Specification Structures and Propositions-as-Types for Concurrency*

Samson Abramsky Simon Gay Rajagopal Nagarajan

Department of Computing,
Imperial College of Science, Technology and Medicine, London, UK.
email: {sa,sjg3,rn4}@doc.ic.ac.uk.

Abstract

Many different notions of "property of interest" and methods of verifying such properties arise naturally in programming. A general framework of "Specification Structures" is presented for combining different notions and methods in a coherent fashion. This is then applied to concurrency in the setting of Interaction Categories. As an example, we present a type system for concurrent processes strong enough to guarantee deadlock-freedom, and expressive enough to allow the treatment of some standard examples from the literature. This is illustrated using the classic Dining Philosophers problem.

This paper appears in *Logics for Concurrency: Structure vs. Automata—Proceedings of the VIIIth Banff Higher Order Workshop* (F. Moller and G. Birtwistle, eds), Springer-Verlag Lecture Notes in Computer Science, 1995.

1 Introduction

Type Inference and *Verification* are two main paradigms for constraining the behaviour of programs in such a way as to guarantee some desirable properties. Although they are generally perceived as rather distinct, on closer inspection it is hard to make any very definite demarcation between them; type inference rules shade into compositional proof rules for a program logic. Indeed, type inference systems, even for the basic case of functional programming languages, span a broad spectrum in terms of expressive power. Thus, ML-style types [31] are relatively weak as regards expressing behavioural constraints, but correspondingly tractable as regards efficient algorithms for "type checking". System F types [21] are considerably more expressive of polymorphic behaviour, and System F typing guarantees Strong Normalization. However, System F cannot express the fact that a program of type $\mathsf{list}[\mathsf{nat}] \Rightarrow \mathsf{list}[\mathsf{nat}]$ is actually a sorting function. Martin-Löf type theory, with dependent types and equality types, can express complete total correctness specifications. In the richer theories, type checking is undecidable [35].

One might try to make a methodological distinction: post-hoc verification vs. constructions with intrinsic properties. However, this is more a distinction between ways in which Type Inference/Verification can be deployed than between these two formal paradigms.

We suggest that it is the rule rather than the exception that there are many different notions of "properties of interest" for a given computational setting. Some examples:

*This research was supported by EPSRC project "Foundational Structures in Computer Science", and EU projects "CONFER" (ESPRIT BRA 6454) and "COORDINATION" (ESPRIT BRA 9102).

- Even in the most basic form of sequential programming, it has proved fruitful to separate out the aspects of partial correctness and termination, and to use different methods for these two aspects [16].
- In the field of static analysis, and particularly in the systematic framework of abstract interpretation [23], a basic ingredient of the methodology is to use a range of non-standard interpretations to gain information about different properties of interest.
- In concurrency, it is standard to separate out classes of properties such as safety, liveness, and fairness constraints, extending into a whole temporal hierarchy, and to apply different methods for these classes [27].

The upshot of this observation is that no one monolithic type system will serve all our purposes. Moreover, we need a coherent framework for moving around this space of different classes of properties.

The basic picture we offer to structure this space is the "tower of categories":

$$\mathbb{C}_0 \leftarrowtail \mathbb{C}_1 \leftarrowtail \mathbb{C}_2 \leftarrowtail \cdots \leftarrowtail \mathbb{C}_k.$$

The idea behind the picture is that we have a semantic universe (category with structure) $\mathbb{C}_0$, suitable for modelling some computational situation, but possibly carrying only a very rudimentary notion of "type" or "behavioural specification". The tower arises by refining $\mathbb{C}_0$ with richer kinds of property, so that we obtain a progressively richer setting for performing specification and verification[1].

We will now proceed to formalize this idea of enriching a semantic universe with a refined notion of property in terms of *Specification Structures*.

2 Specification Structures

The notion of specification structure, at least in its most basic form, is quite anodyne, and indeed no more than a variation on standard notions from category theory. Nevertheless, it provides an alternative view of these standard notions which is highly suggestive, particularly from a Computer Science point of view. Similar notions have been studied, for a variety of purposes, by Burstall and McKinna [28], O'Hearn and Tennent [32], and Pitts [33].

Definition 1 *Let $\mathbb{C}$ be a category. A* specification structure *S over $\mathbb{C}$ is defined by the following data:*

- *a set PA of "properties over A", for each object A of $\mathbb{C}$.*
- *a relation $R_{A,B} \subseteq PA \times \mathbb{C}(A,B) \times PB$ for each pair of objects A, B of $\mathbb{C}$.*

We write $\varphi\{f\}\psi$ for $R_{A,B}(\varphi, f, \psi)$ ("Hoare triples"). This relation is required to satisfy the following axioms, for $f : A \to B$, $g : B \to C$, $\varphi \in PA$, $\psi \in PB$ and $\theta \in PC$:

$$\varphi\{\mathsf{id}_A\}\varphi \tag{s1}$$

$$\varphi\{f\}\psi, \psi\{g\}\theta \Longrightarrow \varphi\{f\,;g\}\theta \tag{s2}$$

[1] Of course, non-linear patterns of refinement—trees or dags rather than sequences—can also be considered, but the tower suffices to establish the main ideas.

The axioms (s1) and (s2) are typed versions of the standard Hoare logic axioms for "sequential composition" and "skip" [16].

Given $\mathbb{C}$ and S as above, we can define a new category $\mathbb{C}_S$. The objects are pairs (A, φ) with $A \in \mathsf{Ob}(\mathbb{C})$ and $\varphi \in PA$. A morphism $f : (A, \varphi) \to (B, \psi)$ is a morphism $f : A \to B$ in $\mathbb{C}$ such that $\varphi\{f\}\psi$.

Composition and identities are inherited from $\mathbb{C}$; the axioms (s1) and (s2) ensure that $\mathbb{C}_S$ is a category. Moreover, there is an evident faithful functor

$$\mathbb{C} \leftarrowtail \mathbb{C}_S$$

given by

$$A \leftarrow\!\shortmid (A, \varphi).$$

In fact, the notion of "specification structure on $\mathbb{C}$" is coextensive with that of "faithful functor into $\mathbb{C}$". Indeed, given such a functor $F : \mathbb{D} \to \mathbb{C}$, we can define a specification structure by:

$$\begin{array}{rcl} PA & = & \{\varphi \in \mathsf{Ob}(\mathbb{D}) \mid F(\varphi) = A\} \\ \varphi\{f\}\psi & \equiv & \exists \alpha \in \mathbb{D}(\varphi, \psi).\ F(\alpha) = f \end{array}$$

(by faithfulness, α is unique if it exists). It is easily seen that this passage from faithful functors to specification structures is (up to equivalence) inverse to that from S to $\mathbb{C} \leftarrowtail \mathbb{C}_S$.

A more revealing connection with standard notions is yielded by the observation that specification structures on $\mathbb{C}$ correspond exactly to lax functors from $\mathbb{C}$ to $\mathcal{R}el$, the category of sets and relations. Indeed, given a specification structure S on $\mathbb{C}$, the object part of the corresponding functor $R : \mathbb{C} \to \mathcal{R}el$ is given by P, while for the arrow part we define

$$R(f) \quad = \quad \{(\varphi, \psi) \mid \varphi\{f\}\psi\}.$$

Then (s1) and (s2) become precisely the statement that R is a lax functor with respect to the usual order-enrichment of $\mathcal{R}el$ by inclusion of relations:

$$\begin{array}{rcl} \mathsf{id}_{R(A)} & \subseteq & R(\mathsf{id}_A) \\ R(f)\,;R(g) & \subseteq & R(f\,;g). \end{array}$$

Moreover, the functor $\mathbb{C} \leftarrowtail \mathbb{C}_S$ is the lax fibration arising from the Grothendieck construction applied to R.

The notion of specification structure acquires more substance when there is additional structure on $\mathbb{C}$ which should be lifted to $\mathbb{C}_S$. Suppose for example that $\mathbb{C}$ is a monoidal category, i.e. there is a bifunctor $\otimes : \mathbb{C}^2 \to \mathbb{C}$, an object I, and natural isomorphisms

$$\begin{array}{rrcl} \mathsf{assoc}_{A,B,C} : & (A \otimes B) \otimes C & \cong & A \otimes (B \otimes C) \\ \mathsf{unitl}_A : & I \otimes A & \cong & A \\ \mathsf{unitr}_A : & A \otimes I & \cong & A \end{array}$$

satisfying the standard coherence equations [26]. A specification structure for $\mathbb{C}$ must then correspondingly be extended with an action

$$\otimes_{A,B} : PA \times PB \to P(A \otimes B)$$

and an element $\mathsf{u} \in PI$ satisfying, for $f : A \to B$, $f' : A' \to B'$ and properties φ, φ', ψ, ψ', θ over suitable objects:

$$\begin{array}{c} \varphi\{f\}\psi, \varphi'\{f'\}\psi' \Longrightarrow \varphi \otimes \varphi'\{f \otimes f'\}\psi \otimes \psi' \\ (\varphi \otimes \psi) \otimes \theta\{\mathsf{assoc}_{A,B,C}\}\varphi \otimes (\psi \otimes \theta) \\ \mathsf{u} \otimes \varphi\{\mathsf{unitl}_A\}\varphi \\ \varphi \otimes \mathsf{u}\{\mathsf{unitr}_A\}\varphi. \end{array}$$

Such an action extends the corresponding lax functor $R : \mathbb{C} \to \mathcal{R}el$ to a lax monoidal functor to $\mathcal{R}el$ equipped with its standard monoidal structure based on the cartesian product.

Now assume that $\mathbb{C}$ is symmetric monoidal closed, with natural isomorphism $\mathsf{symm}_{A,B} : A \otimes B \cong B \otimes A$, and internal hom $\multimap$ given by the adjunction

$$\mathbb{C}(A \otimes B, C) \quad \cong \quad \mathbb{C}(A, B \multimap C).$$

Writing $\Lambda(f) : A \to B \multimap C$ for the morphism corresponding to $f : A \otimes B \to C$ under the adjunction, we require an action

$$\multimap_{A,B} : PA \times PB \to P(A \multimap B)$$

and axioms

$$\varphi \otimes \psi\{\mathsf{symm}_{A,B}\}\psi \otimes \varphi$$

$$(\varphi \multimap \psi) \otimes \varphi\{\mathsf{eval}_{A,B}\}\psi$$

$$\varphi \otimes \psi\{f\}\theta \Longrightarrow \varphi\{\Lambda(f)\}\psi \multimap \varphi.$$

Going one step further, suppose that $\mathbb{C}$ is a $*$-autonomous category, i.e. a model for the multiplicative fragment of classical linear logic [11], with linear negation $(-)^{\perp}$, where for simplicity we assume that $A^{\perp\perp} = A$. Then we require an action

$$(-)^{\perp}_{A} : PA \to PA^{\perp}$$

satisfying

$$\begin{aligned} \varphi^{\perp\perp} &= \varphi \\ \varphi \multimap \psi &= (\varphi \otimes \psi^{\perp})^{\perp}. \end{aligned}$$

Under these circumstances all this structure on $\mathbb{C}$ lifts to $\mathbb{C}_S$. For example, we define

$$\begin{aligned} (A, \varphi) \otimes (B, \psi) &= (A \otimes B, \varphi \otimes_{A,B} \psi) \\ (A, \varphi)^{\perp} &= (A^{\perp}, \varphi^{\perp}_{A}) \\ (A, \varphi) \multimap (B, \psi) &= (A \multimap B, \varphi \multimap_{A,B} \psi). \end{aligned}$$

All the constructions on morphisms in $\mathbb{C}_S$ work exactly as they do in $\mathbb{C}$, the above axioms guaranteeing that these constructions are well-defined in $\mathbb{C}_S$. For example, if $f : (A, \varphi) \to (B, \psi)$ and $g : (A', \varphi') \to (B', \psi')$, then

$$f \otimes g : (A \otimes A', \varphi \otimes \varphi') \to (B \otimes B', \psi \otimes \psi').$$

Moreover, all this structure is preserved by the faithful functor $\mathbb{C} \leftarrowtail \mathbb{C}_S$.

