Ralf Schweimeier

Department of Computing, City University, London, UK
ralf@soi.city.ac.uk

A categorical framework for typing CCS-style process communication

Abstract

Category theory has proved a useful tool in the study of type systems for sequential programming languages. Various approaches have been proposed to use categorical models to examine the type structures appropriate to concurrent systems. In this paper, we outline some of these approaches, such as interaction categories, and argue that they are not appropriate to model the handshake communication mechanism as used e.g. in CCS or the π-calculus. We propose an alternative general categorical framework for examining the type structure of such systems, and exhibit its categorical structure, which is similar to that of existing approaches. We then examine in detail an instance of this framework, based on a simple fragment of CCS. We prove that it is isomorphic to a syntactic category constructed from a process algebra similar to CCS, with a fusion operator, as in the fusion calculus. Thus, we make explicit some of the type structure implicitly present in such a process algebra.

1 Introduction

For sequential programming languages, categorical methods have proved very useful in the study of type structures [11]. There have been some attempts at utilising category theory for the study of concurrent systems. The general idea, as advocated e.g. in [2], is to model types as objects and processes as morphisms. In this setting, constructions in the category correspond to type constructors and process constructors, providing methods for composing processes according to the type discipline. The crucial operation in concurrent systems, interaction, is modelled by the primitive operation in a category, morphism composition. Thus, viewing a process as a morphism $p : A \rightarrow B$, the objects $A$ and $B$ represent the interface of a process, split into two parts. Composition of morphisms $p : A \rightarrow B$ and $q : B \rightarrow C$ yields a process $p; q : A \rightarrow C$ which consists of the processes interacting on the interface of type $B$, which is then hidden from the outside world. This operation corresponds to parallel composition with hiding/restriction in process calculi like CCS [24].

In a graphical presentation of this setting, a process as a morphism $p : 
Schweimeier

(1) Process (2) Composition (3) Tensor product

\[ p_{AB}^{p;q} \]

\[ p_{AC}^{B} \]

\[ pq \]

\[ p \otimes q \]

\[ A \quad p \quad B \]

\[ A \quad p \quad B \quad q \quad C \]

\[ C \quad q \quad D \]

Fig. 1. Graphical view of processes

\( A \rightarrow B \) is viewed as a graph with incoming edges representing the type \( A \) and outgoing edges representing \( B \), as shown in Fig. 1(1). In this presentation, edges represent wires, or channels, on which a process may interact with other processes. The composition of processes \( p : A \rightarrow B \) and \( q : B \rightarrow C \) may be viewed graphically as in Fig. 1(2). In order for two processes to interact, they are connected by their matching interfaces of type \( B \), and in the resulting process \( p; q : A \rightarrow C \) this interface is hidden from the outside world.

The composition of processes without interaction between them is modelled by a tensor product, yielding monoidal structure on the category. For morphisms \( p : A \rightarrow B \) and \( q : C \rightarrow D \), the tensor product \( p \otimes q : A \otimes C \rightarrow B \otimes D \) can be viewed graphically as in Fig. 1(3). The tensor product may be seen as a pairing operation, which groups together two processes without interaction between them, and makes both their interfaces available to the outside world.

There may be other interesting structure, such as duality, expressing a symmetry between the two sides of the interface; or compact closed structure [20,2] or a trace [19,1,36,26] admitting the formation of cyclic networks of processes.

There have been various approaches towards process categories along these lines. Interaction categories [2] are based on a process calculus similar to Synchronous CCS [23], and have a rich type structure, allowing, for example to specify deadlock-freedom [3]. A similar category, based on transition systems of a CCS-like process calculus is introduced in [36] to study the structure of asynchronous processes. Action calculi [27] provide a general setting in which to study a variety of sequential or concurrent calculi, based on the fundamental notions of names and controls. The controls are the basic building blocks of processes, while the names provide points of interaction. Tile models [14] are a general categorical model with a similar intention to action calculi: to give a unifying framework to model a variety of process calculi; they are based on a general notion of transition system. These categorical models all have in common a symmetric monoidal structure, enriched by other structure such as temporal structure to model the dynamics of a process [2], or traced monoidal [19], compact closed [20] or similar structure [7,8] to model the connectivity of processes.

In this paper, we will briefly introduce some of these categorical frameworks
for concurrent systems, and argue that they are inadequate for modelling the handshake communication mechanism found in languages such as CCS [24], the π-calculus [28], or Concurrent ML [31]. We will then motivate and present an alternative categorical framework in which to model types for CCS-style communication. The model is general and should allow the modelling of various process languages, as well as different kinds of type information. We study in detail a particular instance of the model, based on a fragment of the process algebra CCS. In particular, we show that this model arises as a syntactic category of a CCS-style process algebra with fusions, as in the fusion calculus [29].

The structure of this paper is as follows: in the next section, we present some categorical frameworks for processes, and outline why we consider them inappropriate to model CCS-style communication. We then present an alternative general categorical model for processes with type information. In Section 3, we present an instance of this model based on a simple fragment of CCS. We then introduce a process algebra with fusions in Section 4, from which we construct a syntactic category in Section 5, and show it to be isomorphic with the instance of the general model.

2 A Categorical Model for Typing Processes

2.1 Motivation

We would like to define a category of CCS processes, with type information expressing the communication capabilities of a process. In line with interaction categories [2] and the categorical model for asynchronous processes of [36], an obvious approach would be the following:

• Objects are finite sets.
• Morphisms $A \rightarrow B$ are CCS terms whose free names are in $A + B$, viewed up to an appropriate form of bisimilarity.
• Composition of morphisms $p : A \rightarrow B$ and $q : B \rightarrow C$ is given by parallel composition plus restriction: $p; q = (p | q) \setminus B$.

The composition is indeed associative, up to strong bisimilarity, but what are the identities?

In interaction categories of [2], communication is synchronous, as in SCCS [23]. Thus, an identity morphism is a process that continually offers to do the same action on both sides of its interface – it can be seen as a buffer that immediately sends on any message it receives. Because it is synchronous, the receive and send action happen at the same time, and so it cannot be distinguished whether a message was sent through the buffer or not. There is an asynchronous version of an interaction category, where processes are allowed to idle, thus encoding asynchronous processes in a synchronous calculus, similar to asynchronous SCCS [23]. However, the asynchronous interaction cat-
category still contains synchronous processes; in particular, the identities are synchronous buffers.

In an asynchronous setting such as in CCS, however, a buffer will not generally work as an identity for composition. For example, in the setting above, one could try to define the identity morphism \( id_A : A \rightarrow A \) as a CCS process \( B \) with free names in \( A + A \) by the following guarded recursive equation:
\[
B = \sum_{a \in A} a_1.\overline{a}_2.B + a_2.\overline{a}_1.B,
\]
where \( a_1 \) and \( a_2 \) are the two copies of \( a \) in \( A + A \). However, \( B \) fails to be an identity for certain processes. For example, composition with the \( \text{nil} \) process yields:
\[
(\sum_{a \in A} a_1.\overline{a}_2.B + a_2.\overline{a}_1.B | \text{nil})\backslash A_2,
\]
where \( A_2 = \{a_2 | a \in A\} \). This is not equal to \( \text{nil} \), however, because it can perform any action \( a_1 \), for \( a \in A \).

