac{\Rightarrow t \otimes u \xrightarrow{(y \mapsto a) \otimes \perp} t'}{t \otimes u > x \otimes y \Rightarrow x \xrightarrow{y \mapsto a} t'} \\
\frac{t \otimes u > x \otimes y \Rightarrow x y \xrightarrow{a} t'}{\Rightarrow [t \otimes u > x \otimes y \Rightarrow x y] \xrightarrow{a} t'}
\end{array}$$

In $\Rightarrow t \otimes u \xrightarrow{(y \mapsto a) \otimes \perp} t'$ the correspondence between u and y is lost. This difficulty is in fact what led us to HOPLA, and it seems appropriate at this point to give its denotational semantics.

Recall from Section 5.2 the inadequacy of \mathbf{Lin} as a denotational model of process calculi. In that section, we discussed two reasonable responses to this, of which one was to employ lifting $(-)_\perp$ to allow maps to ignore their arguments. The other—more radical—response was to employ a suitable exponential $!(-)$ in order to allow arbitrary copying of arguments.

8 Copying

As for the exponential $!$ of linear logic, there are many possible choices—see [21]. Intuitively, an object of $!\mathbb{P}$ should represent a computation path of an “assembly” of processes each with computation paths 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 is not fixed once and for all. Rather the assembly grows as further copies are invoked, and these copies can be made of 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 subpaths.

With this choice of exponential, there is an obvious inclusion functor $i_{\mathbb{P}} : !\mathbb{P} \rightarrow \widehat{\mathbb{P}}$, playing the role of Yoneda in Section 4.2 and strict Yoneda in Section 6.

In particular, it can be shown that $\widehat{\mathbb{P}}$ with $i_{\mathbb{P}}$ is the free *filtered colimit* completion of $!\mathbb{P}$. Spelled out, given any functor $F : !\mathbb{P} \rightarrow \mathcal{C}$ where \mathcal{C} has all filtered colimits, there is a filtered-colimit preserving functor $G : \widehat{\mathbb{P}} \rightarrow \mathcal{C}$, determined to within isomorphism, such that $G \circ i_{\mathbb{P}} \cong F$ (see [17]):

$$\begin{array}{ccc}
!\mathbb{P} & \xrightarrow{i_{\mathbb{P}}} & \widehat{\mathbb{P}} \\
& \searrow F & \downarrow G \\
& & \mathcal{C}
\end{array}$$

It follows that maps $!\mathbb{P} \rightarrow \mathbb{Q}$ in \mathbf{Lin} correspond, to within isomorphism, to *continuous* (i.e., filtered colimit preserving) functors $\widehat{\mathbb{P}} \rightarrow \widehat{\mathbb{Q}}$.

8.1 A cartesian closed category

Define \mathbf{Cts} to be the category consisting of small categories $\mathbb{P}, \mathbb{Q}, \dots$ as objects and morphisms $F : \mathbb{P} \rightarrow \mathbb{Q}$ the continuous functors $F : \widehat{\mathbb{P}} \rightarrow \widehat{\mathbb{Q}}$; they compose as functors. Clearly \mathbf{Lin} is a subcategory of \mathbf{Cts} , one which shares the same objects. We have

$$\mathbf{Cts}(\mathbb{P}, \mathbb{Q}) \simeq \mathbf{Lin}(!\mathbb{P}, \mathbb{Q})$$

for all small categories \mathbb{P}, \mathbb{Q} . The category \mathbf{Cts} is the coKleisli category of the comonad based on finite-colimit completions. The unit of the corresponding adjunction is given by maps $copy : \mathbb{P} \rightarrow !\mathbb{P}$ of \mathbf{Cts} , which play the role of $\lfloor - \rfloor$ from \mathbf{Aff} . We can easily characterise those maps in \mathbf{Cts} which are in \mathbf{Lin} :

Proposition 8.1 *Suppose $F : \widehat{\mathbb{P}} \rightarrow \widehat{\mathbb{Q}}$ is a functor which preserves filtered colimits. Then, F preserves all colimits iff F preserves finite coproducts.*

So a continuous map is linear iff it preserves sums.