The above example of structure on $\mathbb{C}$ is illustrative. Exactly similar definitions can be given for a range of structures, including:

- models of Classical (or Intuitionistic) Linear Logic including the additives and exponentials [10]
- cartesian closed categories [15]
- models of polymorphism [15].

2.1 Examples of Specification Structures

In each case we specify the category $\mathbb{C}$, the assignment of properties P to objects and the Hoare triple relation.

1. $$\mathbb{C} = \mathcal{S}et, \quad PX = X, \quad a\{f\}b \equiv f(a) = b.$$

   In this case, $\mathbb{C}_S$ is the category of pointed sets.

2. $$\mathbb{C} = \mathcal{R}el, \quad PX = \wp X, \quad S\{R\}T \equiv \forall x \in S.\{y \mid xRy\} \subseteq T.$$

   This is essentially a typed version of dynamic logic [25], with the "Hoare triple relation" specialized to its original setting. If we take

   $$\begin{array}{rcl} S \otimes_{X,Y} T & = & S \times T \\ S_X^{\perp} & = & X \backslash S \end{array}$$

   then $\mathbb{C}_S$ becomes a model of Classical Linear Logic.

3. $$\begin{array}{l} \mathbb{C} = \mathcal{R}el, \quad PX = \{C \subseteq X^2 \mid C = C^o, C \cap \mathsf{id}_X = \varnothing\}, \\ C\{R\}D \equiv xCx', xRy, x'Ry' \Rightarrow yDy'. \end{array}$$

   $$\begin{array}{rcl} C \otimes D & = & \{((x, x'), (y, y')) \mid xCy \wedge x'Dy'\} \\ C_X^{\perp} & = & X^2 \backslash (C \cup \mathsf{id}_X). \end{array}$$

   $\mathbb{C}_S$ is the category of coherence spaces and linear maps [20].

4. $$\begin{array}{l} \mathbb{C} = \mathcal{S}et, \quad PX = \{s : \omega \rightharpoonup X \mid \forall x \in X.\exists n \in \omega.s(n) = x\}, \\ s\{f\}t \equiv \exists n \in w.f \circ s \simeq t \circ \varphi_n \end{array}$$

   where φ_n is the nth partial recursive function in some acceptable numbering [34]. Then $\mathbb{C}_S$ is the category of modest sets, seen as a full subcategory of ω-$\mathcal{S}et$ [10].

5. $$\begin{array}{l} \mathbb{C} = \text{the category of SFP domains}, \\ PD = K\Omega(D)(\text{the compact-open subsets of } D), \\ U\{f\}V \equiv U \subseteq f^{-1}(V). \end{array}$$

   This yields (part of) Domain Theory in Logical Form [2], the other part arising from the local lattice-theoretic structure of the sets PD and its interaction with the global type structure.

6. $\mathbb{C}$ = games and partial strategies, as in [9], PA = all sets of infinite plays, $U\{\sigma\}V$ iff σ is *winning* with respect to U, V in the sense of [7]. Then $\mathbb{C}_S$ is the category of games and winning strategies of [7].

These examples show the scope and versatility of these notions. Let us return to our picture of the tower of categories:

$$\mathbb{C}_0 \leftarrowtail \mathbb{C}_1 \leftarrowtail \mathbb{C}_2 \leftarrowtail \cdots \leftarrowtail \mathbb{C}_k.$$

Such a tower arises by progressively refining $\mathbb{C}_0$ by specification structures $S_1, \ldots, S_k$ so that

$$\mathbb{C}_{i+1} \quad = \quad (\mathbb{C}_i)_{S_{i+1}}.$$

Each such step adds propositional information to the underlying "raw" computational entities (morphisms of $\mathbb{C}_0$). The aim of verification in this framework is to "promote" a morphism from $\mathbb{C}_i$ to $\mathbb{C}_j$, $i < j$. That is, to promote a $\mathbb{C}_0$ morphism $f : A \to B$ to a $\mathbb{C}_k$ morphism

$$f : (A, \varphi_1, \ldots, \varphi_k) \to (B, \psi_1, \ldots, \psi_k)$$

is precisely to establish the "verification conditions"

$$\bigwedge_{i=1}^{k} \varphi_i\{f\}\psi_i.$$

Once this has been done, by whatever means—model checking, theorem proving, manual verification, etc.—the morphism is now available in $\mathbb{C}_k$ to participate in typing judgements there. In this way, a coherent framework for combining methods, including both compositional and non-compositional approaches, begins to open up.

We now turn to the specific applications of this framework which in fact originally suggested it, in the setting of the first author's interaction categories.

3 Interaction Categories

Interaction Categories [1, 3, 4, 6] are a new paradigm for the semantics of sequential and concurrent computation. This term encompasses certain known categories (the category of concrete data structures and sequential algorithms [12], categories of games [7], geometry of interaction categories [8]) as well as several new categories for concurrency. The fundamental examples of concurrent interaction categories are $\mathcal{SP}roc$, the category of synchronous processes, and $\mathcal{ASP}roc$, the category of asynchronous processes. These categories will be defined in this section; others will be constructed later by means of specification structures over $\mathcal{SP}roc$ and $\mathcal{ASP}roc$.

The general picture of these categories is that the objects are types, which we also think of as specifications; the morphisms are concurrent processes which satisfy these specifications; and composition is interaction, i.e. an ongoing sequence of communications. The dynamic nature of composition in interaction categories is one of the key features, and is in sharp contrast to the functional composition typically found in categories of mathematical structures.

There is not yet a definitive axiomatisation of interaction categories, although some possibilities have been considered [18]. The common features of the existing examples are that they have $*$-autonomous structure, which corresponds to the multiplicative fragment of classical linear logic [20]; products and coproducts, corresponding to the additives of linear logic, and additional temporal structure which enables the dynamics of process evolution to be described. Furthermore, $\mathcal{SP}roc$ has suitable structure to interpret the exponentials ! and ?, and is thus a model of full classical linear logic.

3.1 The Interaction Category $\mathcal{SP}roc$

In this section we briefly review the definition of $\mathcal{SP}roc$, the category of synchronous processes. Because the present paper mainly concerns the use of specification structures for deadlock-freedom, we omit the features of $\mathcal{SP}roc$ which will not be needed in later sections. More complete definitions can be found elsewhere [1, 6, 18].

An object of $\mathcal{SP}roc$ is a pair $A = (\Sigma_A, S_A)$ in which Σ_A is an *alphabet* (*sort*) of *actions* (*labels*) and $S_A \subseteq^{nepref} \Sigma_A^*$ is a *safety specification*, i.e. a non-empty prefix-closed subset of Σ_A^*. If A is an object of $\mathcal{SP}roc$, a *process of type* A is a process P with sort Σ_A such that $\mathsf{traces}(P) \subseteq S_A$. Our notion of process is *labelled transition system*, with *strong bisimulation* as the equivalence. We will usually define processes by means of labelled transition rules.

If P is a labelled transition system, $\mathsf{traces}(P)$ is the set of sequences labelling finite paths from the root. The set of sequences labelling finite and infinite paths is $\mathsf{alltraces}(P)$ and the set of sequences labelling infinite paths is $\mathsf{inftraces}(P)$. The following coinductive definition is equivalent to this description.

$$\begin{aligned}
\mathsf{alltraces}(P) &\stackrel{\text{def}}{=} \{\varepsilon\} \cup \{a\sigma \mid P \xrightarrow{a} Q, \sigma \in \mathsf{alltraces}(Q)\} \\
\mathsf{traces}(P) &\stackrel{\text{def}}{=} \{\sigma \in \mathsf{alltraces}(P) \mid \sigma \text{ is finite}\} \\
\mathsf{inftraces}(P) &\stackrel{\text{def}}{=} \{\sigma \in \mathsf{alltraces}(P) \mid \sigma \text{ is infinite}\}.
\end{aligned}$$

The fact that P is a process of type A is expressed by the notation $P : A$.

The most convenient way of defining the morphisms of $\mathcal{SP}roc$ is first to define a $*$-autonomous structure on objects, and then say that the morphisms from A to B are processes of the internal hom type $A \multimap B$. This style of definition is typical of interaction categories; definitions of categories of games [7] follow the same pattern. Given objects A and B, the object $A \otimes B$ has

$$\begin{aligned}
\Sigma_{A\otimes B} &\stackrel{\text{def}}{=} \Sigma_A \times \Sigma_B \\
S_{A\otimes B} &\stackrel{\text{def}}{=} \{\sigma \in \Sigma_{A\otimes B}^* \mid \mathsf{fst}^*(\sigma) \in S_A, \mathsf{snd}^*(\sigma) \in S_B\}.
\end{aligned}$$

The duality is trivial on objects: $A^\perp \stackrel{\text{def}}{=} A$. This means that at the level of types, $\mathcal{SP}roc$ makes no distinction between input and output. Because communication in $\mathcal{SP}roc$ consists of synchronisation rather than value-passing, processes do not distinguish between input and output either.

The definition of $\otimes$ makes clear the extent to which processes in $\mathcal{SP}roc$ are synchronous. An action performed by a process of type $A \otimes B$ consists of a pair of actions, one from the alphabet of A and one from that of B. Thinking of A and B as two ports of the process, synchrony means that at every time step a process must perform an action in every one of its ports.

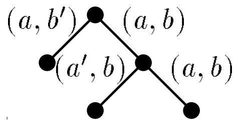
For simplicity, we shall work with $*$-autonomous categories in which $A^{\perp\perp} = A$, and $A \multimap B \stackrel{\text{def}}{=} (A \otimes B^\perp)^\perp$, $A \mathbin{⅋} B \stackrel{\text{def}}{=} (A^\perp \otimes B^\perp)^\perp$. In $\mathcal{SP}roc$, we have $A = A^\perp$, and hence $A \mathbin{⅋} B = A \multimap B = A \otimes B$. Not all interaction categories exhibit this degeneracy of structure: in particular the category $\mathcal{SP}roc_D$ of deadlock-free processes, which will be defined in Section 4, gives distinct interpretations to $\otimes$ and $⅋$.

