Types for Deadlock-freedom: The Synchronous Case

Samson Abramsky
Department of Computer Science,
University of Edinburgh,
Mayfield Road,
Edinburgh EH9 3JZ, UK.
samson@dcs.ed.ac.uk

Simon Gay
Department of Computer Science,
Royal Holloway,
University of London,
Egham, Surrey TW20 0EX, UK.
S.Gay@dcs.rhbnc.ac.uk

Rajagopal Nagarajan
Department of Computing,
Imperial College of Science,
Technology and Medicine,
180 Queen's Gate,
London SW7 2BZ, UK
&
Electronics Research Laboratory,
University of California,
Berkeley, CA 94720, USA.
R.Nagarajan@doc.ic.ac.uk

June 18, 1997

Abstract

Many different notions of "program property", and many different methods of verifying such properties, arise naturally in programming. We present a general framework of *Specification Structures* for combining different notions and methods in a coherent fashion. We then apply the idea of specification structures to concurrency in the setting of Interaction Categories. As a specific example, a certain specification structure defined over the interaction category $\mathcal{SP}roc$ yields a new category $\mathcal{SP}roc_D$ whose type system is strong enough to guarantee deadlock-freedom of concurrent processes. We present some techniques for manipulating typed processes in this category, and show that they allow us to reason about deadlock-freedom in *synchronous networks*, a class of concurrent systems which incorporates both synchronous dataflow programs and systolic algorithms.

1 Introduction

Type Inference and *Verification* are two main paradigms for constraining the behaviour of programs in such a way as to guarantee desirable properties. Although they are generally perceived as 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 [39] are relatively weak as regards expressing behavioural constraints, but correspondingly tractable as regards efficient algorithms for "type checking". System F types [24] 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 [41], with dependent types and equality types, can express complete total correctness specifications. In the richer theories, type checking is undecidable [47].

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 follow.

- 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 [19].
- In the field of static analysis, and particularly in the systematic framework of abstract interpretation [28], 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 [34].

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":

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

The idea behind the picture is that we have a semantic universe (category with structure) $\mathcal{C}_0$, suitable for modelling some computational situation, but possibly carrying only a very rudimentary notion of "type" or "behavioural specification". The tower arises by refining $\mathcal{C}_0$ with richer kinds of property, so that we obtain progressively richer settings for performing specification and verification. 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 remainder of this paper provides a detailed development of this idea in the setting of interaction categories [1, 5, 6], with particular reference to synchronous systems. Section 2 introduces the notion of a *specification structure*, which formalizes the idea of enriching a

semantic universe with a refined notion of property. Section 3 reviews the theory of interaction categories and defines $\mathcal{SP}roc$, a category of synchronous processes. In Section 4 we explicitly state the requirements for a specification structure to be defined over an interaction category such as $\mathcal{SP}roc$, and in Section 5 we define a particular specification structure over $\mathcal{SP}roc$. The result is a category $\mathcal{SP}roc_D$, in which types are strong enough to specify deadlock-freedom, and deadlock-freedom can be verified compositionally. In Section 6 we give an alternative presentation of this specification structure, which is sometimes more convenient for calculations, and prove that it is equivalent. As an application of the theory developed in the rest of the paper, Section 7 analyses the construction of a class of *synchronous networks*, which encompasses both synchronous dataflow programs in languages such as SIGNAL [25] and LUSTRE [26], and systolic algorithms [20]. Finally we compare our theory with other approaches, and discuss the possibilities for further developments.

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 [35], O'Hearn and Tennent [42], and Pitts [44].

Definition 2.1 *Let $\mathcal{C}$ be a category. A* specification structure *S over $\mathcal{C}$ is defined by the following data:*

- *for each object A of $\mathcal{C}$, a set $P_S A$ of "properties over A".*
- *for each pair of objects A, B of $\mathcal{C}$, a relation $S_{A,B} \subseteq P_S A \times \mathcal{C}(A,B) \times P_S B$.*

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

$$\varphi\{\mathsf{id}_A\}\varphi \tag{1}$$

$$\varphi\{f\}\psi, \psi\{g\}\theta \Longrightarrow \varphi\{f\,;g\}\theta \tag{2}$$

The axioms (1) and (2) are typed versions of the standard Hoare logic axioms for "skip" and "sequential composition" [19]. Given $\mathcal{C}$ and S as above, we can define a new category $\mathcal{C}_S$. An object of $\mathcal{C}_S$ is a pair (A, φ) with $A \in \mathsf{ob}\,\mathcal{C}$ and $\varphi \in P_S A$. A $\mathcal{C}_S$-morphism $f : (A, \varphi) \to (B, \psi)$ is a morphism $f : A \to B$ in $\mathcal{C}$ such that $\varphi\{f\}\psi$.

Composition and identities are inherited from $\mathcal{C}$; the axioms (1) and (2) ensure that $\mathcal{C}_S$ is a category. Moreover, there is a faithful functor

$$\mathcal{C} \leftarrow\!\!\!\shortmid \mathcal{C}_S$$

given by

$$A \mapsto (A, \varphi).$$

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

$$\begin{array}{rcl} P_S A & = & \{\varphi \in \mathsf{ob}\,\mathcal{D} \mid F(\varphi) = A\} \\ \varphi\{f\}\psi & \iff & \exists \alpha \in \mathcal{D}(\varphi, \psi).\, F(\alpha) = f \end{array}$$

(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 $\mathcal{C} \leftarrow \mathcal{C}_S$.

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

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

Then (1) and (2) 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 $\mathcal{C} \leftarrow \mathcal{C}_S$ 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 $\mathcal{C}$ which should be lifted to $\mathcal{C}_S$. Suppose for example that $\mathcal{C}$ is a monoidal category, i.e. there is a bifunctor $\otimes : \mathcal{C}^2 \to \mathcal{C}$, an object I, and natural isomorphisms

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

satisfying the standard coherence equations [33]. A specification structure for $\mathcal{C}$ must then correspondingly be extended with an action

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

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

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

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

Now assume that $\mathcal{C}$ 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

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

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

$$\begin{array}{c} (\varphi \otimes \psi)\{\mathsf{symm}_{A,B}\}(\psi \otimes \varphi) \\ ((\varphi \multimap \psi) \otimes \varphi)\{\mathsf{eval}_{A,B}\}\psi \end{array}$$

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

Going one step further, suppose that $\mathcal{C}$ is a $*$-autonomous category, i.e. a model for the multiplicative fragment of classical linear logic [13], 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 of this structure on $\mathcal{C}$ lifts to $\mathcal{C}_S$. 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 $\mathcal{C}_S$ work exactly as they do in $\mathcal{C}$, the above axioms guaranteeing that these constructions are well-defined in $\mathcal{C}_S$. 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 of this structure is preserved by the faithful functor $\mathcal{C} \leftarrow\!\!\!\prec \mathcal{C}_S$.

The above example of structure on $\mathcal{C}$ 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 [11]
- cartesian closed categories [18]
- models of polymorphism [18].

2.1 Examples of Specification Structures

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

1. $\mathcal{C} = \mathcal{S}et$, $P_S X = X$, $a\{f\}b \stackrel{\text{def}}{\Leftrightarrow} f(a) = b$.

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

2. $\mathcal{C} = \mathcal{R}el$, $P_S X = \{*\}$, $*\{R\}* \stackrel{\text{def}}{\Leftrightarrow} \forall x \in A, y, z \in B.\, xRy \wedge xRz \Rightarrow y = z$.

   Then $\mathcal{C}_S$ is the category of sets and partial functions.

3. $\mathcal{C} = \mathcal{R}el$, $PX = \wp X$, $S\{R\}T \stackrel{\text{def}}{\Leftrightarrow} \forall x \in S.\, \{y \mid xRy\} \subseteq T$.

   This is essentially a typed version of dynamic logic [32], with the "Hoare triple relation" specialized to its original setting. If we take

   $$\begin{aligned} S \otimes_{X,Y} T &= S \times T \\ S^{\perp}_{X} &= X \backslash S \end{aligned}$$

   then $\mathcal{C}_S$ becomes a model of classical linear logic.

4. $\mathcal{C} = \mathcal{R}el, P_S X = \{C \subseteq X^2 \mid C = C^o, C \cap \mathsf{id}_X = \varnothing\}$

$$\begin{aligned} C\{R\}D &\overset{\text{def}}{\Leftrightarrow} xCx', xRy, x'Ry' \Rightarrow yDy'. \\ C \otimes D &= \{((x,x'),(y,y')) \mid xCy \wedge x'Dy'\} \\ C^{\perp}_{X} &= X^2 \backslash (C \cup \mathsf{id}_X). \end{aligned}$$

$\mathcal{C}_S$ is the category of coherence spaces and linear maps [23].

5. $\mathcal{C} = \mathcal{S}et, P_S X = \{s : \omega \rightharpoonup X \mid \forall x \in X. \exists n \in \omega. s(n) = x\}$,

$$s\{f\}t \overset{\text{def}}{\Leftrightarrow} \exists n \in w.\, f \circ s \simeq t \circ \varphi_n$$

where φ_n is the nth partial recursive function in some acceptable numbering [45]. Then $\mathcal{C}_S$ is the category of modest sets, seen as a full subcategory of ω-$\mathcal{S}et$ [11].

6. $\mathcal{C}$ = the category of SFP domains,
$P_S D = K\Omega(D)$(the compact-open subsets of D),
$U\{f\}V \overset{\text{def}}{\Leftrightarrow} U \subseteq f^{-1}(V)$.

This yields (part of) Domain Theory in Logical Form [3], the other part arising from the local lattice-theoretic structure of the sets $P_S D$ and its interaction with the global type structure.

7. $\mathcal{C}$ = games and partial strategies, as in [10], PA = all sets of infinite plays, $U\{\sigma\}V$ iff σ is *winning* with respect to U, V in the sense of [8]. Then $\mathcal{C}_S$ is the category of games and winning strategies of [8].

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

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

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

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

Each such step adds propositional information to the underlying "raw" computational entities (morphisms of $\mathcal{C}_0$). The aim of verification in this framework is to "promote" a morphism from $\mathcal{C}_i$ to $\mathcal{C}_j$, $i < j$. That is, to promote a $\mathcal{C}_0$ morphism $f : A \to B$ to a $\mathcal{C}_k$ 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 available in $\mathcal{C}_k$ to participate in typing judgements. In this way, a coherent framework for combining methods, including both compositional and non-compositional approaches, begins to open up. In the remainder of the paper, we will use specification structures defined over interaction categories to construct type systems for the specification and verification of concurrent systems.

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

The theory of *Interaction Categories* has been proposed as a new paradigm for the semantics of sequential and concurrent computation [1, 5, 6]. The term encompasses certain known categories (the category of concrete data structures and sequential algorithms [14], categories of games [8], geometry of interaction categories [9]) as well as several new categories for concurrency. The fundamental examples of concurrent interaction categories are $\mathcal{SP}roc$ [5], the category of synchronous processes, and $\mathcal{ASP}roc$ [6], the category of asynchronous processes. The category $\mathcal{SP}roc$ will be defined in this section; later we will use a specification structure over $\mathcal{SP}roc$ to construct another interaction category.

The general picture of interaction 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 [21]. The common features of the existing examples are that they have $*$-autonomous structure, which corresponds to the multiplicative fragment of classical linear logic [23]; products and coproducts, corresponding to the additives of linear logic, and additional temporal structure which enables the dynamics of process evolution to be described.

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, 21].

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 [36]. It is convenient to work with *synchronization trees* as canonical representatives of strong bisimulation classes. We will write $\mathsf{ST}(\Sigma)$ for the set of synchronization trees over a label-set Σ. We will usually define processes by means of labelled transition rules. The process of type A with no transitions is called nil_A, or sometimes nil if the type is clear from the context. We use the notation $P \xrightarrow{a} P'$ for labelled transitions, and $P \xrightarrow{s}{}^* P'$ when a trace (sequence of actions) s is involved. The empty trace is denoted ε, and concatenation of traces s and t is simply written st. We do not distinguish notationally between the action a and the trace containing only the action a.

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 $\mathsf{inftraces}(P)$. Equivalently, we may use the following coinductive definition.

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

$$\mathsf{inftraces}(P) \stackrel{\text{def}}{=} \{\sigma \in \mathsf{alltraces}(P) \mid \sigma \text{ is infinite}\}.$$

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 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 $*$-autonomous categories of games [8] 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 $A \multimap B \stackrel{\text{def}}{=} (A \otimes B^\perp)^\perp$, $A \parr B \stackrel{\text{def}}{=} (A^\perp \otimes B^\perp)^\perp$. In $\mathcal{SP}roc$, we have $A = A^\perp$, and hence $A \parr B = A \multimap B = A \otimes B$. 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 6, gives distinct interpretations to $\otimes$ and $\parr$.

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. An example of composition is shown in Figure 1. 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

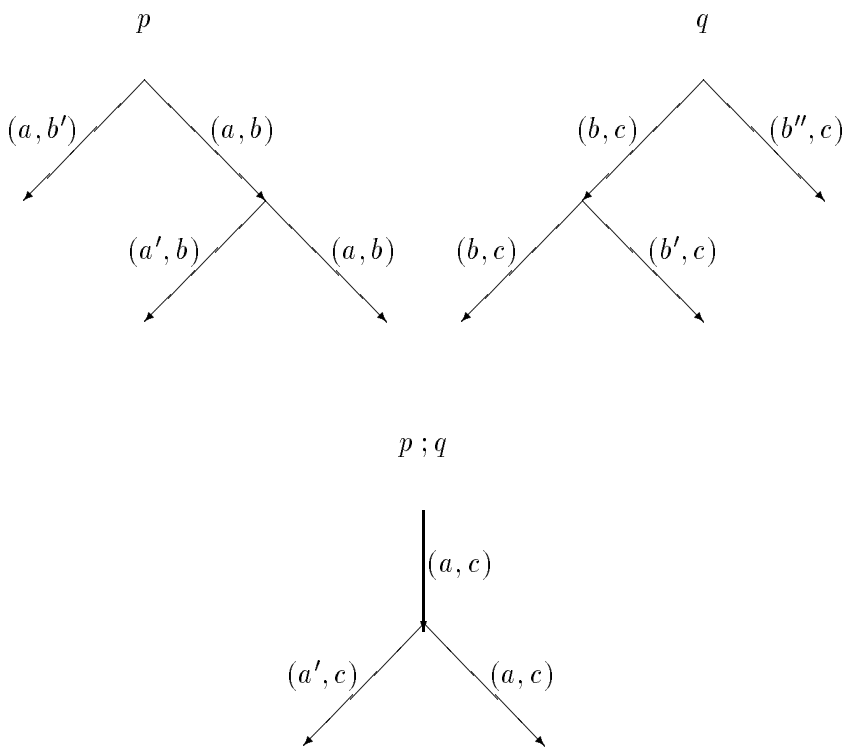


Figure 1: Composition in $\mathcal{SP}roc$

$P \restriction S$, 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.

It is useful at this point to define the object A/s, where A is an object of $\mathcal{SP}roc$ and $s \in S_A$:

$$\begin{aligned} \Sigma_{A/s} &\stackrel{\text{def}}{=} \Sigma_A \\ S_{A/s} &\stackrel{\text{def}}{=} S_A/s. \end{aligned}$$

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 $\Sigma_{A \multimap A}$ is defined by the transition rule

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

Proposition 3.1 *$\mathcal{SP}roc$ is a category.*

Proof To prove that composition is associative and that identities work correctly, the strategy is to show that a suitable relation on processes is a bisimulation, and argue by coinduction. To prove that $p;(q;r) = (p;q);r$ for all $p : A \to B$, $q : B \to C$ and $r : C \to D$, the relation is

$$\{(p\,;(q\,;r),(p\,;q)\,;r) \mid p \in \mathsf{ST}(\Sigma_{A \multimap B}), q \in \mathsf{ST}(\Sigma_{B \multimap C}), r \in \mathsf{ST}(\Sigma_{C \multimap D})\}.$$

To prove that $p\,;\mathsf{id}_B = p$ for all $p : A \to B$, the relation is

$$\{(p\,;\mathsf{id}, p) \mid p \in \mathsf{ST}(\Sigma_{A \multimap B})\}$$

where id has sort $\Sigma_{B \multimap B}$. In each case, the fact that the relation is a bisimulation follows easily from the transition rules defining composition. $\square$

The following definition will be useful later.

Definition 3.2 *If A is an object of $\mathcal{SP}roc$ and $s \in S_A$, then*

$$\Sigma_A(s) \stackrel{\text{def}}{=} \{a \in \Sigma_A \mid sa \in S_A\}.$$

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.

