A Specification Structure for Deadlock-Freedom of Synchronous Processes

(to appear in *Theoretical Computer Science*)

S. Abramsky

Department of Computer Science, University of Edinburgh, James Clerk Maxwell Building, King's Buildings, Mayfield Road, Edinburgh EH9 3JZ, UK

S. J. Gay

Department of Computer Science, Royal Holloway, University of London, Egham, Surrey TW20 0EX, UK

R. Nagarajan

Department of Computing, Imperial College of Science, Technology and Medicine, 180 Queen's Gate, London SW7 2BZ, UK

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 $\mathcal{SP}roc_D$ in which morphisms are deadlock-free concurrent processes and composition is process interaction. $\mathcal{SP}roc_D$ is obtained from $\mathcal{SP}roc$ by adding specification information to the objects which is strong enough to guarantee deadlock-freedom. The main technical contribution is to show that this can be done in a way which is preserved by composition. The methods used to achieve this can be seen as a semantic analogue of those used to prove strong normalization in classical linear logic.

1 Introduction

In this paper we are concerned with a concrete instance of the following general situation. We have a semantic universe (category with structure) $\mathbb{C}_0$,

Preprint submitted to Elsevier Preprint 19 June 1998

suitable for modelling some computational situation, but possibly carrying only a very rudimentary notion of "type" or "behavioural specification". We then refine the objects of $\mathbb{C}_0$ in order to obtain a richer setting for performing specification and verification. 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 the objects contain specification information strong enough to guarantee deadlock-freedom and in which deadlock-freedom is closed under composition. We also define a specification structure R, based on a different approach to specifying deadlock-freedom. In Section 6 we prove that the two specification structures are equivalent, and that up to isomorphism we have constructed just one category of deadlock-free processes. 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 [26] and LUSTRE [27], and systolic algorithms [22]. Finally we compare our theory with other approaches, and discuss current limitations and possibilities for further developments. For a more general discussion of the methodological issues relating to specification structures, see [2].

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 [40], and Pitts [43].

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

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

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

and $\theta \in P_S C$*:*

$$\phi\{\mathsf{id}_A\}\phi \tag{1}$$
$$\phi\{f\}\psi, \psi\{g\}\theta \Longrightarrow \phi\{f\,;g\}\theta. \tag{2}$$

The axioms (1) and (2) are typed versions of the standard Hoare logic axioms for "skip" and "sequential composition" [21]. Given $\mathbb{C}$ and S as above, we can define a new category $\mathbb{C}_S$. An object of $\mathbb{C}_S$ is a pair (A, ϕ) with $A \in \mathsf{ob}\,\mathbb{C}$ and $\phi \in P_S A$. A $\mathbb{C}_S$-morphism $f : (A, \phi) \to (B, \psi)$ is a morphism $f : A \to B$ in $\mathbb{C}$ such that $\phi\{f\}\psi$.

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

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

given by

$$A \hookleftarrow (A, \phi).$$

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

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

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

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

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

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:

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

See [41] for a fuller discussion of how this idea relates to more general notions in category theory.

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

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

satisfying the standard coherence equations [34]. A specification structure for $\mathbb{C}$ must then correspondingly be extended with an action

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

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

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

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

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

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

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

$$\multimap_{A,B} : P_S A \times P_S B \to P_S(A \multimap B)$$

and axioms

$$(\phi \otimes \psi)\{\mathsf{symm}_{A,B}\}(\psi \otimes \phi)$$

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

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

Going one step further, suppose that $\mathbb{C}$ is a $*$-autonomous category, i.e. a model for the multiplicative fragment of classical linear logic [15], with linear negation $(-)^{\perp}$, where for simplicity we assume that $A^{\perp\perp} = A$. Then we require an action

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

satisfying

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

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

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

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

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

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

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

- models of classical (or intuitionistic) linear logic including the additives and exponentials [13]
- cartesian closed categories [20]
- models of polymorphism [20].

2.1 Examples of Specification Structures

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

(1) $\mathbb{C} = \mathcal{S}et$, $P_S X = X$, $a\{f\}b \stackrel{\text{def}}{\Leftrightarrow} f(a) = b$.
In this case, $\mathbb{C}_S$ is the category of pointed sets.

(2) $\mathbb{C} = \mathcal{R}el$, $P_S X = \{*\}$, $*\{R\}* \stackrel{\text{def}}{\Leftrightarrow} \forall x \in A, y, z \in B.\, xRy \wedge xRz \Rightarrow y = z$.
Then $\mathbb{C}_S$ is the category of sets and partial functions.

(3) $\mathbb{C} = \mathcal{R}el$, $P_S X = \wp X$, $S\{R\}T \stackrel{\text{def}}{\Leftrightarrow} \forall x \in S.\, \{y \mid xRy\} \subseteq T$.
This is essentially a typed version of dynamic logic [33], 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 $\mathbb{C}_S$ becomes a model of classical linear logic.

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

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

$\mathbb{C}_S$ is the category of coherence spaces and linear maps [25].

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

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

where ϕ_n is the nth partial recursive function in some acceptable numbering [45]. Then $\mathbb{C}_S$ is the category of modest sets, seen as a full subcategory of ω-$\mathcal{S}et$ [13].

(6) $\mathbb{C}$ = the category of SFP domains,
$P_S D = K\Omega(D)$(the compact-open subsets of D),
$U\{f\}V \stackrel{\text{def}}{\Leftrightarrow} U \subseteq f^{-1}(V)$.
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) $\mathbb{C}$ = games and partial strategies, as in [12], $P_S A$ = all sets of infinite plays, $U\{\sigma\}V$ iff σ is *winning* with respect to U, V in the sense of [10]. Then $\mathbb{C}_S$ is the category of games and winning strategies of [10].

3 The Interaction Category $\mathcal{SProc}$

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 [16], categories of games [10], geometry of interaction categories [11]) as well as several new categories for concurrency. The fundamental examples of concurrent interaction categories are $\mathcal{SProc}$ [5], the category of synchronous processes, and $\mathcal{ASProc}$ [6], the category of asynchronous processes. The category $\mathcal{SProc}$ will be defined in this section; later we will use a specification structure over $\mathcal{SProc}$ 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 [23]. The common features of the existing examples are that they have $*$-autonomous structure, which corresponds to the multiplicative fragment of classical linear logic [25]; products and coproducts, corresponding to the additives of linear logic, and additional temporal structure which enables the dynamics of process evolution to be described.

In this section we briefly review the definition of $\mathcal{SProc}$, the category of synchronous processes. Because the present paper mainly concerns the use of specification structures for deadlock-freedom, we omit the features of $\mathcal{SProc}$ which will not be needed in later sections. More complete definitions can be found elsewhere [1,23].

3.1 Processes

A *labelled transition system* (LTS) [36] is a triple $(S, L, \longrightarrow)$ where S is a set of states, L is a set of labels, and $\longrightarrow\ \subseteq S \times L \times S$ is the transition relation. We write $s \xrightarrow{a} s'$ for $(s, a, s') \in\ \longrightarrow$.

A *process representative* with *sort* or *alphabet* L is a distinguished state of an LTS whose set of labels is L. We will usually define process representatives by giving a list of transitions (which define an LTS if the set of states is taken to consist of all the states appearing in the transitions) and indicating a distinguished state. It is also possible to present a process representative diagrammatically, as a distingushed node of a directed graph with labelled edges, in which case the edges of the graph correspond to transitions.

Given an LTS, strong bisimulation [36] is defined as a relation on the set of states. Because LTSs can be combined by disjoint union, we can consider strong bisimulation to be a relation on the set of all processes with a given sort. All of these definitions are standard; see [36] for a full discussion of the theory of strong bisimulation. We will work with equivalence classes of process representatives modulo strong bisimulation, and use the term *process* to refer to an equivalence class. We will generally write "=" for process equivalence, but may sometimes use "$\sim$" to emphasise that the equivalence is strong bisimulation. We write $\mathsf{Proc}(L)$ for the set of processes with sort L.

3.2 Traces

A trace over a set L is a finite sequence of elements of L. We write L^* for the Kleene closure of L (the set of all traces over L). We write L^ω for the set of infinite sequences of elements of L. We denote the empty trace by ε, and the concatenation of traces s and t by st. We do not distinguish notationally between the element a and the trace consisting of just a.