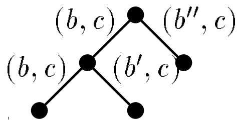
A morphism $p : A \to B$ of $\mathcal{SP}roc$ is a process p of type $A \multimap B$ (so p has to satisfy a certain safety specification). Since $A \multimap B = A \otimes B$ in $\mathcal{SP}roc$, this amounts to saying that a morphism from A to B is a process of type $A \otimes B$. The reason for giving the definition in terms of $\multimap$ is that it sets the pattern for all interaction category definitions, including cases in which there is less degeneracy.

If $p : A \to B$ and $q : B \to C$ then the composite $p \,;\, q : A \to C$ is defined by labelled transitions.

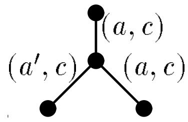
$$\frac{p \xrightarrow{(a,b)} p' \qquad q \xrightarrow{(b,c)} q'}{p \,;\, q \xrightarrow{(a,c)} p' \,;\, q'}$$

At each step, the actions in the common type B have to match. The processes being composed constrain each other's behaviour, selecting the possibilities which agree in B. For example, if p and q are as shown:





then $p\,;q$ is this tree.



This ongoing communication is the "interaction" of interaction categories. If the processes in the definition terminated after a single step, so that each could be considered simply as a set of pairs, then the labelled transition rule would reduce to precisely the definition of relational composition. This observation leads to the $\mathcal{SP}roc$ slogan: *processes are relations extended in time.*

The identity morphisms are synchronous buffers or wires: whatever is received by $\mathsf{id}_A : A \to A$ in the left copy of A is instantaneously transmitted to the right copy (and *vice versa*—there is no real directionality). The following auxiliary definition helps to define the identity processes. If P is a process with sort Σ and $S \subseteq^{nepref} \Sigma^*$ then the process $P \restriction S$, also with sort Σ, is defined by the transition rule

$$\frac{P \xrightarrow{a} Q \qquad a \in S}{P \restriction S \xrightarrow{a} Q \restriction (S/a)}$$

where $S/a \stackrel{\text{def}}{=} \{\sigma \mid a\sigma \in S\}$. Note that the condition $a \in S$ in the transition rule refers to the singleton sequence a rather than the action a. We make no notational distinction between these uses of a.

The identity morphism $\mathsf{id}_A : A \to A$ is defined by $\mathsf{id}_A \stackrel{\text{def}}{=} \mathsf{id} \restriction S_{A \multimap A}$ where the process id with sort Σ_A is defined by the transition rule

$$\frac{a \in \Sigma_A}{\mathsf{id} \xrightarrow{(a,a)} \mathsf{id}.}$$

Proposition 2 *$\mathcal{SP}roc$ is a category.*

Proof: The proof that composition is associative and that identities work correctly uses a coinductive argument to show that suitable processes are bisimilar. Full details can be found elsewhere [1, 6]. $\square$

3.1.1 $\mathcal{SP}roc$ as a $*$-Autonomous Category

The definitions of $\otimes$ and $(-)^\perp$ can now be extended to morphisms, making them into functors. If $p : A \to C$ and $q : B \to D$ then $p \otimes q : A \otimes B \to C \otimes D$ and $p^\perp : C^\perp \to A^\perp$ are defined by transition rules.

$$\frac{p \xrightarrow{(a,c)} p' \qquad q \xrightarrow{(b,d)} q'}{p \otimes q \xrightarrow{((a,b),(c,d))} p' \otimes q'} \qquad\qquad \frac{p \xrightarrow{(a,c)} p'}{p^\perp \xrightarrow{(c,a)} p'^\perp}$$

The tensor unit I is defined by

$$\Sigma_I \stackrel{\text{def}}{=} \{*\} \qquad\qquad S_I \stackrel{\text{def}}{=} \{*^n \mid n < \omega\}.$$

The following notation provides a useful way of defining the structural morphisms needed to specify the rest of the $*$-autonomous structure. If P is a process with sort Σ, and $f : \Sigma \rightharpoonup \Sigma'$ is a partial function, then $P[f]$ is the process with sort Σ' defined by

$$\frac{P \xrightarrow{a} Q \qquad a \in \mathsf{dom}(f)}{P[f] \xrightarrow{f(a)} Q[f].}$$

The canonical isomorphisms $\mathsf{unitl}_A : I \otimes A \cong A$, $\mathsf{unitr}_A : A \otimes I \cong A$, $\mathsf{assoc}_{A,B,C} : A \otimes (B \otimes C) \cong (A \otimes B) \otimes C$ and $\mathsf{symm}_{A,B} : A \otimes B \cong B \otimes A$ are defined as follows. Here we are using a pattern-matching notation to define the partial functions needed for the relabelling operations; for example, $(a, a) \mapsto ((*, a), a)$ denotes the partial function which has the indicated effect when its arguments are equal.

$$\begin{array}{lcl}
\mathsf{unitl}_A & \stackrel{\text{def}}{=} & \mathsf{id}_A[(a, a) \mapsto ((*, a), a)] \\
\mathsf{unitr}_A & \stackrel{\text{def}}{=} & \mathsf{id}_A[(a, a) \mapsto ((a, *), a)] \\
\mathsf{assoc}_{A,B,C} & \stackrel{\text{def}}{=} & \mathsf{id}_{A\otimes(B\otimes C)}[((a, (b, c)), (a, (b, c))) \mapsto ((a, (b, c)), ((a, b), c))] \\
\mathsf{symm}_{A,B} & \stackrel{\text{def}}{=} & \mathsf{id}_{A\otimes B}[((a, b), (a, b)) \mapsto ((a, b), (b, a))].
\end{array}$$

If $f : A \otimes B \to C$ then $\Lambda(f) : A \to (B \multimap C)$ is defined by

$$\Lambda(f) \stackrel{\text{def}}{=} f[((a, b), c) \mapsto (a, (b, c))].$$

The evaluation morphism $\mathsf{Ap}_{A,B} : (A \multimap B) \otimes A \to B$ is defined by

$$\mathsf{Ap}_{A,B} \stackrel{\text{def}}{=} \mathsf{id}_{A\multimap B}[((a, b), (a, b)) \mapsto (((a, b), a), b)].$$

All of the structural morphisms are essentially formed from identities, and the only difference between f and $\Lambda(f)$ is a reshuffling of ports. In each of the above uses of relabelling, the partial function on sorts is defined by means of a pattern-matching notation; the function is only defined for arguments which fit the pattern.

If P is a process of type A then $P[a \mapsto (*, a)]$ is a morphism $I \to A$ which can be identified with P. This agrees with the view of global elements (morphisms from I, in a $*$-autonomous category) as inhabitants of types.

Proposition 3 *$\mathcal{SP}roc$ is a compact closed category.*

Proof: Verifying the coherence conditions for $\otimes$ is straightforward, given the nature of the canonical isomorphisms as relabelled identities. The properties required of Λ and Ap are equally easy to check. Since $(-)^\perp$ is trivial, it is automatically an involution. This gives the $*$-autonomous structure; compact closure follows from the coincidence of $\otimes$ and $\mathbin{⅋}$. □

3.1.2 Compact Closure and Multi-Cut

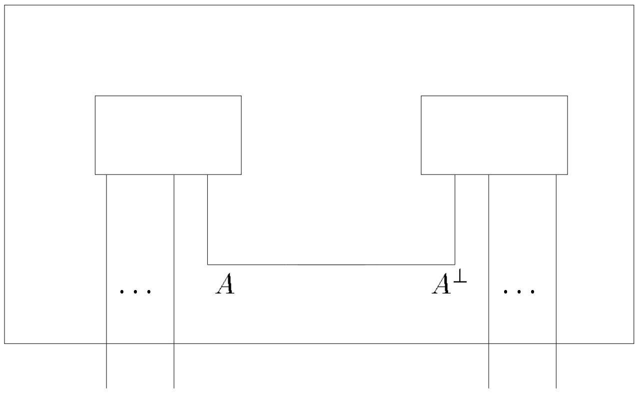
As we have already seen the linear type structure of $\mathcal{SP}roc$ is quite degenerate. Specification structures can be used to enrich the specifications of $\mathcal{SP}roc$ to stronger behavioural properties. This will have the effect of "sharpening up" the linear type structure so that the degeneracies disappear.

Our point here is that the looser type discipline of $\mathcal{SP}roc$ can actually be *useful* in that it permits the flexible construction of a large class of processes within a typed framework. In particular, compact closure validates a very useful typing rule which we call the *multi-cut*. (This is actually Gentzen's MIX rule [19] but we avoid the use of this term since Girard has used it for quite a different rule in the context of Linear Logic.)

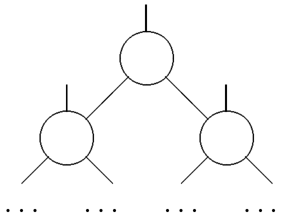
The usual Cut Rule

$$\frac{\vdash \Gamma, A \qquad \vdash \Delta, A^{\perp}}{\vdash \Gamma, \Delta}$$

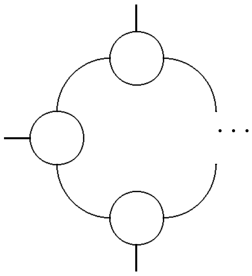
allows us to plug two modules together by an interface consisting of a single "port" [5]:



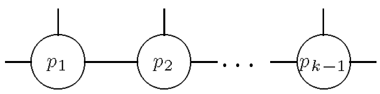
This allows us to connect processes in a tree structure



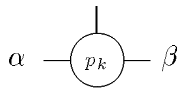
but not to construct cyclic interconnection networks



such as the Scheduler described in [30]. The problem with building a cycle is at the last step where we have already connected



To connect



we must plug both α and β simultaneously into the existing network. This could be done if we had the following "binary" version of the cut rule

$$\frac{\vdash \Gamma, A_1, A_2 \qquad \vdash \Delta, A_1^{\perp}, A_2^{\perp}}{\vdash \Gamma, \Delta}$$

or more generally the "multi-cut":

$$\frac{\vdash \Gamma, \Delta \qquad \vdash \Gamma', \Delta^{\perp}}{\vdash \Gamma, \Gamma'}$$

This rule is not admissible in Linear Logic and cannot in general be interpreted in Linear Categories. However it can always be canonically interpreted in a compact closed category (and hence in particular in $\mathcal{SProc}$) as the following construction shows.