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

The tensor unit I is defined by

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

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 below. We use a pattern-matching notation to define the partial functions needed for the relabelling operations; for example, $(a, a) \mapsto ((\bullet, a), a)$ denotes the partial function which has the indicated effect when its arguments are equal.

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

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.

If P is a process of type A then $P[a \mapsto (\bullet, a)]$ 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.

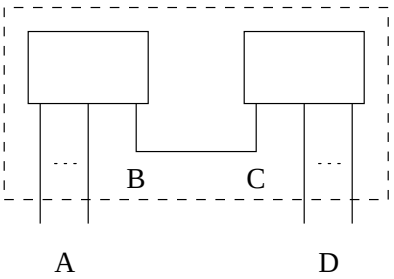


Figure 2: Using the Cut rule to connect modules

Proposition 3.3 *$\mathcal{SP}roc$ 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 $\parr$. □

The following result on relabellings will be useful later.

Lemma 3.4 *If $p \in \mathsf{ST}(\Sigma_{A \multimap B})$, $q \in \mathsf{ST}(\Sigma_{B \multimap C})$ and $f : \Sigma_B \to \Sigma_D$ is an isomorphism, then*

$$p[(a,b) \mapsto (a, f(b))] \,;\, q[(b,c) \mapsto (f(b),c)] = p \,;\, q.$$

Proof It follows from the definitions of relabelling and composition that the relation

$$\{(p[(a,b) \mapsto (a,f(b))] \,;\, q[(b,c) \mapsto (f(b),c)], p \,;\, q) \mid p \in \mathsf{ST}(\Sigma_{A \multimap B}), q \in \mathsf{ST}(\Sigma_{B \multimap C})\}$$

is a bisimulation. □

3.1.2 Compact Closure and Multi-Cut

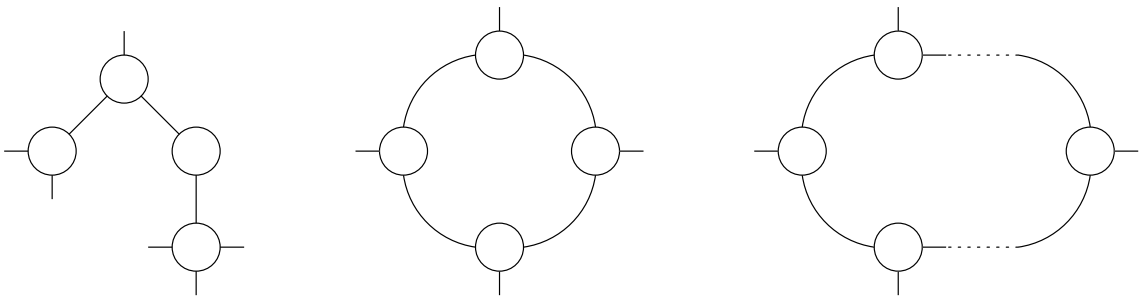
As we have already seen the linear type structure of $\mathcal{SP}roc$ is quite degenerate. Specification structures can be used to enrich the specifications of $\mathcal{SP}roc$ to stronger behavioural properties. This will have the effect of "sharpening up" the linear type structure so that the degeneracies 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 *multi-cut*. (This is actually Gentzen's MIX rule [22] but we avoid the use of this term since Girard has used it for a quite 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 [7], as in Figure 2. This allows us to connect processes in a tree structure, as in Figure 3(a), but not to construct cyclic interconnection networks as in Figure 3(b). The problem with constructing a cycle occurs at the final step, when two processes must be plugged together on two ports simultaneously as in Figure 3(c). Cyclic connections would be supported if



(a) (b) (c)

Figure 3: Cyclic and acyclic networks

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" rule:

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

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

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

$$f : I \to \tilde{A} \otimes \tilde{C} \qquad g : I \to \tilde{B} \otimes \tilde{C}^\perp$$

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

$$\begin{array}{ccccc}
I & \xrightarrow{\sim} & I \otimes I & \xrightarrow{f \otimes g} & (\tilde{A} \otimes \tilde{C}) \otimes (\tilde{B} \otimes \tilde{C}^\perp) \\
\Big\downarrow h & & & & \Big\downarrow \sim \\
\tilde{A} \otimes \tilde{B} & \xleftarrow{\sim} & \tilde{A} \otimes I \otimes \cdots \otimes I \otimes \tilde{B} & \xleftarrow{\mathsf{evaluate}} & \tilde{A} \otimes (C_1 \otimes C_1^\perp) \otimes \cdots \otimes (C_k \otimes C_k^\perp) \otimes \tilde{B}
\end{array}$$

Note that in a compact closed category $I = \perp$ so $A^\perp = A \multimap I$. Arrows labelled by $\sim$ are canonical isomorphisms, and the morphism $\mathsf{evaluate}$ is $\mathsf{id} \otimes \mathsf{Ap} \otimes \cdots \otimes \mathsf{Ap} \otimes \mathsf{id}$.

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. Some related notions, arising in work on coherence in compact closed categories, can be found in the literature [15, 29].

3.2 Products, Coproducts and Non-determinism

The binary coproduct functor $\oplus$ is defined on objects by

$$\begin{array}{rcl} \Sigma_{A\oplus B} & \stackrel{\text{def}}{=} & \Sigma_A + \Sigma_B \\ S_{A\oplus B} & \stackrel{\text{def}}{=} & \{\mathsf{inl}^*(s) \mid s \in S_A\} \\ & \cup & \{\mathsf{inr}^*(s) \mid s \in S_B\}. \end{array}$$

If $p : A \to C$ and $q : B \to D$ then $p \oplus q : A \oplus B \to C \oplus D$ is defined by

$$\begin{array}{rcl} p \oplus q & \stackrel{\text{def}}{=} & p[(a,c) \mapsto (\mathsf{inl}(a), \mathsf{inl}(c))] \\ & + & q[(b,d) \mapsto (\mathsf{inr}(b), \mathsf{inr}(d))]. \end{array}$$

The insertions $\mathsf{inl}_{A,B} : A \to A \oplus B$ and $\mathsf{inr}_{A,B} : B \to A \oplus B$ are defined by

$$\begin{array}{rcl} \mathsf{inl}_{A,B} & \stackrel{\text{def}}{=} & \mathsf{id}_A[(a,a) \mapsto (a, \mathsf{inl}(a))] \\ \mathsf{inr}_{A,B} & \stackrel{\text{def}}{=} & \mathsf{id}_B[(b,b) \mapsto (b, \mathsf{inr}(b))] \end{array}$$

and, for $p : A \to C$, $q : B \to C$, $[p,q] : A \oplus B \to C$ is

$$\begin{array}{rcl} [p,q] & \stackrel{\text{def}}{=} & p[(a,c) \mapsto (\mathsf{inl}(a), c)] \\ & + & q[(b,c) \mapsto (\mathsf{inr}(b), c)]. \end{array}$$

In these definitions, the operation + on processes means non-deterministic sum of labelled transition systems (i.e. the standard CCS +).

Proposition 3.5 *The above definitions make $A \oplus B$ a coproduct of A and B.*

Proof Suppose $p : A \to C$ and $q : B \to C$. We first need to check that $\mathsf{inl} ; [p,q] = p$. The definitions of $[p,q]$ and composition mean that

$$\mathsf{inl} ; [p,q] \xrightarrow{(a,c)} \mathsf{inl} ; p'[(a,c) \mapsto (\mathsf{inl}(a), c)] \iff p \xrightarrow{(a,c)} p'.$$

Since $\mathsf{inl} = \mathsf{id}[(a,a) \mapsto (a, \mathsf{inl}(a))]$, Lemma 3.4 shows that

$$\mathsf{inl} ; p'[(a,c) \mapsto (\mathsf{inl}(a), c)] = \mathsf{id} ; p' = p'.$$

Hence $\mathsf{inl} ; [p,q]$ and p are bisimilar.

Symmetrically, $\mathsf{inr} ; [p,q] = q$.

Now suppose that $h : A \oplus B \to C$ with $\mathsf{inl} ; h = p$ and $\mathsf{inr} ; h = q$. There is a trivial possibility to dispose of: if $h = \mathsf{nil}$ then $p = \mathsf{nil}$ and $q = \mathsf{nil}$, and $[p,q] = \mathsf{nil} = h$.

Otherwise, the type of h means that its first transition has one of two forms: either

$$h \xrightarrow{(\mathsf{inl}(a),c)} h'$$

or

$$h \xrightarrow{(\mathsf{inr}(b),c)} h'.$$

In the first case, because $\mathsf{inl} ; h = p$, we have $p \xrightarrow{(a,c)} p'$ with $\mathsf{inl} ; h' = p'$. The safety specification of $A \oplus B$ means that we can consider h' as a morphism $\mathsf{inl}(A) \to C$, where the object $\mathsf{inl}(A)$ is defined by

$$\begin{array}{rcl} \Sigma_{\mathsf{inl}(A)} & \stackrel{\text{def}}{=} & \{\mathsf{inl}(a) \mid a \in \Sigma_A\} \\ S_{\mathsf{inl}(A)} & \stackrel{\text{def}}{=} & \{\mathsf{inl}^*(s) \mid s \in S_A\}. \end{array}$$

Now we have

$$\begin{aligned} p'[(a,c) \mapsto (\mathsf{inl}(a),c)] &= (\mathsf{inl}\,;h')[(a,c) \mapsto (\mathsf{inl}(a),c)] \\ &= \mathsf{inl}[(a,\mathsf{inl}(a)) \mapsto (\mathsf{inl}(a),\mathsf{inl}(a))]\,;h' \\ &= \mathsf{id}_{\mathsf{inl}(A)}\,;h' \\ &= h'. \end{aligned}$$

Similarly, in the second case we have $q \xrightarrow{(b,c)} q'$ and $h' = q'[(b,c) \mapsto (\mathsf{inr}(b),c)]$.

Finally, note that the first transition of h is the same as that of either $p[(a,c) \mapsto (\mathsf{inl}(a),c)]$ or $q[(b,c) \mapsto (\mathsf{inr}(b),c)]$ as appropriate. Hence h is bisimilar to $p[(a,c) \mapsto (\mathsf{inl}(a),c)] + q[(b,c) \mapsto (\mathsf{inr}(b),c)]$, which is the definition of $[p,q]$. □

Since $\oplus$ is a coproduct, its dual is a product; because all objects of $\mathcal{SP}roc$ are self-dual, this means that $A \oplus B$ is itself also a product of A and B—so, in fact, a biproduct.

We will use the notation $A \,\&\, B$ for the product of A and B, in line with the standard notation for the additive connectives of linear logic [23]. In $\mathcal{SP}roc$, $A \,\&\, B$ is the same object as $A \oplus B$, but we will use the product notation when we want to emphasise the product properties. Exploiting the self-duality of $\mathcal{SP}roc$ objects, we can define projections and pairing as follows.

$$\begin{aligned} \pi_A &\stackrel{\text{def}}{=} \mathsf{inl}^\perp \\ \pi_B &\stackrel{\text{def}}{=} \mathsf{inr}^\perp \\ \langle p,q\rangle &\stackrel{\text{def}}{=} [p,q]^\perp \end{aligned}$$

There is also a zero object $\mathbf{0}$ which has $\Sigma_{\mathbf{0}} \stackrel{\text{def}}{=} \varnothing$ and $S_{\mathbf{0}} \stackrel{\text{def}}{=} \{\varepsilon\}$.

Proposition 3.6 *The object* $\mathbf{0}$ *is initial and terminal in* $\mathcal{SP}roc$.

Proof The only safe trace for $\mathbf{0}$ is the empty trace, so a morphism $A \to \mathbf{0}$ cannot make any transitions and must be nil. Similarly for a morphism $\mathbf{0} \to A$. □

When a category has biproducts and a zero object, it is possible to define a commutative monoid structure on each homset [33]. If $p, q : A \to B$ then $p + q : A \to B$ is defined by

$$\begin{aligned} p+q &\stackrel{\text{def}}{=} A \stackrel{\Delta_A}{\to} A \oplus A \stackrel{[p,q]}{\to} B \\ &= A \stackrel{\langle p,q\rangle}{\to} B \oplus B \stackrel{\nabla_B}{\to} B \end{aligned}$$

where $\Delta_A \stackrel{\text{def}}{=} \langle \mathsf{id}_A, \mathsf{id}_A\rangle$ is the diagonal and $\nabla_B \stackrel{\text{def}}{=} [\mathsf{id}_B, \mathsf{id}_B]$ the codiagonal. The unit is defined by $0_{A\to B} \stackrel{\text{def}}{=} A \to \mathbf{0} \to B$.

In $\mathcal{SP}roc$, this construction yields the non-deterministic sum of CCS (when strong bisimulation is taken as the notion of equivalence). The proof of Proposition 3.6 shows that the unique morphisms into and out of $\mathbf{0}$ are nil processes, and so 0_A is also nil. To unravel the definition of $+$, consider the composition $\langle p,q\rangle\,;\nabla_B$. Pairing creates a union of the behaviours of p and q, but with disjointly labelled copies of B. Composing with ∇_B removes the difference between the two copies. A choice can be made between p and q at the first step, but then the behaviour continues as behaviour of p or behaviour of q.

3.2.1 Time

So far, all of the constructions in $\mathcal{SP}roc$ have been esentially constructions on relations, extended uniformly through time. The next step is to define an operator which allow the temporal structure of the morphisms to be manipulated.

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

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

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

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

Proposition 3.7 *There are isomorphisms*

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

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

Proof $\mathsf{monunit} : I \cong \circ I$ is defined by $\mathsf{monunit} \xrightarrow{(\bullet,*)} \mathsf{id}_I$.

$\mathsf{mon}_{A,B} : (\circ A) \otimes (\circ B) \cong \circ(A \otimes B)$ 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. □

The most important feature of $\circ$ is that it has the following *unique fixed point property* (UFPP) [1].

Proposition 3.8 *For any objects A and B, and any morphisms $f : A \to \circ A$ and $g : \circ B \to B$, there is a unique morphism $It(f,g) : A \to B$ such that*

$$\begin{array}{ccc} A & \xrightarrow{f} & \circ A \\ {\scriptstyle It(f,g)}\downarrow & & \downarrow{\scriptstyle \circ It(f,g)} \\ B & \xleftarrow[g]{} & \circ B \end{array}$$

commutes.

Proof The equational condition that the square commute, namely $It(f,g) = f \,;\, \circ It(f,g) \,;\, g$, can be read as a guarded recursive definition of $It(f,g)$. It is standard in concurrency theory that such a definition has a unique solution [36]. □

We will not go into the applications of this property in the present paper, except to mention that it supports guarded recursive definitions [1, 21] and is an important part of a proposed axiomatisation of interaction categories [21]. The notation $It(f,g)$ is intended to suggest *iteration*.

4 Specification Structures over Interaction Categories

4.1 The Sequence of Definitions

Suppose $\mathcal{C}$ is a $*$-autonomous category with a notion of a set of *processes* of each type, written $\mathsf{Proc}(A)$ (a process P of type A may be identified with a morphism $P : I \to A$). The

following sequence of steps provides a convenient way to define a specification structure S. This sequence will be used in the present paper when defining specification structures over $\mathcal{SP}roc$; it mirrors the sequence already used in the definition of $\mathcal{SP}roc$ itself.

1. Define $P_S A$ for each A.
2. For each A, define a relation of *satisfaction*: $\models_A \subseteq \mathsf{Proc}(A) \times P_S A$.
3. Define $(-)^\perp_A$.
4. Define $\otimes_{A,B}$ and hence $\parr_{A,B}$ and $\multimap_{A,B}$.
5. Define the Hoare triple relation by $\theta\{f\}\varphi \stackrel{\text{def}}{\Leftrightarrow} f \models_{A\multimap B} \theta \multimap \varphi$.
6. Verify that the desired structure of $\mathcal{C}$, including the $*$-autonomous structure, lifts to $\mathcal{C}_S$.

For reference, we will now list the definitions and conditions which are needed to lift the relevant structure of $\mathcal{C}$ to $\mathcal{C}_S$. In the present paper, we are interested in the $*$-autonomous structure, products and coproducts, the unit delay functor, and the UFPP.