In an LTS $(S, L, \longrightarrow)$ we introduce the notation $s \xrightarrow{t}{}^* s'$. If $t = a_1 a_2 \dots a_n$ then $s \xrightarrow{t}{}^* s'$ means $\exists s_0 = s, s_1, \dots, s_n = s' \in S. \forall i \in \{1, \dots, n\}. s_{i-1} \xrightarrow{a_i} s_i$.

The transition $s \xrightarrow{\varepsilon}{}^* s'$ exists if and only if $s = s'$.

If P is a process with sort L, then $\mathsf{alltraces}(P) \subseteq ({}^*L) \cup L^\omega$ and $\mathsf{traces}(P) \subseteq ({}^*L)$ are defined coinductively by

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

If P is thought of as a distinguished node in a labelled directed graph, then $\mathsf{traces}(P)$ is the set of sequences labelling finite paths from the distinguished node.

Given any set L, the process nil_L has sort L and no transitions (it is the unique state in the LTS $(\{\mathsf{nil}_L\}, L, \varnothing)$). We will abbreviate nil_L by nil if the sort is clear from the context.

3.3 The Category

An object of $\mathcal{SP}roc$ is a pair $A = (\Sigma_A, S_A)$ in which Σ_A is a set of *actions* and $S_A \subseteq^{nepref} \Sigma_A^*$ is a *safety specification*, i.e. a non-empty prefix-closed subset of Σ_A^*. We often refer to Σ_A as the *sort* or *alphabet* of 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 $\mathsf{traces}(P) \subseteq S_A$. There is always at least one process of type A, namely nil_{Σ_A}, which may also be written nil_A or simply nil. 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 [10] follow the same pattern. Given objects A and B, the object $A \otimes B$ has

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

For any sets X, Y and function $f : X \to Y$, $f^* : X^* \to Y^*$ is the trace extension of f.

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. Later, however, we will construct a category in which this distinction is present.

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

For simplicity, we shall work with $*$-autonomous categories in which $A^{\perp\perp} = A$, and $A \multimap B \stackrel{\text{def}}{=} (A \otimes B^\perp)^\perp$, $A \mathbin{\wp} B \stackrel{\text{def}}{=} (A^\perp \otimes B^\perp)^\perp$. In $\mathcal{SP}roc$, we have $A = A^\perp$, and hence $A \mathbin{\wp} B = A \multimap B = A \otimes B$. 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 5, gives distinct interpretations to $\otimes$ and $\wp$.

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 $A \otimes B$.

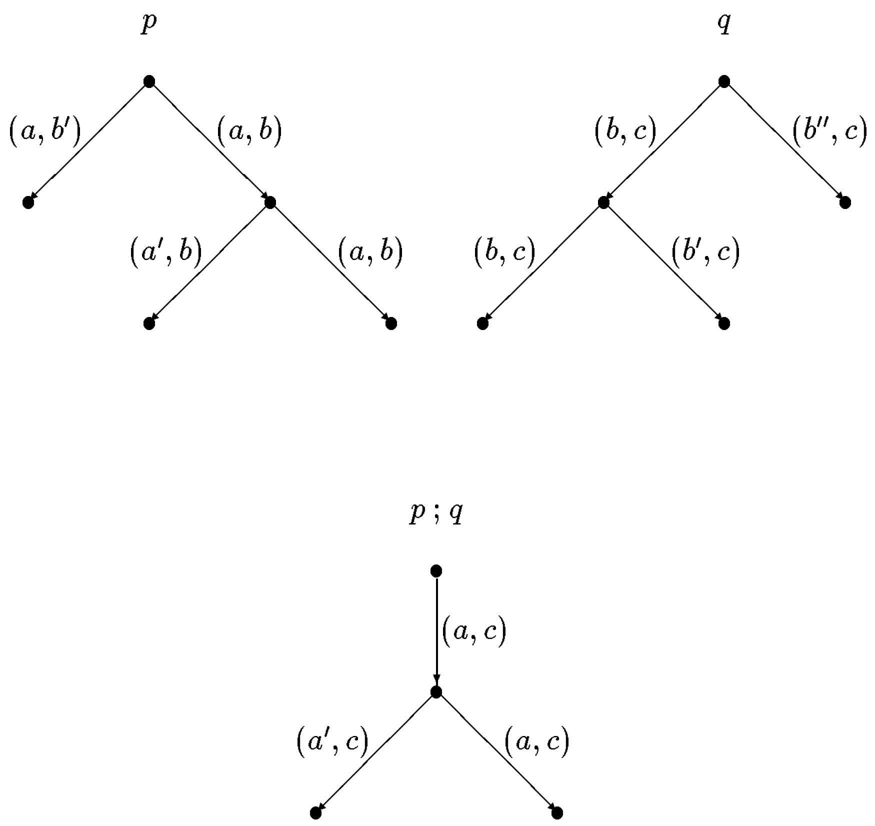

Fig. 1. Composition in $\mathcal{SP}roc$

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, in the style of Plotkin's "structural operational semantics" [44].

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

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.*

We have to prove that composition is well-defined with respect to strong bisimulation, and that a composite process satisfies the appropriate safety specification.

Proposition 2 *If $p, p' : A \to B$ and $q, q' : B \to C$ with $p \sim p'$ and $q \sim q'$ then $p \,;\, q \sim p' \,;\, q'$.*

PROOF. It is straightforward to show that the relation

$$\{(p \,;\, q, p' \,;\, q') \mid p, p' \in \mathsf{Proc}(\Sigma_{A \multimap B}), q, q' \in \mathsf{Proc}(\Sigma_{B \multimap C}), p \sim p', q \sim q'\}$$

is a strong bisimulation. For more details of proof techniques for strong bisimulation, see [36]. $\square$

Proposition 3 *If $\mathsf{traces}(p) \subseteq S_{A \multimap B}$ and $\mathsf{traces}(q) \subseteq S_{B \multimap C}$ then $\mathsf{traces}(p;q) \subseteq S_{A \multimap C}$.*

PROOF. Suppose $s \in \mathsf{traces}(p \,;\, q)$. Then there are $t \in \mathsf{traces}(p)$ and $u \in \mathsf{traces}(q)$ such that $\mathsf{fst}^*(s) = \mathsf{fst}^*(t)$, $\mathsf{snd}^*(s) = \mathsf{snd}^*(u)$, and $\mathsf{snd}^*(t) = \mathsf{fst}^*(u)$. From the definitions of $S_{A \multimap B}$ and $S_{B \multimap C}$, $\mathsf{fst}^*(t) \in S_A$ and $\mathsf{snd}^*(u) \in S_C$. Hence $s \in S_{A \multimap C}$. $\square$

The following definitions will be useful in the rest of this section and later.

Definition 4 *If A is an object of $\mathcal{SP}roc$ and $s \in S_A$, then*

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

If P is a process with sort Σ and $S \subseteq^{nepref} \Sigma^*$ then the process $P \upharpoonright S$, also with sort Σ, is defined by the transition rule

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

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.

If A is an object of $\mathcal{SP}roc$ and $s \in S_A$ then the object A/s is defined by:

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

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 identity morphism $\mathsf{id}_A : A \to A$ is defined by $\mathsf{id}_A \stackrel{\text{def}}{=} \mathsf{id} \lceil S_{A \multimap A}$ 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 5 *$\mathcal{SP}roc$ 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 strong bisimulation. 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

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

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

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

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. □

3.4 $\mathcal{SP}roc$ 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.

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

As with composition, it is straightforward to check that $\otimes$ respects strong bisimulation and that $p \otimes q$ satisfies the required safety specification.

The tensor unit I is defined by

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

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.

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

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.

Proposition 6 *$\mathcal{SP}roc$ 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 $⅋$. $\square$

The following result on relabellings will be useful later.