Let

$$\Gamma = A_1, \ldots A_m, \Gamma' = B_1, \ldots B_n, \Delta = C_1, \ldots C_k.$$

We write

$$\tilde{A} = A_1 \otimes \cdots \otimes A_m, \quad \tilde{B} = B_1 \otimes \cdots \otimes B_n, \quad \tilde{C} = C_1 \otimes \cdots \otimes C_k$$

$$\tilde{C}^{\perp} = (C_1 \otimes \cdots \otimes C_k)^{\perp} \cong C_1^{\perp} \otimes \cdots \otimes C_k^{\perp}$$

Suppose that the proofs of $\vdash \Gamma, \Delta$ and $\vdash \Gamma', \Delta'$ are interpreted by morphisms

$$f : I \longrightarrow \tilde{A} \otimes \tilde{C}, \quad g : I \longrightarrow \tilde{B} \otimes \tilde{C}^{\perp}$$

respectively. Then we can construct the required morphism $I \longrightarrow \tilde{A} \otimes \tilde{B}$ as follows:

$$
\begin{tikzcd}
I \arrow[d, "\wr"', "\text{(unit)}"] \\
I \otimes I \arrow[d, "f \otimes g"] \\
(\tilde{A} \otimes \tilde{C}) \otimes (\tilde{B} \otimes \tilde{C}^{\perp}) \arrow[d, "\wr"', "\text{(canonical isos)}"] \\
\tilde{A} \otimes ((C_1 \otimes C_1^{\perp}) \otimes \cdots \otimes (C_k \otimes C_k^{\perp})) \otimes \tilde{B} \arrow[d, "\text{(evaluation)}"] \\
\tilde{A} \otimes I \otimes \cdots \otimes I \otimes \tilde{B} \arrow[d, "\wr"', "\text{(unit)}"] \\
\tilde{A} \otimes \tilde{B}
\end{tikzcd}
$$

(Note that in a compact closed category $I = \perp$ so $A^{\perp} = A \multimap I$.)

In the case where $k = 1$ this construction is the internalization of composition in the category (using the autonomous structure) so it properly generalizes the standard interpretation of Cut. For some related notions which have arisen in work on coherence in compact closed categories, see [13, 24].

3.1.3 $\mathcal{SP}roc$ as a Linear Category

$\mathcal{SP}roc$ also has structure corresponding to the linear logic exponentials ! and ?. We will not need this structure in the present paper; details can be found elsewhere [1, 6, 18].

3.1.4 Non-determinism

We can define the non-deterministic sum exactly as in CCS. If $p, q : A \to B$ then $p + q$ is defined by

$$
\frac{p \xrightarrow{(a,b)} p'}{p + q \xrightarrow{(a,b)} p'} \qquad \frac{q \xrightarrow{(a,b)} q'}{p + q \xrightarrow{(a,b)} q'}
$$

For any objects A and B, there is a morphism $\mathsf{nil} : A \to B$ which has no transitions.

The non-deterministic sum and the nil morphisms exist for quite general reasons: $\mathcal{SP}roc$ has biproducts, and it is standard that this yields a commutative monoid structure on every homset [26]. In the present paper, we have defined $+$ directly as we will not make any other use of the products and coproducts.

3.1.5 Time

So far, none of the constructions in $\mathcal{SP}roc$ have made use of the fact that morphisms are *processes* with dynamic behaviour. Everything that has been discussed applies equally well to the category of sets and relations. The next step is to justify the claim that $\mathcal{SP}roc$ looks like "relations extended in time" by defining some structure which allows the temporal aspects of the category to be manipulated.

The basic construction dealing with time is the unit delay functor $\bigcirc$. It is defined on objects by

$$\begin{aligned} \Sigma_{\bigcirc A} &\stackrel{\text{def}}{=} \{*\} + \Sigma_A \\ S_{\bigcirc A} &\stackrel{\text{def}}{=} \{\varepsilon\} \cup \{*\sigma \mid \sigma \in S_A\}. \end{aligned}$$

It is notationally convenient to write $*$ instead of $\mathsf{inl}(*)$, assuming that $* \notin \Sigma_A$. Given $f : A \to B$, $\bigcirc f : \bigcirc A \to \bigcirc B$ is defined by the single transition $\bigcirc f \xrightarrow{(*,*)} f$.

It is straightforward to check that $\bigcirc$ is indeed a functor. In fact it is a strict monoidal functor.

Proposition 4 *There are isomorphisms*

$$\mathsf{mon}_{A,B} : (\bigcirc A) \otimes (\bigcirc B) \to \bigcirc(A \otimes B)$$

(natural in A and B) and $\mathsf{monunit} : I \to \bigcirc I$.

Proof: $\mathsf{monunit} : I \cong \bigcirc I$ is defined by

$$\mathsf{monunit} \xrightarrow{(\bullet,*)} \mathsf{id}_I$$

where $\Sigma_I = \{\bullet\}$. $\mathsf{mon}_{A,B} : (\bigcirc A) \otimes (\bigcirc B) \cong \bigcirc(A \otimes B)$ is defined by

$$\mathsf{mon}_{A,B} \xrightarrow{((*,*),*)} \mathsf{id}_{A \otimes B}.$$

In both cases the inverse is obtained by considering the process as a morphism in the opposite direction. It is easy to check that these are isomorphisms and that mon is natural. $\square$

The most important feature of $\bigcirc$ is that it has the *unique fixed point property* (UFPP) [6]: for any objects A and B, and any morphisms $f : A \to \bigcirc A$ and $g : \bigcirc B \to B$ there is a unique morphism $h : A \to B$ such that

$$\begin{tikzcd} A \arrow[r, "f"] \arrow[d, "h"'] & \bigcirc A \arrow[d, "\bigcirc h"] \\ B & \bigcirc B \arrow[l, "g"] \end{tikzcd}$$

commutes. We will not go into the applications of this property in the present paper, except to mention that it supports guarded recursive definitions [1, 6, 18] and is an important part of a proposed axiomatisation of interaction categories [18].

Apart from $\bigcirc$ there are two other delay functors: the initial delay δ and the propagated delay Δ. These are the same as the operators used by Milner [29, 30] to construct CCS from SCCS, and they can also be used to construct asynchronous processes in the synchronous framework of $\mathcal{SProc}$. However, when analysing asynchronous problems it is much more convenient to work in a different category, $\mathcal{ASProc}$, which we will define shortly. For this reason, we will give only the basic definitions of the delay functors here, and not dwell on their properties.

The functors δ and Δ are defined on objects by

$$\begin{aligned}
\Sigma_{\delta A} &\stackrel{\text{def}}{=} \mathbf{1} + \Sigma_A \\
S_{\delta A} &\stackrel{\text{def}}{=} \{*^n \sigma \mid (n < \omega) \wedge (\sigma \in S_A)\} \\
\Sigma_{\Delta A} &\stackrel{\text{def}}{=} \mathbf{1} + \Sigma_A \\
S_{\Delta A} &\stackrel{\text{def}}{=} \{\varepsilon\} \cup \{a_1 *^{n_1} a_2 *^{n_2} a_3 \ldots \mid (n_i < \omega) \wedge (a_1 a_2 a_3 \ldots \in S_A)\}
\end{aligned}$$

and on morphisms by transition rules.

$$\frac{}{\delta f \xrightarrow{(*,*)} \delta f} \qquad \frac{f \xrightarrow{(a,b)} f'}{\delta f \xrightarrow{(a,b)} f'} \qquad \frac{f \xrightarrow{(a,b)} f'}{\Delta f \xrightarrow{(a,b)} \delta \Delta f'}$$

Both of these functors are monads. Full details can be found elsewhere [1, 6, 18].

3.2 The Interaction Category $\mathcal{ASProc}$

The theory of interaction categories is not restricted to the synchronous model of concurrency which underlies the category $\mathcal{SProc}$. There is also a category of asynchronous processes, $\mathcal{ASProc}$, which we will now define. In this context, asynchrony means the capacity to delay; in particular, an asynchronous process can delay in some of its ports while performing observable actions in others. Because we do not wish to distinguish between processes which differ only in the amount by which they delay at certain points, we now consider processes to be labelled transition systems modulo observation equivalence (weak bisimulation) [30] rather than strong bisimulation.

In CCS there is a single silent action, τ, which is used by all processes to represent delay. In the typed framework of interaction categories we no longer have a global set of actions, so it is necessary to specify a silent action τ_A in each type A. Thus an object of $\mathcal{ASProc}$ contains an extra piece of information compared to an object of $\mathcal{SProc}$. This enables observation equivalence classes of typed processes to be defined: when considering processes of type A, the action τ_A is used as the silent action in the standard definition of observation equivalence.

The approach we will take to defining operations on asynchronous processes is to define them by labelled transition rules, and then check that they are well-defined on observation equivalence classes.

The definition of $\mathcal{ASProc}$ in this section is slightly different from the original definition [4], where an action was taken to be a *set* of labels and the silent action was $\varnothing$. The definition used here emphasises the essential difference between $\mathcal{ASProc}$ and $\mathcal{SProc}$, namely the introduction of τ_A and the use of observation equivalence.

3.3 $\mathcal{ASProc}$ as a Category

An object of $\mathcal{ASProc}$ is a triple $A = (\Sigma_A, \tau_A, S_A)$, in which Σ_A is a set of actions, $\tau_A \in \Sigma_A$ is the *silent action*, $S_A \subseteq^{nepref} \mathsf{ObAct}(A)^*$ is a safety specification, and $\mathsf{ObAct}(A) \stackrel{\text{def}}{=} \Sigma_A - \{\tau_A\}$ is the set of observable actions of A.

A *process* with *sort* Σ and *silent action* $\tau \in \Sigma$ is an observation equivalence class of synchronisation trees with label set Σ.

A *process P of type A*, written $P : A$, is a process P with sort Σ_A and silent action τ_A such that $\mathsf{obtraces}(P) \subseteq S_A$, where

$$\begin{array}{rcl} \mathsf{allobtraces}(P) & \stackrel{\text{def}}{=} & \{\varepsilon\} \cup \{a\sigma \mid P \stackrel{a}{\Longrightarrow} Q, \sigma \in \mathsf{allobtraces}(Q)\} \\ \mathsf{obtraces}(P) & \stackrel{\text{def}}{=} & \{\sigma \in \mathsf{allobtraces}(P) \mid \sigma \text{ is finite}\} \\ \mathsf{infobtraces}(P) & \stackrel{\text{def}}{=} & \{\sigma \in \mathsf{allobtraces}(P) \mid \sigma \text{ is infinite}\} \end{array}$$

Just as in $\mathcal{SProc}$ the morphisms are defined via the object part of the $*$-autonomous structure. Given objects A and B, the object $A \otimes B$ has

$$\begin{array}{rcl} \Sigma_{A\otimes B} & \stackrel{\text{def}}{=} & \Sigma_A \times \Sigma_B \\ \tau_{A\otimes B} & \stackrel{\text{def}}{=} & (\tau_A, \tau_B) \\ S_{A\otimes B} & \stackrel{\text{def}}{=} & \{\sigma \in \mathsf{ObAct}(\Sigma_{A\otimes B})^* \mid \sigma{\restriction}A \in S_A, \sigma{\restriction}B \in S_B\} \end{array}$$

where, for $\alpha \in \mathsf{ObAct}(\Sigma_{A\otimes B})$,

$$\alpha{\restriction}A \stackrel{\text{def}}{=} \begin{cases} \mathsf{fst}(\alpha) & \text{if } \mathsf{fst}(\alpha) \neq \tau_A \\ \varepsilon & \text{otherwise} \end{cases}$$

and for $\sigma \in \mathsf{ObAct}(\Sigma_{A\otimes B})^*$, $\sigma{\restriction}A$ is obtained by concatenating the individual $\alpha{\restriction}A$. The projection $\sigma{\restriction}B$ is defined similarly. Notice that taking $\tau_{A\otimes B} = (\tau_A, \tau_B)$ means that a process with several ports delays by simultaneously delaying in its individual ports.