4.2 $*$-Autonomous Structure

For every pair A, B of objects we need an operation

$$\otimes_{A,B} : P_S A \times P_S B \to P_S(A \otimes B).$$

When writing this and similar operations, we will usually omit the subscripts. To define the functor $\otimes : \mathcal{C}_S \times \mathcal{C}_S \to \mathcal{C}_S$ we need, for every $\theta, \varphi, \psi, \rho \in P_S A, P_S B, P_S C, P_S D$, $f : A \to C$ and $g : B \to D$,

$$\theta\{f\}\psi, \varphi\{g\}\rho \quad \Rightarrow \quad (\theta \otimes \varphi)\{f \otimes g\}(\psi \otimes \rho).$$

Then

$$(A, \theta) \otimes (B, \varphi) \quad \stackrel{\text{def}}{=} \quad (A \otimes B, \theta \otimes_{A,B} \varphi).$$

We need $I_S \in P_S I$ in order to define the tensor unit in $\mathcal{C}_S$ by (I, I_S).

To lift the symmetric monoidal structure of $\mathcal{C}$ to $\mathcal{C}_S$ we need the following conditions, for every $\theta, \varphi, \psi \in P_S A, P_S B, P_S C$.

$$(\theta \otimes (\varphi \otimes \psi))\{\mathsf{assoc}_{A,B,C}\}((\theta \otimes \varphi) \otimes \psi)$$
$$(I_S \otimes \theta)\{\mathsf{unitl}_A\}\theta$$
$$(\theta \otimes I_S)\{\mathsf{unitr}_A\}\theta$$
$$(\theta \otimes \varphi)\{\mathsf{symm}_{A,B}\}(\varphi \otimes \theta)$$

For each object A we need an operation

$$(-)^\perp_A : P_S A \to P_S(A^\perp)$$

and we can then define

$$(A, \theta)^\perp \quad \stackrel{\text{def}}{=} \quad (A^\perp, \theta^\perp_A).$$

In order to define the functorial action of $(-)^\perp$ on $\mathcal{C}_S$ we need, for every $\theta, \varphi \in P_S A, P_S B$ and $f : A \to B$,

$$\theta\{f\}\varphi \quad \Rightarrow \quad \varphi^\perp_B\{f^\perp\}\theta^\perp_A.$$

The operations

$$\begin{aligned} \parr_{A,B} &: P_S A \times P_S B \to P_S(A \parr B) \\ \multimap_{A,B} &: P_S A \times P_S B \to P_S(A \multimap B) \end{aligned}$$

can be defined by

$$\begin{aligned} \theta \parr_{A,B} \varphi &\stackrel{\text{def}}{=} (\theta_A^{\perp} \otimes_{A^{\perp},B^{\perp}} \varphi_B^{\perp})^{\perp}_{A^{\perp} \otimes B^{\perp}} \\ \theta \multimap_{A,B} \varphi &\stackrel{\text{def}}{=} (\theta \otimes_{A,B^{\perp}} \varphi_B^{\perp})^{\perp}_{A \otimes B^{\perp}} \end{aligned}$$

and the property $\perp_S$ by

$$\perp_S \stackrel{\text{def}}{=} I_S^{\perp}.$$

To lift the closed structure we need, for every $\theta, \varphi, \psi \in P_S A, P_S B, P_S C$ and $f : A \otimes B \to C$,

$$(\theta \otimes \varphi)\{f\}\psi \quad \Rightarrow \quad \theta\{\Lambda(f)\}(\varphi \multimap \psi)$$

and

$$((\theta \multimap \varphi) \otimes \theta)\{\mathsf{Ap}_{A,B}\}\varphi.$$

4.3 Products and Coproducts

For every pair A, B of objects we need an operation

$$\&_{A,B} : P_S A \times P_S B \to P_S(A \,\&\, B).$$

This enables the product in $\mathcal{C}_S$ to be defined on objects by

$$(A, \theta) \,\&\, (B, \varphi) \stackrel{\text{def}}{=} (A \,\&\, B, \theta \,\&_{A,B}\, \varphi).$$

For functoriality, it must be the case that for every $\theta, \varphi, \psi, \rho \in P_S A, P_S B, P_S C, P_S D$, $f : A \to C$ and $g : B \to D$,

$$\theta\{f\}\psi, \varphi\{g\}\rho \quad \Rightarrow \quad (\theta \,\&\, \varphi)\{f \,\&\, g\}(\psi \,\&\, \rho).$$

Additionally, we need

$$\begin{aligned} &(\theta \,\&\, \varphi)\{\pi_A\}\theta \\ &(\theta \,\&\, \varphi)\{\pi_B\}\varphi \end{aligned}$$

and, for $f : C \to A$, $g : C \to B$,

$$\psi\{f\}\theta, \psi\{g\}\varphi \quad \Rightarrow \quad \psi\{\langle f, g\rangle\}(\theta \,\&\, \varphi).$$

The treatment of coproducts is dual.

4.4 Unit Delay

For each object A we need an operation

$$\circ_A : P_S A \to P_S(\circ\, A)$$

in order to define the unit delay on $\mathcal{C}_S$ by

$$\circ(A, \theta) \stackrel{\text{def}}{=} (\circ\, A, \circ_A \theta).$$

For functoriality we need, for each $f : A \to B$ and $\theta, \varphi \in P_S A, P_S B$,

$$\theta\{f\}\varphi \quad \Rightarrow \quad \circ\,\theta\{\circ\, f\}\circ\varphi.$$

Lifting the UFPP to $\mathcal{C}_S$ requires that if $f : A \to \circ\, A$ and $g : \circ\, B \to B$ with $\theta, \varphi \in P_S A, P_S B$,

$$\theta\{f\}\circ\theta, \circ\,\varphi\{g\}\varphi \quad \Rightarrow \quad \theta\{It(f, g)\}\varphi.$$

5 A Specification Structure for Deadlock-Freedom

Throughout this paper, deadlock means termination. A more refined treatment might consider *unsuccessful* termination as deadlock; the view taken here is that all termination is unsuccesful. In fact, given the synchronous nature of $\mathcal{SP}roc$, successful termination is not especially interesting because all processes would have to terminate simultaneously. A process may have both terminating and non-terminating behaviours, but a *deadlock-free* process is one which has no maximal finite behaviours. For example, the process $a.b.\mathsf{nil}$ can deadlock; the process P defined by $P = a.P + b.\mathsf{nil}$ can deadlock although it can also generate the infinite trace a^ω; the process Q defined by $Q = a.b.Q$ is deadlock-free.

Deadlock-freedom is not preserved by composition: two processes may individually be deadlock-free, but when forced to communicate they could deadlock each other by being unable to agree on a sequence of actions to perform. For example, if the CCS processes P and Q are defined by $P = a.b.P$ and $Q = \bar{a}.c.Q$ then composing them means forming the process $(P \mid Q) \setminus \{a, b, c\}$. In this process, P and Q can communicate for a single step, but then deadlock occurs because P must do b next while Q can only do c.

In order to construct a category of deadlock-free processes which are guaranteed to remain deadlock-free when composed with each other, more information is needed than just the fact that a process runs forever. The rest of this section describes one approach to building suitable extra information into the types, by constructing a specification structure over $\mathcal{SP}roc$. In Section 6 we define an alternative specification structure, and prove that the two are equivalent.

The first construction of a specification structure for deadlock-freedom takes a property over a type to be a set of processes of that type. This is clearly the most general notion of property, and has no inherent connection with deadlocks. For these properties to say anything about deadlock-freedom, the sets of processes must be carefully chosen in a way which will now be described.

5.1 Fundamentals

Definition 5.1 *A process P of type A* converges, *written $P\downarrow$, if whenever $P \xrightarrow{s}{}^{*} Q$ there is $a \in \Sigma_A$ and a process R such that $Q \xrightarrow{a} R$.*

Convergence means deadlock-freedom; the reason for the choice of terminology is an analogy with proofs of strong normalisation in Classical Linear Logic [23, 4].

Definition 5.2 *Given processes P and Q of type A, the process $P \sqcap Q$ of type A is defined by*

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

For each type A, the orthogonality *relation on the set of processes of type A is defined by*

$$P \perp Q \quad \stackrel{\text{def}}{\Leftrightarrow} \quad (P \sqcap Q)\downarrow .$$

If P and Q are orthogonal, then they represent possible projections, in the shared type, of two successfully communicating processes. For example, if $R : A \to B$ and $S : B \to C$ then the behaviours of R and S in their ports of type B can be described as processes P and Q of type B; orthogonality of P and Q corresponds to non-deadlocking communication between R and S when they are combined into $R\,;S$.

Because only deadlock-free processes are of interest in this section, it is convenient to restrict attention to those types of $\mathcal{SP}roc$ whose safety specifications do not force termination. Such types are called *progressive*.

Definition 5.3 *An object A of $\mathcal{SP}roc$ is progressive if*

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

The full subcategory of $\mathcal{SP}roc$ consisting of just the progressive objects is denoted by $\mathcal{SP}roc_{pr}$.

$\mathcal{SP}roc_{pr}$ inherits all the structure of $\mathcal{SP}roc$, apart from the zero object. The specification structure for deadlock-freedom will be defined over $\mathcal{SP}roc_{pr}$.

Definition 5.4 *For each object A of $\mathcal{SP}roc_{pr}$, $\mathsf{Proc}(A)$ is the set of convergent processes of type A. The orthogonality relation is extended to sets of processes by defining, for $U, V \subseteq \mathsf{Proc}(A)$ and $P : A$,*

$$\begin{aligned} P \perp U &\stackrel{\text{def}}{\Leftrightarrow} \forall Q \in U.P \perp Q \\ U \perp V &\stackrel{\text{def}}{\Leftrightarrow} \forall P \in U.P \perp V. \end{aligned}$$

Orthogonality then generates an operation of negation on sets of processes, defined by

$$U^{\perp} \stackrel{\text{def}}{=} \{P \in \mathsf{Proc}(A) \mid P \perp U\}.$$

Defining $(-)^{\perp}$ in this way from a symmetric orthogonality relation yields a self-adjoint Galois connection [17] and the following lemma holds for general reasons.

Lemma 5.5 *For all $U, V \subseteq \mathsf{Proc}(A)$,*

$$\begin{array}{lrcl} 1. & U \subseteq V & \Rightarrow & V^{\perp} \subseteq U^{\perp} \\ 2. & U & \subseteq & U^{\perp\perp} \\ 3. & U^{\perp} & = & U^{\perp\perp\perp}. \end{array}$$

Proof

1. If $P \in V^{\perp}$ then $\forall Q \in V.P \perp Q$. Hence $\forall Q \in U.P \perp Q$. Hence $P \in U^{\perp}$.
2. If $P \in U$ and $Q \in U^{\perp}$ then $P \perp Q$. Hence $\forall Q \in U^{\perp}.P \perp Q$. Hence $P \in U^{\perp\perp}$.
3. From 2 we have $U^{\perp} \subseteq U^{\perp\perp\perp}$. Also, applying 1 to 2 gives $U^{\perp\perp\perp} \subseteq U^{\perp}$. □

We will also need some additional results.

Definition 5.6 *For any object A of $\mathcal{SP}roc_{pr}$, the process $\mathsf{max}_A : A$ is defined by*

$$\frac{a \in S_A}{\mathsf{max}_A \xrightarrow{a} \mathsf{max}_{A/a}}.$$

Lemma 5.7 *For any $P \in \mathsf{Proc}(A)$, $P \perp \mathsf{max}_A$.*

Proof $P \sqcap \mathsf{max}_A = P$, so $P \perp \mathsf{max}_A$ because $P\!\downarrow$. □

Lemma 5.8 *If $U \subseteq \mathsf{Proc}(A)$ then $U^{\perp} \neq \varnothing$.*

Proof Lemma 5.7 implies that $\mathsf{max}_A \in U^{\perp}$. □

5.2 The Specification Structure

Following the sequence of definitions listed in Section 4, we can now define the specification structure D.

Definition 5.9

1. $P_D A \stackrel{\text{def}}{=} \{U \subseteq \mathsf{Proc}(A) \mid (U \neq \varnothing) \wedge (U^{\perp\perp} = U)\}$.
2. $P \models U \stackrel{\text{def}}{\Leftrightarrow} P \in U$.
3. $(-)^{\perp} : P_D A \to P_D A$ *has already been defined.*
4. $$\begin{array}{rcl} U \otimes V & \stackrel{\text{def}}{=} & \{P \otimes Q \mid P \in U, Q \in V\}^{\perp\perp} \\ U \parr V & \stackrel{\text{def}}{=} & (U^{\perp} \otimes V^{\perp})^{\perp} \\ U \multimap V & \stackrel{\text{def}}{=} & (U \otimes V^{\perp})^{\perp} \\ I_D & \stackrel{\text{def}}{=} & \{\mathsf{max}_I\}. \end{array}$$
5. $U\{f\}V \stackrel{\text{def}}{\Leftrightarrow} f \models U \multimap V$.
6. *The definitions required to lift the additive and temporal structure of $\mathcal{SP}roc$ to $\mathcal{SP}roc_D$ are*
$$\begin{array}{rcl} U \oplus V & \stackrel{\text{def}}{=} & (\{P[\mathsf{inl}] \mid P \in U\} \cup \{Q[\mathsf{inr}] \mid Q \in V\})^{\perp\perp} \\ U \mathbin{\&} V & \stackrel{\text{def}}{=} & (U^{\perp} \oplus V^{\perp})^{\perp} \\ \circ U & \stackrel{\text{def}}{=} & \{\circ P \mid P \in U\}. \end{array}$$

The following points are worth noting.

- Lemma 5.8 ensures that clause 3 of the definition makes sense, by guaranteeing that $U^{\perp} \in P_D A$ for each $U \in P_D A$.
- For any $U \subseteq \mathsf{Proc}(A)$, $U^{\perp\perp}$ is the smallest $^{\perp\perp}$-invariant set of processes containing U.
- There are now separate definitions relating to product ($\&$) and coproduct ($\oplus$). We will prove later that these connectives are distinct in the specification structure D.

We need to check that every set of processes defined in Definition 5.9 is $^{\perp\perp}$-invariant. In every case except that of $\circ$, this follows from the fact (Lemma 5.5) that for every $U \subseteq \mathsf{Proc}(A)$, $U^{\perp}$ is $^{\perp\perp}$-invariant.

Lemma 5.10 *If $U \in P_D A$ then $\circ U \in P_D(\circ A)$.*

Proof First, $\circ U \neq \varnothing$ because $U \neq \varnothing$.

Next, note that for $P, Q \in \mathsf{Proc}(\circ A)$, there are $P', Q' \in \mathsf{Proc}(A)$ such that $P = \circ P'$ and $Q = \circ Q'$. Furthermore, $P \perp Q \iff P' \perp Q'$.

This means that

$$\begin{array}{rcl} (\circ U)^{\perp} & = & \{\circ P \mid P \in U\}^{\perp} \\ & = & \{\circ Q \mid Q \in U^{\perp}\} \\ & = & \circ(U^{\perp}) \end{array}$$

and hence $(\circ U)^{\perp\perp} = \circ(U^{\perp\perp}) = \circ U$. □

We can now prove that D satisfies the specification structure axioms.

Lemma 5.11 *If $U \subseteq V \subseteq \mathsf{Proc}(A)$ then $\mathsf{id}_A \in U \multimap V$.*

Proof We need $\mathsf{id}_A \in (U \otimes V^\perp)^\perp$. Now,

$$\begin{aligned}(U \otimes V^\perp)^\perp &= \{P \otimes Q \mid P \in U, Q \in V^\perp\}^{\perp\perp\perp} \\ &= \{P \otimes Q \mid P \in U, Q \in V^\perp\}^\perp\end{aligned}$$

so it is enough to show $\mathsf{id}_A \perp \{P \otimes Q \mid P \in U, Q \in V^\perp\}$. Let $P \in U$ and $Q \in V^\perp$. $U \subseteq V$ implies $V^\perp \subseteq U^\perp$, so $Q \in U^\perp$ and hence $P \perp Q$. For any common trace s of id_A and $P \otimes Q$, $\mathsf{fst}^*(s)$ is a trace of P and $\mathsf{snd}^*(s)$ is a trace of Q, and $\mathsf{fst}^*(s) = \mathsf{snd}^*(s)$. So there is an action a such that $\mathsf{fst}^*(s)a$ is a trace of P and $\mathsf{snd}^*(s)a$ is a trace of Q. Hence (a, a) is an action such that $s(a, a)$ is a trace of both id_A and $P \otimes Q$. This means that $(\mathsf{id}_A \sqcap (P \otimes Q))\!\downarrow$, and so $\mathsf{id}_A \perp P \otimes Q$. □

For the next two lemmas, a slight abuse of notation is useful. If $f : A \to B$ and $P : A$, there is a process $P\,;f$ of type B obtained by regarding P as a morphism $I \to A$, composing with f, and then regarding the resulting morphism $I \to B$ as a process of type B. Similarly, if $Q : B^\perp$ there is a process $f\,;Q$ of type A.