There is an isomorphism $!(\mathbb{P} \& \mathbb{Q}) \cong !\mathbb{P} \times !\mathbb{Q}$ making \mathbf{Cts} cartesian closed; this immediately allows us to interpret the simply-typed lambda calculus with pairing in \mathbf{Cts} . Products $\mathbb{P} \& \mathbb{Q}$ in \mathbf{Cts} are given as in \mathbf{Lin} but now viewing the projections as continuous functors. The function space $\mathbb{P} \rightarrow \mathbb{Q}$ is given by $(!\mathbb{P})^{\text{op}} \times \mathbb{Q}$. Like \mathbf{Aff} , the category \mathbf{Cts} does not have coproducts and so we again resort to the coproduct of \mathbf{Lin} to construct a prefixed sum. The definition is the same that of Section 6.6, we just need to replace $(-)_\perp$ with $!(-)$ and $\lfloor - \rfloor$ with $copy$.

8.2 Equivalence

The category $!\mathbb{P}$ has an initial element \perp , given by the empty colimit, and so we can decompose any presheaf X over $!\mathbb{P}$ as a sum of rooted presheaves. This is the key to a correspondence with the operational semantics of HOPLA. We may interpret $\mathbb{P} : a : \mathbb{P}'$ as a map $\mathbb{P} \rightarrow !\mathbb{P}'$ of \mathbf{Lin} by letting the judgement for β in Figure 10 denote restriction to $!\mathbb{P}'_\beta$, and inductively interpreting $u \mapsto a, (a, -), (-, a)$ as the denotation of a precomposed with the map given by application to $\llbracket u \rrbracket$, and first and second projections, respectively. The rules are then sound in the sense that if $t \xrightarrow{a} t'$ then (identifying a term with its denotation) $copy\ t'$ is a rooted component of $a(t)$, and in fact, there is a 1-1 correspondence between derivations with conclusion $t \xrightarrow{a} t'$, for some t' , and the rooted components of $a(t)$, so the rules are also complete.

8.3 Bisimulation

Interestingly, although the operational bisimulation of Park/Milner is a congruence for HOPLA, maps of \mathbf{Cts} do not in general preserve open maps. This suggests that one should look at other notions of open map, cf. Section 6.2. Indeed, for each ‘‘inclusion’’ $i_{\mathbb{P}} : !\mathbb{P} \rightarrow \widehat{\mathbb{P}}$, there is a corresponding notion of $i_{\mathbb{P}}$ -bisimulation. Unfortunately for our choice of exponential, $i_{\mathbb{P}}$ -bisimulation degenerates into isomorphism, but there are choices of exponential whose corresponding $i_{\mathbb{P}}$ -bisimulation coincide with open-map bisimulation, and one may hope to recover operational bisimulation as $i_{\mathbb{P}}$ -bisimulation for some choice of exponential (or expose it as a union of such bisimulations). In fact, it appears that the correspondence between the denotational and operational semantics can be proved abstractly, depending only on properties of the prefixed sum, so that it holds also for other choices of exponential.

9 Independence

Returning to the operational semantics of ALLAN, consider again the term B of Example 7.9. In the operational semantics, we may derive transitions

$$\begin{array}{ccc} \Rightarrow [B > x \otimes y \Rightarrow x \otimes a.y] & & \\ \begin{array}{c} \nearrow a \otimes \perp \\ \searrow \perp \otimes a \end{array} & & \\ \Rightarrow B & \Rightarrow [B > x \otimes y \Rightarrow x \otimes y] & \end{array}$$