Lemma 7 *If $p \in \mathsf{Proc}(\Sigma_{A \multimap B})$, $q \in \mathsf{Proc}(\Sigma_{B \multimap C})$ and $f : \Sigma_B \to \Sigma_D$ is a bijection, 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

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

is a bisimulation. $\square$

3.5 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*. The usual Cut rule

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

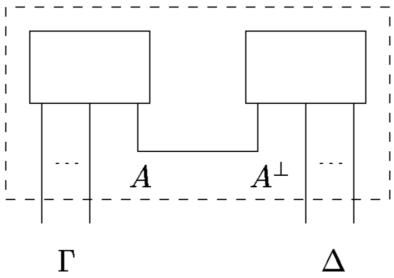

Fig. 2. Using the Cut rule to connect modules

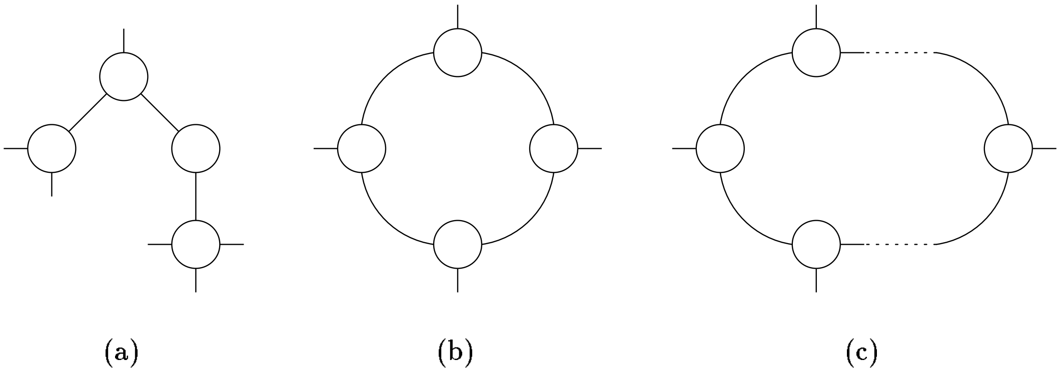

Fig. 3. Cyclic and acyclic networks

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 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 $\mathcal{SProc}$) as the following construction shows.

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

$$\begin{tikzcd}
I \arrow[r, "\sim"] \arrow[dd, "h"'] & I \otimes I \arrow[r, "f \otimes g"] & (\tilde{A} \otimes \tilde{C}) \otimes (\tilde{B} \otimes \tilde{C}^{\perp}) \arrow[d, "\sim"] \\
 & & \tilde{A} \otimes (C_1 \otimes C_1^{\perp}) \otimes \cdots \otimes (C_k \otimes C_k^{\perp}) \otimes \tilde{B} \arrow[d, "\mathsf{evaluate}"] \\
\tilde{A} \otimes \tilde{B} & & \tilde{A} \otimes I \otimes \cdots \otimes I \otimes \tilde{B} \arrow[ll, "\sim"']
\end{tikzcd}$$

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 **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 [17,30].

The above use of compact closed structure to interpret cyclic networks goes back to [5,6]. In recent work, Joyal, Street and Verity [29] have axiomatised feedback in monoidal categories (in their terminology a *trace*) and observed that every compact closed category is traced. This provides an appropriate general setting for the above calculation. For a discussion of connections between traced monoidal categories and computational structures, see [8].

It is useful to introduce some notation for the operation of cycle formation. If $P \in \mathsf{Proc}(\Sigma_{A_1 \otimes \cdots \otimes A_n \otimes B \otimes B^{\perp}})$ then $\overline{P} \in \mathsf{Proc}(\Sigma_{A_1 \otimes \cdots \otimes A_n})$ is defined by the transition rule

$$\frac{P \xrightarrow{(a_1,\ldots,a_n,b,b)} Q}{\overline{P} \xrightarrow{(a_1,\ldots,a_n)} \overline{Q}.}$$

If $P : A_1 \otimes \cdots \otimes A_n \otimes B \otimes B^\perp$ then $\overline{P} : A_1 \otimes \cdots \otimes A_n$ (it is straightforward to check that the necessary safety specification is satisfied).

3.6 Products, Coproducts and Non-determinism

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

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

In the following definitions we use the operation $+$ on processes for non-deterministic sum of labelled transition systems (the standard $+$ operation of CCS [36]). If $p : A \to C$ and $q : B \to D$ then $p \oplus q : A \oplus B \to C \oplus D$ is defined by

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

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

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

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

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

Proposition 8 *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 7 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

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

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 $[p, q]$. $\square$

Since $\oplus$ is a coproduct, its dual is a product; because all objects of $\mathcal{SProc}$ 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 [25]. In $\mathcal{SProc}$, $A \,\&\, B$ 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 $\mathcal{SProc}$ 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 9 *The object* $\mathbf{0}$ *is initial and terminal in* $\mathcal{SProc}$.

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 $\mathbf{0} \to A$. $\square$

When a category has biproducts and a zero object, it is possible to define a commutative monoid structure on each homset [34]. 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 $\mathcal{SProc}$, this construction yields the non-deterministic sum of CCS (when strong bisimulation is taken as the notion of equivalence). The proof of Proposition 9 shows that the unique morphisms into and out of $\mathbf{0}$ are nil processes, and so $0_{A \to B}$ 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.

3.7 Time

So far, all of the constructions in $\mathcal{SP}roc$ have been essentially 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

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

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 10 *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 11 *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*

$$\begin{tikzcd} A \arrow[r, "f"] \arrow[d, "It(f,g)"'] & \circ A \arrow[d, "\circ It(f,g)"] \\ B & \circ B \arrow[l, "g"] \end{tikzcd}$$

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]. $\square$

We will not go into the applications of this property in the present paper, except to mention that it supports guarded recursive definitions [1,23] and is an important part of a proposed axiomatisation of interaction categories [23]. The notation $It(f,g)$ is intended to suggest *iteration.*

4 Specification Structures over Interaction Categories

4.1 The Sequence of Definitions

Suppose $\mathbb{C}$ is a $*$-autonomous category with a notion of a set of elements of each type, written $\mathsf{Elem}(A)$ (in $\mathcal{SP}roc$ an element of type A is a process P of type A, which 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*: $\models_A \subseteq \mathsf{Elem}(A) \times P_S A$.

(3) Define $(-)^{\perp}_{A}$.
(4) Define $\otimes_{A,B}$ and hence $\mathbin{⅋}_{A,B}$ and $\multimap_{A,B}$.
(5) Define the Hoare triple relation by $\theta\{f\}\phi \overset{\text{def}}{\Leftrightarrow} f \models_{A \multimap B} \theta \multimap \phi$.
(6) Verify that the desired structure of $\mathbb{C}$, including the $*$-autonomous structure, lifts to $\mathbb{C}_S$.

For reference, we will now list the definitions and conditions which are needed to lift the relevant structure of $\mathbb{C}$ to $\mathbb{C}_S$. 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 $\otimes : \mathbb{C}_S \times \mathbb{C}_S \to \mathbb{C}_S$ we need, for every $\theta, \phi, \psi, \rho \in P_S A, P_S B, P_S C, P_S D$, $f : A \to C$ and $g : B \to D$,

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

Then

$$(A, \theta) \otimes (B, \phi) \overset{\text{def}}{=} (A \otimes B, \theta \otimes_{A,B} \phi).$$

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

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

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

For each object A we need an operation

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

and we can then define

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

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

$$\theta\{f\}\phi \Rightarrow \phi_B^{\perp}\{f^{\perp}\}\theta_A^{\perp}.$$

The operations

$$\begin{aligned} \mathbin{\rotatebox[origin=c]{180}{$\&$}}_{A,B} &: P_S A \times P_S B \to P_S(A \mathbin{\rotatebox[origin=c]{180}{$\&$}} B) \\ \multimap_{A,B} &: P_S A \times P_S B \to P_S(A \multimap B) \end{aligned}$$

can be defined by

$$\theta \mathbin{\rotatebox[origin=c]{180}{$\&$}}_{A,B} \phi \stackrel{\text{def}}{=} (\theta_A^{\perp} \otimes_{A^{\perp},B^{\perp}} \phi_B^{\perp})^{\perp}_{A^{\perp} \otimes B^{\perp}}$$
$$\theta \multimap_{A,B} \phi \stackrel{\text{def}}{=} (\theta \otimes_{A,B^{\perp}} \phi_B^{\perp})^{\perp}_{A \otimes B^{\perp}}$$

and the property $\perp_S$ by

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

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

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

and

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

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 $\mathbb{C}_S$ to be defined on objects by

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

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

$$\theta\{f\}\psi, \phi\{g\}\rho \Rightarrow (\theta \mathbin{\&} \phi)\{f \mathbin{\&} g\}(\psi \mathbin{\&} \rho).$$

Additionally, we need

$$(\theta \mathbin{\&} \phi)\{\pi_A\}\theta$$
$$(\theta \mathbin{\&} \phi)\{\pi_B\}\phi$$

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

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

The treatment of coproducts is dual.