In [36], a category is obtained by defining the identities to be buffers, and restricting the sets of morphisms to those processes \( p \) that are buffered, i.e. \( B;p:B \). The buffers are then obviously identities. In [35], axioms are given to classify buffered processes. These axioms are quite strong. They require, for example, that a process can at every state do an input transition on each input channel. For first-in-first-out buffers, they require that from each state there is at most one output transition.

So if we want a category of all CCS processes, we need to look for different identities. A natural choice for identities would be the \( \text{nil} \) process: morphism composition is parallel composition, and the \( \text{nil} \) process is the identity for parallel composition. We propose to view processes as separated into an interface part and a behaviour part. Then the structural morphisms such as identities should have an “empty” behaviour, the \( \text{nil} \) process and merely provide operations for connecting other processes. Such a setting should be parametrised on a notion of interface, and a notion of process behaviour, and thus provide a flexible framework in which to give various kinds of type information to different process calculi or models.

A similar idea is present in action calculi [27], and in tile models [14], where morphisms are built from constructors from a given signature, and identities and other structural morphisms are just “wires” for connecting processes, and do not have a computational behaviour. In these models, existing calculi need to be encoded into the framework, by giving a suitable signature. In contrast to this, we would like the morphisms of our category to be given directly by the processes in a calculus or model of processes, as in interaction categories [2], where morphisms are synchronisation trees, or the models for asynchronous processes of [36], where morphisms are transition systems.

2.2 The Model

We will now formally define a category \( \textbf{Proc} \) of processes, as process behaviour plus interface information, as outlined in the previous section. The model is parametrised on a category \( \mathbf{A} \), in which to model types (of interfaces), a category \( \mathbf{P} \), in which to model process behaviours, and a functor \( ar : \mathbf{P} \rightarrow \mathbf{A} \), which
Schweimeier

assigns an *arity*, or type, to each process behaviour. We require the functor to be a co-fibration [17], with appropriate symmetric monoidal structure.

**Definition 2.1** Let \( \pi : \mathcal{C} \to \mathcal{B} \) be a functor. A morphism \( u : X \to Y \) in \( \mathcal{C} \) above \( f = \pi u : A \to B \) in \( \mathcal{B} \) is called co-cartesian, if for any \( v : X \to Z \) in \( \mathcal{C} \) such that \( \pi v = f ; g \) for some \( g : B \to \pi Z \) in \( \mathcal{B} \), there is a unique \( w : Y \to Z \) above \( g \), such that \( u ; w = v \).

A functor \( \pi : \mathcal{C} \to \mathcal{B} \) is called a co-fibration, if for every object \( X \) in \( \mathcal{C} \) and morphism \( f : \pi X \to B \), there exist an object \( Y \) above \( B \) and a co-cartesian morphism \( u : X \to Y \) above \( f \). We call \( u \) a co-cartesian lifting of \( f \) and \( X \), and write \( f^*(X) \) for \( Y \), and \( \bar{f}(X) \) for \( u \). By abuse of language, we sometimes refer to \( f^*(X) \) as a co-cartesian lifting.

**Definition 2.2** Let \((\mathcal{C}, \otimes, I)\) and \((\mathcal{B}, \odot, J)\) be symmetric monoidal categories. A strict symmetric monoidal co-fibration is a functor \( \pi : \mathcal{C} \to \mathcal{B} \) which is a strict symmetric monoidal functor and a co-fibration, and for morphisms \( f : A \to A' \) and \( g : B \to B' \) in \( \mathcal{B} \), and objects \( X \) above \( A \) and \( Y \) above \( B \) we have that \( \bar{f}(X) \odot \bar{g}(Y) : X \otimes Y \to f^*(X) \odot g^*(Y) \) is a co-cartesian lifting of \( f \odot g : A \odot B \to A' \odot B' \).

We are now ready to list the required ingredients for our model:

- A finitely cocomplete category \( \mathcal{A} \), i.e. it has an initial object, binary coproducts, and pushouts. It has a symmetric monoidal structure given by the coproduct \( + \) as the tensor product and the initial object \( 0 \) as the tensor unit.
- A symmetric monoidal category \( \mathcal{P} \). We write \( \oplus \) for the tensor product and \( \circ \) for the tensor unit.
- A strict symmetric monoidal co-fibration \( \text{ar} : \mathcal{P} \to \mathcal{A} \).

The structure on \( \mathcal{A} \) is needed to model an empty interface, the combination of interfaces, and the identification of “wires”. We write \( ! : 0 \to A \) for the unique morphism from the initial object to \( A \), and for morphisms \( f : A \to C \) and \( g : B \to C \) we write the mediating morphism from \( A + B \) to \( C \) as \( [f, g] : A + B \to C \). We mean the symmetric monoidal structure on \( \mathcal{A} \) defined by coproduct and initial object when we require \( \text{ar} \) to be a symmetric monoidal functor.

In \( \mathcal{P} \), the objects are intended to be process behaviours, where the tensor product, written \( P \oplus Q \), is parallel composition, and the tensor unit, written \( 0 \), is the nil process.

The fact that \( \text{ar} \) is monoidal means that the interface of the nil process is empty, and the interfaces in parallel composition are kept disjoint. The co-cartesian lifting \( f^*(P) \) of an object \( P \) with \( \text{ar}(P) = X \) with respect to a morphism \( f : X \to Y \) is intended to model the substitution of the free names in \( P \) (of type \( X \)), according to the map \( f \). We write \( P\{f\} \) for \( f^*(P) \) in order to emphasise this fact.
In Section 3, we construct a strict symmetric monoidal co-fibration from a process algebra similar to CCS. In [37], models of concurrency such as transition systems and synchronisation trees are shown to be co-fibrations, with the co-cartesian lifting given by relabelling. More generally, presheaf models for concurrency [10] are often co-fibrations over a category of sets of labels.

We will now assume a symmetric monoidal co-fibration $ar : \mathbb{P} \to \mathbb{A}$ as given and fixed, and define a category $\text{Proc} = \text{Proc}(ar)$ and its structure.

The category
The objects of the category $\text{Proc}$ are the objects of $\mathbb{A}$. A morphism $p : A \to B$ in $\text{Proc}$ is given by a diagram

\[
\begin{array}{ccc}
B & \to &  \\
\downarrow & & \downarrow \\
A \rightarrow X & \searrow & \\
& & \\
\end{array}
\]

in $\mathbb{A}$ (i.e. a co-span $(s, t) : A \to B$ in $\mathbb{A}$, see [4,6]), and an object $P$ in $\mathbb{P}$ with $ar(P) = X$. In analogy with the notation in type theory, we will write $ar(P) = X$ as $X \vdash P$, and write a morphism in $\text{Proc}$ as the diagram

\[
\begin{array}{ccc}
B & \to &  \\
\downarrow & & \downarrow \\
A \rightarrow X \vdash P & \searrow & \\
& & \\
\end{array}
\]

Intuitively, the morphisms $s$ and $t$ describe how the interface $X$ of the process $P$ is made available to the outside world, via an interface split into two parts, $A$ and $B$.