But we do not have a general result saying how such a diagram can be completed into a diamond corresponding to

$$\begin{array}{ccc} & (a \otimes \epsilon, \mathbf{0}) & \\ \begin{array}{c} \nearrow a \otimes \perp \\ \searrow \perp \otimes a \end{array} & & \begin{array}{c} \searrow \perp \otimes a \\ \nearrow a \otimes \perp \end{array} \\ (\epsilon \otimes \epsilon, \mathbf{0}) & & (a \otimes a, \mathbf{0}) \\ \begin{array}{c} \searrow \perp \otimes a \\ \nearrow a \otimes \perp \end{array} & & \\ & (\epsilon \otimes a, \mathbf{0}) & \end{array}$$

in the category of elements. In other words, as it stands the operational semantics gives an *interleaving model* of the tensor fragment, whereas the presheaf semantics seems to be an *independence model*. This distinction is fundamental in concurrency theory. An interleaving model explains concurrency in terms of nondeterminism, whereas concurrency is taken as basic in independence models. For the process B this means that the two

transitions above are considered *independent*, they can happen in any order, and having done both, the resulting state is silent about which occurred first.

The diamond property holds for the process B , but until now we have been unable to prove (or disprove) whether it holds in general for the operational semantics of the tensor fragment. However, the considerations about the independence present in the presheaf semantics of the tensor fragment has led us to realise that definable presheaves can be represented by *event structures* [26], a standard independence model of concurrency. In detail, categories of elements of definable presheaves can be seen as posets of configurations of event structures.

Event structures were originally used as concrete representations of domains. Interestingly, in our application, event structures may play both these roles, interpreting types as well as terms of the tensor fragment.

The material below describes work in progress.

9.1 Types as event structures

Definition 9.1 A *prime event structure with binary conflict*, or just *event structure*, is a triple $\mathbb{P} = (E, \leq, \smile)$ where E is a set of *events* upon which a partial order \leq of *causality* and a binary, symmetric, irreflexive relation \smile of *conflict* are defined. This data must satisfy

$$\begin{aligned} [e] &=_{\text{def}} \{e' \mid e' \leq e\} \text{ is finite for every } e \in E \\ e \smile e' \leq e'' &\Rightarrow e \smile e'' \text{ for all } e, e', e'' \in E \end{aligned}$$

Notation 9.2 We use Girard's notation: \complement is the complement of \smile , the reflexive closure of which is written \succsim with complement \frown . Specifying one relation clearly determines all of them.

Definition 9.3 A *configuration* of $\mathbb{P} = (E, \leq, \smile)$ is a finite subset p of E which is down-closed: $\forall e \in p. [e] \subseteq p$ and consistent: $\forall e, e' \in p. e \frown e'$. The poset of configurations of \mathbb{P} ordered by inclusion is also written \mathbb{P} . Note that \mathbb{P} contains both \emptyset and $[e]$ for any $e \in E$. Below, we'll consistently write $\perp \in \mathbb{P}$ for the empty configuration. Two configurations $p, p' \in \mathbb{P}$ are *compatible*, written $p \uparrow p'$, if they have an upper bound in \mathbb{P} .

We list some basic constructions of event structures:

Lifting If $\mathbb{P} = (E, \leq, \smile)$ is an event structure, then its *lifting*, written \mathbb{P}_\perp , is the event structure given by

$$\begin{aligned} \text{events: } & \{\perp\} \cup \{[e] \mid e \in E\} \\ \text{causality: } & p \leq p' \text{ iff } p \subseteq p' \\ \text{conflict: } & p \smile p' \text{ iff } p \not\uparrow_{\mathbb{P}} p' \end{aligned}$$

The poset \mathbb{P}_\perp is isomorphic to \mathbb{P} with a new least element added.

Sum If $\mathbb{P}_i = (E_i, \leq_i, \smile_i)$ are event structures for $i \in I$, then their *sum*, written $\Sigma_{i \in I} \mathbb{P}_i$, is the event structure given by

$$\begin{aligned} \text{events: } & \{i : e \mid e \in E_i\} \\ \text{causality: } & i : e \leq j : e' \text{ iff } i = j \text{ and } e \leq_i e' \\ \text{conflict: } & i : e \smile j : e' \text{ iff } i \neq j \text{ or } (i = j \text{ and } e \smile_i e') \end{aligned}$$

The poset $\Sigma_{i \in I} \mathbb{P}_i$ is isomorphic to the coalesced sum of pointed posets \mathbb{P}_i , $i \in I$. The non- \perp elements will be written $i : p$ for $p \in \mathbb{P}_i$.