4.4 Unit Delay

For each object A we need an operation

$$\circ_A \colon P_S A \to P_S(\circ A)$$

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

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

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

$$\theta\{f\}\phi \Rightarrow \circ\theta\{\circ f\}\circ\phi.$$

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

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

A major example of a specification structure over $\mathcal{SProc}$ is the subject of the rest of this paper. For illustrative purposes, we now give two examples of constructions which are *not* specification structures.

(1) Consider a variation on $\mathcal{SProc}$ in which objects are simply alphabets and there is no safety requirement on morphisms. Formally, consider the full subcategory of $\mathcal{SProc}$ consisting of all objects of the form $A = (\Sigma_A, \Sigma_A^*)$. Call this subcategory $\mathcal{USProc}$ ("unsafe $\mathcal{SProc}$").

It might appear that we can define a specification structure S on $\mathcal{USProc}$ by

$$\begin{array}{rcl} P_S(A) & = & \{X \mid X \subseteq^{nepref} \Sigma_A^*\} \\ p \models X & \Longleftrightarrow & \mathsf{traces}(p) \subseteq X \end{array}$$

or equivalently, in terms of the Hoare triple relation,

$$\begin{array}{rcl} X\{f\}Y & \Longleftrightarrow & \{\mathsf{fst}^*(s) \mid s \in \mathsf{traces}(f)\} \subseteq X \\ & \wedge & \{\mathsf{snd}^*(s) \mid s \in \mathsf{traces}(f)\} \subseteq Y. \end{array}$$

The intention would be that $\mathcal{USProc}_S$ is equivalent to $\mathcal{SProc}$, in other words that $\mathcal{SProc}$ could be defined by first defining $\mathcal{USProc}$ and then adding the safety specifications by means of a specification structure.

However, S does not satisfy the specification structure axioms: if $X \subset \Sigma_A^*$ then we do not have $X\{\mathsf{id}_A\}X$, because $\{\mathsf{fst}^*(s) \mid s \in \mathsf{traces}(\mathsf{id}_A)\} = \Sigma_A^*$.

(2) Given any set Σ_U, consider the full subcategory of $\mathcal{SProc}$ with just the object $U = (\Sigma_U, \Sigma_U^*)$. Call this category $\mathcal{TSProc}$, ("typeless $\mathcal{SProc}$"). We might attempt to define a specification structure S over $\mathcal{TSProc}$, with the intention that $\mathcal{TSProc}_S$ is equivalent to $\mathcal{USProc}$ as in the previous example:

$$\begin{array}{rcl} P_S(U) & = & \wp\Sigma_U \\ p \models X & \Longleftrightarrow & \mathsf{traces}(p) \subseteq X^* \end{array}$$

Again, we could instead define the Hoare triple relation:

$$\begin{array}{rcl} X\{f\}Y & \Longleftrightarrow & \{\mathsf{fst}^*(s) \mid s \in \mathsf{traces}(f)\} \subseteq X^* \\ & \wedge & \{\mathsf{snd}^*(s) \mid s \in \mathsf{traces}(f)\} \subseteq Y^*. \end{array}$$

As before, the specification structure axioms are not satisfied, because the identity morphism in $\mathcal{TSProc}$ does not lift to an identity morphism on an object (U, X) of $\mathcal{TSProc}_S$. If $X \subset \Sigma_U$ then we do not have $X\{\mathsf{id}_U\}X$ because there are traces s of id_U which contain actions in $\Sigma_U - X$.

These examples indicate that an alphabet and a safety specification constitute the minimum type information which can be used as the basis for an interaction category.

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. We will build suitable additional information into the objects by constructing a specification structure over $\mathcal{SP}roc$. In fact we will construct two specification structures, based on different approaches to specifying deadlock-freedom. It turns out, however, that the two specification structures are equivalent in a strong sense, and lead to isomorphic categories. Each approach has its advantages: the *ready specification* approach is easier to motivate, while the *sets of processes* approach appears more general. In both cases, there is a strong analogy between the treatment of deadlock-freedom and proofs of strong normalisation in Classical Linear Logic [4,25].

In Section 5.1 we discuss ready specifications and define the specification structure R. However, we do not immediately prove that R *is* a specification structure. In Section 5.4 we define a specification structure D, based on sets of processes, such that $\mathcal{SP}roc_D$ is the desired category of deadlock-free processes; this section contains most of the proofs necessary to establish that the structure of $\mathcal{SP}roc$ lifts to $\mathcal{SP}roc_D$. In Section 6 we prove that D and R are equivalent. This equivalence enables us to deduce that R is a specification structure, and to fill in the proofs omitted from Section 5.4. It turns out that although the proofs of the basic specification structure properties are most easily carried out in D, the proofs that the product and coproduct structure lift are more easily expressed in the language of ready specifications. Finally we prove that the categories $\mathcal{SP}roc_D$ and $\mathcal{SP}roc_R$ are isomorphic.

5.1 Ready Specifications

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 12 *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*

$$\mathsf{initials}(P) \stackrel{\text{def}}{=} \{x \in \Sigma_A \mid \exists Q. P \xrightarrow{x} Q\}$$

$$\mathsf{readies}(P) \stackrel{\text{def}}{=} \{(s, X) \mid (P \xrightarrow{s}{}^{*} Q) \wedge (X = \mathsf{initials}(Q))\}.$$

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 13 *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 $(s, \varnothing) \in \mathsf{readies}(P)$. Again equivalently, if and only if whenever $P \xrightarrow{s}{}^{*} Q$ there is $a \in \Sigma_A$ and a process R such that $Q \xrightarrow{a} R$. We use the notation $P{\downarrow}$ to indicate that P is deadlock-free.*

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 [14,18,28]. There, however, they are used to define semantic alternatives to bisimulation; the use made of ready pairs in this paper is very different.

The key definition is that of *orthogonality* of ready pairs.

Definition 14 *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, then $(s, X) \perp (t, Y)$ 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. If there are two ports, in which the respective sets of actions X and Y can occur, then connecting them together only results in 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 15 *Let* $\theta \subseteq \mathsf{RP}(A)$ *for some object* A*.*

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

Defining $(-)^{\perp}$ in this way from a symmetric orthogonality relation yields a self-adjoint Galois connection [19] and the following lemma holds for general reasons.

Lemma 16 *For* $\theta, \phi \subseteq \mathsf{RP}(A)$*,*

$$\begin{aligned}&1.\ \theta \subseteq \phi \Rightarrow \phi^{\perp} \subseteq \theta^{\perp} \\ &2.\ \qquad \theta \subseteq \theta^{\perp\perp} \\ &3.\ \ \theta^{\perp\perp\perp} = \theta^{\perp}.\end{aligned}$$

Definition 17 *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* $\mathsf{RS}(A)$*.*

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

Definition 18 *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 $\mathcal{SP}roc_{pr}$.

Proposition 19 *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 17. 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 $x \in \{x\}$. $\square$

Corollary 20 *If A is progressive then* $\mathsf{RS}(A) \neq \varnothing$.

Proposition 21 *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 Lemma 16.
(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. $\square$

Corollary 22 *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 $\{(s, \Sigma_A(s)) \mid s \in S_A\} = \mathsf{RP}(A)^{\perp} \subseteq \theta^{\perp}$. $\square$

5.2 The Specification Structure R

Following the sequence of definitions in Section 4.1, we can define the specification structure R over $\mathcal{SP}roc_{pr}$.