Cospans naturally form a bicategory [4]; in order to obtain a category, we define equality of morphisms up to isomorphism of $X \vdash P$. Let two morphisms $p, q \in \text{Proc}(A, B)$ be equivalent, $p \sim q$, if there exists an isomorphism $u : P \to Q$ in $\mathbb{P}$, making the following diagram in $\mathbb{A}$ commute:

\[
\begin{array}{ccc}
B & \to &  \\
\downarrow & & \downarrow \\
A \rightarrow X \vdash P & \searrow \ar(u) & \searrow \vdash Q \\
& & \downarrow \ar(u) & \\
& & Y \vdash Q & \\
\end{array}
\]

For an object $A$, the identity morphism $id_A$ in $\text{Proc}$ is given by

\[
\begin{array}{ccc}
A & \to &  \\
\downarrow & & \downarrow \\
A \rightarrow A \vdash \emptyset \{!A\} & \searrow & \\
& & \\
\end{array}
\]
where $\mathbb{O}$ is the tensor unit in $\mathbb{P}$ with $ar(\mathbb{O}) = 0$, and $!_A$ is the unique morphism in $A$ from 0 to $A$. Thus, an identity morphism has the “empty” behaviour $\mathbb{O}$, and connects the two parts of the interface via the identities $id_A$.

Morphism composition is defined by the following diagram, where the square is a pushout of the morphisms $t$ and $s'$ in $A$, and $[e_1, e_2]$ is the mediating morphism from the coproduct $X + Y$ to $Z$. Note that $ar(P \otimes Q) = X + Y$ because $ar$ is strict monoidal. Intuitively, composition is defined as “interaction”, identifying those parts of the interfaces $X$ and $Y$ which are mapped to by $s'$ and $t$, and renaming the channel names in $P$ and $Q$ accordingly. This “identification” is modelled by a pushout in $A$, and the “renaming” is modelled by a co-cartesian lifting.

\[
p : A \rightarrow B \quad q : B \rightarrow C \quad p; q : A \rightarrow C
\]

\[
\begin{array}{ccc}
B \xrightarrow{s'} Y \vdash Q & & B \xrightarrow{s'} Y \vdash Q \\
\downarrow t & & \downarrow e_2 \\
A \xrightarrow{\tau} X \vdash P & & A \xrightarrow{\tau} X \vdash P \\
& & \vdash Z \vdash (P \otimes Q)\{[e_1, e_2]\}
\end{array}
\]

**Monoidal structure**

The tensor unit is defined to be the initial object 0 in $A$. On objects, the tensor product is defined by the coproduct: $A \otimes B = A + B$. On morphisms, the tensor product is defined by the following diagram.

\[
p : A \rightarrow B \quad q : C \rightarrow D \quad p \otimes q : A \otimes C \rightarrow B \otimes D
\]

\[
\begin{array}{ccc}
B & & B + D \\
\downarrow t & & \downarrow s' + t' \\
A \xrightarrow{\tau} X \vdash P & & A + C \xrightarrow{\tau} X + Y \vdash P \otimes Q
\end{array}
\]

The structural isomorphisms for associativity, left unit, and symmetry of the tensor product are defined by the following diagrams, where the morphisms $\alpha_{ABC}, \rho_A$ and $\sigma_{AB}$ are the structural isomorphisms which define the symmetric monoidal structure in $A$ in terms of the coproduct and initial object, and $!$ denotes the unique morphism from 0.
Bimonoids

The category **Proc** has some further structure, similar to interaction categories [2], the categorical models of asynchronous processes [36], action calculi [27,15,38], and tile models [14]. We will now define this structure; for brevity, we write as if \( C \) was strict monoidal and omit the associativity and unit isomorphisms.

**Definition 2.3** Let \( C \) be a symmetric monoidal category. A **commutative monoid** in \( C \) is an object \( A \) in \( C \) together with morphisms \( \otimes \): \( A \otimes A \to A \) and \( \nu \): \( I \to A \) satisfying axioms (1)–(3) below. Dually, a **commutative comonoid** is an object \( A \) together with morphisms \( \otimes \): \( A \to A \otimes A \) and \( \omega \): \( A \to I \) satisfying axioms (4)–(6). A **commutative bimonoid** is an object which is a commutative monoid and a commutative comonoid.

\[
\begin{align*}
(1) & \ (\triangleright_A \otimes id_A) \triangleright_A = (id_A \otimes \triangleright_A) \triangleright_A = \triangleright_A; \ (\triangleright_A \otimes \nu) = (id_A \otimes \triangleright_A) \\
(2) & \ (\nu \otimes id_A) \triangleright_A = \triangleright_A \\
(3) & \ \sigma_{AA} \triangleright_A = \triangleright_A \\
(4) & \ (\triangleleft_A \otimes id_A) \triangleleft_A = (id_A \otimes \triangleleft_A); \ (\triangleleft_A \otimes \nu) = (id_A \otimes \triangleleft_A) \\
(5) & \ (\omega \otimes id_A) = \omega_A \\
(6) & \ \sigma_{AA} \triangleleft_A = \triangleleft_A
\end{align*}
\]

**Definition 2.4** A **category with discrete bimonoids** is a symmetric monoidal category \( C \) where every object is a commutative bimonoid, satisfying the following coherence conditions.

\[
\begin{align*}
(7) & \ \triangleright_{A \otimes A} = (id_A \otimes \sigma_{AA} \otimes id_A); \ (\triangleright_A \otimes \triangleright_A) \\
(8) & \ \triangleleft_{A \otimes A} = (\triangleleft_A \otimes \triangleleft_A); \ (\triangleleft_A \otimes \nu_A) \\
(9) & \ \nu_{A \otimes A} = \nu_A \otimes \nu_A \\
(10) & \ \omega_{A \otimes A} = \omega_A \otimes \omega_A \\
(11) & \ (\triangleleft_A \otimes id_A); \ (id_A \otimes \triangleright_A) = \triangleright_A; \ (\triangleleft_A \otimes \nu_A) \\
(12) & \ \triangleleft_A; \ (\triangleright_A \otimes id_A) > id_A
\end{align*}
\]

It follows from Axiom (11) that categories with discrete bimonoids are compact closed [20], where the duality is the identity on objects, and the unit and counit are given by \( \nu_A; \triangleleft_A \) and \( \triangleright_A; \omega_A \), respectively. Consequently, they are also traced monoidal categories [19], with the trace defined in terms of the compact closed structure.

Similar categorical structures are examined in [7,8]. The most similar to ours are the match-share categories, which are almost categories with discrete bimonoids, except that they lack the \( \nu \) and \( \omega \) morphisms; and the part-monoidal categories, which are categories with discrete bimonoids, and additionally satisfy the axiom \( \nu_A; \omega_A = id_A \). Symmetric action calculi [38] have a very similar categorical structure. They are, in fact, part-monoidal categories.