Lifted sum If \mathbb{P}_α for $\alpha \in A$ are event structures, then their *lifted sum*, written $\Sigma_{\alpha \in A} \alpha.\mathbb{P}_\alpha$, is the event structure $\Sigma_{\alpha \in A} \alpha.\mathbb{P}_{\alpha\perp}$. The corresponding poset is isomorphic to the lifting of the separated sum of the \mathbb{P}_α . The non- \perp elements will be written $\alpha.p$ for $p \in \mathbb{P}_\alpha$.

Tensor If $\mathbb{P}_i = (E_i, \leq_i, \smile_i)$ are event structures for $i = 0, 1$ then their *tensor product*, written $\mathbb{P}_0 \otimes \mathbb{P}_1$, is the event structure given by

$$\begin{aligned} \text{events: } & \{i : e \mid e \in E_i\} \\ \text{causality: } & i : e \leq j : e' \text{ iff } i = j \text{ and } e \leq_i e' \\ \text{conflict: } & i : e \smile j : e' \text{ iff } i = j \text{ and } e \smile_i e' \end{aligned}$$

The poset $\mathbb{P}_0 \otimes \mathbb{P}_1$ is isomorphic to the usual cartesian product of posets. The non- \perp elements will be written $p \otimes q$ for $p \in \mathbb{P}_0$ and $q \in \mathbb{P}_1$ (not both \perp).

Above If $\mathbb{P} = (E, \leq, \smile)$ is an event structure with $p \in \mathbb{P}$, then the event structure $p^*\mathbb{P}$ has events $\{e \in E \mid \forall e' \in p. e \frown e'\}$, *i.e.* those events that may extend the configuration p . Causality and conflict are simply restrictions of \leq and \smile to these events. The poset $p^*\mathbb{P}$ of configurations is isomorphic to the poset above $p \in \mathbb{P}$.

The constructions above allow us to interpret the types $\mathbb{P}, \mathbb{Q}, \dots$ of the tensor fragment as event structure. Beware that because of the empty configuration, all types are now interpreted as *pointed* posets, and so correspond to \mathbb{P}_\perp of Section 6.

Recursive types are obtained as lubs in the (large) cpo of event structures ordered by a substructure ordering, see [26], with respect to which the constructions above are continuous.

The actions of Section 7.4.2 are recovered as a sub-language of the following language of configurations of type interpretations, called *paths* below:

$$p, q ::= \perp \mid \alpha.p \mid p \otimes q$$

Here, the syntax is restricted to rule out “synonyms” of \perp like $\perp \otimes \perp$. A well-formed path $p : \mathbb{P}$, interpreted as the obvious element $p \in \mathbb{P}$, is derived using the rules below:

$$\frac{}{\perp : \mathbb{P}} \quad \frac{p : \mathbb{P}_\beta \text{ where } \beta \in A}{\beta.p : \sum_{\alpha \in A} \alpha.\mathbb{P}_\alpha}$$

$$\frac{p : \mathbb{P} \quad q : \mathbb{Q}}{p \otimes q : \mathbb{P} \otimes \mathbb{Q}} \quad (p, q) \neq (\perp, \perp)$$

Notice that because of the restriction, there is a 1-1 correspondence between elements of path orders \mathbb{P} and derivations of $p : \mathbb{P}$. We’ll confuse syntax and denotations for path orders and paths.