The duality is trivial on objects: $A^\perp \stackrel{\text{def}}{=} A$.

A morphism $p : A \to B$ of $\mathcal{ASProc}$ is a process p such that $p : A \multimap B$.

If $p : A \to B$ and $q : B \to C$, then the composite $p\,;q : A \to C$ is defined by labelled transitions.

$$\frac{p \xrightarrow{(a,\tau_B)} p'}{p\,;q \xrightarrow{(a,\tau_C)} p'\,;q} \qquad \frac{q \xrightarrow{(\tau_B,c)} q'}{p\,;q \xrightarrow{(\tau_A,c)} p\,;q'}$$

$$\frac{p \xrightarrow{(a,b)} p' \qquad q \xrightarrow{(b,c)} q'}{p\,;q \xrightarrow{(a,c)} p'\,;q'}$$

The first two rules allow either process to make a transition independently, if no communication is required. The third rule allows the processes to communicate by performing the same action in the port B. Any of the actions a, b, c can be τ; if $b = \tau_B$ in the third rule, then two simultaneous independent transitions are made.

It is necessary to prove that composition is well-defined on observation equivalence classes, but we will not give the details here.

As in $\mathcal{SProc}$, it is straightforward to prove that if $f : A \to B$ and $g : B \to C$, then $f\,;g$ satisfies the safety specification necessary to be a morphism $A \to C$.

Although $\mathcal{ASProc}$ is a category of asynchronous processes, the identity morphisms are still *synchronous* buffers. As a candidate identity morphism, a synchronous buffer seems likely to work, given the definition of composition; of course, once it has been shown to be an identity, no other choice is possible.

The identity morphism $\mathsf{id}_A : A \to A$ is defined as in $\mathcal{SProc}$: $\mathsf{id}_A \stackrel{\text{def}}{=} \mathsf{id}{\restriction}S_{A\multimap A}$ where the process id with sort Σ_A is defined by

$$\frac{a \in \Sigma_A}{\mathsf{id} \xrightarrow{(a,a)} \mathsf{id}.}$$

Just as in $\mathcal{SP}roc$, if P is a process with sort Σ and $S \subseteq^{nepref} \Sigma^*$ then the process $P \upharpoonright S$, also with sort Σ, is defined by the transition rule

$$\frac{P \xrightarrow{a} Q \qquad a \in S}{P \upharpoonright S \xrightarrow{a} Q \upharpoonright (S/a).}$$

3.4 $\mathcal{ASP}roc$ as a $*$-Autonomous Category

If $p : A \to C$ and $q : B \to D$ then $p \otimes q : A \otimes B \to C \otimes D$ and $p^\perp : C^\perp \to A^\perp$ are defined by transition rules. The rules for $\otimes$ illustrate the asynchronous nature of $\mathcal{ASP}roc$; the two processes can make transitions either independently or simultaneously.

$$\frac{p \xrightarrow{(a,c)} p'}{p \otimes q \xrightarrow{((a,\tau_B),(c,\tau_D))} p' \otimes q} \qquad \frac{q \xrightarrow{(b,d)} q'}{p \otimes q \xrightarrow{((\tau_A,b),(\tau_C,d))} p \otimes q'}$$

$$\frac{p \xrightarrow{(a,c)} p' \qquad q \xrightarrow{(b,d)} q'}{p \otimes q \xrightarrow{((a,b),(c,d))} p' \otimes q'} \qquad \frac{p \xrightarrow{(a,c)} p'}{p^\perp \xrightarrow{(c,a)} p'^\perp}$$

The tensor unit I is defined by

$$\Sigma_I \stackrel{\text{def}}{=} \{\tau_I\} \qquad\qquad S_I \stackrel{\text{def}}{=} \{\varepsilon\}.$$

The morphisms expressing the symmetric monoidal closed structure are defined as in $\mathcal{SP}roc$, by combining identities.

Proposition 5 *$\mathcal{ASP}roc$ is a compact closed category.*

3.4.1 Non-determinism

It turns out that $\mathcal{ASP}roc$ has only *weak* biproducts. The construction of an addition on the homsets can still be carried out, but it yields $\tau.P + \tau.Q$. This is not surprising, as the CCS operation $+$ is not well-defined on observation equivalence classes. In later sections we will often construct processes by means of guarded sums such as $a.P + b.Q$, which can be given direct definitions in terms of labelled transitions.

3.5 Time

In $\mathcal{ASP}roc$, the delay monads δ and Δ are less meaningful than in $\mathcal{SP}roc$, since delay is built into all the definitions. But the unit delay functor $\bigcirc$ is still important. On objects it is defined by

$$\begin{aligned}
\Sigma_{\bigcirc A} &\stackrel{\text{def}}{=} \{*\} + \Sigma_A \\
\tau_{\bigcirc A} &\stackrel{\text{def}}{=} \tau_A \\
S_{\bigcirc A} &\stackrel{\text{def}}{=} \{\varepsilon\} \cup \{*\sigma \mid \sigma \in S_A\}.
\end{aligned}$$

If $f : A \to B$ then $\bigcirc f : \bigcirc A \to \bigcirc B$ is defined by the transition $\bigcirc f \xrightarrow{(*,*)} f$.

Proposition 6 *$\bigcirc$ is a functor, and has the UFPP.*

Proof: As in $\mathcal{SP}roc$. □

4 Specification Structures for Deadlock-Freedom

4.1 The Synchronous Case

We shall now describe a specification structure D for $\mathcal{SP}roc$ such that $\mathcal{SP}roc_D$ will be a category of deadlock-free processes, closed under all the type constructions described above (and several more omitted in this introductory account). This specification structure has a number of remarkable features:

- The typing rule for composition in $\mathcal{SP}roc_D$ will be a compositional proof rule for plugging together deadlock-free processes while preserving deadlock-freedom. Rules of this kind are known to be difficult to obtain [17].
- The concepts and techniques used in defining this specification structure and verifying that it has the required properties represent a striking transfer of techniques from Proof Theory (Tait-Girard proofs of Strong Normalization [21]) to concurrency. This is made possible by our framework of interaction categories and specification structures.

We begin with some preliminary definitions. Firstly we define a binary process combinator $p \sqcap q$ by the transition rule

$$\frac{p \xrightarrow{a} p' \qquad q \xrightarrow{a} q'}{p \sqcap q \xrightarrow{a} p' \sqcap q'}.$$

Note that $p \sqcap q$ is the meet of p and q with respect to the simulation pre-order [30].

Let $\mathsf{ST}_{\mathcal{L}}$ be the set of processes labelled over $\mathcal{L}$. We define, for $p \in \mathsf{ST}_{\mathcal{L}}$:

$$p{\downarrow} \quad \equiv \quad \forall s \in \mathcal{L}^*, q \in \mathsf{ST}_{\mathcal{L}}.p \xrightarrow{s} q \Rightarrow \exists a \in \mathcal{L}, r \in \mathsf{ST}_{\mathcal{L}}.q \xrightarrow{a} r.$$

We read $p{\downarrow}$ as "p is deadlock-free", i.e. it can never evolve into the nil process.

An important point is that we need to restrict attention to those objects of $\mathcal{SP}roc$ whose safety specifications do not force processes to deadlock. The object A is *progressive* if

$$\forall s \in S_A.\exists a \in \Sigma_A.sa \in S_A.$$

By considering just the progressive objects, we can be sure that there are deadlock-free processes of every type.

Finally, the key definition:

$$p \perp q \equiv (p \sqcap q){\downarrow}\ .$$

We can think of $p \perp q$ as expressing the fact that "p passes the test q"; but note that $\_ \perp \_$ is symmetric so the rôles of p and q, tester and testee, can be interchanged freely.

Now we lift this symmetric relation to a self-adjoint Galois connection on sets of processes in a standard fashion [14]:

$$\begin{aligned} p \perp U &\equiv \forall q \in U.\, p \perp q \\ U^{\perp} &\equiv \{p \mid p \perp U\}. \end{aligned}$$

Since $(-)^{\perp}$ is a self-adjoint Galois connection, it satisfies

$$U^{\perp\perp\perp} \equiv U^{\perp}.$$

We are now ready to define the specification structure D on the subcategory of $\mathcal{SP}roc$ which consists of just the progressive objects.

$$PA \quad = \quad \{U \subseteq \mathsf{ST}_{\Sigma_A} \mid \forall p \in U.\, (p : A) \wedge (p{\downarrow}), U \neq \varnothing, U = U^{\perp\perp}\}.$$

(Compare with the definition of *candidats* in [20], and with Linear Realizability Algebras [8]).

$$
\begin{aligned}
U \otimes V &\stackrel{\text{def}}{=} \{p \otimes q \mid p \in U, q \in V\}^{\perp\perp} \\
U \parr V &\stackrel{\text{def}}{=} (U^{\perp} \otimes V^{\perp})^{\perp} \\
U \multimap V &\stackrel{\text{def}}{=} (U \otimes V^{\perp})^{\perp}.
\end{aligned}
$$

The Hoare triple relation is defined by means of a satisfaction relation between processes and properties. If $p : A$ and $U \in PA$ then

$$p \models U \iff p \in U$$

and

$$U\{f\}V \iff f \models U \multimap V.$$

Proposition 7 *These definitions yield a specification structure D on $\mathcal{SP}roc$; all the type structure on $\mathcal{SP}roc$ described in Section 3 can be lifted to $\mathcal{SP}roc_D$.*

We illustrate the proof of this proposition by sketching the verification of the key case for composition, i.e. the cut rule:

$$\frac{U\{p\}V \qquad V\{q\}W}{U\{p\,;q\}W} \qquad\qquad \frac{p : (A,U) \to (B,V) \qquad q : (B,V) \to (C,W)}{p\,;q : (A,U) \to (C,W)}$$

To verify $U\{p\,;q\}W$ we must show that, for all $r \in U$ and $s \in W^{\perp}$, $r \otimes s \perp p\,;q$, i.e. that $((r \otimes s) \sqcap (p\,;q))\!\downarrow$. By some elementary process algebra,

$$((r \otimes s) \sqcap (p\,;q)) = (r\,;p) \sqcap (q\,;s)$$

where we regard r as a morphism $r : I \to A$ and s as a morphism $s : C \to I$. Thus it suffices to prove that

$$(r\,;p) \perp (q\,;s)$$

which holds since $r \in U, p \in U \multimap V$ implies $r\,;p \in V$ and similarly $s \in W^{\perp}, q \in V \multimap W$ implies that $q\,;s \in V^{\perp}$.

It can be shown that in general, $U \otimes V$ is properly included in $U \parr V$, and hence in $\mathcal{SP}roc_D$ the operations of $\otimes$ and $\parr$ are distinct. Thus the specification structure leads to a category which, as a model of linear logic, is less degenerate than the original $\mathcal{SP}roc$.

4.2 The Asynchronous Case

The category $\mathcal{SP}roc_D$ allows synchronous problems to be analysed for deadlock-freedom, but there are also many asynchronous systems which we would like to verify. The obvious approach to reasoning about asynchronous deadlock-freedom is to use the delay operators of $\mathcal{SP}roc_D$ to represent asynchrony, and then proceed as before. However, experience has shown that when this is done the only deadlock-free behaviours which the types can guarantee to exist are those in which all the processes in the system delay. Hence we need a version of the specification structure D over $\mathcal{ASP}roc$. This section describes the construction, and illustrates it with an application to the dining philosophers problem.