Definition 23

(1) $P_R A \stackrel{\text{def}}{=} \{\theta \in \mathsf{RS}(A) \mid \theta^{\perp\perp} = \theta\}$.
(2) $P \models \theta \stackrel{\text{def}}{\Leftrightarrow} \mathsf{readies}(P) \subseteq \theta$.
(3) $(-)^{\perp} : P_R A \to P_R A$ *has already been defined.*

(4)

$$\begin{aligned}
\theta \mathbin{\mathcal{V}} \phi &\stackrel{\text{def}}{=} \{(s, U \times V) \mid (\mathsf{fst}^*(s), U) \in \theta^\perp, (\mathsf{snd}^*(s), V) \in \phi^\perp\}^\perp \\
\theta \otimes \phi &\stackrel{\text{def}}{=} (\theta^\perp \mathbin{\mathcal{V}} \phi^\perp)^\perp \\
\theta \multimap \phi &\stackrel{\text{def}}{=} \theta^\perp \mathbin{\mathcal{V}} \phi \\
I_R &\stackrel{\text{def}}{=} \{(s, \{*\}) \mid s \in S_I\}.
\end{aligned}$$

(5) $\theta\{f\}\phi \stackrel{\text{def}}{=} f \models \theta \multimap \phi$.

(6) The definitions required to lift the additive and temporal structure of $\mathcal{SP}roc$ to $\mathcal{SP}roc_R$ are

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

Proposition 24 *If $\theta \in P_R A$ and $P \models \theta$ then $P\!\downarrow$.*

PROOF. Follows from the fact that θ contains only proper ready pairs. $\Box$

Proposition 25 *If $\theta \in P_R A$ then $\theta \supseteq \{(s, \Sigma_A(s)) \mid s \in S_A\}$.*

PROOF. Follows from Corollary 22, because $\theta = (\theta^\perp)^\perp$. $\Box$

As mentioned earlier, we do not yet prove that R satisfies the specification structure axioms.

5.3 Example

To illustrate the way in which ready specifications rule out deadlocking compositions, it is convenient to restrict attention to the first step of process behaviour. Then a process $p : A \to B$ is simply a subset of $\Sigma_A \times \Sigma_B$ (i.e. a relation between A and B), ready pairs are replaced by ready sets (the traces are discarded, and a ready specification is a set of ready sets), and satisfaction becomes $p \models \theta \iff p \subseteq \theta$.

Orthogonality is defined on ready sets by $X \perp Y \iff X \cap Y \neq \varnothing$, and is lifted to ready specifications as before. The definitions of $\mathbin{\mathcal{V}}$ on ready specifications, and I_R, become

$$\begin{aligned}\theta \mathbin{⅋} \phi &= \{U \times V \mid U \in \theta^\perp, V \in \phi^\perp\}^\perp \\ I_R &= \{\{*\}\}\end{aligned}$$

and as before, $I_R^\perp = I_R$. Composition of processes is relational composition, and deadlock-freedom is non-emptiness.

Consider the object A defined by $\Sigma_A = \{a, b\}$ (safety specifications are now irrelevant). There are four ready specifications over A:

$$\begin{aligned}\theta &= \{\{a, b\}\} \\ \phi &= \{\{a, b\}, \{a\}\} \\ \psi &= \{\{a, b\}, \{b\}\} \\ \theta^\perp &= \{\{a, b\}, \{a\}, \{b\}\}.\end{aligned}$$

It is easy to check that these are all ready specifications, i.e. are $^{\perp\perp}$-invariant, and that there are no other possibilities. Note that $\phi^\perp = \phi$ and $\psi^\perp = \psi$.

Define $p : I \to A$ and $q : A \to I$ by $p = \{(*, a)\}$ and $q = \{(b, *)\}$. The processes p and q are composable in $\mathcal{SProc}_R$ if and only if there is $\alpha \in P_R A$ such that $p : (I, I_R) \to (A, \alpha)$ and $q : (A, \alpha) \to (I, I_R)$, i.e. $p \models I_R \multimap \alpha$ and $q \models \alpha \multimap I_R$. Since $p \,;\, q = \varnothing$, we expect that there should not be such an α. The small number of possibilities for α make it straightforward to calculate $I_R \multimap \alpha$ and $\alpha \multimap I_R$ in each case, and check satisfaction directly.

$$\begin{aligned}I_R \multimap \theta &= I_R^\perp \mathbin{⅋} \theta \\ &= I_R \mathbin{⅋} \theta \\ &= \{U \times V \mid U \in I_R^\perp, V \in \theta^\perp\}^\perp \\ &= \{\{*\} \times V \mid V \in \theta^\perp\}^\perp \\ &= \{(*, a), (*, b)\}, \{(*, a)\}, \{(*, b)\}\}^\perp \\ &= \{\{(*, a), (*, b)\}\}\end{aligned}$$

As expected from the $*$-autonomous structure, $I_R \multimap \theta$ is isomorphic to θ in a straightforward way; similar calculations yield:

$$\begin{aligned}I_R \multimap \phi &= \{\{(*, a), (*, b)\}, \{(*, a)\}\} \\ I_R \multimap \psi &= \{\{(*, a), (*, b)\}, \{(*, b)\}\} \\ I_R \multimap \theta^\perp &= \{\{(*, a), (*, b)\}, \{(*, a)\}, \{(*, b)\}\}.\end{aligned}$$

Calculating $\alpha \multimap I_R$ in each case is similar, and as expected the result is isomorphic to $\alpha^\perp$.

$$\theta \multimap I_R = \{\{(a, *), (b, *)\}, \{(a, *)\}, \{(b, *)\}\}$$

$$\begin{aligned}
\phi \multimap I_R &= \{\{(a,*),(b,*)\},\{(a,*)\}\} \\
\psi \multimap I_R &= \{\{(a,*),(b,*)\},\{(b,*)\}\} \\
\theta^\perp \multimap I_R &= \{\{(a,*),(b,*)\}\}
\end{aligned}$$

We can now see that the following relationships hold:

$$\begin{array}{ll}
p \models I_R \multimap \phi & \qquad q \models \theta \multimap I_R \\
p \models I_R \multimap \theta^\perp & \qquad q \models \psi \multimap I_R
\end{array}$$

but there is no choice of ready specification over A which allows p and q to be composed.

The same ready specifications can be used to illustrate various cases in which processes can be composed: for example, if $r : I \to A$ and $s : A \to I$ are defined by $r = \{(*,a),(*,b)\}$ and $s = \{(a,*)\}$ then we can achieve $r : (I, I_R) \to (A, \alpha)$ and $s : (A, \alpha) \to (I, I_R)$ by taking α to be θ, ϕ or $\theta^\perp$.

5.4 Sets of Processes

In this section we use the same notion of deadlock-freedom, but take a different approach to constructing the properties required for a specification structure.

Definition 26 *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*

$$P \perp Q \overset{\text{def}}{\Leftrightarrow} (P \sqcap Q)\Downarrow .$$

Intuitively, $P \perp Q$ means that if P and Q are run in synchronous lockstep then they do not deadlock each other. As an example of how this is used, consider $R : A \to B$ and $S : B \to C$. 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 $R\,;S$.

Definition 27 *For each object A of $\mathcal{SP}roc_{pr}$, $\mathsf{Proc}(A)$ is the set of deadlock-free processes of type A. The orthogonality relation is extended to sets of processes by defining, for $U, V \subseteq \mathsf{Proc}(A)$ and $P : A$,*

$$P \perp U \overset{\text{def}}{\Leftrightarrow} \forall Q \in U. P \perp Q$$
$$U \perp V \overset{\text{def}}{\Leftrightarrow} \forall P \in U. P \perp V.$$

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

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

For this new notion of orthogonality, we have the same result as Lemma 16.

Lemma 28 *For all $U, V \subseteq \mathsf{Proc}(A)$,*

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

We will also need some additional results.