Lemma 5.12 *If $f : A \to B$, $U \in P_D A$, $V \in P_D B$, $f \in U \multimap V$ and $P \in U$, then $P\,;f \in V$.*

Proof To show that $P\,;f \in V$, we will consider an arbitrary $Q \in V^\perp$ and show that $P\,;f \perp Q$, thus establishing $P\,;f \in V^{\perp\perp} = V$.

Let $Q \in V^\perp$ and let s be a common trace of $P\,;f$ and Q. The definition of composition means that there is a trace t of f such that $\mathsf{fst}^*(t)$ is a trace of P and $\mathsf{snd}^*(t) = s$. We have $f \in U \multimap V$, and so $f \perp (P \otimes Q)$. Hence there is an action (a, b) such that $t(a, b)$ is a trace of f, $\mathsf{fst}^*(t)a$ is a trace of P and $\mathsf{snd}^*(t)b$ is a trace of Q. Then sb is a common trace of $P\,;f$ and Q, so $((P\,;f) \sqcap Q)\!\downarrow$ as required. □

Lemma 5.13 *If $f : A \to B$, $U \in P_D A$, $V \in P_D B$, $f \in U \multimap V$ and $Q \in V^\perp$, then $f\,;Q \in U^\perp$.*

Proof Similar to the proof of Lemma 5.12. □

Notation When s and t are traces of equal length, we will write $s \;\mathsf{zip}\; t$ for the unique trace u such that $\mathsf{fst}^*(u) = s$ and $\mathsf{snd}^*(u) = t$.

Proposition 5.14 *D is a specification structure over $\mathcal{SP}roc_{pr}$.*

Proof The first requirement is that if A is any object of $\mathcal{SP}roc_{pr}$ and $U \in P_D A$, $U\{\mathsf{id}_A\}U$. This follows from Lemma 5.11.

Next, suppose that A, B, C are objects of $\mathcal{SP}roc_{pr}$ and $U \in P_D A, V \in P_D B$ and $W \in P_D C$. If $f : A \to B$ and $g : B \to C$ with $U\{f\}V$ and $V\{g\}W$, we need $U\{f\,;g\}W$. Thus the goal is to prove that $f\,;g \perp \{P \otimes R \mid P \in U, R \in W^\perp\}$.

Take $P \in U$ and $Q \in W^\perp$. We need to prove that $(f\,;g) \perp (P \otimes Q)$. Any common trace of $f\,;g$ and $P \otimes Q$ arises from traces s and t such that $f \xrightarrow{s}{}^* f'$, $g \xrightarrow{t}{}^* g'$, $P \xrightarrow{\mathsf{fst}^*(s)}{}^* P'$, $Q \xrightarrow{\mathsf{snd}^*(t)}{}^* Q'$ and $\mathsf{snd}^*(s) = \mathsf{fst}^*(t)$. Then we have

$$f\,;g \xrightarrow{\mathsf{fst}^*(s)\;\mathsf{zip}\;\mathsf{snd}^*(t)} f'\,;g' \qquad P \otimes Q \xrightarrow{\mathsf{fst}^*(s)\;\mathsf{zip}\;\mathsf{snd}^*(t)} P' \otimes Q'.$$

This gives $P\,;f \xrightarrow{\mathsf{snd}^*(s)}{}^* P'\,;f'$ and $g\,;Q \xrightarrow{\mathsf{fst}^*(t)}{}^* g'\,;Q'$. By Lemmas 5.12 and 5.13, $P\,;f \in V$ and $g\,;R \in V^\perp$. Hence $(P\,;f) \perp (g\,;R)$, so there is b such that $P'\,;f' \xrightarrow{b} P''\,;f''$ and $g'\,;Q' \xrightarrow{b} g''\,;Q''$. By the definition of composition, there are a and c such that $P' \xrightarrow{a} P''$, $f' \xrightarrow{(a,b)} f''$, $g' \xrightarrow{(b,c)} g''$ and $Q' \xrightarrow{c} Q''$. Hence $f'\,;g' \xrightarrow{(a,c)} f''\,;g''$ and $P' \otimes Q' \xrightarrow{(a,c)} P'' \otimes Q''$, as required. □

It is now legitimate to talk about the category $\mathcal{SP}roc_D$ of deadlock-free processes. In order to prove that the $*$-autonomous structure of $\mathcal{SP}roc$ lifts to $\mathcal{SP}roc_D$, we need to check the various conditions listed in Section 4. As an example of the style of proof required, we will verify one case.

Proposition 5.15 *If $U \in P_D A$ and $V \in P_D B$, then $(U \otimes V)\{\mathsf{symm}_{A,B}\}(V \otimes U)$.*

Proof We need $\mathsf{symm} \in (U \otimes V) \multimap (V \otimes U)$, i.e. $\mathsf{symm} \in ((U \otimes V) \otimes (V \otimes U)^\perp)^\perp$, or equivalently $\mathsf{symm} \perp \{P \otimes Q \mid P \in U \otimes V, Q \in (V \otimes U)^\perp\}$.

First, suppose that $Q \in (V \otimes U)^\perp = \{R \otimes S \mid R \in V, S \in U\}^{\perp\perp\perp}$, i.e.

$$Q \perp \{R \otimes S \mid R \in V, S \in U\}.$$

Defining $Q' \stackrel{\text{def}}{=} Q[(b,a) \mapsto (a,b)]$, it is clear that $Q' \perp \{S \otimes R \mid S \in U, R \in V\}$ and so $Q' \in (U \otimes V)^\perp$.

Now suppose that $P \in U \otimes V$ and $Q \in (V \otimes U)^\perp$, and s is a common trace of symm and $P \otimes Q$. The definition of symm means that $\mathsf{fst}^*(s) = \mathsf{snd}^*(s)[(b,a) \mapsto (a,b)]$. Also, $\mathsf{fst}^*(s)$ is a trace of P and $\mathsf{snd}^*(s)[(b,a) \mapsto (a,b)]$ is a trace of Q'. Because $P \perp Q'$, there is an action (a,b) available to both P and Q' after this trace. So Q can do (b,a) after the corresponding trace, and $P \otimes Q$ can do $((a,b),(b,a))$ after s. This action is also available to symm after s. Hence $\mathsf{symm} \perp P \otimes Q$, as required. □

We also need to check that the products and coproducts lift to $\mathcal{SP}roc_D$. The necessary proofs will be postponed until Section 6, as they turn out to be more easily formulated in the language of ready specifications.

We will, however, prove that the UFPP lifts to $\mathcal{SP}roc_D$.

Proposition 5.16 *Let A, B be objects of $\mathcal{SP}roc_{pr}$, $U \in P_D A$ and $V \in P_D B$. Let $f : (A,U) \to (\circ A, \circ U)$ and $g : (\circ B, \circ V) \to (B,V)$, and let $h : A \to B$ be the unique morphism in $\mathcal{SP}roc$ satisfying $h = f\,;\circ h\,;g$. Then $h : (A,U) \to (B,V)$ in $\mathcal{SP}roc_D$.*

Proof We need to prove that $h \models U \multimap V$, i.e. that $h \perp \{P \otimes Q \mid P \in U, Q \in V^\perp\}$. We will prove, by induction on the length of s, that

$$\forall s.\forall P \in U.\forall Q \in V^\perp. \left(\begin{array}{ll} & (P \otimes Q) \sqcap h \xrightarrow{s}{}^* (P' \otimes Q') \sqcap h' \\ \Rightarrow & \exists (a,b).(P' \otimes Q') \sqcap h' \xrightarrow{(a,b)} (P'' \otimes Q'') \sqcap h'' \end{array} \right)$$

(Base case) $s = \epsilon$. Take $P \in U$, $Q \in V^\perp$, $R \in \circ(U^\perp)$, $S \in \circ V$. Because $f \perp P \otimes R$ there is a such that $f \xrightarrow{(a,*)} f'$ and $P \xrightarrow{a} P'$. Because $g \perp S \otimes Q$ there is b such that $g \xrightarrow{(*,b)} g'$ and $Q \xrightarrow{b} Q'$. The transitions of f and g give $h \xrightarrow{(a,b)} h'$, and we also have $P \otimes Q \xrightarrow{(a,b)} P' \otimes Q'$.

(Inductive step) Assuming the result for traces of length n, consider traces of length $n+1$. Take $P \in U$ and $Q \in V^\perp$. Because $f \models U \multimap \circ U$ and $g \models \circ V \multimap V$, Lemmas 5.12 and 5.13 give $P\,;f \in \circ U$ and $g\,;Q \in \circ V$. Suppose $h \sqcap (P \otimes Q) \xrightarrow{s\ \mathsf{zip}\ t}{}^* h' \sqcap (P' \sqcap Q')$. This means

that there are traces u and v with $f \xrightarrow{s \,\mathsf{zip}\, (*u)}{}^* f'$, $\circ h \xrightarrow{(*u)\,\mathsf{zip}\,(*v)}{}^* k$ and $g \xrightarrow{(*v)\,\mathsf{zip}\,t}{}^* g'$. Also $P \xrightarrow{s}{}^* P'$ and $Q \xrightarrow{t}{}^* Q'$. Writing $P\,;f = \circ\, R$ and $g\,;Q = \circ\, S$, we have $R \xrightarrow{u}{}^* R' = P'\,;f'$ and $S \xrightarrow{v}{}^* S' = g'\,;Q'$.

Now we have $h \sqcap (R \otimes S) \xrightarrow{u\,\mathsf{zip}\,v}{}^* k \sqcap (R' \otimes S')$ with $R \in U$, $S \in V^\perp$ and $\mathsf{length}((){u}\ \mathsf{zip}\ v) = n$. By the induction hypothesis, there is (c,d) such that $k \sqcap (R \otimes S) \xrightarrow{(a,b)}{}^* k' \sqcap (R' \otimes S')$. So there is (a,b) such that $P' \xrightarrow{a} P''$, $f' \xrightarrow{(a,c)} f''$, $Q' \xrightarrow{b} Q''$ and $g' \xrightarrow{(d,b)} g''$. Hence $h' \xrightarrow{(a,b)} f''\,;k'\,;g''$ and $P' \otimes Q' \xrightarrow{(a,b)} P'' \otimes Q''$. □

For later work it is useful to have a supply of properties over each type.

Proposition 5.17 *For every object A of $\mathcal{SP}roc_{pr}$, $\{\mathsf{max}_A\}^\perp = \mathsf{Proc}(A)$ and $\mathsf{Proc}(A)^\perp = \{\mathsf{max}_A\}$.*

Proof For any $P \in \mathsf{Proc}(A)$, $P \perp \mathsf{max}_A$. Hence $\mathsf{Proc}(A) \perp \{\mathsf{max}_A\}$. This means that $\{\mathsf{max}_A\}^\perp \supseteq \mathsf{Proc}(A)$; also, $\{\mathsf{max}_A\}^\perp \subseteq \mathsf{Proc}(A)$. This gives $\{\mathsf{max}_A\}^\perp = \mathsf{Proc}(A)$.

For the second part, we already have $\{\mathsf{max}_A\} \subseteq \mathsf{Proc}(A)^\perp$. Now suppose that $P \neq \mathsf{max}_A$. There is a process P', a trace s and an action $a \in \Sigma_A$ such that $sa \in S_A$ and $P \xrightarrow{s}{}^* P'$ but P' cannot do a. Define Q to be the same process as P, except that the node P' is replaced by Q' where $Q' = a : \mathsf{max}_{A/sa}$. Then $P \sqcap Q$ does not converge, so $P \not\perp \mathsf{Proc}(A)$. □

Corollary 5.18 $\{\mathsf{max}_A\}^{\perp\perp} = \{\mathsf{max}_A\}$ *and* $\mathsf{Proc}(A)^{\perp\perp} = \mathsf{Proc}(A)$.

Definition 5.19 *For each object A of $\mathcal{SP}roc_{pr}$, define two properties over A: $\mathsf{in}_A \stackrel{\text{def}}{=} \{\mathsf{max}_A\}$ and $\mathsf{out}_A \stackrel{\text{def}}{=} \mathsf{Proc}(A)$. Thus we have $\mathsf{in}_A^\perp = \mathsf{out}_A$ and $\mathsf{out}_A^\perp = \mathsf{in}_A$.*

A port of type (A, in_A) represents an input because the possible behaviour is described by max_A which is always prepared to engage in any action. A port of type (A, out_A) represents a possibly non-deterministic output; no information is available about its possible behaviour.

Proposition 5.20 *For all objects A,*

$$(s \in S_A \Rightarrow \exists! a \in \Sigma_A . sa \in S_A) \iff (\mathsf{in}_A = \mathsf{out}_A).$$

Corollary 5.21 $I_D = \mathsf{in}_I = \mathsf{out}_I = I_D^\perp$.

There are a few useful results on combinations of in and out properties.

Proposition 5.22 *For any objects A and B of $\mathcal{SP}roc_{pr}$,*

1. $\mathsf{in}_A \otimes \mathsf{in}_B = \mathsf{in}_{A \otimes B}$
2. $\mathsf{out}_A \parr \mathsf{out}_B = \mathsf{out}_{A \parr B}$
3. $\mathsf{out}_A \otimes \mathsf{out}_B = \mathsf{out}_{A \otimes B}$
4. $\mathsf{in}_A \parr \mathsf{in}_B = \mathsf{in}_{A \parr B}$

Proof

1. Follows from the fact that $\mathsf{max}_A \otimes \mathsf{max}_B = \mathsf{max}_{A \otimes B}$.
2. Follows from 1 by duality.

3. Since $M_A \otimes M_B = \{P \otimes Q \mid P \in M_A, Q \in M_B\}^{\perp\perp}$, it is enough to prove

$$\{P \otimes Q \mid P \in M_A, Q \in M_B\}^{\perp} = \{\mathsf{max}_{A \otimes B}\}.$$

Clearly

$$\mathsf{max}_{A \otimes B} \perp \{P \otimes Q \mid P \in M_A, Q \in M_B\}.$$

Suppose that $R \in \mathsf{Proc}(A \otimes B)$ and $R \neq \mathsf{max}_{A \otimes B}$. At some point in the tree of R, there is an action (a, b) which is unavailable. For simplicity, say that R cannot do (a, b). Then if $P = a : \mathsf{max}_{A/a}$ and $Q = b : \mathsf{max}_{B/b}$, $(P \otimes Q) \not\perp R$.

4. Follows from 3 by duality. □

Proposition 5.23 *If $P : A_1 \parr \cdots \parr A_n$ in $\mathcal{SP}roc$, the A_i are progressive and $P\downarrow$, then $P : (A_1, \mathsf{out}_{A_1}) \parr \cdots \parr (A_n, \mathsf{out}_{A_n})$ in $\mathcal{SP}roc_D$.*

Proof It is immediate that if $P : A$ in $\mathcal{SP}roc$, A is progressive and P is deadlock-free, then $P : (A, \mathsf{out}_A)$ in $\mathcal{SP}roc_D$. By Proposition 5.22, the result can be obtained by applying this observation to the type $A_1 \parr \cdots \parr A_n$. □

5.3 Loss of Degeneracy

The degeneracies present in $\mathcal{SP}roc$ (coincidence of $\otimes$ and $\parr$, coincidence of $\&$ and $\oplus$) do not appear in $\mathcal{SP}roc_D$.

Proposition 5.24 *Define $\mathcal{SP}roc$ objects A and B by $\Sigma_A = \{a, b\}$, $S_A = \Sigma_A^*$, $\Sigma_B = \{c, d\}$, $S_B = \Sigma_B^*$. Then $\mathsf{out}_A \otimes \mathsf{in}_B \neq \mathsf{out}_A \parr \mathsf{in}_B$.*

Proof We have

$$\begin{aligned} \mathsf{out}_A \otimes \mathsf{in}_B &= \{P \otimes \mathsf{max}_B \mid P \in \mathsf{Proc}(A)\}^{\perp\perp} \\ \mathsf{out}_A \parr \mathsf{in}_B &= (\mathsf{in}_A \otimes \mathsf{out}_B)^{\perp} \\ &= \{\mathsf{max}_A \otimes Q \mid Q \in \mathsf{Proc}(B)\}^{\perp}. \end{aligned}$$

Defining processes X and Y of type $A \otimes B$ by

$$\begin{aligned} X &= (b, c) : X + (a, d) : X \\ Y &= (a, c) : Y + (b, d) : Y \end{aligned}$$

it is easy to see

$$\begin{aligned} X &\in \{P \otimes \mathsf{max}_B \mid P \in \mathsf{Proc}(A)\}^{\perp} \\ Y &\in \{\mathsf{max}_A \otimes Q \mid Q \in \mathsf{Proc}(B)\}^{\perp}. \end{aligned}$$

But $X \not\perp Y$, which means that $X \notin \{P \otimes \mathsf{max}_B \mid P \in \mathsf{Proc}(A)\}^{\perp\perp}$. □

Loss of compact closure is to be expected, as in general the arbitrary formation of cycles can lead to deadlock. Later in the paper we will present ways of constructing cyclic processes in particular cases.