When we try to define a specification structure for deadlock-freedom over $\mathcal{ASP}roc$, two complications arise which were not present in the synchronous case. The first is to do with *divergence*, or *livelock*. Suppose there are morphisms $f : A \to B$ and $g : B \to C$, each of which runs forever but only does actions in B. Then even if f and g do not deadlock each other, the result of composing them is a morphism which does no observable actions at all—under observation equivalence this is the same as the nil process, which is deadlocked. This shows that when dealing with asynchronous processes, it is insufficient simply to guarantee that processes can always communicate with each other when composed. The second technical problem is that because convergence of a process will mean the ability to continue performing observable actions, there are no convergent processes of type I in $\mathcal{ASP}roc$, and hence no properties over I. This means that the asynchronous deadlock-free category will have no tensor unit; in order to retain the ability to use the $*$-autonomous structure in calculations, a different object will have to be used instead. We will not discuss this issue in the present paper.

To solve the first problem we can adapt Hoare's solution of a similar problem [22]. He considers processes with one input and one output, which can be connected together in sequence. This is actually quite close to the categorical view in some ways: these processes have the "shape" of morphisms and can be composed, although there are no identity processes. More to the point, he is interested in conditions on processes which ensure that connecting them together does not lead to divergence. Restating the question in the categorical framework, if $f : A \to B$ and $g : B \to C$, what is the condition that $f\,;g$ does not diverge? Hoare's solution is to specify that f should be *left-guarded* or g *right-guarded*. Left-guardedness means that every infinite trace of f should contain infinitely many observable actions in A; similarly, right-guardedness means that every infinite trace of g should contain infinitely many observable actions in C. If f is left-guarded it has no infinite behaviours which only involve actions in B, so no matter what g does there can be no divergent behaviour of $f\,;g$. Symmetrically, if g is right-guarded then $f\,;g$ does not diverge. If a process is to be a morphism in a category, it must be composable both on the left and on the right; this means that it needs to be both left-guarded and right-guarded. Requiring that a morphism be both left- and right-guarded, i.e. that every infinite trace must contain infinitely many observable actions in both ports, amounts to a specification of *fairness*. What we need for deadlock-freedom is a category in which all the morphisms are fair in this sense. This issue only arises in the asynchronous case, since in a synchronous category it is impossible for an infinite trace of a process to have anything other than an infinite sequence of actions in each port.

4.2.1 The Category $\mathcal{FP}roc$

The category $\mathcal{FP}roc$ (fair processes) has objects $A = (\Sigma_A, \tau_A, S_A, F_A)$. The first three components of an object are exactly as in $\mathcal{ASP}roc$. The fourth, F_A, is a subset of $\mathsf{ObAct}(A)^\omega$ such that all finite prefixes of any trace in F_A are in S_A. The interaction category operations on objects are defined as in $\mathcal{ASP}roc$, with the addition that

$$\begin{aligned} F_{A^\perp} &\stackrel{\text{def}}{=} F_A \\ F_{A\otimes B} &\stackrel{\text{def}}{=} \{s \in \mathsf{ObAct}(A\otimes B)^\omega \mid s{\restriction}A \in F_A, s{\restriction}B \in F_B\} \\ F_{\bigcirc A} &\stackrel{\text{def}}{=} \{*s \mid s \in F_A\}. \end{aligned}$$

A process in $\mathcal{FP}roc$ is almost the same as a process in $\mathcal{ASP}roc$, except that there now has to be a way of specifying which of the infinite traces of a synchronisation tree are to be considered as actual infinite behaviours of the process. This is done by working with pairs (P, T_P) in which P is an $\mathcal{ASP}roc$ process and $\varnothing \neq T_P \subseteq \mathsf{infobtraces}(P)$. Only the infinite traces in T_P are viewed as behaviours of P, even though the tree P may have many other infinite traces. There is a condition for this specification of valid infinite traces to be compatible with transitions: if $P \stackrel{a}{\Longrightarrow} Q$ then $T_P \supseteq \{as \mid s \in T_Q\}$.

A process of type A in $\mathcal{F}Proc$ is a pair (P, T_P) as above, in which P is a process of type (Σ_A, τ_A, S_A) in $\mathcal{AS}Proc$, and $T_P \subseteq F_A$. Equivalence of processes is defined by

$$(P, T_P) = (Q, T_Q) \stackrel{\text{def}}{\Leftrightarrow} (P \approx Q) \wedge (T_P = T_Q)$$

where the relation $\approx$ is observation equivalence; thus equivalence in $\mathcal{F}Proc$ is a refinement of equivalence in $\mathcal{AS}Proc$.

As usual, a morphism from A to B is a process of type $A \multimap B$. The identity morphism on A in $\mathcal{F}Proc$ is $(\mathsf{id}_A, F_{A \multimap A})$ where id_A is the identity on (Σ_A, τ_A, S_A) in $\mathcal{AS}Proc$. It will often be convenient to refer to $\mathcal{F}Proc$ processes by their first components, and just consider the second components as extra information when necessary; thus the process (P, T_P) may simply be written P.

For composition, if $(f, T_f) : A \to B$ and $(g, T_g) : B \to C$ then $(f, T_f) ; (g, T_g) \stackrel{\text{def}}{=} (f ; g, T_{f;g})$ where

$$\begin{aligned} T_{f;g} \stackrel{\text{def}}{=} \; & \{s \in \mathsf{infobtraces}(f ; g) \mid \\ & \exists t \in T_f, u \in T_g.[t{\restriction}A = s{\restriction}A, t{\restriction}B = u{\restriction}B, u{\restriction}C = s{\restriction}C]\}. \end{aligned}$$

It is straightforward to check that if $T_f \subseteq F_{A \multimap B}$ and $T_g \subseteq F_{B \multimap C}$ then $T_{f;g} \subseteq F_{A \multimap C}$.

The functorial action of $\otimes$ is defined by $(f, T_f) \otimes (g, T_g) \stackrel{\text{def}}{=} (f \otimes g, T_{f \otimes g})$ where, for $f : A \to C$ and $g : B \to D$,

$$\begin{aligned} T_{f \otimes g} \stackrel{\text{def}}{=} \; & \{s \in \mathsf{infobtraces}(f \otimes g) \mid \\ & s{\restriction}(A, C) \in T_f, s{\restriction}(B, D) \in T_g, s \in F_{A \otimes B \multimap C \otimes D}\}. \end{aligned}$$

This definition discards the infinite behaviours of $f \otimes g$ which correspond to unfair interleavings. Effectively, this means that we are assuming the existence of a fair scheduler for parallel composition; keeping the treatment of fairness at the level of specifications, we do not say anything about how such a scheduler might be implemented.

$\mathcal{F}Proc$ inherits the $*$-autonomous structure of $\mathcal{AS}Proc$, because all the structural morphisms, being essentially identities, are fair and the abstraction operation does not affect fairness. The exception to this is that there is no tensor unit: $\mathsf{ObAct}(I) = \varnothing$, so it is not possible to define F_I.

Proposition 8 *$\mathcal{F}Proc$ is a compact closed category without units.*

The definition of $\mathcal{F}Proc$ is very close to the definition of a specification structure over $\mathcal{AS}Proc$—additional properties (the fairness specifications) are defined at each type, and satisfaction of these properties by processes is defined. However, $\mathcal{F}Proc$ does not actually arise from a specification structure. The reason is the assumption of fair interleaving in the definition of $\otimes$. When a specification structure S is defined over a category $\mathbb{C}$, functors on $\mathbb{C}$ are lifted to $\mathbb{C}_S$ by checking that their actions in $\mathbb{C}$ preserve the Hoare triple relations. By contrast, the $\otimes$ functors on $\mathcal{F}Proc$ and $\mathcal{AS}Proc$ have different actions on morphisms.

The specification structure for deadlock-freedom can now be defined over the progressive subcategory $\mathcal{F}Proc_{pr}$ of $\mathcal{F}Proc$, which now consists of those objects for which every safe trace can be extended to a valid infinite trace. The definitions are very similar to those for $\mathcal{S}Proc$. The essential difference is that convergence of a process means the ability to keep doing *observable* actions. Furthermore, the choice of next action should not commit the process to a branch of behaviour which can lead only to a disallowed infinite trace. If $P : A$ then $P{\downarrow}$ means

- whenever $P \stackrel{s}{\Longrightarrow} Q$ there is $a \in \mathsf{ObAct}(A)$ and a process R such that $Q \stackrel{a}{\Longrightarrow} R$, and there is $t \in \mathsf{infobtraces}(R)$ such that $sat \in T_P$.

The definition of equivalence of $\mathcal{F}Proc$ processes P and Q, requiring $P \approx Q$ and $T_P = T_Q$, permits the possibility that although P and Q are not observation equivalent it is only the presence of branches corresponding to invalid infinite traces which causes observation equivalence to fail. However, if a process is convergent then there is no branch along which all infinite traces are invalid, so this situation does not arise. In the specification structure for deadlock-freedom over $\mathcal{F}Proc$, a property is a set of convergent processes and satisfaction is membership, just as in the synchronous case. This means that all the deadlock-free processes considered are convergent, and the equivalence behaves well for them. It is not, however, possible to require that $\mathcal{F}Proc$ should consist *only* of convergent processes, because convergence in itself is not preserved by composition. It is only when convergence is combined with satisfaction of suitable deadlock-free types that composition works.

Given P and Q of type A in $\mathcal{AS}Proc$, $P \sqcap Q$ is defined exactly as in $\mathcal{S}Proc$:

$$\frac{P \xrightarrow{a} P' \qquad Q \xrightarrow{a} Q'}{P \sqcap Q \xrightarrow{a} P' \sqcap Q'.}$$

If P and Q have type A in $\mathcal{F}Proc$ and $T_P \cap T_Q \neq \varnothing$, then $P \sqcap Q$ can be converted into an $\mathcal{F}Proc$ process of type A by defining $T_{P \sqcap Q} \stackrel{\text{def}}{=} T_P \cap T_Q$. Orthogonality is now defined by

$$P \perp Q \quad \stackrel{\text{def}}{\Leftrightarrow} \quad T_P \cap T_Q \neq \varnothing \text{ and } (P \sqcap Q)\!\downarrow .$$

It is extended to sets of processes exactly as in the synchronous case. For each object A, $P_D A$ is again the set of ${}^{\perp\perp}$-invariant sets of convergent processes of type A. Satisfaction is membership, and all of the operations on properties are defined exactly as before.

The proof that D satisfies the composition axiom is similar to the proof in the synchronous case, but is complicated by the presence of silent actions. If $f : A \to B$ and $g : B \to C$ in $\mathcal{F}Proc$, communication between f and g when $f\,;g$ is formed can include periods in which the common action in B is τ_B. This means that several cases arise in the proof, depending on whether f and g are delaying or performing observable actions, and fairness is crucial in dealing with them. Details of the proof can be found in [18].