Definition 29 *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 30 *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 $P\downarrow$. $\square$

Lemma 31 *If $U \subseteq \mathsf{Proc}(A)$ then $U^{\perp} \neq \varnothing$.*

PROOF. Lemma 30 implies that $\mathsf{max}_A \in U^{\perp}$. $\square$

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

Definition 32

(1) $P_D A \stackrel{\text{def}}{=} \{U^\perp \mid U \subseteq \mathsf{Proc}(A)\}$
(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)

$$\begin{aligned} U \otimes V &\stackrel{\text{def}}{=} \{P \otimes Q \mid P \in U, Q \in V\}^{\perp\perp} \\ U \mathbin{\wp} V &\stackrel{\text{def}}{=} (U^\perp \otimes V^\perp)^\perp \\ U \multimap V &\stackrel{\text{def}}{=} (U \otimes V^\perp)^\perp \\ I_D &\stackrel{\text{def}}{=} \{\mathsf{max}_I\}. \end{aligned}$$

(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*

$$\begin{aligned} U \oplus V &\stackrel{\text{def}}{=} (\{P[\mathsf{inl}] \mid P \in U\} \cup \{Q[\mathsf{inr}] \mid Q \in V\})^{\perp\perp} \\ U \mathbin{\&} V &\stackrel{\text{def}}{=} (U^\perp \oplus V^\perp)^\perp \\ \circ U &\stackrel{\text{def}}{=} \{\circ P \mid P \in U\}. \end{aligned}$$

The following points are worth noting.

- Lemma 31 and Lemma 28 ensure that for each $U \in P_D(A)$, $U \neq \varnothing$ and $U^{\perp\perp} = U$.
- Lemma 31 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 32 is ${}^{\perp\perp}$-invariant. In every case except that of $\circ$, this follows from the fact that for every $U \subseteq \mathsf{Proc}(A)$, $U^\perp$ is ${}^{\perp\perp}$-invariant (Lemma 28).

Lemma 33 *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

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

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

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

Lemma 34 *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 $\mathsf{id}_A \perp P \otimes Q$. $\square$

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 35 *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. $\square$

Lemma 36 *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 35. $\square$

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 $\mathsf{snd}^*(u) = t$.

Theorem 37 *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 34.

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 $f\,;g \perp \{P \otimes Q \mid P \in U, R \in W^\perp\}$.

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 35 and 36, $P\,;f \in V$ 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. $\square$

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 38 *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. $\square$

We also need to check that the products and coproducts lift to $\mathcal{SP}roc_D$. It turns out that the necessary proofs are more easily formulated in the language of ready specifications, and they will be postponed until Section 6, when ready specifications have been shown to be equivalent to sets of processes.

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

Proposition 39 *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

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

(Base case) $s = \varepsilon$. 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 35 and 36 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 $P' \otimes Q' \xrightarrow{(a,b)} P'' \otimes Q''$. $\square$

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

Proposition 40 *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 $P \sqcap Q$ is not deadlock-free, so $P \not\perp \mathsf{Proc}(A)$. $\square$

Corollary 41 $\{\mathsf{max}_A\}^{\perp\perp} = \{\mathsf{max}_A\}$ *and* $\mathsf{Proc}(A)^{\perp\perp} = \mathsf{Proc}(A)$.

Definition 42 *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 43 *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 44 $I_D = \mathsf{in}_I = \mathsf{out}_I = I_D^{\perp}$.

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

Proposition 45 *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. $\mathsf{out}_A \,⅋\, \mathsf{out}_B = \mathsf{out}_{A ⅋ B}$
3. $\mathsf{out}_A \otimes \mathsf{out}_B = \mathsf{out}_{A \otimes B}$
4. $\mathsf{in}_A \,⅋\, \mathsf{in}_B = \mathsf{in}_{A ⅋ B}$.

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 46 *If $P : A_1 \invamp \cdots \invamp A_n$ in $\mathcal{SP}roc$, the A_i are progressive and $P\!\downarrow$, then $P : (A_1, \mathsf{out}_{A_1}) \invamp \cdots \invamp (A_n, \mathsf{out}_{A_n})$ 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 45, the result can be obtained by applying this observation to the type $A_1 \invamp \cdots \invamp A_n$. □

5.6 Loss of Degeneracy

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

Proposition 47 *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 $\mathsf{out}_A \otimes \mathsf{in}_B \neq \mathsf{out}_A \invamp \mathsf{in}_B$.*

PROOF. We have

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

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

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

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

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

PROOF. We have

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

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 $\mathsf{out}_A \mathbin{\&} \mathsf{out}_B$. $\square$

Although $\otimes$ and $\mathbin{⅋}$ are distinct in $\mathcal{SProc}_D$, the Mix rule is still valid.

Proposition 49 *For any objects (A, U), (B, V) of $\mathcal{SProc}_D$, we have $\mathsf{id}_{A \otimes B} : (A, U) \otimes (B, V) \to (A, U) \mathbin{⅋} (B, V)$.*

PROOF. By Lemma 34 it is enough to show that $U \otimes V \subseteq U \mathbin{⅋} V$, i.e.

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

This follows from

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

which in turn follows from

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

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 $(P \otimes Q) \perp (R \otimes S)$. $\square$

6 Equivalence

We will now prove that for each object A of $\mathcal{SP}roc_{pr}$ there is a bijection between $P_D A$ and $P_R A$, and that this bijection preserves all the operations on properties exactly. Also, satisfaction of properties is preserved. This will enable us to deduce that R is a specification structure (which has not yet been proved), and that $\mathcal{SP}roc_R$ and $\mathcal{SP}roc_D$ are isomorphic.

6.1 Equivalence of D and R

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

Lemma 50 *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 51 *Let $U \in P_D A$ and $\theta \in P_R A$.*

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

Proposition 52 *If $\theta \in P_R A$ 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 53 *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 50. Hence $P \perp R$, i.e. $P \in U^{\perp\perp} = U$. Thus $FG(U) \subseteq U$. □

Lemma 54 *Satisfaction is preserved by the correspondence between D and R, 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 $P \in FG(U) = U$. $\Box$

Lemma 55 *$F(\theta^\perp) \subseteq F(\theta)^\perp$ 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 $(s, X) \perp (t, Y)$. $\Box$

Lemma 56 *If $\theta \in P_R A$, there is a set of processes $\{P_i \mid i \in I\}$ (where I is some indexing set) such that*

$$\bigcup_{i \in I} \mathsf{readies}(P_i) = \theta.$$

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 $X \neq \Sigma_A(\varepsilon)$. This means that the states (ε, X) can be taken as the processes P_i. $\Box$

Proposition 57 $F(\theta^\perp) = F(\theta)^\perp$.

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 $P \in F(\theta^\perp)$. $\Box$

Proposition 58 $G(U^\perp) = G(U)^\perp$.

PROOF.

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

Corollary 59 *$F(\theta) = F(\theta)^{\perp\perp}$ and $G(U) = G(U)^{\perp\perp}$.*

Proposition 60 *If $\theta \in P_R A$ and $\phi \in P_R B$ then $F(\theta \mathbin{⅋} \phi) = F(\theta) \mathbin{⅋} F(\phi)$.*

PROOF. It is enough to show that $F(\theta^{\perp} \mathbin{⅋} \phi^{\perp}) = F(\theta^{\perp}) \mathbin{⅋} F(\phi^{\perp})$. Now,

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

and

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

so we need to show that

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

We will consider the two inclusions separately. If

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

then

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

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

For the other inclusion, suppose that $R \perp \{P \otimes Q \mid P \in F(\theta), Q \in F(\phi)\}$. We need to show that $\mathsf{readies}(R) \perp \{(s, A \times B) \mid (\mathsf{fst}^*(s), A) \in \theta, (\mathsf{snd}^*(s), B) \in \phi\}$. Let $(s, X) \in \mathsf{readies}(R)$, $(\mathsf{fst}^*(s), A) \in \theta$ and $(\mathsf{snd}^*(s), B) \in \phi$. By Lemma 56 there are P and Q such that $P \in F(\theta)$, $Q \in F(\phi)$, $(\mathsf{fst}^*(s), A) \in \mathsf{readies}(P)$ and $(\mathsf{snd}^*(s), B) \in \mathsf{readies}(Q)$. 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 $(a, b) \in X \cap (A \times B)$. $\square$

Corollary 61 $F(\theta \otimes \phi) = F(\theta) \otimes F(\phi)$.

PROOF. This follows from the fact that F preserves $\invamp$ and $(-)^\perp$, and duality of $\otimes$ and $\invamp$. $\square$

Corollary 62 $G(U \invamp V) = G(U) \invamp G(V)$ *and* $G(U \otimes V) = G(U) \otimes G(V)$.