Lemma 9.4 *Suppose $p : \mathbb{P}$. Then the poset $p^*\mathbb{P}$ is (to within isomorphism) again a path order.*

Accordingly, when $p : \mathbb{P}$, we’ll treat the expression $p^*\mathbb{P}$ as a syntactic synonym for a path order, according to

$$\begin{aligned} \perp^*\mathbb{P} &= \mathbb{P} \\ (\beta.p)^*(\sum_{\alpha \in A} \alpha.\mathbb{P}_\alpha) &= p^*\mathbb{P}_\beta \\ (p \otimes q)^*(\mathbb{P} \otimes \mathbb{Q}) &= p^*\mathbb{P} \otimes q^*\mathbb{Q} \end{aligned}$$

The language of patterns of Section 7.4.1 is obtained from the language of paths by replacing the \perp ’s with distinct variables and removing the restriction in the tensor rule. For any pattern $\Pi \Vdash p : \mathbb{P}$, we have $\Pi \cong (p\perp)^*\mathbb{P}$. Given a pattern $x_1 : \mathbb{P}_1, \dots, x_k : \mathbb{P}_k \Vdash p : \mathbb{P}$ and paths $p_1 : \mathbb{P}_1, \dots, p_k : \mathbb{P}_k$, we’ll write $p(\vec{p})$ for the path obtained by applying p to the p_i . Because of the syntactic restriction on paths, this is not just substitution. As an example, we have $(x \otimes y)(\perp, \perp) = \perp$.

Having represented types, paths, and patterns of the tensor fragment using event structures, we turn to terms. To this end, we’ll need morphisms between event structures.

9.2 Terms as spans

Definition 9.5 Consider two event structures $X = (E_1, \leq_1, \smile_1)$ and $\mathbb{P} = (E_2, \leq_2, \smile_2)$. A morphism $\eta : X \rightarrow \mathbb{P}$ is a function $\eta : E_1 \rightarrow E_2$ that for all $e, e' \in E_1$ satisfies $\eta[e] \supseteq [\eta e]$ and $e \smile_1 e' \Rightarrow \eta e \smile_2 \eta e'$.

Lemma 9.6 *Let $\eta : X \rightarrow \mathbb{P}$ be a morphism of event structures. Then η sends configurations of X to configurations of \mathbb{P} and*

$$\forall x \in X. \forall e, e' \in x. \eta e = \eta e' \Rightarrow e = e'$$

i.e., η is injective on configurations.

Definition 9.7 With the notation of Definition 9.5, the morphism $\eta : X \rightarrow \mathbb{P}$ is *synchronous*, if for all $e \in E_1$, $\eta[e] = [\eta e]$.

Lemma 9.8 *Let $\eta : X \rightarrow \mathbb{P}$ be a synchronous morphism. Then for all $x \in X$ and $p \in \mathbb{P}$, we have*

$$p \subseteq \eta x \Rightarrow \exists! x' \in X. x' \subseteq x \text{ and } \eta x' = p,$$

i.e., a smaller “output” configuration is obtained from a unique smaller “input” configuration. We’ll write $x|_p$ for the unique subconfiguration x' of x .

A synchronous morphism $\eta : X \rightarrow \mathbb{P}$ induces a pre-sheaf X over \mathbb{P} as follows:

$$X(p) = \{x \in X \mid \eta x = p\}$$

If $p \subseteq p'$, X maps $x \in X(p')$ to $x|_p$. Note that X is rooted because $X(\perp)$ is the singleton consisting of the empty configuration of X . Further, the category of elements of X is isomorphic to the poset X .

Using synchronous morphisms and the constructions on event structures above, we are able to interpret most of the tensor fragment. But for pattern match, we need to be able to interpret open terms, and so we need an event structure representation of the maps of **Aff**, which up to isomorphism are *rooted* profunctors $F : \mathbb{P} \rightarrow \mathbb{Q}$ (recall that \mathbb{P} and \mathbb{Q} are pointed!). We shall make no distinction between F as a functor $\mathbb{P} \rightarrow \widehat{\mathbb{Q}}$ and F as a bifunctor $\mathbb{P} \times \mathbb{Q}^{\text{op}} \rightarrow \mathbf{Set}$.