A discrete bimonoid structure on **Proc** is defined by the following diagrams.
Schweimeier

\[ \triangleleft_A : A \rightarrow A \otimes A \quad \omega_A : A \rightarrow I \quad \triangleright_A : A \otimes A \rightarrow A \quad \nu_A : I \rightarrow A \]

\[ A + A \xrightarrow{[id, id_A]} A \vdash O \{!_A\} \quad A \xrightarrow{id_A} A \vdash O \{!_A\} \quad A \xrightarrow{id_A} 0 \vdash O \{!_A\} \quad A \xrightarrow{id_A} A \vdash O \{!_A\} \]

**Theorem 2.5** Proc is a symmetric monoidal category with discrete bimonoid structure.

See [33] for the proof.

### 3 A category of CCS processes

In this section, we define an instance of the general framework presented above, by defining a category of process behaviours \( P \) consisting of terms in a process algebra based on a small fragment of CCS [24], and give these process terms arities in the category \( \text{FinSet} \) of finite sets and functions, thus defining a symmetric monoidal co-fibration \( ar_{CCS} : P \rightarrow A \) as required in the definition of Proc.

**Signature**

We give a multi-sorted signature for CCS terms by types \( N, A \) and \( P \) for names, actions and processes, and function symbols

- \( \text{in} : (N) \rightarrow A \) Input action
- \( \text{out} : (N) \rightarrow A \) Output action
- \( \tau : () \rightarrow A \) Silent action
- \( \text{0} : () \rightarrow P \) The nil process
- \( \cdot : (A, P) \rightarrow P \) Prefixing
- \( : (P, P) \rightarrow P \) Parallel composition
- \( + : (P, P) \rightarrow P \) Non-deterministic choice

As usual, we write the binary operators in infix form. In examples, we will sometimes use the more familiar \( a.p \) and \( \bar{a}.p \) for \( \text{in}(a).p \) and \( \text{out}(a).p \), respectively. We assume that substitution binds tightest, and prefixing binds tighter than parallel composition and choice. The definition of the set of free variables of type \( T \) in a term \( t \), \( \text{fv}_T(t) \), is standard, as is the substitution \( t\{f\} \) of free variables of type \( T \) in a term \( t \) according to a function \( f : X \rightarrow Y \) (where \( \text{fv}_T(t) \subseteq X \)), simultaneously substituting \( f(a) \) for each \( a \in \text{fv}_T(t) \).

**Equations**

We impose a set of equations \( E_{CCS} \) on process terms, given in Fig. 2. These are just the equations for making parallel composition into a commutative
monoid multiplication with 0 as the unit – this is enough to define a symmetric monoidal structure for the co-fibration.

**Algebra**

The free CCS-algebra is given by the term algebra. We are only interested in processes, i.e. terms of type \( P \), and will not allow any free variables except those of type \( N \), i.e. free names. So we work with the term algebra \( T_C(X) \) of terms of type \( P \) with free names in a given set \( X \) defined as follows:

\[
T_C(X) = \{ t | t \text{ is a CCS-term of type } P, \text{fv}_N(t) \subseteq X, \text{ and } \text{fv}_A(t) = \text{fv}_P(t) = \emptyset \}
\]

and

\[
\tilde{T}_C(X) = \{ \left[ t \right] =_C | t \in T_C(X) \}, \text{ where } \left[ t \right]_C \text{ is the } =_C\text{-equivalence class of } t.
\]

In this way, we can view a process in \( T_C(X) \) as given in a context, \( X \vdash p \), where the free names of \( p \) are contained in \( X \). Clearly, the process algebra CCS \([24]\), with (strong or weak) bisimilarity as the congruence on terms, is a \((CCS,E_{CCS})\)-algebra.

We can extend \( T_C \) and \( \tilde{T}_C \) to functors \( \text{FinSet} \rightarrow \text{Set} \): For \( f : A \rightarrow B \) in \( \text{FinSet} \) define \( T_C(f) : T_C(A) \rightarrow T_C(B) \) by \( T_C(f)(t) \overset{\text{def}}{=} \{ f \} \). Similarly for \( \tilde{T}_C \) – it is easy to check that substitution is well-defined on equivalence classes of terms \(^1\).

Because a functor \( \mathbb{C} \rightarrow \text{Set} \) is a (discrete) co-indexed category, we can apply the Grothendieck construction \([17]\) to obtain a (discrete) co-fibration. We perform the construction for \( \tilde{T}_C \), and use the CCS-algebra structure to make the co-fibration into a symmetric monoidal functor.

**Definition 3.1** The category \( \int \tilde{T}_C \) is given by

- **Objects** Pairs \((A,p)\), where \( A \) is a finite set, and \( p \in \tilde{T}_C(A) \). We also write \( A \vdash p \).
- **Morphisms** A morphism \( f : (A,p) \rightarrow (B,q) \) is given by a function \( f : A \rightarrow B \) such that \( \tilde{T}_C(f)(p) = q \), i.e. \( p(f) =_C q \). Morphism composition and identities are inherited from \( \text{FinSet} \).
- **Tensor product** On objects, the tensor product is defined as

\[
(A,p) \otimes (B,q) \overset{\text{def}}{=} A + B, p\{in_1 \} | q\{in_2 \}
\]

Given morphisms \( f : (A,p) \rightarrow (A',p') \) and \( g : (B,q) \rightarrow (B',q') \), \( f \otimes g : A \otimes B \rightarrow A' \otimes B' \) is defined as \( f + g : A + B \rightarrow A' + B' \).

\(^1\) Unfortunately, this does not quite work for CCS bisimilarity; for example, \( \bar{a} | b \sim \bar{a}.b+b.\bar{a} \), but \( (\bar{a} | b)\{a/b\} \neq (\bar{a}.b+b.\bar{a})\{a/b\} \). In this case, we need a stronger form of bisimilarity, which respects substitution, such as the hyperbisimilarity of fusion calculus \([29]\).
The tensor unit $I$ is given by the object $(\emptyset, 0)$. Associativity, unit, and symmetry isomorphisms are inherited from the coproduct structure on $\text{FinSet}$.

- **Co-Fibration** There is an obvious strict symmetric monoidal functor $ar_{CCS} : \int \tilde{T}_C \rightarrow \text{FinSet}$ given by $ar_{CCS}(f : (A, p) \rightarrow (B, q)) = f : A \rightarrow B$.

The functor $ar_{CCS}$ is a discrete co-fibration: Given an arrow $f : A \rightarrow B$ in $\text{FinSet}$ and an object $(A, p)$ in $\int \tilde{T}_C$, there is a unique arrow in $\int \tilde{T}_C$ above $f$ with domain $(A, p)$, given by $f : (A, p) \rightarrow (A, p\{f\})$. It follows from the equations in Fig. 2 that $\int \tilde{T}_C$ is symmetric monoidal, and that $ar_{CCS}$ is a symmetric monoidal co-fibration.