The proof that identity morphisms satisfy the correct properties is the same as in the synchronous case. Hence

Proposition 9 *D is a specification structure over $\mathcal{F}Proc_{pr}$.*

The asynchronous deadlock-free category is called $\mathcal{F}Proc_D$. For each type A there is a process max_A which has every non-deadlocking behaviour allowed by the safety specification of A. It is defined by

$$\frac{a \in \mathsf{ObAct}(A)}{\mathsf{max}_A \xrightarrow{a} \mathsf{max}_{A/a}}$$

with $T_{\mathsf{max}_A} = F_A$. Note that a process max_A could be defined in this way for any $\mathcal{F}Proc$ object A; max_A is simply the process which exhibits every behaviour permitted by the safety specification S_A. In general max_A might have deadlocking behaviours, but because we are working in $\mathcal{F}Proc_{pr}$, every safe trace can be extended indefinitely and so max_A never terminates.

The process max_A is orthogonal to every convergent process of type A: writing $\mathsf{Proc}(A)$ for the set of all convergent processes of type A, we have $\mathsf{max}_A \perp \mathsf{Proc}(A)$. In fact,

$\mathsf{Proc}(A)^\perp = \{\mathsf{max}_A\}$. $\mathsf{Proc}(A)$ is a valid property over A, as is $\{\mathsf{max}_A\}$, and they are mutually related by $(-)^\perp$. The deadlock-free type $(A, \{\mathsf{max}_A\})$ specifies an input port, because it forces all possible actions to be accepted. The type $(A, \mathsf{Proc}(A))$ specifies an output, because any selection of actions is allowed. From now on, we denote $\mathsf{Proc}(A)$ and $\{\mathsf{max}_A\}$ by out_A and in_A respectively, so that $\mathsf{in}_A^\perp = \mathsf{out}_A$ and $\mathsf{out}_A^\perp = \mathsf{in}_A$. It is not hard to prove

Proposition 10 $\mathsf{out}_A \invamp \mathsf{out}_B = \mathsf{out}_{A \invamp B}$.

If $P : A$ in $\mathcal{FProc}$ and $P\!\downarrow$ then $P \models \mathsf{out}_A$ and so $P : (A, \mathsf{out}_A)$ in $\mathcal{FProc}_D$. Combined with the previous result, this gives

Proposition 11 *If $P : A_1 \invamp \cdots \invamp A_n$ in $\mathcal{FProc}$ and $P\!\downarrow$, then in $\mathcal{FProc}_D$,*

$$P : (A_1, \mathsf{out}_{A_1}) \invamp \cdots \invamp (A_n, \mathsf{out}_{A_n}).$$

This result is very useful for applications, as we shall see in the next section. Another useful fact is that if the safety specification of A is such that in every state there is a unique allowable next action, then $\mathsf{in}_A = \mathsf{out}_A$.

4.3 Constructing Cyclic Networks

The deadlock-free categories $\mathcal{SProc}_D$ and $\mathcal{FProc}_D$ are not compact closed, which means that the categorical structure no longer supports the construction of arbitrary process networks. Any non-cyclic structure can be constructed, using the fact that the category is $*$-autonomous, but additional proof rules are needed to form cycles.

Suppose that $P : (\Gamma, U) \invamp (X, V) \invamp (X^\perp, V^\perp)$ in $\mathcal{FProc}_D$. There is an obvious condition that forming $\overline{P}$ by connecting the X and $X^\perp$ ports should not cause a deadlock: that every trace s of P with $s{\restriction}X = s{\restriction}X^\perp$ can be extended by an action $(\bar{a}, x, x)$ of P. The action x could be τ_X, as it is permissible for the sequence of communications between the X and $X^\perp$ ports to pause, or the action tuple $\bar{a}$ could be τ_Γ, but not both. Again, to obtain $\overline{P} : (\Gamma, U)$ in $\mathcal{FProc}_D$ it is also necessary to ensure that the specification U can still be satisfied while the communication is taking place.

The possibility of divergence does not have to be considered separately. It is conceivable that $\overline{P}$ could have a non-deadlocking infinite behaviour in which no observable actions occur in Γ, but the corresponding behaviour of P would be unfair because it would neglect the ports in Γ. Thus it is sufficient to state a condition which guarantees that forcing X and $X^\perp$ to communicate does not affect the actions available in the other ports. This condition can be expressed in terms of ready pairs. The definition of $\mathsf{readies}(P)$ for an $\mathcal{FProc}$ process P of type A is

$$\begin{aligned} \mathsf{initials}(P) &\stackrel{\text{def}}{=} \{a \in \mathsf{ObAct}(A) \mid \exists Q. P \stackrel{a}{\Longrightarrow} Q\} \\ \mathsf{readies}(P) &\stackrel{\text{def}}{=} \{(s, X) \mid \exists Q.[(P \stackrel{s}{\Longrightarrow} Q) \wedge (X = \mathsf{initials}(Q))]\}. \end{aligned}$$

The condition $\mathsf{cycle}(P)$ is now

- For every $(s, A) \in \mathsf{readies}(P)$ such that $s{\restriction}X = s{\restriction}X^\perp$, and every action $(\bar{a}, x, y) \in A$, there is $z \in \Sigma_X$ such that $\tau_{\Gamma \invamp X \invamp X^\perp} \neq (\overline{a}, z, z) \in A$.

This leads to a proof rule for cycle formation.

$$\frac{P : (\Gamma, U) \invamp (X, V) \invamp (X^\perp, V^\perp) \quad \mathsf{cycle}(P)}{\overline{P} : (\Gamma, U)}$$

This rule illustrates one of the main features of our approach—the combination of type-theoretic and traditional verification techniques. Typically, the construction of a process will be carried out up to a certain point by means of the linear combinators, and its correctness will be guaranteed by the properties of the type system. This phase of the verification procedure is completely compositional. However, if cyclic connections are to be formed, some additional reasoning about the behaviour of the process is needed. The nature of this reasoning is embodied in the above proof rule. The rule is not compositional, in the sense that the internal structure of P must be examined to some extent in order to validate the condition $\mathsf{cycle}(P)$, but the departure from compositionality is only temporary. Once the hypotheses of the proof rule have been established, the result is that $\overline{P}$ has a type, and can be combined with other processes purely on the basis of that type.

5 The Dining Philosophers

The problem of the dining philosophers [22] provides a good example of working with the category of asynchronous deadlock-free processes. Our analysis of it will make use of the proof rule for cycle formation, introduced in the previous section, and thus illustrates the combination of type-theoretic arguments with more traditional reasoning. The example itself is well-known in the concurrency literature, but it is worth reviewing the scenario here before plunging into an analysis.

In a college there are five philosophers, who spend their lives seated around a table. In the middle of the table is a large bowl of spaghetti; also on the table are five forks, one between each pair of philosophers. Each philosopher spends most of his time thinking, but occasionally becomes hungry and wants to eat. In order to eat, he has to pick up the two nearest forks; when he has finished eating, he puts the forks down again. The problem consists of defining a concurrent system which models this situation; there are then various questions which can be asked about its behaviour. One is about deadlock-freedom: is it possible for the system to reach a state in which nothing further can happen, for example because the forks have been picked up in an unsuitable way? Another is about fairness: do all the philosophers get a chance to eat, or is it possible for one of them to be excluded forever? The reason for looking at the dining philosphers example in this paper is to illustrate techniques for reasoning about deadlock-freedom, but because of the way in which the asynchronous deadlock-free category has been constructed, fairness has to be considered as well.

A philosopher can be modelled as a process with five possible actions: eating, picking up the left fork, putting down the left fork, picking up the right fork, and putting down the right fork. Calling these actions e, lu, ld, ru, rd respectively, a CCS definition of a philosopher could be $P = lu.ru.e.ld.rd.P$. There is no action corresponding to thinking: a philosopher is deemed to be thinking at all times, unless actually doing something else. In $\mathcal{ASProc}$ a philosopher has three ports: one for the eating action and one each for the left and right forks. The type of the fork ports is X, defined by $\Sigma_X \stackrel{\text{def}}{=} \{u, d, \tau_X\}$ and with S_X requiring u and d to alternate, starting with u. The type of the eating port is Y defined by $\Sigma_Y \stackrel{\text{def}}{=} \{e, \tau_Y\}$ and with S_Y allowing all traces. The philosopher process can be typed as $P : X^\perp \mathbin{⅋} Y \mathbin{⅋} X$.

A fork has four actions, lu, ld, ru and rd. For the usage of these names by the fork to match their usage by the philosophers, the necessary convention is that if a fork does the action lu, it has been picked up from the right. A possible definition of a fork is $K = lu.ld.K + ru.rd.K$ and it can be typed as $K : X^\perp \mathbin{⅋} X$.

Five philosophers and five forks can be connected together in the desired configuration,

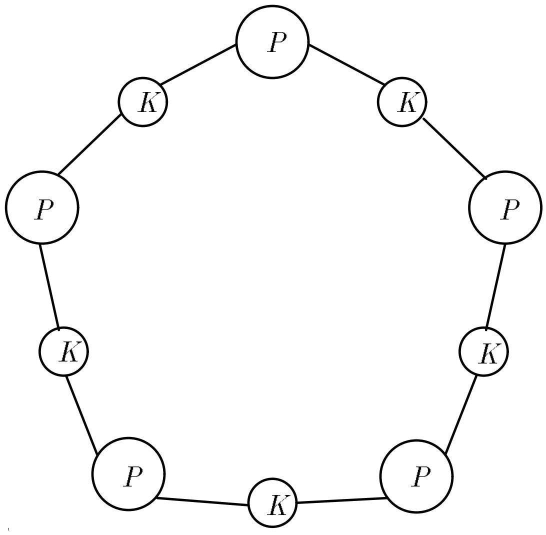


Figure 1: Process Configuration for the Dining Philosophers

illustrated in Figure 1, by using the compact closed structure of $\mathcal{ASProc}$, as usual. The next step is to transfer everything to $\mathcal{FProc}$ and then to $\mathcal{FProc}_D$.

To construct the P and K processes in $\mathcal{FProc}$, fairness specifications must be added to the types X and Y, and the acceptable infinite behaviours of P and K must be specified. This will be done in such a way that P and K satisfy the appropriate fairness specifications. For both X and Y the fairness specification can simply be all infinite traces. This means that there is no fairness requirement on the actions within a port, but only between ports. For the types of the philosopher and the fork, $F_{X^\perp \mathbin{⅋} Y \mathbin{⅋} X}$ consists of the infinite traces whose projections into the three ports are all infinite, and similarly $F_{X^\perp \mathbin{⅋} X}$.

To convert the $\mathcal{ASProc}$ process P into an $\mathcal{FProc}$ process, it is sufficient to take $T_P = \mathsf{infobtraces}(P)$. It is then clear that $T_P \subseteq F_{X^\perp \mathbin{⅋} Y \mathbin{⅋} X}$ because the behaviour of P simply cycles around all the available actions. Also, P is convergent because its behaviour consists of just one infinite branch. However, K has unfair infinite behaviours—for example, there is an infinite trace in which the ru and rd actions never appear. Thus T_K must be defined in such a way as to eliminate these undesirable infinite traces, and this can easily be done by taking $T_K = F_{X^\perp \mathbin{⅋} X}$. Then K is convergent, because any of its finite behaviours can be extended to a fair infinite behaviour by choosing a suitable interleaving from that point on. This approach means that this section is not addressing the issue of how fairness can be achieved in the dining philosophers problem—to do that, the implementation of a fair scheduler would have to be considered. As already stated, this problem has been introduced as an example of compositional reasoning about deadlock-freedom; fairness only appears in the minimal possible way needed for the categorical approach to be applicable.