Let \mathbb{P} and \mathbb{Q} be path orders. We’ll consider spans $\langle \mu, X, \eta \rangle$ of the form

$$\begin{array}{ccc} & X & \\ \mu \swarrow & & \searrow \eta \\ \mathbb{P} & & \mathbb{Q} \end{array}$$

Here, X is an event structure and η is a synchronous morphism, while μ is a strict, monotone map of posets. This is equivalent to requiring that μ maps events of X to configurations of \mathbb{P} such that

$$e \leq e' \Rightarrow \mu e \subseteq \mu e' \quad \text{and} \quad e \smile e' \Rightarrow \mu e \uparrow_{\mathbb{P}} \mu e'$$

We write $\mathbf{Spn}(\mathbb{P}, \mathbb{Q})$ for the class of such spans. They induce profunctors $F = F(\mu, X, \eta) : \mathbb{P} \dashrightarrow \mathbb{Q}$ as follows:

$$F(p, q) = \{x \in X \mid \mu x \subseteq p \text{ and } \eta x = q\}.$$

If $p \subseteq p'$, then $F(p \subseteq p', q)$ maps $x \in F(p, q)$ to itself as an element of $F(p', q)$, while for $q' \subseteq q$ in \mathbb{Q} , $F(p, q' \subseteq q)$ maps $x \in F(p, q)$ to $x|_{q'}$. Note that F is rooted in the sense that for any $p \in \mathbb{P}$, $F(p, \perp)$ is the singleton containing the empty configuration of X . A crucial observation is that F is *stable*: as a functor $\mathbb{P} \rightarrow \widehat{\mathbb{Q}}$, it preserves pullbacks. Since \mathbb{P} has all pullbacks, given by intersection of configurations, this intuitively means that any output of F is obtained from a unique, minimal input. In detail, if $x \in F(p, q)$ for some p, q , then $\mu x \subseteq p$ is the unique, minimal input needed for this computation.

One can now show that the poset of configurations of X is isomorphic to the category of elements of $\int^{p \in \mathbb{P}} Fp$. This fact can be interpreted as saying that F can be viewed as a process with computation paths in \mathbb{Q} , with needed input assigned to each computation state.

Being a morphism of event structures, η induces a strict, monotone map of posets $X \rightarrow \mathbb{Q}$. So μ and η can be seen as two maps of the same kind, suggesting how to compose spans. Let $\langle \mu_0, X_0, \eta_0 \rangle$ be a span in $\mathbf{Spn}(\mathbb{P}, \mathbb{Q})$ and $\langle \mu_1, X_1, \eta_1 \rangle$ a span in $\mathbf{Spn}(\mathbb{Q}, \mathbb{R})$. We can then form a pullback in the category of (pointed) posets and (strict) monotone maps:

$$\begin{array}{ccccc} & & X & & \\ & \swarrow \pi_0 & & \searrow \pi_1 & \\ X_0 & & & & X_1 \\ \mu_0 \swarrow & & \eta_0 \mu_1 & & \searrow \eta_1 \\ \mathbb{P} & & \mathbb{Q} & & \mathbb{R} \end{array}$$

So, the poset X is (up to isomorphism) given by

$$\{(x_0, x_1) \in X_0 \times X_1 \mid \eta_0 x_0 = \mu_1 x_1\}$$

—with componentwise order, and π_0, π_1 the obvious projections. The non- \perp elements of X will be written (x_0, x_1) with $x_0 \in X_0$ and $x_1 \in X_1$. In fact, X is isomorphic to the poset of configurations of an event structure $X = (E, \leq, \smile)$ with

$$\text{events: } \{(x_0, e_1) \in X_0 \times E_1 \mid \eta_0 x_0 = \mu_1 e_1\}$$

$$\text{causality: } (x_0, e_1) \leq (x'_0, e'_1) \text{ iff } x_0 \subseteq x'_0 \text{ and } e_1 \leq_1 e'_1$$

$$\text{conflict: } (x_0, e_1) \smile (x'_0, e'_1) \text{ iff } x_0 \not\uparrow_0 x'_0 \text{ or } e_1 \smile_1 e'_1$$

This is an event structure and the map $\pi_1 : E \rightarrow E_1$ taking (x_0, e_1) to e_1 is synchronous. Since strict morphisms compose we have

$$\langle \mu_0 \circ \pi_0, X, \eta_1 \circ \pi_1 \rangle \in \mathbf{Spn}(\mathbb{P}, \mathbb{R}).$$

As is to be expected from defining composition via pullbacks, associativity holds only up to isomorphism, suggesting a bicategory with objects $\mathbb{P}, \mathbb{Q}, \dots$ and spans as arrows. In fact, we expect spans to arise as some sub-bicategory of \mathbf{Prof} , the bicategory of profunctors. However, the exact relationship is yet to be determined.