**Proposition 3.2** The functor $ar_{CCS} : \int \tilde{T}_C \rightarrow \text{FinSet}$ is a symmetric monoidal co-fibration.

We can now use the functor $ar_{CCS} : \int \tilde{T}_C \rightarrow \text{FinSet}$ to define an instance $\text{Proc}_{CCS}$ of the process model defined in the previous section. A morphism in $\text{Proc}_{CCS}$ is then given by a diagram

\[
\begin{array}{ccc}
B & \rightarrow & A \\
\downarrow & & \downarrow s \\
A & \xrightarrow{p} & X
\end{array}
\]

where $A$, $B$ and $X$ are finite sets (of channel names), and $p$ is a CCS process with free names in $X$. The functions $s$ and $t$ describe how the free names of $p$ are made available to the outside world: each name $a \in A$ represents the name $s(a)$ in $p$, and each name $b \in B$ represents the name $t(b)$ in $p$. Those names in $p$ which are not mapped to by $s$ or $t$ are hidden. Some of the names in $X$ may not occur in $p$, but may be made available in the interfaces $A$ and $B$.

**Example 3.3** The CCS processes $a.x.b$ and $\bar{x}.\bar{b}$ can be made into morphisms $p : \{a, b, c\} \rightarrow \{x, b\}$ and $q : \{x, b\} \rightarrow \{b\}$ in $\text{Proc}_{CCS}$ as in the following diagram, which also shows their composition $p; q : \{a, b, c\} \rightarrow \{b\}$.

\[
\begin{array}{ccc}
\{b\} & \rightarrow & \{x, b\} \\
\downarrow & & \downarrow \quad \quad \quad \downarrow \\
\{a, b, c\} & \xrightarrow{\bar{x}.\bar{b}} & \{a, b, x\} \xrightarrow{a.x.b} \{a, b, x\} \xrightarrow{a.x.b | \bar{x}.\bar{b}}
\end{array}
\]

where $c \mapsto a$ denotes the function which maps $c$ to $a$, and all other elements to themselves. Then, in the composition $p; q$, the name $a$ is made available to the left-hand interface twice, as both $a$ and $c$; the name $b$ is made available to both the left-hand and right-hand interface, but may also perform an internal
communication; and \( x \) is hidden from the outside world and may only be used for an internal communication. \( \square \)

4 A process algebra for CCS with fusions

We will now define a process algebra \( FCCS \), which extends \( CCS \) by restriction of names, and a construct for equating names, similar to fusions in fusion calculi \([29,16]\). We will establish a normal form result, which shows that every process can be written as a \( CCS \)-process, with some “global” fusions and restrictions, resembling the action of a co-span on a set of names, as in the category \( \text{Proc} \) of Section 2. This will enable us, in the following section, to define a syntactic category with structure like \( \text{Proc} \), and eventually show it to be isomorphic to \( \text{Proc}_{CCS} \).

Signature

The signature for \( FCCS \) is a multi-sorted binding signature \([13]\). The types are \( N, A \) and \( P \) for names, actions and processes; the function symbols are

\[
\begin{align*}
\text{in} : (N) & \rightarrow A \quad \text{Input action} & 0 : () & \rightarrow P \quad \text{The nil process} \\
\text{out} : (N) & \rightarrow A \quad \text{Output action} & . : (A,P) & \rightarrow P \quad \text{Prefixing} \\
\tau : () & \rightarrow A \quad \text{Silent action} & \mid : (P,P) & \rightarrow P \quad \text{Parallel composition} \\
\quad & & + : (P,P) & \rightarrow P \quad \text{Non-deterministic choice} \\
\nu : ([N]P) & \rightarrow P \quad \text{Restriction} \\
= : (N,N) & \rightarrow P \quad \text{Fusion}
\end{align*}
\]

where the notation \( \nu : ([N]P) \rightarrow P \) indicates that \( \nu \) is a binding operator which takes one argument of type \( P \), and binds one variable of type \( N \) in that argument.

The definition of the set \( \text{fv}_T(t) \) of free variables of type \( T \) in a term \( t \) is standard. We write \( \text{fn}(t) \) for the set of free names \( \text{fv}_N(t) \). For a term \( t \) with \( \text{fn}(t) \subseteq X \), and a function \( f : X \rightarrow Y \), we write \( t\{f\} \) for the capture-avoiding simultaneous substitution of names \( f(a) \) for \( a \).

Equations

We impose a set of equations on \( FCCS \)-terms, given in Fig. 3. We write \( E_{FCCS} \) for the set of equations, and \( =_{F} \) for the congruence generated by it. The equations are those from \( CCS \), and some axioms ensuring that fusions act as an equivalence relation on names, and that fusions and restrictions are “global” in a sense to be made precise below. A similar equational theory is given as a structural congruence for a fusion calculus in \([16]\).
that play a particular role. The axioms (=-Refl), (=-Symm), and (=-Trans) ensure those terms which consist only of a parallel composition of fusions generate an equivalence relation on the free names of a process. Those terms which consist only of a parallel composition of fusions generate an equivalence relation on the free names of a process – fusions generate an equivalence relation on the free names of a process. Then, \( \tilde{T}_F \) is the free \((FCCS, E_{FCCS})\)-algebra. See \[33\] for the technical details.

The intuition behind the fusion construct is the identification of names in a process – fusions generate an equivalence relation on the free names of a process. Those terms which consist only of a parallel composition of fusions play a particular role. The axioms (=-Refl), (=-Symm), and (=-Trans) ensure that \( =_F \)-equivalence classes of such terms are in one-to-one correspondence.

### Algebra

The free \( FCCS \)-algebra is given by the term algebra for \( FCCS \). For a finite set \( A \) (of names), define \( T_F(A) \) as the set of all \( FCCS \)-terms with free names in \( A \), up to \( \alpha \)-equivalence: \( T_F(A) = \{ [t]_{\equiv_\alpha} \mid t \text{ is a } FCCS \text{-term of type } P, f_{\nu_N}(t) \subseteq X, \text{ and } f_{\nu_A}(t) = f_{\nu_P}(t) = \emptyset \} \). Similarly, the term algebra for \( FCCS \) satisfying the equations \( E_{FCCS} \) is given by the terms modulo \( =_F \), the congruence generated by \( E_{FCCS} \): \( \tilde{T}_F(A) = \{ [t]_{=_{\tilde{T}}_F} \mid t \in T_F(A) \} \).

Formally, an algebra for a binding signature is a presheaf \( \text{FinSet} \to \text{Set} \) \[13\]. We can make \( T_F \) and \( \tilde{T}_F \) into presheaves, defining the action on morphisms by name substitution. Then, \( T_F \) is the free \( FCCS \)-algebra, and \( \tilde{T}_F \) is the free \((FCCS, E_{FCCS})\)-algebra. See \[33\] for the technical details.