Typing the philosopher and fork processes in $\mathcal{FProc}_D$ requires suitable properties over the types X and Y. For Y, out_Y can be used. Because Y has only one observable action, $\mathsf{out}_Y = \mathsf{out}_Y^{\perp\perp}$. Similarly for X, the set out_X can be used, and because the safety specification of X is such that in each state there is only one action available, $\mathsf{out}_X = \mathsf{out}_X^{\perp\perp}$. Because $K : X^\perp \mathbin{⅋} X$ in $\mathcal{FProc}$ and K is convergent, $K \models \mathsf{out}_{X^\perp} \mathbin{⅋} \mathsf{out}_X$ and so $K : (X, \mathsf{out}_X)^\perp \mathbin{⅋} (X, \mathsf{out}_X)$ in $\mathcal{FProc}_D$. Similarly, $P : (X, \mathsf{out}_X)^\perp \mathbin{⅋} (Y, \mathsf{out}_Y) \mathbin{⅋} (X, \mathsf{out}_X)$ in $\mathcal{FProc}_D$. These typings mean that any number of philosophers and forks can be connected together in a line, and the resulting process is guaranteed to be deadlock-free.

Interestingly, this applies not only to the "correct" configuration in which philosophers and forks alternate, but also to other possibilities such as a sequence of forks with no philosophers.

The interesting step of the construction consists of completing the cycle by connecting the X and $X^{\perp}$ ports at opposite ends of a chain in which forks and philosophers alternate. Because $\mathcal{FProc}_D$ is not compact closed, the proof rule of the previous section must be used. First of all, some traditional analysis based on reasoning about the state of the system is useful. For the moment, the e actions can be ignored as they do not have any impact on deadlocks in this system. The following cases cover all possibilities for a state.

1. If there is P_i such that both adjacent forks are down, it can pick up the left fork.

2. If there is P_i whose right fork is up and whose left fork is down, it can either put down the right fork (if it has just put down the left fork) or pick up the left fork (if its neighbour has the right fork).

3. If all forks are up and some P_i has both its forks, it can put down the left fork.

4. If all forks are up and every P_i has just one fork, they all have their left forks, and there is a deadlock.

The last case is the classic situation in which the dining philosophers may deadlock—each philosopher in turn picks up the left fork, and then they are stuck. In terms of ready sets, there is a state in which every possible next action has non-matching projections in the two X ports.

In Hoare's formulation of the dining philosophers problem [22] the philosophers are not normally seated, but have to sit down before attempting to pick up their forks. This means that the possibility of deadlock can be removed by adding a *footman*, who controls when the philosophers sit down. The footman ensures that at most four philosophers are seated at any one time, which means that there is always a philosopher with an available fork on both sides; in this way, the deadlocked situation is avoided. However, implementing this solution involves a major change to the system: there is a new process representing the footman, the philosopher processes have extra ports on which they interact with the footman, and consequently their types need to be re-examined. It is more convenient to use an alternative approach, which will now be described.

One of the philosophers is replaced by a variant, P', which picks up the forks in the opposite order. So $P' = ru.lu.e.rd.ld.P'$ in CCS notation. Intuitively, this prevents the deadlocking case from arising, because even if the four Ps each pick up their left fork, P' is still trying to pick up its right fork (which is already in use) and so one of the Ps has a chance to pick up its right fork as well. The check that there are no deadlocks takes the form of a case analysis, as before.

1. If all the forks are up and some philosopher has both its forks, it can put one of them down, whether it is P or P'.

2. If all the forks are up and every philosopher has just one, either they all have their left fork or all the right. If they all have their left fork, then P' can put down its left fork. If they all have their right fork, then any P can put down its right fork.

3. If two adjacent forks are down, then the philosopher in between them can pick one of them up, whether it is P or P'.

4. Otherwise there is the configuration $u - \texttt{phil}_1 - d - \texttt{phil}_2 - u - \texttt{phil}_3$.

- If $\texttt{phil}_2$ is P and has its right fork, it can put down the right fork.
- If $\texttt{phil}_2$ is P and doesn't have its right fork, it can pick up the left fork.
- If $\texttt{phil}_2$ is P' and has its right fork, it can pick up the left fork.
- If $\texttt{phil}_2$ is P' and doesn't have its right fork, then $\texttt{phil}_3$ must be P and has its left fork. Then if $\texttt{phil}_3$'s right fork is down, $\texttt{phil}_3$ can pick it up. If the right fork is up and $\texttt{phil}_3$ has it, it can put down the left fork. Otherwise, $\texttt{phil}_4$ is P and has its left fork. Continuing this argument for each $\texttt{phil}_i$ with $i \geqslant 4$ leads eventually to either a possible action, or cyclically back to $i = 1$ and the deduction that $\texttt{phil}_1$ has its left fork. In the latter case, since $\texttt{phil}_1$ is P, it can pick up its right fork.

To recast this argument in terms of checking the condition on the final cyclic connection, suppose that the final connection is between the P' process and the fork on its right. Each case of the argument either produces a communication between P' and this fork, or produces a communication elsewhere in the cycle, which means that there is an action of the system in which the two ports to be connected both delay. This shows that the cycle condition is satisfied, and the proof rule can be applied.

Acknowledgements

We would like to thank Rick Blute, Robin Cockett, Phil Scott and David Spooner for their detailed reviews of this paper.

References

[1] S. Abramsky, S. J. Gay, and R. Nagarajan. Interaction categories and foundations of typed concurrent programming. In M. Broy, editor, *Deductive Program Design: Proceedings of the 1994 Marktoberdorf International Summer School*, NATO ASI Series F: Computer and Systems Sciences. Springer-Verlag, 1995. Also available as `theory/papers/Abramsky/marktoberdorf.ps.gz` via anonymous ftp to `theory.doc.ic.ac.uk`.

[2] S. Abramsky. Domain theory in logical form. *Annals of Pure and Applied Logic*, 51:1–77, 1991.

[3] S. Abramsky. Interaction Categories (Extended Abstract). In G. L. Burn, S. J. Gay, and M. D. Ryan, editors, *Theory and Formal Methods 1993: Proceedings of the First Imperial College Department of Computing Workshop on Theory and Formal Methods*, pages 57–70. Springer-Verlag Workshops in Computer Science, 1993.

[4] S. Abramsky. Interaction Categories and communicating sequential processes. In A. W. Roscoe, editor, *A Classical Mind: Essays in Honour of C. A. R. Hoare*, pages 1–15. Prentice Hall International, 1994.

[5] S. Abramsky. Proofs as processes. *Theoretical Computer Science*, 135:5–9, 1994.

[6] S. Abramsky. Interaction Categories I: Synchronous processes. Paper in preparation, 1995.

[7] S. Abramsky and R. Jagadeesan. Games and full completeness for multiplicative linear logic. *Journal of Symbolic Logic*, 59(2):543 – 574, June 1994.

[8] S. Abramsky and R. Jagadeesan. New foundations for the geometry of interaction. *Information and Computation*, 111(1):53–119, 1994.

[9] S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF (extended abstract). In M. Hagiya and J. C. Mitchell, editors, *Theoretical Aspects of Computer Software. International Symposium TACS'94*, number 789 in Lecture Notes in Computer Science, pages 1–15, Sendai, Japan, April 1994. Springer-Verlag.

[10] A. Asperti and G. Longo. *Categories, Types and Structures : An introduction to category theory for the working computer scientist.* Foundations of Computing Series. MIT Press, 1991.

[11] M. Barr. $*$-autonomous categories and linear logic. *Mathematical Structures in Computer Science*, 1(2):159–178, July 1991.

[12] G. Berry and P.-L. Curien. Theory and practice of sequential algorithms: the kernel of the applicative language CDS. In J. C. Reynolds and M. Nivat, editors, *Algebraic Semantics*, pages 35–84. Cambridge University Press, 1985.

[13] R. Blute. Linear logic, coherence and dinaturality. *Theoretical Computer Science*, 115(1):3–41, 1993.

[14] P. M. Cohn. *Universal Algebra*, volume 6. D. Reidel, 1981.

[15] R. L. Crole. *Categories for Types.* Cambridge University Press, 1994.

[16] J. W. de Bakker. *Mathematical Theory of Program Correctness.* Prentice Hall International, 1980.

[17] W. P. de Roever. The quest for compositionality—a survey of assertion based proof systems for concurrent programs, Part I: Concurrency based on shared variables. In *Proceedings of the IFIP Working Conference*, 1985.

[18] S. J. Gay. *Linear Types for Communicating Processes.* PhD thesis, University of London, 1995. Available as `theory/papers/Gay/thesis.ps.gz` via anonymous ftp to `theory.doc.ic.ac.uk`.

[19] G. Gentzen. Investigations into logical deduction. In M. E. Szabo, editor, *The Collected Papers of Gerhard Gentzen.* North-Holland, 1969.

[20] J.-Y. Girard. Linear Logic. *Theoretical Computer Science*, 50(1):1–102, 1987.

[21] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, 1989.

[22] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.

[23] N. D. Jones and F. Nielson. Abstract interpretation. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 4. Oxford University Press, 1995. To appear.

[24] G. M. Kelly and M. L. Laplaza. Coherence for compact closed categories. *Journal of Pure and Applied Algebra*, 19:193–213, 1980.

[25] D. C. Kozen and J. Tiuryn. Logics of programs. In van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 789–840. North Holland, 1990.

[26] S. Mac Lane. *Categories for the Working Mathematician.* Springer-Verlag, Berlin, 1971.

[27] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems.* Springer-Verlag, 1992.

[28] J. McKinna and R. Burstall. Deliverables: A categorical approach to program development in type theory. In *Proceedings of Mathematical Foundation of Computer Science*, 1993.

[29] R. Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25:267–310, 1983.

[30] R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[31] R. Milner, M. Tofte, and R. Harper. *The Definition of Standard ML.* MIT Press, 1990.

[32] P. W. O'Hearn and R. D. Tennent. Relational parametricity and local variables. In *Proceedings, 20th ACM Symposium on Principles of Programming Languages.* ACM Press, 1993.

[33] A. M. Pitts. Relational properties of recursively defined domains. In *8th Annual Symposium on Logic in Computer Science*, pages 86–97. IEEE Computer Society Press, Washington, 1993.

[34] R. Soare. *Recursively Enumerable Sets and Degrees.* Perspectives in Mathematical Logic. Springer-Verlag, Berlin, 1987.

[35] J. B. Wells. Typability and type checking in the second-order λ-calculus are equivalent and undecidable. In *Proceedings, Ninth Annual IEEE Symposium on Logic in Computer Science.* IEEE Computer Society Press, 1994.