PROOF.

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

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

Proposition 63 $F(\theta \,\&\, \phi) = F(\theta) \,\&\, F(\phi)$ *and* $G(U \,\&\, V) = G(U) \,\&\, G(V)$.

PROOF.

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

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

Proposition 64 $F(\circ\theta) = \circ F(\theta)$ *and* $G(\circ U) = \circ G(U)$.

PROOF.

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

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

Proposition 65 $F(\mathsf{RP}(A)) = \mathsf{out}_A$ *and* $F(\mathsf{RP}(A)^\perp) = \mathsf{in}_A$*.*

PROOF. It is sufficient to prove the first statement.

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

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.

6.2 Products and Coproducts

Now that the specification structures D and R have been shown to be equivalent, any calculations relating to deadlock-freedom can be carried out in whichever setting is more convenient. We have not yet proved that products and coproducts lift to $\mathcal{SP}roc_D$; we will now present the proof in terms of ready specifications. By duality, it is sufficient to consider products.

Lemma 66 *Let A, B be objects of $\mathcal{SP}roc_{pr}$ with $\theta \in P_R A$, $\phi \in P_R B$.*

(1) If $s \neq \varepsilon$ then $(\mathsf{inl}^*(s), X) \in \theta \,\&\, \phi \iff (s, \{x \mid \mathsf{inl}(x) \in X\}) \in \theta$.
(2) If $s \neq \varepsilon$ then $(\mathsf{inr}^*(s), X) \in \theta \,\&\, \phi \iff (s, \{x \mid \mathsf{inr}(x) \in X\}) \in \phi$.
(3) $(\varepsilon, X) \in \theta \,\&\, \phi \iff \exists U, V.(\epsilon, U) \in \theta, (\epsilon, V) \in \phi$ 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 $(\mathsf{inl}^*(s), X) \in \theta \,\&\, \phi$, the definition of $\theta \,\&\, \phi$ 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 $\theta \,\&\, \phi$ 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 $(\varepsilon, V) \in \phi$.
($\Leftarrow$) Suppose $X = \mathsf{inl}(U) \cup \mathsf{inr}(V)$ with $(\epsilon, U) \in \theta$ and $(\epsilon, V) \in \phi$. For any $(\epsilon, W) \in \theta^\perp$, $W \cap U \neq \varnothing$ and hence $W \cap X \neq \varnothing$. For any $(\epsilon, Z) \in \phi^\perp$, $Z \cap V \neq \varnothing$ and hence $X \cap V \neq \varnothing$. Thus $(\epsilon, X) \in \theta \,\&\, \phi$. $\square$

Proposition 67 *Let A, B, C be objects of $\mathcal{SP}roc_{pr}$. Let $f : A \to B$ and $g : A \to C$ with $\theta\{f\}\phi$ and $\theta\{g\}\psi$, where $\theta, \phi, \psi \in P_R A, P_R B, P_R C$.*

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

PROOF.

(1) We need

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

Consider any s, U, V, X with

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

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 66, 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 $(\varepsilon, W') \in \phi$, 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 $\mathsf{id}_A \models \theta^\perp \mathbin{⅋} \theta$, 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 $(t \mathbin{\mathsf{zip}} t, Y) \perp (t \mathbin{\mathsf{zip}} \mathsf{snd}^*(s), W \times V)$, 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

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

Consider any s, U, V, X with

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

There are three cases.

(a) $s = \varepsilon$, so that

$$\begin{aligned} X = {} & \mathsf{initials}(f)[(a, b) \mapsto (a, \mathsf{inl}(b))] \\ & \cup \mathsf{initials}(g)[(a, c) \mapsto (a, \mathsf{inr}(c))] \end{aligned}$$

and, by Lemma 66, $V = \mathsf{inl}(V_1) \cup \mathsf{inr}(V_2)$ with $(\varepsilon, V_1) \in \phi^\perp$ and $(\varepsilon, V_2) \in \psi^\perp$. Then $f \models \theta^\perp \mathbin{⅋} \phi$ 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 $(t, V_1) \in \phi^\perp$. Then $f \models \theta^\perp \mathbin{⅋} \phi$ 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. $\square$

6.3 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 68 *If $P \in U$ and* $\mathsf{readies}(P) = \mathsf{readies}(Q)$ *then* $Q \in U^{\perp\perp}$.

PROOF. Follows from Proposition 53. Alternatively, note that 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 deadlock-free processes must be the union of a collection of ready-equivalence classes of deadlock-free processes. So membership of ${}^{\perp\perp}$-invariant sets cannot distinguish processes more finely than ready-equivalence.

6.4 The categories $\mathcal{SP}roc_D$ and $\mathcal{SP}roc_R$ are isomorphic

Theorem 69 *The categories $\mathcal{SP}roc_D$ and $\mathcal{SP}roc_R$ are* isomorphic, *i.e. there are functors $\mathcal{F} : \mathcal{SP}roc_D \leftrightarrow \mathcal{SP}roc_R : \mathcal{G}$ with $\mathcal{FG} = \mathcal{I}_{\mathcal{SP}roc_R}$ and $\mathcal{GF} = \mathcal{I}_{\mathcal{SP}roc_D}$.*

PROOF. 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 R 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 $\mathcal{SP}roc_R$. 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 [26] and LUSTRE [27], and systolic algorithms [22].

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 constructed 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.

In addition, Proposition 73 gives a sufficient condition for the safe construction of cycles which, *a priori*, might be unsafe. The condition is that when a cycle is formed by connecting one of the outputs of a network to one of the inputs, the output must be *independent* of the input. Independence means that the output produced at any time depends only on the input received at previous times. In the synchronous dataflow language LUSTRE, cycle formation is restricted so that every cycle contains a pre node; the pre operator in LUSTRE introduces an output which is independent of one input. Hence any legal cycle in a LUSTRE program satisfies our condition. Furthermore, our condition is the natural specialisation of Wadge's *cycle sum test* [47] to the synchronous case. Wadge attaches an integer weight to every path from input to output in every node, corresponding to a delay in causality. Computations of history-independent functions have a weight of 0, the pre node has a weight of $+1$, and a node which produces output only *after* consuming input has a negative weight. Deadlock in a dataflow network occurs when an output causally depends on itself; if the sum of the weights around every cycle is strictly positive, then this cannot happen. Wadge does not assume synchrony, and hence has to consider negative weights; in the synchronous setting, all weights must be non-negative, and the cycle sum test reduces to checking for the presence of at least one strictly positive weight in every cycle. The cycle-formation condition established in Proposition 73 is therefore as general as Wadge's cycle sum test, given that we are working synchronously.

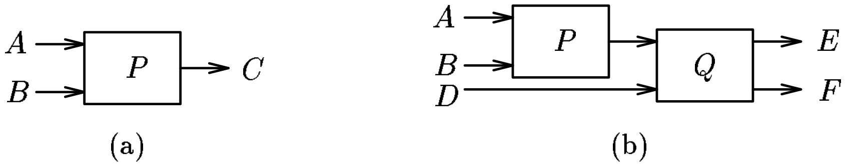

Fig. 4. Two simple networks

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 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 70 *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 71 *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*

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

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

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

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

i.e. that

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

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 $(a_1, \ldots, a_n) \in X$.