The intuition behind the fusion construct is the identification of names in a process – fusions generate an equivalence relation on the free names of a process. Those terms which consist only of a parallel composition of fusions play a particular role. The axioms (=-Refl), (=-Symm), and (=-Trans) ensure that \( =_F \)-equivalence classes of such terms are in one-to-one correspondence.
with equivalence relations, via

\[ [a_1 = b_1] \mid \cdots \mid [a_n = b_n] \leftrightarrow \{(a_1, b_1), \ldots, (a_n, b_n)\}^* \]

where \( R^* \) denotes the reflexive, symmetric and transitive closure of \( R \). From now on, we will therefore often identify equivalence relations on \( A \) and parallel compositions of fusions in \( T_F(A) \), and write, e.g. \( R | p \) for \([a_1 = b_1] | [a_n = b_n] \mid p\), where \( R = \{(a_1, b_1), \ldots, (a_n, b_n)\}^* \).

Fusions are intended to identify names in a process. We formalise this in the following proposition, which states that names related by the equivalence relation corresponding to a parallel composition of fusions can be substituted for each other.

**Proposition 4.1** Let \( p \in T_F(A) \) and \( f : A \to A \) such that \((a, f(a)) \in R \) for all \( a \in A \). Then \( R | p =_F R | p f \).

In particular, \([a = b] | p =_F [a = b] | p^{(b/a)}\).

**Normal forms**

We will now state a theorem establishing that every FCCS-term has a normal form where all the fusions and restrictions are at top level. This result is crucial for the translation from the syntactic category based on FCCS to the category \( \text{Proc}_{CCS} \) of Section 3: for a normal form in \( T_F \), we obtain a term in \( T_C \) by removing the top level fusions and restrictions. The fusions and restrictions are then modelled by a co-span.

**Theorem 4.2** There exists a natural transformation \( nf : T_F \to T_F \) whose component \( nf_A \) assigns a normal form to each \( p \in T_F(A) \) such that

\[ p =_F nf_A(p) \equiv R \mid (\nu X)p' \]

for some equivalence relation \( R \subseteq A^2 \), some finite set \( X \) such that \( A \cap X = \emptyset \), and some \( p' \in T_C(A + X) \).

Furthermore, if \( R \mid (\nu X)p' =_F S \mid (\nu Y)q' \), where \( p' \in T_C(A + X) \) and \( q' \in T_C(A + Y) \), then \( R = S \), and there exists a bijection \( \sigma : X \to Y \) such that \( p'\{[-]_R + \sigma\} \equiv_{C} q'\{[-]_R + \text{id}_Y\} \).

See [33] for the proof. The theorem contains several statements: 1. A normal form \( R \mid (\nu X)p' =_F p \) exists for each term \( p \in T_F(A) \). 2. The function \( nf_A \) assigns one such normal form to each \( p \in T_F(A) \). It is obtained by using the axioms of Fig. 3 to lift fusions and restrictions to the top level. 3. The statement that \( nf \) is a natural transformation means that normal forms are preserved by substitution. 4. The last statement concerns uniqueness of the normal form: in a normal form, the equivalence relation \( R \) is unique, and the CCS term \( p' \) is unique up to \( =_C \), and renaming of the bound names \( X \), and of \( R \)-equivalent names in \( A \).
Example 4.3 We illustrate how the normal form of a \( \text{FCCS} \)-term is computed.

\[
(\nu x)(a.x \mid (\nu y)([x = y] \mid \bar{y}.\bar{c}) \mid (\nu z)(z.b \mid [c = z] \mid [a = b]))
=_{F} (\nu x y z)(a.x \mid [x = y] \mid \bar{y}.\bar{c} \mid z.b \mid [c = z] \mid [a = b]) \quad (\nu\text{-Scope})
=_{F} (\nu x y z)(a.x \mid [x = y] \mid \bar{x}.\bar{c} \mid c.b \mid [c = z] \mid [a = b]) \quad (\text{Proposition 4.1})
=_{F} (\nu x)(a.x \mid \bar{x}.\bar{c} \mid c.b \mid [a = b]) \quad (=\text{-Discard})
=_{F} [a = b] \mid (\nu x)(a.x \mid \bar{x}.\bar{c} \mid c.b) \quad (\nu\text{-Scope}) \quad \Box
\]

Note how different fusions are lifted outside a restriction: in \([a = b]\), both names are free, so the fusion can be lifted outside the restriction; in \([c = z]\), \(c\) is free and \(z\) is bound, so the fusion and the restriction on \(z\) get discarded using (= Discard); and in \([x = y]\), both names are bound – the two names collapse into one.

5 A syntactic category based on CCS with fusions

From the term algebra for \( \text{FCCS} \), we will now define a syntactic category with structure similar to \( \text{Proc} \), and show that it is in fact isomorphic to \( \text{Proc}_{\text{CCS}} \) defined in Section 3.

The terms in the process algebra \( T_{F} \) are given in a context \((A \text{ for terms in } T_{F}(A))\). So in analogy with the motivation in the introduction, a morphism \( p : A \rightarrow B \) is a process \( p \) with its interface (the context) split into two parts \( A \) and \( B \): \( A + B \vdash p \). Composition is given by parallel composition plus restriction. Along similar lines, it is shown in [12] how interaction categories may be constructed as syntactic categories of certain process calculi.

Assuming that \( A, B, \) and \( C \) are disjoint, we define composition of \( A + B \vdash p \) and \( B + C \vdash q \) as \( A + C \vdash (\nu B)(p \mid q) \). We need to be slightly more careful, however, because we want to identify different copies of a name in a disjoint union, e.g. the two copies of \( A \) in the coproduct \( A + A \), as in the definition of the identity morphism in \( \text{Proc} \). So we use injections to make sure that the sets involved are mutually disjoint. We choose coproducts in \( \text{FinSet} \) as \( A_{1} + \cdots + A_{n} = \text{in}_{1}(A_{1}) \cup \cdots \cup \text{in}_{n}(A_{n}) \) for \( n \geq 2 \). Then the injections are given by \( \text{in}_{i} : A_{i} \rightarrow A_{1} + \cdots + A_{n} \), for each \( 1 \leq i \leq n \). In order to facilitate the definition of the syntactic category, we extend the restriction and fusion operations, and substitution to sets of names as follows.

\[
[g(A) = h(A)] \overset{\text{def}}{=} [g(a_{1}) = h(a_{1})] \mid \cdots \mid [g(a_{n}) = h(a_{n})]
(\nu f(A))p \overset{\text{def}}{=} (\nu f(a_{1})) \cdots (\nu f(a_{n}))p
p\{g(A) / f(A)\} \overset{\text{def}}{=} p\{g(a_{1}), \ldots, g(a_{n}) / f(a_{1}), \ldots, f(a_{n})\}
\]

61
where \( A = \{a_1, \ldots, a_n\} \), and \( f : A \to B \) and \( g : A \to C \) are injections. We write \((\nu A)p\) for \((\nu \text{id}_A(A))p\).