In any case, the spans can be used to provide an alternative denotational semantics to the tensor fragment. Below, we'll write $\mathcal{P}[-], \mathcal{S}[-]$ for the interpretations as (rooted) profunctors and spans, respectively. We have

Proposition 9.9 *Let t be any term of the tensor fragment. Then $\mathcal{P}[[t]]$ is isomorphic to the profunctor induced by $\mathcal{S}[[t]]$.*

PROOF: By structural induction on (the typing derivation) of t . \square

9.3 Stable operational semantics

The event structure semantics may be exploited to obtain a new, simpler operational semantics. Rather than working with closed terms, we may derive both output and the corresponding minimal input for *open* terms, using judgements of the form

$$x_1 : p_1, \dots, x_k : p_k \vdash t \xrightarrow{q} t'.$$

Here, we have $x_1 : \mathbb{P}_1, \dots, x_k : \mathbb{P}_k \vdash t : \mathbb{Q}$ and p_1, \dots, p_k are paths of type $\mathbb{P}_1, \dots, \mathbb{P}_k$, respectively, while q is a path of type \mathbb{Q} . The operational rules are shown in Figure 23. We can show that the rules are type-correct in the sense that for a derived transition as above, we have

$$x_1 : p_1^* \mathbb{P}_1, \dots, x_k : p_k^* \mathbb{P}_k \vdash t' : q^* \mathbb{Q}.$$

Intuitively, this says that t' has the type of a process obtained by letting t use some input and produce some output. Semantically, there is a 1-1 correspondence between derivations with conclusion of the above form and configurations $x \in [[t]]$ with $\mu x = p_1 \otimes \dots \otimes p_k$ and $\eta x = q$. Except for recursion, this can be proved by structural induction on t , and so we avoid the use of a size measure. Further, we can show the diamond property for this semantics.

$$\begin{array}{c}
\text{(i)} \frac{}{x : p \vdash x \xrightarrow{p} x} \quad \text{(w)} \frac{\gamma \vdash t \xrightarrow{q} t'}{\gamma, x : \perp \vdash t \xrightarrow{q} t'} \quad \text{(e)} \frac{\gamma, x : p, y : q, \delta \vdash t \xrightarrow{r} t'}{\gamma, y : q, x : p, \delta \vdash t \xrightarrow{r} t'} \\
\text{(r)} \frac{\gamma \vdash t[\text{rec } x.t/x] \xrightarrow{p} t'}{\gamma \vdash \text{rec } x.t \xrightarrow{p} t'} \quad \text{(s)} \frac{\gamma \vdash t_j \xrightarrow{p} t'}{\gamma \vdash \sum_{i \in I} t_i \xrightarrow{p} t'} \quad j \in I \quad \text{(a)} \frac{\gamma \vdash t \xrightarrow{p} t'}{\gamma \vdash \alpha.t \xrightarrow{\alpha.p} t'} \\
\text{(t)} \frac{\gamma \vdash t \xrightarrow{p} t' \quad \delta \vdash u \xrightarrow{q} u'}{\gamma, \delta \vdash t \otimes u \xrightarrow{p \otimes q} t' \otimes u'} \\
\text{(m)} \frac{\delta \vdash u \xrightarrow{p(\vec{p})} u' \quad \gamma, \vec{x} : \vec{p} \vdash t \xrightarrow{q} t'}{\gamma, \delta \vdash [u > p(\vec{x}) \Rightarrow t] \xrightarrow{q} [u' > \vec{x} \Rightarrow t']} \quad \text{(b)} \frac{}{\vec{x} : \vec{\perp} \vdash t \xrightarrow{\perp} t}
\end{array}$$

Figure 23: ALLAN stable operational semantics

The event structure semantics suggests a stable function space, for which we seem to be able to provide an operational counterpart by refining the above rules. On the down side, at higher order the profunctors induced by spans (most probably) don't coincide with those obtained from the function space of the presheaf semantics. The difference is analogous to that between the domain theory of Dana Scott and the stable domain theory of Gérard Berry.