Proposition 5.25 *Define $\mathcal{SP}roc$ objects A and B by $\Sigma_A = \{a\}$, $S_A = \Sigma_A^*$, $\Sigma_B = \{b\}$, $S_B = \Sigma_B^*$. Then $\mathsf{out}_A \oplus \mathsf{out}_B \neq \mathsf{out}_A \,\&\, \mathsf{out}_B$.*

Proof We have

$$\begin{aligned}
\mathsf{out}_A \oplus \mathsf{out}_B &= (\{\mathsf{max}_A\} \cup \{\mathsf{max}_B\})^{\perp\perp} \\
&= \{\mathsf{max}_A, \mathsf{max}_B\}^{\perp\perp} \\
\mathsf{out}_A \,\&\, \mathsf{out}_B &= (\mathsf{out}_A \oplus \mathsf{out}_B)^{\perp} \\
&= \{\mathsf{max}_A, \mathsf{max}_B\}^{\perp},
\end{aligned}$$

omitting inl and inr for clarity. Now, $\{\mathsf{max}_A, \mathsf{max}_B\}^{\perp} = \{\mathsf{max}_A + \mathsf{max}_B\}$, but

$$\{\mathsf{max}_A + \mathsf{max}_B\}^{\perp} \supseteq \{\mathsf{max}_A + \mathsf{max}_B, \mathsf{max}_A, \mathsf{max}_B\}$$

and so $\mathsf{out}_A \oplus \mathsf{out}_B$ is strictly larger than $\mathsf{out}_A \,\&\, \mathsf{out}_B$. □

Although $\otimes$ and $\parr$ are distinct in $\mathcal{SP}roc_D$, the Mix rule is still valid.

Proposition 5.26 *For any objects (A,U), (B,V) of $\mathcal{SP}roc_D$, we have $\mathsf{id}_{A\otimes B} : (A,U) \otimes (B,V) \multimap (A,U) \parr (B,V)$.*

Proof By Lemma 5.11 it is enough to show that $U \otimes V \subseteq U \parr V$, i.e.

$$\{P \otimes Q \mid P \in U, Q \in V\}^{\perp\perp} \quad \subseteq \quad \{R \otimes S \mid R \in U^{\perp}, S \in V^{\perp}\}^{\perp}.$$

This follows from

$$\{R \otimes S \mid R \in U^{\perp}, S \in V^{\perp}\} \quad \subseteq \quad \{P \otimes Q \mid P \in U, Q \in V\}^{\perp}$$

which in turn follows from

$$\{R \otimes S \mid R \in U^{\perp}, S \in V^{\perp}\} \quad \perp \quad \{P \otimes Q \mid P \in U, Q \in V\}.$$

Take $P \in U$, $Q \in V$, $R \in U^{\perp}$, $S \in V^{\perp}$. If $(P \otimes Q) \sqcap (R \otimes S) \xrightarrow{s}{}^{*} (P' \otimes Q') \sqcap (R' \otimes S')$ then $P \sqcap R \xrightarrow{\mathsf{fst}^*(s)}{}^{*} P' \sqcap R'$ and $Q \sqcap S \xrightarrow{\mathsf{snd}^*(s)}{}^{*} Q' \sqcap S'$. Because $P \perp R$ and $Q \perp S$ there are a, b such that $P' \xrightarrow{a} P''$, $R' \xrightarrow{a} R''$, $Q' \xrightarrow{b} Q''$, $S' \xrightarrow{b} S''$. This implies $(P' \otimes Q') \sqcap (R' \otimes S') \xrightarrow{(a,b)} (P'' \otimes Q'') \sqcap (R'' \otimes S'')$, and so $(P \otimes Q) \perp (R \otimes S)$. □

6 Ready Specifications

The specification structure presented in the previous section results in a category which supports compositional verification of deadlock-freedom, but there is perhaps a lack of intuition about the use of sets of processes. There is an alternative specification structure for deadlock-freedom, which was in fact the first to be discovered, and which is more easily motivated. It turns out to be equivalent to the sets of processes approach.

6.1 Fundamentals

The reason why deadlock-freedom is not generally preserved by composition in $\mathcal{SP}roc$ is that two deadlock-free processes may, when forced to communicate, reach states from which no further communication is possible even though both processes have more actions available. This observation leads to the idea that if a type is to guarantee compositional deadlock-freedom, it must specify something about which actions a process must be prepared to perform in certain states. The way in which this information is captured is via

the notions of ready pair and ready specification. We will see that ready specifications arise naturally as "datatypes extended in time".

Consider a function $f : A_1 \times \cdots \times A_n \to B$, with each of the types $A_1, \ldots, A_n$ and B determining a set of values. Suppose that in one computation step f receives n inputs and simultaneously produces an output. In each of the n inputs, f is prepared to receive any value—indeed, it is precisely this property which characterises them as inputs—while in the output, f is free to choose which value appears. More generally, if f is a relation rather than a function, we can think of the A_i and B as the types of ports rather than inputs or outputs. In this case, the picture of which values may appear in each port is more complex—some input values may not be accepted, and some outputs may be non-deterministic. More generally still, f may be a process with dynamic behaviour extending through time. Its behaviour can still be characterised by the sets of values which may appear in each port, but now these sets depend on the state of the process. If we indicate the state of a process by the sequence of actions which led to that state, then the following definitions are very natural.

Definition 6.1 *A* ready pair *over an $\mathcal{SP}roc$ object $A = (\Sigma_A, S_A)$ is a pair (s, X) in which $s \in S_A$ and $X \subseteq \Sigma_A$, such that $\forall x \in X.sx \in S_A$. The set X is the* ready set *of the ready pair. A* proper *ready pair is a ready pair (s, X) with $X \neq \varnothing$. The set of proper ready pairs over an object A is denoted by $\mathsf{RP}(A)$. If P is a process of type A, then*

$$\begin{aligned} \mathsf{initials}(P) &\stackrel{\text{def}}{=} \{x \in \Sigma_A \mid \exists Q.P \xrightarrow{x} Q\} \\ \mathsf{readies}(P) &\stackrel{\text{def}}{=} \{(s, X) \mid (P \xrightarrow{s}{}^{*} Q) \wedge (X = \mathsf{initials}(Q))\}. \end{aligned}$$

For any process P, $\mathsf{readies}(P)$ is the set of ready pairs (s, X) representing the actions (those in X) in which P is ready to engage after performing a sequence s of actions. Note that $\mathsf{readies}(P)$ does not necessarily consist entirely of proper ready pairs.

Definition 6.2 *A process P of type A is* deadlock-free *if and only if there is no trace $s \in S_A$ such that $P \xrightarrow{s}{}^{*} \mathsf{nil}$. Equivalently, if and only if there is no trace $s \in S_A$ such that $(s, \varnothing) \in \mathsf{readies}(P)$.*

This is equivalent to the definition of convergence used in Section 5. For example,

$$\mathsf{readies}(a.b.\mathsf{nil} + a.c.\mathsf{nil}) = \{(\varepsilon, \{a\}), (a, \{b\}), (a, \{c\}), (ab, \varnothing), (ac, \varnothing)\}$$

and if $P = a.P$,

$$\mathsf{readies}(P) = \{(a^n, \{a\} \mid n < \omega)\}.$$

The idea of a ready pair, and the related notions of *failures* and *refusals*, appear in the process algebra literature [12, 16, 27]. There, however, they are used to define semantic alternatives to bisimulation; the use made of ready pairs in this paper is very different.

We use a orthogonality relation on ready pairs, rather than on processes as in Section 5.

Definition 6.3 *The orthogonality relation $\perp$ on $\mathsf{RP}(A)$ is defined by*

$$(s, X) \perp (t, Y) \stackrel{\text{def}}{\Leftrightarrow} ((s = t) \Rightarrow X \cap Y \neq \varnothing).$$

The idea is that if (s, X) and (t, Y) are ready pairs of two processes which are supposed to be communicating, $(s, X) \perp (t, Y)$ means that if they have been communicating so far $(s = t)$ there is some action which they are both prepared to do next $(X \cap Y \neq \varnothing)$ and thus

continue the communication. Again, this is a natural generalisation of the requirement in functional programming that an output always be connected to an input. If there are two ports, in which the respective sets of actions X and Y can occur, then connecting them together only leads to correct communication if $X \cap Y \neq \varnothing$. Taking the varying states of the processes into account leads to the definition of orthogonality of ready pairs.

We lift the orthogonality relation to an operation of negation on sets of ready pairs.

Definition 6.4 *Let $\theta \subseteq \mathsf{RP}(A)$ for some object A.*

$$\begin{array}{rcl} (s,X) \perp \theta & \stackrel{\text{def}}{\Leftrightarrow} & \forall (t,Y) \in \theta . (s,X) \perp (t,Y) \\ \theta^{\perp} & \stackrel{\text{def}}{=} & \{(s,X) \in \mathsf{RP}(A) \mid (s,X) \perp \theta\}. \end{array}$$

Proposition 6.5 *For $\theta, \varphi \subseteq \mathsf{RP}(A)$,*

$$\begin{array}{lrcl} 1. & \theta \subseteq \varphi & \Rightarrow & \varphi^{\perp} \subseteq \theta^{\perp} \\ 2. & \theta & \subseteq & \theta^{\perp\perp} \\ 3. & \theta^{\perp\perp\perp} & = & \theta^{\perp}. \end{array}$$

Definition 6.6 *A ready specification over an object A is a non-empty set θ of proper ready pairs over A, satisfying*

- $((s,X) \in \theta) \wedge (x \in X) \Rightarrow \exists Y.(sx, Y) \in \theta$
- $(sx, Y) \in \theta \Rightarrow \exists X.[(s,X) \in \theta \wedge x \in X]$.

The set of ready specifications over A is denoted by $\mathsf{RS}(A)$.

Proposition 6.7 *If A is progressive then $\mathsf{RP}(A) \in \mathsf{RS}(A)$.*

Proof We need to check that $\mathsf{RP}(A)$ satisfies the closure conditions of Definition 6.6. For the first, suppose that $(s,X) \in \mathsf{RP}(A)$ and $x \in X$. Then $sx \in S_A$ by the definition of ready pair. Since A is progressive, there is $a \in \Sigma_A$ such that $sxa \in S_A$. This means that $(sx, \{a\}) \in \mathsf{RP}(A)$.

For the second, observe that if $(sx, Y) \in \mathsf{RP}(A)$ then $(s, \{x\}) \in \mathsf{RP}(A)$ with $x \in \{x\}$. $\Box$

Corollary 6.8 *If A is progressive then $\mathsf{RS}(A) \neq \varnothing$.*

Proposition 6.9 *For any object A of $\mathcal{SP}roc_{pr}$, the following hold.*

1. $\mathsf{RP}(A)^{\perp\perp} = \mathsf{RP}(A)$
2. $\mathsf{RP}(A)^{\perp} = \{(s, \Sigma_A(s))\}) \mid s \in S_A\}$.

Proof

1. $\mathsf{RP}(A)^{\perp\perp}$ is a set of ready pairs, so $\mathsf{RP}(A)^{\perp\perp} \subseteq \mathsf{RP}(A)$. Also $\mathsf{RP}(A) \subseteq \mathsf{RP}(A)^{\perp\perp}$ by Proposition 6.5.
2. It is clear that $\{(s, \Sigma_A(s)) \mid s \in S_A\} \perp \mathsf{RP}(A)$. Conversely, suppose that $(s,X) \perp \mathsf{RP}(A)$. Because $(s, \{x\}) \in \mathsf{RP}(A)$ for every $x \in \Sigma_A$ such that $sx \in S_A$, the definition of orthogonality means that $x \in X$ for each such x. Hence $X = \Sigma_A(s)$ as claimed. $\Box$

Corollary 6.10 *For any object A of $\mathcal{SP}roc_{pr}$, and $\theta \subseteq \mathsf{RP}(A)$,*

$$\theta^{\perp} \supseteq \{(s, \Sigma_A(s))\}) \mid s \in S_A\}.$$

Proof If $\theta \subseteq \mathsf{RP}(A)$ then $\{(s, \Sigma_A(s))\}) \mid s \in S_A\} = \mathsf{RP}(A)^{\perp} \subseteq \theta^{\perp}$. □

6.2 The Specification Structure D'

Again following the sequence of definitions in Section 4.1, we can define the specification structure D' over $\mathcal{SP}roc_{pr}$.

Definition 6.11

1. $P_{D'}A \stackrel{\text{def}}{=} \{\theta \in \mathsf{RS}(A) \mid \theta^{\perp\perp} = \theta\}$.
2. $P \models \theta \stackrel{\text{def}}{\Leftrightarrow} \mathsf{readies}(P) \subseteq \theta$.
3. $(-)^{\perp} : P_{D'}A \to P_{D'}A$ *has already been defined.*
4. $$\begin{aligned} \theta \parr \varphi &\stackrel{\text{def}}{=} \{(s, U \times V) \mid (\mathsf{fst}^*(s), U) \in \theta^{\perp}, (\mathsf{snd}^*(s), V) \in \varphi^{\perp}\}^{\perp} \\ \theta \otimes \varphi &\stackrel{\text{def}}{=} (\theta^{\perp} \parr \varphi^{\perp})^{\perp} \\ \theta \multimap \varphi &\stackrel{\text{def}}{=} \theta^{\perp} \parr \varphi \\ I_{D'} &\stackrel{\text{def}}{=} \{(s, \{*\}) \mid s \in S_I\}. \end{aligned}$$
5. $\theta\{f\}\varphi \stackrel{\text{def}}{=} f \models \theta \multimap \varphi$.
6. *The definitions required to lift the additive and temporal structure of $\mathcal{SP}roc$ to $\mathcal{SP}roc_{D'}$ are*

$$\begin{aligned} \theta \,\&\, \varphi &\stackrel{\text{def}}{=} (\{(\mathsf{inl}^*(s), \mathsf{inl}(X)) \mid (s, X) \in \theta^{\perp}\} \cup \{(\mathsf{inr}^*(t), \mathsf{inr}(Y)) \mid (t, Y) \in \varphi^{\perp}\})^{\perp} \\ \theta \oplus \varphi &\stackrel{\text{def}}{=} (\theta^{\perp} \,\&\, \varphi^{\perp})^{\perp} \\ \circ\theta &\stackrel{\text{def}}{=} (\{(\varepsilon, \{*\})\} \cup \{(*s, X) \mid (s, X) \in \theta^{\perp}\})^{\perp} \end{aligned}$$

Proposition 6.12 *If $\theta \in P_{D'}A$ and $P \models \theta$ then P is deadlock-free.*

Proof Follows from the fact that θ contains only proper ready pairs. □

Proposition 6.13 *If $\theta \in P_{D'}A$ then $\theta \supseteq \{(s, \Sigma_A(s))\}) \mid s \in S_A\}$.*

Proof Follows from Corollary 6.10, because $\theta = (\theta^{\perp})^{\perp}$. □

We will now prove that there is an isomorphism between properties in D' and properties in D. This will enable us to deduce that D' is a specification structure, and that $\mathcal{SP}roc_{D'}$ and $\mathcal{SP}roc_D$ are isomorphic.

6.3 Equivalence of D and D'

We will now prove that for each object A of $\mathcal{SP}roc_{pr}$ there is a bijection between $P_D A$ and $P_{D'} A$, and this bijection preserves all the operations on properties exactly. Also, satisfaction of properties is preserved. It turns out that this leads to an *isomorphism* between $\mathcal{SP}roc_D$ and $\mathcal{SP}roc_{D'}$.

We now have two versions of orthogonality and of every operation on properties. To avoid confusion we will stick to the convention of using $\theta, \varphi, \ldots$ for ready specifications and $U, V, \ldots$ for sets of processes, and begin by proving that the two notions of orthogonality are compatible.