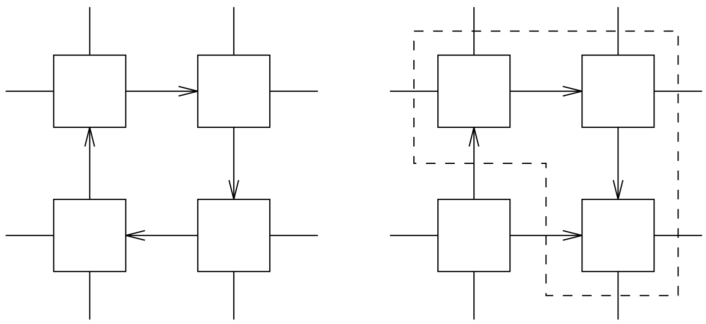

Fig. 5. The two kinds of cycle

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 $(a_1, \ldots, a_n) \in X_1 \times \cdots \times X_n$. $\square$

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

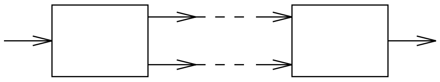

Fig. 6. A double connection between nodes

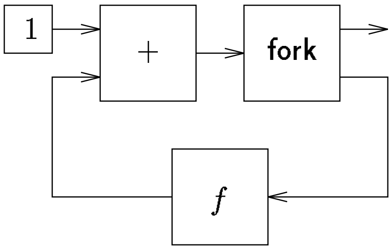

Fig. 7. A network with feedback

phisms gives

$$P : (A, \mathsf{out}_A) \to (B, \mathsf{out}_B) \mathbin{\rotatebox[origin=c]{180}{$\&$}} (C, \mathsf{out}_C)$$
$$Q : (B, \mathsf{out}_B) \otimes (C, \mathsf{out}_C) \to (D, \mathsf{out}_D)$$

or equivalently

$$P : (A, \mathsf{out}_A) \to (B \mathbin{\rotatebox[origin=c]{180}{$\&$}} C, \mathsf{out}_B \mathbin{\rotatebox[origin=c]{180}{$\&$}} \mathsf{out}_C)$$
$$Q : (B \otimes C, \mathsf{out}_B \otimes \mathsf{out}_C) \to (D, \mathsf{out}_D).$$

By Proposition 45 we have $\mathsf{out}_B \mathbin{\rotatebox[origin=c]{180}{$\&$}} \mathsf{out}_C = \mathsf{out}_{B \mathbin{\rotatebox[origin=c]{180}{$\&$}} C}$ and $\mathsf{out}_B \otimes \mathsf{out}_C = \mathsf{out}_{B \otimes C}$. Combined with the fact that $B \otimes C = B \mathbin{\rotatebox[origin=c]{180}{$\&$}} C$, this means that P and Q are composable, and we obtain $P ; Q : (A, \mathsf{out}_A) \to (D, \mathsf{out}_D)$, or equivalently $P ; Q : (A, \mathsf{in}_A) \mathbin{\rotatebox[origin=c]{180}{$\&$}} (D, \mathsf{out}_D)$. 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 72 *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 73 *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

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

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 $i \in \{1, \ldots, n\}$, $(s_i, X_i) \in \theta_i^\perp$, and $(s, X) \in \mathsf{readies}(\overline{P})$, where $s = s_1$ zip $\ldots$ 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 $\mathsf{readies}(P) \subseteq \mathsf{in} \parr \cdots \parr \mathsf{in} \parr \mathsf{out} \parr \cdots \parr \mathsf{out} \parr \mathsf{in} \parr \mathsf{out}$, 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. $\square$

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 [23] 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 73 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 [23]. Hence it is sufficient to check that there is a source somewhere in every cycle. In LUSTRE, the only way of introducing a source is by means of a **pre** node, hence the requirement that every cycle in a LUSTRE program must contain at least one **pre**.

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 **in.out** represents an infinite alternation of input and output. The structure of $\mathcal{SP}roc_D$ is rich enough to include semantic versions of such communication patterns over any $\mathcal{SP}roc$ object. Continuing the example, the interpretation of the sequence **in.out** over the $\mathcal{SP}roc$ object 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 presented a semantic 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 semantic universes allowing us to express increasingly strong constraints on computational behaviour. We have illustrated this idea by defining a specification structure over $\mathcal{SP}roc$, a category of synchronous processes. The resulting category, $\mathcal{SProc}_D$, is a semantic universe in which compositional verification of deadlock-freedom can be discussed. We have presented two equivalent definitions of $\mathcal{SProc}_D$, one based on the idea of a specification as a set of processes, the other based on the notion of ready specification. As a simple application, we have shown that $\mathcal{SProc}_D$ supports the specification and verification of deadlock-free synchronous networks; examples of synchronous networks include synchronous dataflow programs and systolic algorithms.

We have described $\mathcal{SProc}_D$ as a "semantic universe" without explicitly presenting a semantic function with a particular language as its domain. We think of the objects of $\mathcal{SProc}_D$ as semantic models of types which specify communication behaviour, as indicated briefly in Section 7.3. Previous work on interaction categories includes the definition of a typed process calculus [23,24] in which types correspond to safety specifications. This process calculus has a denotational semantics in which types are interpreted by $\mathcal{SProc}$ objects and processes by $\mathcal{SProc}$ morphisms. (In fact, the typed process calculus can be interpreted in any category satisfying certain axioms which capture the essential structure of $\mathcal{SProc}$.) We hope to extend this process calculus so that its type system specifies communication patterns of the kind mentioned in Section 7.3, and then use $\mathcal{SProc}_D$ to give a denotational semantics to the new process calculus. The aim of our research on interaction categories is to understand the semantic structure of complex behavioural types, and work from that understanding towards a language which can be given a categorical semantics.

Several type systems for concurrency have been proposed recently. All of them start with a syntax (often based on the π-calculus), an operational semantics and a collection of typing rules, and prove that correct typing guarantees certain operational properties. 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 [32], the order of usage of channels [31], subtyping [42], types for choice and branching behaviour [46]. The distinguishing features of our semantic approach are as follows. First, it is based on a category-theoretic description of the collective structure of processes and their relationship to specifications. Second, we have proposed a methodology (via the notion of a specification structure) for treating a range of program properties and verification techniques within a single framework. Finally, because we have a semantic model of specifications or types, arbitrarily complex combinations of input and output ports can be treated in a uniform way. 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. Of course, we still need to formalise a syntax of processes and types in order to complete the picture of a language with its categorical semantics.

The other way in which the theory described in this paper could be extended and developed is by removing the assumption of synchrony. Progress has already been made on an asynchronous version of the theory, by applying the sets of processes approach to the asynchronous interaction category $\mathcal{ASProc}$ [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 [2,23] and improved versions in [9,39]; a full report of this area will be the subject of a future paper.

Beyond the issues of synchrony and a formal syntax, there are two respects in which our theory of deadlock-freedom is 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 our specifications 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 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 project "Foundational Structures in Computer Science", 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 and prooftree packages were 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. Retracing some paths in process algebra. In U. Montanari and V. Sassone, editors, *CONCUR'96: Proceedings of the 7th International Conference on Concurrency Theory*, volume 1119 of *LNCS*. Springer-Verlag, 1996.

[9] S. Abramsky, S. Gay, and R. Nagarajan. A type-theoretic approach to deadlock-freedom of asynchronous systems. In M. Abadi and T. Ito, editors, *Theoretical Aspects of Computer Software. International Symposium TACS'97*, number 1281 in Lecture Notes in Computer Science, pages 295–320, Sendai, Japan, September 1997. Springer-Verlag.

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

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

[12] 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.

[13] 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.

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

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

[16] 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.

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

[18] 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.

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

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

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

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

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

[24] S. J. Gay and R. Nagarajan. A typed calculus of synchronous processes. In *Proceedings, Tenth Annual IEEE Symposium on Logic in Computer Science.* IEEE Computer Society Press, 1995.

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

[26] P. Guernic, T. Gautier, M. Borgne, and C. Maire. Programming real-time applications with SIGNAL. *Proceedings of the IEEE*, 79(9):1321–1336, September 1991.

[27] 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.

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

[29] A. Joyal, R. Street, and D. Verity. Traced monoidal categories. *Mathematical Proceedings of the Cambridge Philosophical Society*, 119(3):425–446, 1996.

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

[31] N. Kobayashi. A partially deadlock-free typed process calculus. In *Proceedings, Twelfth Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, 1997.

[32] 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.

[33] 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.

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

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

[40] P. W. O'Hearn and R. D. Tennent. Parametricity and local variables. *J. ACM*, 42(3):658–709, May 1995.

[41] D. Pavlovic and S. Abramsky. Specifying interaction categories. In E. Moggi and G. Rosolini, editors, *Category Theory and Computer Science '97*, volume 1290 of *LNCS*, pages 147–158. Springer Verlag, 1997.

[42] 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.

[43] A. M. Pitts. Relational properties of domains. *Information and Computation*, 127:66–90, 1996.

[44] G. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, 1981.

[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] W. W. Wadge. An extensional treatment of dataflow deadlock. *Theoretical Computer Science*, 13:3–15, 1981.