**Definition 5.1** The category \textbf{Synt} is given by the following data:

- **Objects**: Finite sets.
- **Morphisms**: A morphism \( p : A \to B \) is an \textit{FCCS} term \( A + B \vdash p \), viewed up to equality in \( \tilde{T}_F(A + B) \).
- **Identities**: The identities \( \text{id}_A : A \to A \) are given by

\[
A + A \vdash [\text{in}_1(A) = \text{in}_2(A)]
\]

- **Composition**: Given morphisms \( A + B \vdash p : A \to B \) and \( B + C \vdash q : B \to C \), their composition \( p; q : A \to C \) is defined as

\[
A + C \vdash (\nu \text{in}_3(B))(p\{\text{in}_3(B)/\text{in}_2(B)\} \mid q\{\text{in}_3(B)/\text{in}_1(B)\})
\]

where \( \text{in}_1 : B \to B + C \), \( \text{in}_2 : B \to A + B \) and \( \text{in}_3 : B \to A + C + B \).

- **Tensor**: On objects, the tensor is given by the coproduct: \( A \otimes B \overset{\text{def}}{=} +B \). The tensor unit \( I \) is defined to be the empty set: \( I \overset{\text{def}}{=} \). Given morphisms \( A + B \vdash p : A \to B \) and \( C + D \vdash q : C \to D \), their tensor product \( p \otimes q : A \otimes C \to B \otimes D \) is defined as

\[
(A + C) + (B + D) \vdash p\{\text{in}_{AB}\} \mid q\{\text{in}_{CD}\}
\]

where \( \text{in}_{AB} \) and \( \text{in}_{CD} \) are the obvious injections from \( A + B \) and \( C + D \) into \( (A + C) + (B + D) \).

- **Structural isomorphisms**: The natural isomorphisms defining the symmetric monoidal structure are defined as

\[
\alpha_{ABC} : ((A + B) + C) + (A + (B + C)) \vdash [\text{in}_1(\text{in}_1(A)) = \text{in}_2(\text{in}_1(A))] \mid [\text{in}_1(\text{in}_1(B)) = \text{in}_2(\text{in}_1(B))] \mid [\text{in}_1(\text{in}_1(C)) = \text{in}_2(\text{in}_1(C))]
\]

\[
\rho_A : (A + \emptyset) + A \vdash [\text{in}_1(\text{in}_1(A)) = \text{in}_2(A)]
\]

\[
\sigma_{AB} : (A + B) + (B + A) \vdash [\text{in}_1(\text{in}_1(A)) = \text{in}_2(\text{in}_1(A))] \mid [\text{in}_1(\text{in}_2(B)) = \text{in}_2(\text{in}_1(B))]
\]
• **Bimonoids**: The bimonoid structure in **Synt** is defined as follows:

\[
\begin{align*}
\triangleleft_A : & \quad A + (A + A) \vdash [in_1(A) = in_2(in_1(A))] \mid [in_1(A) = in_2(in_2(A))] \\
\omega_A : & \quad A + \emptyset \vdash 0 \\
\triangleright_A : & \quad (A + A) + A \vdash [in_1(in_1(A)) = in_2(A)] \mid [in_1(in_2(A)) = in_2(A)] \\
\nu_A : & \quad \emptyset + A \vdash 0
\end{align*}
\]

**Theorem 5.2** **Synt** is a symmetric monoidal category with discrete bimonoids.

**Proof.** We prove the axioms for identities and associativity of morphism composition. See [33] for the full proof.

**Identities**

Let \((A + B \vdash p) : A \to B\).

\[
\begin{align*}
id_A ; p = (\nu \; in_3(A))([in_1(A) = in_2(A)] \{i_{in_3(A) / in_2(A)}\} \mid p\{i_{in_3(A) / in_1(A)}\}) \\
= (\nu \; in_3(A))([in_1(A) = in_3(A)] \mid p\{i_{in_3(A) / in_1(A)}\}) \\
= (\nu \; in_3(A))([in_1(A) = in_3(A)] \mid p\{i_{in_3(A) / in_1(A)}\}) \{i_{in_1(A) / in_3(A)}\} \quad \text{(Proposition 4.1)} \\
= (\nu \; in_3(A))([in_1(A) = in_3(A)] \mid p) \\
= (\nu \; in_3(A))([in_1(A) = in_3(A)] \mid p) \quad \text{(\(\nu\)-Scope)} \\
= 0 \mid p \quad \text{(-Discard)} \\
= p \quad \text{(-Unit)}
\end{align*}
\]

**Associativity of composition**

Let \((A + B \vdash p) : A \to B\), \((B + C \vdash q) : B \to C\) and \((C + D \vdash r) : C \to D\). For simplicity, assume that \(A\), \(B\), \(C\) and \(D\) are mutually disjoint, and omit the injections.

\[
\begin{align*}
(p ; q) ; r = (\nu C)((\nu B)(p \mid q) \mid r) \\
= (\nu C)(\nu B)((p \mid q) \mid r) \quad \text{(\(\nu\)-Scope)} \\
= (\nu B)(\nu C)((p \mid q) \mid r) \quad \text{(\(\nu\)-Comm)} \\
= (\nu B)(\nu C)(p \mid (q \mid r)) \quad \text{(|-Assoc)} \\
= (\nu B)(p \mid (\nu C)(q \mid r)) \quad \text{(\(\nu\)-Scope)} \\
= p ; (q ; r)
\end{align*}
\]
We will now define structure-preserving translations between the category \( \text{Proc}_{\text{CCS}} \) and the syntactic category \( \text{Synt} \), and show that they form an isomorphism.

**Translation from \( \text{Proc}_{\text{CCS}} \) to \( \text{Synt} \)**

We define a functor \( ps : \text{Proc}_{\text{CCS}} \to \text{Synt} \). On objects, \( ps \) is the identity:

\[
ps(A) \overset{\text{def}}{=} A.
\]

Let \( P : A \to B \) in \( \text{Proc}_{\text{CCS}} \) be given by the diagram

\[
\begin{array}{c}
B \\
\downarrow \\
A \xrightarrow{s} X \\
\end{array} p
\]

Then define \( ps(P) : A \to B \) as

\[
A + B \vdash (\nu \text{in}_3(X))([\text{in}_1(A) = \text{in}_3(s(A))] \mid [\text{in}_2(B) = \text{in}_3(t(B))] \mid p\{\text{in}_3\})
\]

where the \( \text{in}_i \) are the injections into \( A + B + X \).

Thus, the functions \( s \) and \( t \), which make some names of \( p \) visible to the outside world, are translated into fusions. These fusions correspond to an equivalence relation on \( A + B \) defined by \( s \) and \( t \) as \((a, b) \in (A + B)^2 \mid [s, t](a) = [s, t](b)\). Note that in the translation \( ps(P) \), all names of \( p \) are restricted; however, the fusions make visible exactly those names made visible by \( s \) and \( t \).