Lemma 6.14 *If $P, Q \in \mathsf{Proc}(A)$, then $P \perp Q \iff \mathsf{readies}(P) \perp \mathsf{readies}(Q)$.*

Proof Suppose $P \perp Q$, $(s, X) \in \mathsf{readies}(P)$ and $(s, Y) \in \mathsf{readies}(Q)$. So $P \xrightarrow{s}{}^* P'$ and $Q \xrightarrow{s}{}^* Q'$, and orthogonality of P and Q implies that there is an action a such that $P' \xrightarrow{a} P''$ and $Q' \xrightarrow{a} Q''$. This means that $a \in \mathsf{initials}(P') = X$ and $a \in \mathsf{initials}(Q') = Y$, so $(s, X) \perp (s, Y)$.

Conversely suppose $\mathsf{readies}(P) \perp \mathsf{readies}(Q)$, $P \xrightarrow{s}{}^* P'$ and $Q \xrightarrow{s}{}^* Q'$. Then we have $(s, \mathsf{initials}(P')) \in \mathsf{readies}(P)$ and $(s, \mathsf{initials}(Q')) \in \mathsf{readies}(Q)$, and this implies

$$\mathsf{initials}(P') \cap \mathsf{initials}(Q') \neq \varnothing.$$

Thus there is an action a such that $P' \xrightarrow{a} P''$ and $Q' \xrightarrow{a} Q''$, so $P \perp Q$. □

Definition 6.15 *Let $U \in P_D A$ and $\theta \in P_{D'} A$.*

$$\begin{aligned} F(\theta) &\stackrel{\text{def}}{=} \{P \in \mathsf{Proc}(A) \mid \mathsf{readies}(P) \subseteq \theta\} \\ G(U) &\stackrel{\text{def}}{=} \bigcup \{\mathsf{readies}(P) \mid P \in U\}. \end{aligned}$$

Proposition 6.16 *If $\theta \in P_{D'} A$ then $GF(\theta) = \theta$.*

Proof If $(s, X) \in GF(\theta)$ there is $P \in F(\theta)$ with $(s, X) \in \mathsf{readies}(P)$. This means that $(s, X) \in \theta$, because $P \in F(\theta) \Rightarrow \mathsf{readies}(P) \subseteq \theta$. Hence $GF(\theta) \subseteq \theta$.

If $(s, X) \in \theta$ then establishing $(s, X) \in GF(\theta)$ requires $P \in F(\theta)$ such that $(s, X) \in \mathsf{readies}(P)$. This means finding P with $\mathsf{readies}(P) \subseteq \theta$ and $(s, X) \in \mathsf{readies}(P)$. We know that $(t, \Sigma_A(t)) \in \theta$ for any trace $t \in S_A$. In max_A there is a unique state reachable by the trace s. By removing branches from this state, we can construct a process P with the required property. □

Proposition 6.17 *If $U \in P_D A$ then $FG(U) = U$.*

Proof If $P \in U$ then $\mathsf{readies}(P) \subseteq G(U)$, so $P \in FG(U)$. Hence $U \subseteq FG(U)$.

If $P \in FG(U)$ then $\mathsf{readies}(P) \subseteq G(U)$. So for any $(s, X) \in \mathsf{readies}(P)$ there is $Q \in U$ such that $(s, X) \in \mathsf{readies}(Q)$. If $R \in U^\perp$ and $(t, Y) \in \mathsf{readies}(R)$, this means that $(s, X) \perp (t, Y)$, because $R \perp Q$ and by Lemma 6.14. Hence $P \perp R$, i.e. $P \in U^{\perp\perp} = U$. Thus $FG(U) \subseteq U$. □

Lemma 6.18 *Satisfaction is preserved by the correspondence between D and D', i.e.*

$$\begin{aligned} \mathsf{readies}(P) \subseteq \theta &\iff P \in F(\theta) \\ P \in U &\iff \mathsf{readies}(P) \subseteq G(U). \end{aligned}$$

Proof The only non-trivial part is $\mathsf{readies}(P) \subseteq G(U) \Rightarrow P \in U$; the others follow easily from the definitions of F and G. If $\mathsf{readies}(P) \subseteq G(U)$ then $P \in FG(U) = U$. □

Lemma 6.19 *$F(\theta^\perp) \subseteq F(\theta)^\perp$ and $G(U^\perp) \subseteq G(U)^\perp$.*

Proof If $P \in F(\theta^\perp)$ and $Q \in F(\theta)$ then $\mathsf{readies}(P) \subseteq \theta^\perp$ and $\mathsf{readies}(Q) \subseteq \theta$, so $P \perp Q$.

If $(s, X) \in G(U^\perp)$ and $(t, Y) \in G(U)$ then $\exists P \in U^\perp.(s, X) \in \mathsf{readies}(P)$ and $\exists Q \in U.(t, Y) \in \mathsf{readies}(Q)$. Since $P \perp Q$, $\mathsf{readies}(P) \perp \mathsf{readies}(Q)$ and hence $(s, X) \perp (t, Y)$. □

Lemma 6.20 *If $\theta \in P_{D'}A$, there are processes $P_1, \ldots, P_n$ such that*

$$\mathsf{readies}(P_1) \cup \ldots \cup \mathsf{readies}(P_n) = \theta.$$

Proof Define a labelled transition system whose states are the ready pairs in θ, with transitions defined by

$$\frac{a \in X}{(s, X) \xrightarrow{a} (sa, Y).}$$

We have $(s, \Sigma_A(s)) \in \theta$ for each $s \in S_A$. So for any pair $(sa, Y) \in \theta$ there is the transition $(s, \Sigma_A(s)) \xrightarrow{a} (sa, Y)$, which means that every state is reachable from $(\varepsilon, \Sigma_A(\varepsilon))$, except for any (ε, X) with $X \neq \Sigma_A(\epsilon)$. This means that the states (ε, X) can be taken as the processes $P_1, \ldots, P_n$. □

Proposition 6.21 *$F(\theta^\perp) = F(\theta)^\perp$.*

Proof It is enough to prove $F(\theta)^\perp \subseteq F(\theta^\perp)$. If $P \in F(\theta)^\perp$ then $P \perp F(\theta)$. Let $Q_1, \ldots, Q_n$ be such that $\mathsf{readies}(Q_1) \cup \ldots \cup \mathsf{readies}(Q_n) = \theta$. Each $Q_i \in F(\theta)$, hence $P \perp Q_i$ for each i. So $\mathsf{readies}(P) \perp \theta$ and hence $\mathsf{readies}(P) \subseteq \theta^\perp$. Thus $P \in F(\theta^\perp)$. □

Proposition 6.22 *$G(U^\perp) = G(U)^\perp$.*

Proof

$$\begin{aligned} G(U)^\perp &= G(F(G(U)^\perp)) \\ &= G(F(G(U))^\perp) \\ &= G(U^\perp). \end{aligned}$$

□

Corollary 6.23 *$F(\theta) = F(\theta)^{\perp\perp}$ and $G(U) = G(U)^{\perp\perp}$.*

Proposition 6.24 *If $\theta \in P_{D'}A$ and $\varphi \in P_{D'}B$ then $F(\theta \otimes \varphi) = F(\theta) \otimes F(\varphi)$.*

Proof It is enough to show that $F(\theta^\perp \otimes \varphi^\perp) = F(\theta^\perp) \otimes F(\varphi^\perp)$. Now,

$$\theta^\perp \otimes \varphi^\perp = \{(s, A \times B) \mid (\mathsf{fst}^*(s), A) \in \theta, (\mathsf{snd}^*(s), B) \in \varphi\}^\perp$$

and

$$\begin{aligned} F(\theta^\perp) \parr F(\varphi^\perp) &= F(\theta)^\perp \parr F(\varphi)^\perp \\ &= (F(\theta) \otimes F(\varphi))^\perp \\ &= \{P \otimes Q \mid P \in F(\theta), Q \in F(\varphi)\}^\perp \end{aligned}$$

so we need to show that

$$F[\{(s, A \times B) \mid (\mathsf{fst}^*(s), A) \in \theta, (\mathsf{snd}^*(s), B) \in \varphi\}^\perp] = \{P \otimes Q \mid P \in F(\theta), Q \in F(\varphi)\}^\perp.$$

We will consider the two inclusions separately. If

$$R \in F[\{(s, A \times B) \mid (\mathsf{fst}^*(s), A) \in \theta, (\mathsf{snd}^*(s), B) \in \varphi\}^\perp]$$

then

$$\mathsf{readies}(R) \perp \{(s, A \times B) \mid (\mathsf{fst}^*(s), A) \in \theta, (\mathsf{snd}^*(s), B) \in \varphi\}.$$

For any s, A, B with $(\mathsf{fst}^*(s), A) \in \theta$ and $(\mathsf{snd}^*(s), B) \in \varphi$, $(s, X) \in \mathsf{readies}(R)$ implies that there is $(a, b) \in X$ with $a \in A$ and $b \in B$. So if $\mathsf{readies}(P) \subseteq \theta$ and $\mathsf{readies}(Q) \subseteq \varphi$, $R \perp (P \otimes Q)$.

For the other inclusion, suppose that $R \perp \{P \otimes Q \mid P \in F(\theta), Q \in F(\varphi)\}$. We need to show that $\mathsf{readies}(R) \perp \{(s, A \times B) \mid (\mathsf{fst}^*(s), A) \in \theta, (\mathsf{snd}^*(s), B) \in \varphi\}$. Let $(s, X) \in \mathsf{readies}(R)$, $(\mathsf{fst}^*(s), A) \in \theta$ and $(\mathsf{snd}^*(s), B) \in \varphi$. By Lemma 6.20 there are P and Q such that $(\mathsf{fst}^*(s), A) \in \mathsf{readies}(P)$, $(\mathsf{snd}^*(s), B) \in \mathsf{readies}(Q)$, $P \in F(\theta)$ and $Q \in F(\varphi)$. Because $R \perp P \otimes Q$, after the trace s there is an action (a, b) available to both R and $P \otimes Q$. Hence $(a, b) \in X \cap (A \times B)$. □

Corollary 6.25 *$F(\theta \otimes \varphi) = F(\theta) \otimes F(\varphi)$.*

Proof This follows from the fact that F preserves $\parr$ and $(-)^\perp$, and duality of $\otimes$ and $\parr$. □

Corollary 6.26 *$G(U \parr V) = G(U) \parr G(V)$ and $G(U \otimes V) = G(U) \otimes G(V)$.*

Proof

$$\begin{aligned} G(U \parr V) &= G(FG(U) \parr FG(V)) \\ &= GF(G(U) \parr G(V)) \\ &= G(U) \parr G(V). \end{aligned}$$

Again, $G(U \otimes V) = G(U) \otimes G(V)$ follows easily. □

Proposition 6.27 *$F(\theta \mathbin{\&} \varphi) = F(\theta) \mathbin{\&} F(\varphi)$ and $G(U \mathbin{\&} V) = G(U) \mathbin{\&} G(V)$.*

Proof

$$\begin{aligned} F(\theta \mathbin{\&} \varphi) &= F((\theta \mathbin{\&} \varphi)^{\perp\perp}) \\ &= (F(\theta \mathbin{\&} \varphi))^{\perp\perp} \\ &= \{P \mid P \models \theta \mathbin{\&} \varphi\}^{\perp\perp} \\ &= \{Q[\mathsf{inl}] + R[\mathsf{inr}] \mid Q \models \theta, R \models \varphi\}^{\perp\perp} \\ &= \{Q[\mathsf{inl}] + R[\mathsf{inr}] \mid Q \in F(\theta), R \in F(\varphi)\}^{\perp\perp} \end{aligned}$$

$$= F(\theta) \,\&\, F(\varphi).$$

$$\begin{aligned} G(U \,\&\, V) &= G(F(G(U)) \,\&\, F(G(V))) \\ &= G(F(G(U) \,\&\, G(V))) \\ &= G(U) \,\&\, G(V). \end{aligned}$$

□

Proposition 6.28 *$F(\circ\theta) = \circ F(\theta)$ and $G(\circ U) = \circ G(U)$.*

Proof

$$\begin{aligned} F(\circ\theta) &= \{P \mid P \models \circ\theta\} \\ &= \{\circ Q \mid Q \models \theta\} \\ &= \circ F(\theta). \end{aligned}$$

$$\begin{aligned} G(\circ U) &= G(\circ F(G(U))) \\ &= G(F(\circ G(U))) \\ &= \circ G(U). \end{aligned}$$

□

Proposition 6.29 *$F(\mathsf{RP}(A)) = \mathsf{out}_A$ and $F(\mathsf{RP}(A)^\perp) = \mathsf{in}_A$.*

Proof It is sufficient to prove the first statement.

$$\begin{aligned} F(\mathsf{RP}(A)) &= \{P \in \mathsf{Proc}(A) \mid \mathsf{readies}(P) \subseteq \mathsf{RP}(A)\} \\ &= \mathsf{Proc}(A) \\ &= \mathsf{out}_A. \end{aligned}$$

□

From the definitions of $\mathsf{RP}(A)$ and $\mathsf{RP}(A)^\perp$, we have that in the ready specifications formulation, $\mathsf{in}_A = \{(s, \Sigma_A(s)) \mid s \in S_A\}$ and $\mathsf{out}_A = \{(s, X) \mid s \in S_A, \varnothing \neq X \subseteq \Sigma_A(s)\}$. This provides an alternative view of why the properties in and out correspond to input and output. A port of type in is always ready to receive any action in the available alphabet, whereas a port of type out can enter states in which arbitrary subsets of the alphabet are not available.

6.4 Products and Coproducts

Now that the specification structures D and D' have been shown to be equivalent, any calculations relating to deadlock-freedom can be carried out in whichever setting is more convenient. The proof that products and coproducts lift to $\mathcal{SP}roc_D$ was omitted from Section 5; we will now present it in terms of ready specifications, which turns out to be easier. By duality, it is sufficient to consider products.

Lemma 6.30 *Let A, B be objects of $\mathcal{SP}roc_{pr}$ with $\theta \in P_{D'}A$, $\varphi \in P_{D'}B$.*

1. *If $s \neq \varepsilon$ then $(\mathsf{inl}^*(s), X) \in \theta \,\&\, \varphi \iff (s, \{x \mid \mathsf{inl}(x) \in X\}) \in \theta$.*

2. *If* $s \neq \varepsilon$ *then* $(\mathsf{inr}^*(s), X) \in \theta \,\&\, \varphi \iff (s, \{x \mid \mathsf{inr}(x) \in X\}) \in \varphi$.

3. $(\varepsilon, X) \in \theta \,\&\, \varphi \iff \exists U, V.(\epsilon, U) \in \theta, (\epsilon, V) \in \varphi$ *and* $X = \mathsf{inl}(U) \cup \mathsf{inr}(V)$.

Proof

1. ($\Rightarrow$) We will show that $(s, \{x \mid \mathsf{inl}(x) \in X\}) \perp (t, Y)$ for every $(t, Y) \in \theta^\perp$. It is sufficient to consider $(s, Y) \in \theta^\perp$; we then need $\{x \mid \mathsf{inl}(x) \in X\} \cap Y \neq \varnothing$. Since $(\mathsf{inl}^*(s), X) \in \theta \,\&\, \varphi$, the definition of $\theta \,\&\, \varphi$ implies that $X \cap \mathsf{inl}(Y) \neq \varnothing$, and we are done.

   ($\Leftarrow$) We need $(\mathsf{inl}^*(s), X) \perp (\mathsf{inl}^*(s), \mathsf{inl}(U))$ for every $(s, U) \in \theta^\perp$. Since

   $$(s, \{x \mid \mathsf{inl}(x) \in X\}) \in \theta$$

   we have $U \cap \{x \mid \mathsf{inl}(x) \in X\} \neq \varnothing$, and so $X \cap \mathsf{inl}(U) \neq \varnothing$.

2. An identical argument.

3. ($\Rightarrow$) Take $U \stackrel{\text{def}}{=} \{x \mid \mathsf{inl}(x) \in X\}$, $V \stackrel{\text{def}}{=} \{x \mid \mathsf{inr}(x) \in X\}$, so that $X = \mathsf{inl}(U) \cup \mathsf{inr}(V)$. To show that $(\varepsilon, U) \in \theta$, consider $(\varepsilon, Y) \in \theta^\perp$. The definition of $\theta \,\&\, \varphi$ implies $(\varepsilon, X) \perp (\varepsilon, \mathsf{inl}(Y))$ and so $X \cap \mathsf{inl}(Y) \neq \varnothing$. Hence $U \cap Y \neq \varnothing$, i.e. $(\varepsilon, U) \perp (\varepsilon, Y)$. So $(\varepsilon, U) \in \theta^{\perp\perp} = \theta$.

   An identical argument shows that $(\varepsilon, V) \in \varphi$.