References

- [1] S. Abramsky. Computational interpretation of linear logic. Tech. Report 90/20, Dept. of Computing, Imperial College, 1990.
- [2] T. Braüner. *An Axiomatic Approach to Adequacy*. BRICS Dissertation Series DS-96-4, 1996.
- [3] G. L. Cattani. *Presheaf Models for Concurrency*. BRICS Dissertation Series DS-99-1, 1999.
- [4] G. L. Cattani, M. P. Fiore, and G. Winskel. A theory of recursive domains with applications to concurrency. In *Proc. LICS'98*.
- [5] G. L. Cattani, A. J. Power, and G. Winskel. A categorical axiomatics for bisimulation. In *Proc. of CONCUR'98*, LNCS 1466, 1998.
- [6] G. L. Cattani, I. Stark, and G. Winskel. Presheaf models for the π -calculus. In *Proc. CTCS'97*, LNCS 1290.
- [7] M. P. Fiore, G. L. Cattani, and G. Winskel. Weak bisimulation and open maps. In *Proc. LICS'99*.
- [8] T. T. Hildebrandt. A fully abstract presheaf semantics for SCCS with finite delay. In *Proc. CTCS'99*, ENTCS 29.
- [9] T. T. Hildebrandt. *Categorical Models for Concurrency: Independence, Fairness and Dataflow*. BRICS Dissertation Series DS-00-1, 2000.
- [10] T. T. Hildebrandt, P. Panangaden, and G. Winskel. Relational semantics of non-deterministic dataflow. In *Proc. CONCUR'98*, LNCS 1466.
- [11] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21:666–677, 1978.
- [12] D. J. Howe. Proving congruence of bisimulation in functional programming languages. *Information and Computation*, 124(2):103–112, 1996.
- [13] B. Jacobs. Semantics of weakening and contraction. *Annals of Pure and Applied Logic*, 69:73–106, 1994.
- [14] A. Joyal, M. Nielsen, and G. Winskel. Bisimulation from open maps. *Information and Computation*, 127:164–185, 1996.
- [15] A. Joyal and I. Moerdijk. A completeness theorem for open maps. *Annals of Pure and Applied Logic*, 70:51–86, 1994.
- [16] G. Kahn. The semantics of a simple language for parallel programming. *Information Processing*, 74:471–475, 1974.
- [17] G. M. Kelly. *Basic concepts of enriched category theory*. London Math. Soc. Lecture Note Series 64, CUP, 1982.
- [18] A. Levy. *Basic Set Theory*. Springer-Verlag, 1979.
- [19] S. Mac Lane and I. Moerdijk. *Sheaves in Geometry and Logic*. Springer-Verlag, 1992.
- [20] M. Nygaard. *Towards an Operational Understanding of Presheaf Models*. Progress report, University of Aarhus, 2001.
- [21] M. Nygaard and G. Winskel. Linearity in process languages. In *Proc. LICS'02*.
- [22] M. Nygaard and G. Winskel. A higher-order process language. In *Proc. CONCUR'02*.
- [23] D. Park. Concurrency and automata on infinite sequences. In *Proc. 5th GI Conference*, LNCS 104, 1981.
- [24] R. Milner. *A Calculus of Communicating Systems*. LNCS 92, 1980.
- [25] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
- [26] G. Winskel. Event structures. LNCS 255, 1987.
- [27] G. Winskel. A presheaf semantics of value-passing processes. In *Proc. CONCUR'96*, LNCS 1119.
- [28] G. Winskel and M. Nielsen. Models for concurrency. In *Handbook of Logic in Computer Science*, vol 4, OUP, 1995.