**Example 5.3** Consider the morphism \( p : \{a, b, c\} \to \{x, b\} \) and \( q : \{x, b\} \to \{b\} \) in Ex. 3.3. Their translations, and their normal forms according to Thm. 4.2, are given by

\[
\begin{align*}
\{a, b, c, x', b'\} & \vdash (\nu a'' b'' x'')([a = a''] \mid [b = b''] \mid [c = a''] \mid [x' = x''] \mid [b' = b''] \mid a''.x''.b'') \\
& =_F [a = c] \mid [b = b'] \mid a.x.b
\end{align*}
\]

\[
\begin{align*}
\{x, b, b'\} & \vdash (\nu x''b'')(\{x = x''\} \mid [b = b''] \mid [b' = b''] \mid x''.b') \\
& =_F [b = b'] \mid \bar{x}.b
\end{align*}
\]

The composition \( ps(p); ps(q) \) in \( \text{Synt} \) is then given by

\[
\begin{align*}
\{a, b, c, b'\} & \vdash (\nu x''b'')([a = c] \mid [b = b''] \mid a.x''.b] \mid [b'' = b'] \mid x''.b') \\
& =_F [a = c] \mid [b = b'] \mid (\nu x)(a.x.b \mid \bar{x}.b)
\end{align*}
\]

The translation of \( p; q \) is

\[
\begin{align*}
\{a, b, c, b'\} & \vdash (\nu a'' b'' x'')([a = a''] \mid [b = b''] \mid [c = a''] \mid [b' = b''] \mid a''.x''.b'' \mid x''.b') \\
\end{align*}
\]

A simple calculation shows that its normal form is equal to the normal form of \( ps(p); ps(q) \). \( \square \)
Translation from Synt to Proc$_{CCS}$

Now we define a functor $sp : \text{Synt} \to \text{Proc}_{CCS}$. On objects, $sp$ is the identity: $sp(A) \overset{\text{def}}{=} A$. Let $(A + B \vdash p) : A \to B$ be a morphism in $\text{Synt}$. Then, by Thm. 4.2, $p = F(R | (\nu X)(p'))$, where $p'$ is in $T_C(A + B + X)$. We define $sp(A + B \vdash p)$ as the diagram

$$
\begin{array}{ccc}
B & \xrightarrow{\{\nu X\}} & (A + B)/R + X \\
\downarrow^{(m_2(-))_R} & & \downarrow^{R + X \vdash p'\{[-]_R\}} \\
A & \xrightarrow{\{\nu X\}_R} & (A + B)/R + X \vdash p'\{[-]_R\}
\end{array}
$$

In order to translate the FCCS-process $p$, we remove all the fusions and restrictions from $p$ according to Thm. 4.2, obtaining a CCS-process $p'$. Then we model the fusions and restrictions in $p$ by a co-span which “identifies” exactly those names identified by the fusions, and which “hides” exactly the restricted names.

**Example 5.4** The translation by $ps$ of the process $p; q$ in Example 3.3 is $[a = c] \parallel [b = b'] \parallel (\nu X)(a.x.b \mid \bar{x}.\bar{b})$. Translating it back to Proc$_{CCS}$ using $sp$, we obtain

$$
\begin{array}{c}
\{b'\} \\
\downarrow \\
\{a, b, c\} \rightarrow \{a, c\}, \{b, b'\}, x \vdash \{a, c\}.x.\{b, b'\} \mid \bar{x}.\{b, b'\}
\end{array}
$$

This is equal in Proc$_{CCS}$ to $p; q$, via the isomorphism $f : \{a, c\}, \{b, b'\}, x \rightarrow \{a, b, x\}$; $f(\{a, c\}) = a$, $f(\{b, b'\}) = b$, $f(x) = x$. \hfill \Box

The examples illustrate some of the properties of the translations. In fact, the translations form a structure preserving isomorphism between Proc$_{CCS}$ and Synt; this is summarised in the following theorem, whose proof may be found in [33].

**Theorem 5.5** The functors $ps$ and $sp$ preserve the symmetric monoidal and the discrete bimonoid structure, and form an isomorphism between Proc$_{CCS}$ and Synt.

This isomorphism result gives us some insight into the nature of the models obtained by the construction in Section 2. When applied to a fragment of CCS, one obtains a category which is isomorphic to a syntactic category constructed from the same process algebra enriched with restriction and fusions. One would expect that any such category Proc – and possibly any category with discrete bimonoids – is suitable to model restriction and the identification of names.

On the other hand, the result implies that the type structure of Proc is already implicitly present in a process algebra like FCCS. A process calculus
based on CCS with fusions, whose behavioural equivalence satisfies the FCCS axioms, is presented in [33]. Similarly, it should be possible to define a FCCS algebra from the fusion calculus [29] and the explicit fusion calculus [16]. For the latter, there is a strong connection with symmetric action calculi [38], which have similar structure to Proc.

6 Conclusion

We have presented a general model for processes with type information, based on a category of process behaviours, a category of types, and a functor expressing the interface of a process. This category has rich categorical structure, allowing to express the connectivity of processes. We examined in detail an instance of this model based on a fragment of CCS, and showed it to be isomorphic to a syntactic category of a CCS-like process algebra with a fusion operator. Thus, we exhibited the type structure implicitly present in such a process algebra.

Our research was influenced by work on interaction categories [2]; we highlighted some problems that arise when trying to model CCS-style interaction in interaction categories and similar approaches [36], thus motivating our alternative approach.

Action calculi [27], and tile models [14] provide general categorical models for concurrent systems, similar to ours in their aims, and in the categorical structure. Like in our approach, there is a distinction between the behaviour and the type structure (or connectivity) of a process. In our model, however, as in [2;36], a category can be built directly from the terms of a process algebra, or transition systems, factored by some behavioural equivalence; in action calculi or tile models, a process calculus needs to be encoded in the language of the respective model. Our model has a very similar structure to action calculi (in particular the symmetric action calculi [38]) and tile models; it is left for further research to establish a formal connection to these models.

There has been some work on a graphical presentation for action calculi [27], and tile models [7,8], similar to our graphical presentation used in the introduction. In [7,8], certain categories of graphs are shown to be the free categories with certain structure. It should be possible to obtain a similar result for categories with discrete bimonoids. Such a result would give a formal basis to graphical presentations of processes, and provide an intuitive method for reasoning about concurrent programs, by graphical reasoning [5,18,34].

We have presented a concrete instance of our model, which is based on a very simple process calculus based on CCS. In order to prove the generality of our model, more expressive calculi should be fitted into the model, such as the π-calculus [28]. The category of types \( \mathcal{A} \) for the π-calculus will need to take into account the dynamic creation of names, thus leading to more complex type structure. It might be possible to build on research on presheaf models for concurrency [10,9] – they have rich categorical structure, e.g. they are
often co-fibrations over a suitable category of names.

There has been much research into types for process calculi, expressing properties such as the correct use of channel names \([25,30,22]\), deadlock freedom \([21,3]\), access control \([32,40]\), and the dynamic behaviour of processes \([2,39]\). It needs to be investigated if such expressive type information can be fitted into our model, by defining appropriate categories \( \mathcal{A} \) for type information, and showing that they have the necessary categorical structure.

**Acknowledgments**

I would like to thank my D.Phil. supervisors Alan Jeffrey, Marcelo Fiore and Matthew Hennessy for their valuable assistance with this work. Thanks also to Julian Rathke for helpful comments and discussions.

**References**


URL http://fpl.cs.depaul.edu/ajeffrey/premon/