   ($\Leftarrow$) Suppose $X = \mathsf{inl}(U) \cup \mathsf{inr}(V)$ with $(\epsilon, U) \in \theta$ and $(\epsilon, V) \in \varphi$. For any $(\epsilon, W) \in \theta^\perp$, $W \cap U \neq \varnothing$ and hence $W \cap X \neq \varnothing$. For any $(\epsilon, Z) \in \varphi^\perp$, $Z \cap V \neq \varnothing$ and hence $X \cap V \neq \varnothing$. Thus $(\epsilon, X) \in \theta \,\&\, \varphi$. □

Proposition 6.31 *Let* A, B, C *be objects of* $\mathcal{SP}roc_{pr}$ *and let* $\theta, \varphi, \psi \in P_{D'}A, P_{D'}B, P_{D'}C$. *Let* $f : A \to B$ *and* $g : A \to C$ *with* $\theta\{f\}\varphi$ *and* $\theta\{g\}\psi$.

1. $(\theta \,\&\, \varphi)\{\pi_1\}\theta$
2. $(\theta \,\&\, \varphi)\{\pi_2\}\varphi$
3. $\theta\{\langle f, g\rangle\}(\varphi \,\&\, \psi)$

Proof

1. We need

   $$\begin{aligned} \mathsf{readies}(\pi_1) &\subseteq (\theta \,\&\, \varphi) \multimap \theta \\ &= (\theta \,\&\, \varphi)^\perp \parr \theta \\ &= \{(s, U \times V) \mid (\mathsf{fst}^*(s), U) \in \theta \,\&\, \varphi, (\mathsf{snd}^*(s), V) \in \theta^\perp\}^\perp. \end{aligned}$$

   Consider any s, U, V, X with

   $$\begin{aligned} (\mathsf{fst}^*(s), U) &\in \theta \,\&\, \varphi \\ (\mathsf{snd}^*(s), V) &\in \theta^\perp \\ (s, X) &\in \mathsf{readies}(\pi_1). \end{aligned}$$

   For some $(t \;\mathsf{zip}\; t, Y) \in \mathsf{readies}(\mathsf{id}_A)$ we have $s = \mathsf{inl}^*(t) \;\mathsf{zip}\; t$ and

   $$X = \{(\mathsf{inl}(a), b) \mid (a, b) \in Y\}.$$

So $\mathsf{fst}^*(s) = \mathsf{inl}^*(t)$ and $\mathsf{snd}^*(s) = t$.

By Lemma 6.30, either (1) $\mathsf{fst}^*(s) = \varepsilon$ and $U = \mathsf{inl}(W) \cup \mathsf{inr}(W')$ for some W,W' with $(\varepsilon, W) \in \theta$ and $(\varepsilon, W') \in \varphi$, in which case we will say $(t, W) \in \theta$ with $t = \varepsilon$; or (2) $\mathsf{fst}^*(s) \neq \varepsilon$ and $U = \mathsf{inl}(W)$ for some W with $(t, W) \in \theta$.

Because $\mathsf{id}_A \models \theta^\perp \parr \theta$, i.e.

$$\mathsf{readies}(\mathsf{id}_A) \perp \{(u, Z \times T) \mid (\mathsf{fst}^*(u), Z) \in \theta, (\mathsf{snd}^*(u), T) \in \theta^\perp\},$$

in either case we have $(t \ \mathsf{zip} \ t, Y) \perp (t \ \mathsf{zip} \ \mathsf{snd}^*(s), W \times V)$, so $Y \cap (W \times V) \neq \varnothing$. Hence $X \cap (U \times V) \neq \varnothing$, since X and U are defined by relabelling Y and W.

2. A symmetrical argument.

3. We need

$$\mathsf{readies}(\langle f, g \rangle) \quad \perp \quad \{(s, U \times V) \mid (\mathsf{fst}^*(s), U) \in \theta, (\mathsf{snd}^*(s), V) \in (\varphi \mathbin{\&} \psi)^\perp\}.$$

Consider any s, U, V, X with

$$\begin{array}{rcl} (s, X) & \in & \mathsf{readies}(\langle f, g \rangle) \\ (\mathsf{fst}^*(s), U) & \in & \theta \\ (\mathsf{snd}^*(s), V) & \in & (\varphi \mathbin{\&} \psi)^\perp. \end{array}$$

There are three cases.

(a) $s = \varepsilon$, so that $X = \mathsf{initials}(f)[(a, b) \mapsto (a, \mathsf{inl}(b))] \cup \mathsf{initials}(g)[(a, c) \mapsto (a, \mathsf{inr}(c))]$ and, by Lemma 6.30, $V = \mathsf{inl}(V_1) \cup \mathsf{inr}(V_2)$ with $(\varepsilon, V_1) \in \varphi^\perp$ and $(\varepsilon, V_2) \in \psi^\perp$. Then $f \models \theta^\perp \parr \varphi$ implies that $\mathsf{initials}(f) \cap (U \times V_1) \neq \varnothing$, and hence $X \cap (U \times V) \neq \varnothing$.

(b) $s \neq \varepsilon$ and $\mathsf{snd}^*(s) = \mathsf{inl}^*(t)$, so that $X = Y[(a, b) \mapsto (a, \mathsf{inl}(b))]$ with

$$(\mathsf{fst}^*(s) \ \mathsf{zip} \ t, Y) \in \mathsf{readies}(f)$$

and $V = \mathsf{inl}(V_1)$ with $(t, V_1) \in \varphi^\perp$. Then $f \models \theta^\perp \parr \varphi$ implies that $(\mathsf{fst}^*(s) \ \mathsf{zip} \ t, Y) \perp (\mathsf{fst}^*(s) \ \mathsf{zip} \ t, U \times V_1)$; so $Y \cap (U \times V_1) \neq \varnothing$, and hence $X \cap (U \times V) \neq \varnothing$.

(c) A symmetrical case. □

6.5 Ready Equivalence and $^{\perp\perp}$-invariance

It is possible that $\mathsf{readies}(P) = \mathsf{readies}(Q)$ with $P \neq Q$, and in this case the processes P and Q satisfy exactly the same ready specifications. It is not possible for distinct processes to be contained in exactly the same sets of processes: if $P \neq Q$ then $P \notin \{Q\}$. So it appears possible that sets of processes may make finer distinctions than ready specifications. However, if distinct processes have the same readies, they must be in the same $^{\perp\perp}$-invariant sets of processes, as shown by the following proposition.

Proposition 6.32 *If $P \in U$ and $\mathsf{readies}(P) = \mathsf{readies}(Q)$ then $Q \in U^{\perp\perp}$.*

Proof If $R \in U^\perp$ then $\mathsf{readies}(R) \perp \mathsf{readies}(P)$. So $\mathsf{readies}(R) \perp \mathsf{readies}(Q)$, which means that $Q \in U^{\perp\perp}$. □

Defining two processes to be *ready-equivalent* if they have the same readies, this result says that a $^{\perp\perp}$-invariant set of processes must be the union of a collection of ready-equivalence classes. So membership of $^{\perp\perp}$-invariant sets cannot distinguish processes more finely than ready-equivalence.

6.6 $\mathcal{SP}roc_D$ and $\mathcal{SP}roc_{D'}$ are Isomorphic

Because of the extremely strong connection which we have established between the specification structures D and D', it turns out that the corresponding categories are *isomorphic*, i.e. that there are functors $\mathcal{F} : \mathcal{SP}roc_D \leftrightarrows \mathcal{SP}roc_{D'} : \mathcal{G}$ with $\mathcal{FG} = \mathcal{I}_{\mathcal{SP}roc_{D'}}$ and $\mathcal{GF} = \mathcal{I}_{\mathcal{SP}roc_D}$.

Given an object (A, U) of $\mathcal{SP}roc_D$, $\mathcal{F}(A, U) \stackrel{\text{def}}{=} (X, F(U))$. Given a morphism $f : (A, U) \to (B, V)$ of $\mathcal{SP}roc_D$, $\mathcal{F}(f) \stackrel{\text{def}}{=} f : (A, F(U)) \to (B, F(V))$. Note that if $f : (A, U) \to (B, V)$ then we have $f \models U \multimap V$ and, because of the equivalence of satisfaction in D and D' and the fact that F preserves the linear connectives, this gives $f \models F(U) \multimap F(V)$ also. Hence f really is a morphism $(A, F(U)) \to (B, F(V))$ in $\mathcal{SP}roc_{D'}$. Because $\mathcal{F}$ does not change morphisms, composition and identities are trivially preserved. The functor $\mathcal{G}$ is defined similarly, and the fact that $\mathcal{F}$ and $\mathcal{G}$ are mutually inverse follows from the fact that F and G are mutually inverse. Furthermore, $\mathcal{F}$ and $\mathcal{G}$ preserve all of the structure of the categories; again, this is simply because F and G preserve all the structure.

7 Synchronous Networks

In this section we will consider some applications of our techniques to systems of practical interest. There is a class of concurrent systems to which our theory is very well suited; we call these systems *synchronous networks*. A synchronous network consists of a number of processes or nodes, each with various ports, which are connected together in some configuration. The key points are that once the network has been constructed, its topology does not change; and that the entire system is subject to the synchrony hypothesis with which we have been working throughout. The two main examples of synchronous networks are synchronous dataflow programs, written in languages such as SIGNAL [25] and LUSTRE [26], and systolic algorithms [20].

Given that the topology of a network never changes, the operation of categorical composition (parallel composition + hiding) is suitable for forming a fixed, private connection between two nodes. As we have already seen in Section 3 the structure of a compact closed category such as $\mathcal{SP}roc$ allows arbitrary networks to be constructed by means of categorical operations. We are also interested in constructing networks in $\mathcal{SP}roc_D$, to ensure deadlock-freedom; however, loss of compact closure means that cyclic networks cannot be cconstructed without some additional analysis. By suitable use of the deadlock-free types in and out, and their properties, we are able to identify which cycles can always be safely constructed, and formulate an additional proof rule for those which may be unsafe.

7.1 Networks in a Compact Closed Category

Suppose we are working in a compact closed category, potentially one in which $(-)^\perp$ is non-trivial. Suppose also that for each datatype used by a particular network, there is an object in the category suitable for modelling a port of that type. The $(-)^\perp$ operation is used to switch between input and output, in the sense that a port of type A must be

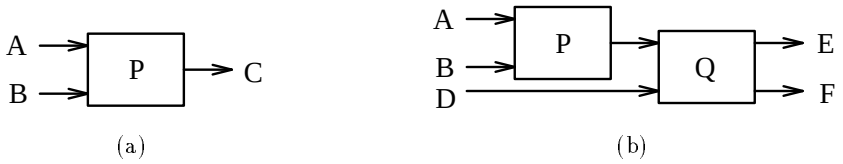


Figure 4: Two simple networks

connected to a port of type $A^\perp$, but at this stage we have not chosen which of A and $A^\perp$ is input and which is output.

In general, when working with a $*$-autonomous category, a node with n ports of types $A_1, \ldots, A_n$ is represented by a process of type $A_1 \parr \cdots \parr A_n$, i.e. a morphism $P : I \to A_1 \parr \cdots \parr A_n$. The closed structure allows types to be moved across the arrow; in a compact closed category we do not have to worry about the effect that this has on the connectives, and we can replace every connective by $\otimes$. The only condition is that when a type is moved across the arrow, $(-)^\perp$ must be applied. For example, a process P with three ports of types $A^\perp$, $B^\perp$ and C could be represented as $P : C^\perp \to A^\perp \otimes B^\perp$, $P : A \otimes C^\perp \to B^\perp$, $P : A \otimes B \to C$, and so on. If we wish to interpret $A^\perp$ and $B^\perp$ as input types and C as an output type, then the last of these makes the most sense, and we might draw the process as in Figure 4(a). In this way, any desired network can be constructed as a morphism in the category, with the calculation described in Section 3 being used to form cycles. For example, the morphism corresponding to the network in Figure 4(b) is

$$(P \otimes \mathsf{id}_D)\,;Q : A \otimes B \otimes D \to E \otimes F$$

where the morphisms corresponding to the individual nodes are $P : A \otimes B \to C$ and $Q : C \otimes D \to E \otimes F$.

7.2 Deadlock-Free Types

We will now consider ways of typing the nodes of a network in $\mathcal{SP}roc_D$. In many cases it is possible to identify each port of a process as either an input or an output, and this allows us to use the types in and out. Since $\mathcal{SP}roc$ is based on synchronization rather than value-passing, we need to define what it means for ports of an $\mathcal{SP}roc$ process to be inputs.

Definition 7.1 *Let $P : A_1 \parr \cdots \parr A_n$ in $\mathcal{SP}roc$ and let $J \subseteq \{1, \ldots, n\}$. P is* receptive *in the ports J if whenever $P \xrightarrow{s}{}^* Q$ and $\forall j \in J\,.\, a_j \in \Sigma_{A_j}(\pi_j^*(s))$ then for each $i \in \{1, \ldots, n\} - J$ there is $a_i \in \Sigma_{A_i}(\pi_i^*(s))$ such that $Q \xrightarrow{(a_1,\ldots,a_n)} R$ for some R.*

Receptivity in a set of ports means those ports correspond to inputs and are independently able to receive arbitrary values. When a dataflow node is modelled by an $\mathcal{SP}roc$ process, that process is receptive in each port which we consider to be an input of the node.

Proposition 7.2 *Let $P : A_1 \parr \cdots \parr A_n$ be any $\mathcal{SP}roc$ process and let J be the set of ports in which it is receptive. Defining $\theta_i \in P_D A_i$ by*

$$\theta_i \stackrel{\text{def}}{=} \begin{cases} \mathsf{in} & \text{if } i \in J \\ \mathsf{out} & \text{otherwise} \end{cases}$$

gives $P : (A_1, \theta_1) \parr \cdots \parr (A_n, \theta_n)$ in $\mathcal{SP}roc_D$.

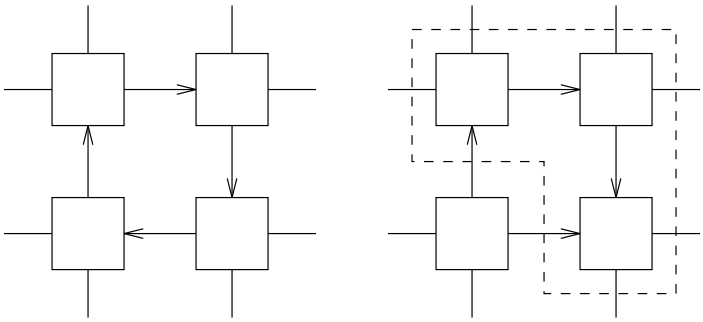


Figure 5: The two kinds of cycle

Proof We will use the ready specification formulation of deadlock-free types. Without loss of generality, assume that $J = \{1, \ldots, m\}$. We need to show that

$$\mathsf{readies}(P) \quad \subseteq \quad \{(s, X_1 \times \cdots \times X_n) \mid \forall i.(\pi_i^*(s), X_i) \in \theta_i^\perp\}^\perp$$

i.e. that

$$\mathsf{readies}(P) \quad \perp \quad \{(s, X_1 \times \cdots \times X_n) \mid \forall i.(\pi_i^*(s), X_i) \in \theta_i^\perp\}.$$

Pick $(s_1, X_1) \ldots (s_n, X_n)$ and X such that for each $i \in \{1, \ldots, n\}$, $(s_i, X_i) \in \theta_i^\perp$, and $(s, X) \in \mathsf{readies}(P)$, where $s = s_1 \mathsf{\ zip\ } \ldots \mathsf{\ zip\ } s_n$. We need to show that $(X_1 \times \cdots \times X_n) \cap X \neq \varnothing$.

For each $i \in \{1, \ldots, m\}$ pick $a_i \in X_i$. Because $(s, X) \in \mathsf{readies}(P)$ there is a process Q such that $P \xrightarrow{s}{}^* Q$ and $X = \mathsf{initials}(Q)$. Because P is receptive in ports $\{1, \ldots, m\}$, for each $j \in \{m+1, \ldots, n\}$ there is $a_j \in \Sigma_{A_j}(s_j)$ such that $Q \xrightarrow{(a_1, \ldots, a_n)} R$ for some R. Hence $(a_1, \ldots, a_n) \in X$.

For each $j \in \{m+1, \ldots, n\}$ we have $\theta_j^\perp = \mathsf{in}$, so $X_j = \Sigma_{A_j}(s_j)$, and $a_j \in X_j$. Hence $(a_1, \ldots, a_n) \in X_1 \times \cdots \times X_n$. □

This result allows any node to be assigned a type on the basis of a classification of its ports as inputs or outputs. If a network is constructed in $\mathcal{SP}roc_D$ according to the type discipline, this corresponds to obeying the constraint that every connection is between an output and an input. As we know, the result is a network which will not deadlock. The type system of $\mathcal{SP}roc_D$ does not allow cyclic connections to be established; however, cycles are very likely to be present in any interesting network, and we need to be able to construct them.

Now that we have identified certain ports as inputs, it is possible to see that not all cycles have the same structure. In Figure 5 the arrows point from outputs to inputs. Each of the two networks contains a cycle, but the patterns of flow of data are different. In the cycle on the right, one node has two outputs coming from it; if the part of the network enclosed in dashed lines is considered as a single node, this means that the cycle can be constructed by simultaneously connecting two outputs from one node to two inputs of another. The cycle on the left does not have this property, and represents a genuine feedback loop. In general, consider a polygon with n sides and orient each side by adding an arrow in one direction or the other. Starting from any vertex, follow the arrows; either we return to the initial vertex, or we arrive at a vertex with two arrows pointing at it. The first case corresponds to a feedback loop; in the second case, a dual argument shows that there is also a vertex with two arrows pointing away from it.

This means that any cycle which is not a feedback loop can be reduced to a simultaneous connection of two outputs from one process to two inputs of another, as in Figure 6. In

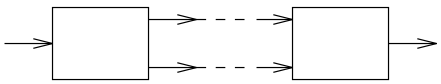


Figure 6: A double connection between nodes

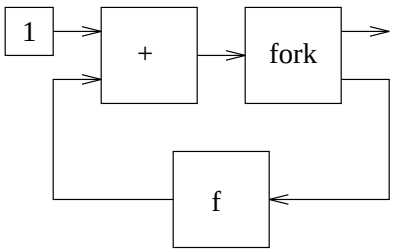


Figure 7: A network with feedback

$\mathcal{SP}roc_D$ we have processes $P : (A, \mathsf{in}_A) \parr (B, \mathsf{out}_B) \parr (C, \mathsf{out}_C)$ and $Q : (B, \mathsf{in}_B) \parr (C, \mathsf{in}_C) \parr (D, \mathsf{out}_D)$. Writing P and Q as morphisms gives

$$\begin{array}{rcl} P & : & (A, \mathsf{out}_A) \to (B, \mathsf{out}_B) \parr (C, \mathsf{out}_C) \\ Q & : & (B, \mathsf{out}_B) \otimes (C, \mathsf{out}_C) \to (D, \mathsf{out}_D) \end{array}$$

or equivalently

$$\begin{array}{rcl} P & : & (A, \mathsf{out}_A) \to (B \parr C, \mathsf{out}_B \parr \mathsf{out}_C) \\ Q & : & (B \otimes C, \mathsf{out}_B \otimes \mathsf{out}_C) \to (D, \mathsf{out}_D). \end{array}$$

By Proposition 5.22 we have $\mathsf{out}_B \parr \mathsf{out}_C = \mathsf{out}_{B \parr C}$ and $\mathsf{out}_B \otimes \mathsf{out}_C = \mathsf{out}_{B \otimes C}$. Combined with the fact that $B \otimes C = B \parr C$, this means that P and Q are composable, and we obtain $P\,;Q : (A, \mathsf{out}_A) \to (D, \mathsf{out}_D)$, or equivalently $P\,;Q : (A, \mathsf{in}_A) \parr (D, \mathsf{out}_D)$. Hence $P\,;Q$ is a deadlock-free process.

We now have to deal with the case of a feedback loop. As an example of the use of feedback in dataflow programming, consider the network in Figure 7. The node 1 produces the sequence $111\ldots$ and the function f is defined on streams by $f(\sigma) = 0\sigma$. The fork node simply copies its input, and the $+$ node outputs at each instant the sum of the inputs received at the same instant. The output x is defined by $x = 111\ldots + f(x)$, and the least solution of this equation (i.e. the least fixed point of $\lambda x.111\ldots + 0x$) is $x = 1234\ldots$. The significant feature of f is that its first output token is independent of any input, and subsequently there is always a delay of one time unit between an input being received and the corresponding output being produced. For a dataflow network to be free of deadlock, every feedback loop should contain a node such as f. In LUSTRE, the corresponding node is called pre, and the language specifies that every loop must contain at least one pre. We will now give a semantic formulation of this property of nodes, and show that it yields a sufficient condition for the formation of deadlock-free cycles.

Definition 7.3 *Let $P : (A_1, \mathsf{in}) \parr \cdots \parr (A_m, \mathsf{in}) \parr (B_1, \mathsf{out}) \parr \cdots \parr (B_n, \mathsf{out})$ in $\mathcal{SP}roc_D$. Output i of P is independent of input j if whenever $P \xrightarrow{s}{}^{*} Q$, $\forall a_1, \ldots, a_{j-1}, a_{j+1}, \ldots, a_m$ $\exists b$ such that for all R, $Q \xrightarrow{(a_1,\ldots,a_m,b_1,\ldots,b_n)} R \Rightarrow b_i = b$.*

Proposition 7.4 *Suppose $P : (A_1, \mathsf{in}) \parr \cdots \parr (A_m, \mathsf{in}) \parr (A_{m+1}, \mathsf{out}) \parr \cdots \parr (A_n, \mathsf{out}) \parr (A_{n+1}, \mathsf{in}) \parr (A_{n+1}, \mathsf{out})$ in $\mathcal{SP}roc_D$ and let $\overline{P}$ be the $\mathcal{SP}roc$ process obtained by connecting ports (A_{n+1}, out) and (A_{n+1}, in) of P. If the output at port (A_{n+1}, out) of P is independent of the input at port (A_{n+1}, in), then $\overline{P} : (A_1, \mathsf{in}) \parr \cdots \parr (A_m, \mathsf{in}) \parr (A_{m+1}, \mathsf{out}) \parr \cdots \parr (A_n, \mathsf{out})$ in $\mathcal{SP}roc_D$.*

Proof We need to show that

$$\mathsf{readies}(P) \quad \bot \quad \{(s, X_1 \times \cdots \times X_n) \mid \forall i.(\pi_i^*(s), X_i) \in \theta_i^\bot\}$$

where $\theta_1 \ldots \theta_m$ are in and $\theta_{m+1} \ldots \theta_n$ are out.

Pick $(s_1, X_1) \ldots (s_n, X_n)$ and X such that for each $i \in \{1, \ldots, n\}$, $(s_i, X_i) \in \theta_i^\bot$, and $(s, X) \in \mathsf{readies}(\overline{P})$, where $s = s_1 \ \mathsf{zip} \ldots \mathsf{zip}\ s_n$. We need to show that $(X_1 \times \cdots \times X_n) \cap X \neq \varnothing$.

The definition of $\overline{P}$ means that there is a trace t over A_{n+1} and a set Y such that $(s \ \mathsf{zip}\ t \ \mathsf{zip}\ t, Y) \in \mathsf{readies}(P)$, and $X = \{(x_1, \ldots, x_n) \mid \exists y.(x_1, \ldots, x_n, y, y) \in Y$. Because the output at port (A_{n+1}, out) of P is independent of the input at port (A_{n+1}, in), for any $x_1, \ldots, x_n$ there is b such that $(x_1, \ldots, x_n, y, z) \in Y \Rightarrow z = b$.

Let $X_{n+1} = \{b\}$ so that $(t, X_{n+1}) \in \mathsf{out}$, and let $X_{n+2} = \Sigma_{A_{n+1}}(t)$ so that $(t, X_{n+2}) \in \mathsf{in}$. Because $\mathsf{readies}(P) \subseteq \mathsf{in} \otimes \cdots \otimes \mathsf{in} \otimes \mathsf{out} \otimes \cdots \otimes \mathsf{out} \otimes \mathsf{in} \otimes \mathsf{out}$, there is $(a_1, \ldots, a_n, y, z) \in (X_1 \times \cdots \times X_{n+2}) \cap Y$.

We have $z = b$, dependent only on $a_1, \ldots, a_n$. Because $X_{n+1} = \{b\}$, $y = b$. So we have $(a_1, \ldots, a_n, b, b) \in Y$, and hence $(a_1, \ldots, a_n) \in X$. Therefore $(X_1 \times \cdots \times X_n) \cap X \neq \varnothing$, as required. □

We will use the term *source* to describe an output which is independent of any input which forms part of a cycle under consideration. In previous work [21] the term *source* has been used to describe an output which is independent of *all* inputs, but here we will use this weaker definition. The process P in Proposition 7.4 represents the network at the last stage of construction, just before formation of the cycle. In practice, and in line with the LUSTRE condition that every loop contains a pre node, we would like to deduce that the appropriate output of P is a source from the fact that one of the nodes used to construct P has a source. It can be shown, assuming that the outputs of nodes depend functionally on the inputs and that nodes are deterministic (these conditions are always satisfied for a language such as LUSTRE), that sources are preserved by composition [21]. Hence it is sufficient to check that there is a source somewhere in every cycle.

7.3 Generalisations

In our analysis of networks, we have simply identified each port as either an input or an output. However, we can imagine more general situations in which a particular port may behave in different ways at different times; for example, being receptive at the first step (and thus behaving as an input) but subsequently behaving as an output. In general, consider any finite sequence of in and out symbols, and interpret such a sequence as specifying the repeating unit of a communication pattern. For example, the sequence $\mathsf{in.out}$ represents an infinite alternation of input and output. The type system of $\mathcal{SP}roc_D$ is rich enough to include semantic versions of such communication patterns over any $\mathcal{SP}roc$ type. Continuing the example, the interpretation of the sequence $\mathsf{in.out}$ over the $\mathcal{SP}roc$ type A would be the ready specification

$$\{(s, \Sigma_A(s)) \mid \mathsf{length}(s) \text{ is even}\} \cup \{(s, X) \mid X \subseteq \Sigma_A(s), \mathsf{length}(s) \text{ is odd}\}.$$

A detailed development of this idea, which is a subject for future work, should lead to interesting connections with the type system proposed by Takeuchi *et al.* [46].

8 Conclusions

We have proposed a type-theoretic view of the specification and verification of concurrent systems. The relevant technical machinery is the notion of *specification structures*, which provides a systematic approach to the construction of a hierarchy of type systems expressing increasingly strong specifications. We have illustrated this idea by defining a specification structure over $\mathcal{SP}roc$, a category of synchronous processes. The resulting category, $\mathcal{SP}roc_D$, has a type system which permits compositional verification of deadlock-freedom. We have presented two equivalent definitions of $\mathcal{SP}roc_D$, one based on the idea of a type as a set of processes, the other based on the notion of ready specification. As a simple application, we have shown that $\mathcal{SP}roc_D$ supports the types necessary to specify and verify deadlock-freedom of synchronous networks; examples of synchronous networks include synchronous dataflow programs and systolic algorithms.

A number of type systems for concurrency have been proposed recently. Many of them are based on the idea of identifying ports or channels as input or outputs, and checking that outputs are always connected to inputs. There are several variations which include information about how many times channels are used [31], the order of usage of channels [30], subtyping [43], types for choice and branching behaviour [46]. The distinguishing features of our approach are as follows. First, it is based on interaction categories, a theory which emphasises the collective structure of processes and describes this structure using the language of category theory. Second, we have proposed a methodology (via the notion of a specification structure) for treating a range of program properties within a single framework, and combining type-checking with other verification techniques. Finally, we have taken a more semantic view of our type system. In our examples, the types assigned to individual ports correspond to either inputs or outputs, but semantically an arbitrary combination of such types can be treated on the same footing as any other type. This means that our arguments for correctness of networks, although intuitively based on considerations of input vs. output and information flow, are formalised within a uniform semantic setting.

There are many ways in which the theory described in this paper could be extended and developed. Progress has already been made on an asynchronous version of the theory, by applying the sets of processes approach to the asynchronous interaction category $\mathcal{ASP}roc$ [6]. The result is a category of deadlock-free processes in which the global synchrony condition is not present. Preliminary versions of this work have appeared in [2, 21] and an improved version in [40]; a full report of this area will be the subject of a future paper. Another avenue of investigation is the development of a formal syntax, incorporating value-passing rather than simply synchronisation, with which to describe processes in $\mathcal{SP}roc_D$. We have also mentioned, in Section 7, the possibility of giving a semantics to general communication patterns similar to those in the type system of Takeuchi *et al.* [46].

There are two major respects in which our deadlock-free type system is perhaps a little too restrictive. First, we have not yet addressed the issue of mobility [37, 38], which has featured prominently in recent research on concurrency theory. Second, the property guaranteed by type system is extremely strong—all processes must run forever. This is the reason why, in our applications, extra analysis is needed in order to construct cyclic networks. Most other proposed type systems for concurrency use types to guarantee slightly weaker properties—for example, that any communication which occurs must be correct, but not that communication must always continue. This problem is alleviated slightly by the asynchronous version of our theory, which incorporates a notion of successful termination, but we would like to find a modification of the theory which would make the type system weaker but correspondingly more flexible. Static analysis techniques, as well as type-checking techniques, may then be appropriate for establishing program properties.

Acknowledgements

This research was partly supported by the EPSRC projects "Foundational Structures in Computer Science", "Typed Concurrent Object-Oriented Languages: foundations, methods and tools" and the EU projects "CONFER" (ESPRIT BRA 6454) and "Coordination" (ESPRIT BRA 9102). The third author was also funded by the Ptolemy project, which is supported by the Defense Advanced Research Projects Agency (DARPA), the State of California MICRO program, and the following companies: The Alta Group of Cadence Design Systems, Dolby Laboratories, Hewlett Packard, Hitachi, Hughes Space and Communications, LG Electronics, Lockheed Martin ATL, NEC, Philips, and Rockwell. Paul Taylor's commutative diagrams package was used in the production of the 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.

[2] S. Abramsky, S. J. Gay, and R. Nagarajan. Specification structures and propositions-as-types for concurrency. In G. Birtwistle and F. Moller, editors, *Logics for Concurrency: Structure vs. Automata—Proceedings of the VIIIth Banff Higher Order Workshop*, volume 1043 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

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

[4] S. Abramsky. Computational Interpretations of Linear Logic. *Theoretical Computer Science*, 111:3–57, 1993.

[5] 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.

[6] 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.

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

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

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

[10] 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.

[11] 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.

[12] J. C. M. Baeten and W. P. Weijland. *Process Algebra*, volume 18 of *Tracts in Theoretical Computer Science*. Cambridge Univ. Press, 1990.

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

[14] 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.

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

[16] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *Journal of the ACM*, 31:560–599, 1984.

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

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

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

[20] M. A. Frumkin. *Systolic Computations*, volume 83 of *Mathematics and its Applications (Soviet Series)*. Kluwer Academic Publishers, 1992.

[21] S. J. Gay. *Linear Types for Communicating Processes*. PhD thesis, University of London, 1995.

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

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

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

[25] P. Guernic, T. Gautier, M. Borgne, and C. Maire. Programming real-time applications with SIGNAL. *Proceedings of the IEEE*, 79(9):1305–1320, September 1991.

[26] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous data flow programming language LUSTRE. *Proceedings of the IEEE*, 79(9):1305–1320, September 1991.

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

[28] 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.

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

[30] N. Kobayashi. A partially deadlock-free typed process calculus. Manuscript, 1996.

[31] N. Kobayashi, B. C. Pierce, and D. N. Turner. Linearity and the pi-calculus. In *Proceedings, 23rd ACM Symposium on Principles of Programming Languages*, 1996.

[32] 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.

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

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

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

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

[37] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I. *Information and Computation*, 100(1):1–40, September 1992.

[38] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, II. *Information and Computation*, 100(1):41–77, September 1992.

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

[40] R. Nagarajan. *Typed Concurrent Programs: Specification & Verification.* PhD thesis, University of London, 1997. To appear.

[41] B. Nordström, K. Petersson, and J. M. Smith. *Programming in Martin-Löf's Type Theory: An Introduction*, volume 7 of *International Series of Monographs on Computer Science*. Oxford University Press, 1990.

[42] 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.

[43] B. Pierce and D. Sangiorgi. Types and subtypes for mobile processes. In *Proceedings, Eighth Annual IEEE Symposium on Logic in Computer Science.* IEEE Computer Society Press, 1993.

[44] 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.

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

[46] K. Takeuchi, K. Honda, and M. Kubo. An interaction-based language and its typing system. In *Proceedings of the 6th European Conference on Parallel Languages and Architectures*, number 817 in Lecture Notes in Computer Science. Springer-Verlag, 1994.

[47] 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.