DOI 10.1007/s00165-003-0018-1
BCS © 2003
Formal Aspects of Computing (2003) 15: 299–318

Formal Aspects of Computing

Intensional and Extensional Semantics of Dataflow Programs

Simon Gay[1] and Rajagopal Nagarajan[2]

[1]Department of Computing Science, University of Glasgow, Glasgow, UK
[2]Department of Computer Science, University of Warwick, Coventry, UK

Abstract. We compare two semantic models of dataflow programs: a synchronous version of the classical Kahn semantics, and a new semantics in a category of synchronous processes. We consider the Kahn semantics to be extensional, as it describes the functions computed by dataflow nodes, and the categorical semantics to be intensional, as it describes the step-by-step production of output tokens from input tokens. Assuming that programs satisfy Wadge's cycle sum condition and are therefore deadlock-free, we prove that the two semantics are equivalent. This equivalence result amounts to a proof that function composition in the extensional semantics is faithfully modelled by the detailed interactions of the intensional semantics, and provides further insight into the nature of dataflow computation.

Keywords: Categorical semantics; Dataflow; Interaction categories; Kahn semantics; Synchronous computation

1. Introduction

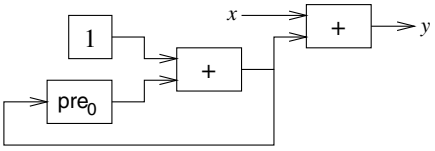
Dataflow is a simple model of parallel computing in which a program consists of a collection of nodes, each with a number of inputs and outputs, connected into a fixed network. Abstractly, the nodes can be viewed as independent parallel processes, communicating via data channels; a range of concrete implementations are possible, with varying degrees of parallelism.

Small dataflow programs can usefully be presented graphically, but to cater for larger programs dataflow languages typically use an equational syntax. Figure 1 illustrates the two styles. The correspondence between the graphical and equational presentations is not usually formalised.

1.1. Synchronous and Asynchronous Execution

Dataflow programs can be executed either synchronously or asynchronously. We take synchronous execution to mean that tokens are produced in step with a global clock; at each time step, every node consumes one token from each of its inputs and adds one token to its output. The connections between nodes transmit tokens instantaneously. Asynchronous execution means that the connections are unbounded buffers and each node produces

Correspondence and offprint requests to: Simon Gay, Department of Computing Science, University of Glasgow, Glasgow G12 8QQ, UK. Email: simon@dcs.gla.ac.uk



$$\begin{array}{rcl} y & = & x + z \\ z & = & 1 + w \\ w & = & \mathsf{pre}_0(z) \end{array} \qquad \text{or} \qquad \begin{array}{rcl} y & = & x + z \\ z & = & 1 + \mathsf{pre}_0(z) \end{array}$$

Fig. 1. A dataflow network: graphical and equational descriptions.

more output whenever it has received sufficient input to do so. The difference between the synchronous and asynchronous models is highlighted by the interpretation of the pre_0 node, which outputs 0 followed by a copy of its input stream. Asynchronously, the node is fully specified by the equation $\mathsf{pre}_0(x) = 0x$ and the rate of production of tokens is not determined. Synchronously, the node outputs 0 at the first step and at each subsequent step outputs the token received at the previous step. This paper is concerned with dataflow programs using the synchronous execution model we have described.

1.2. Synchronous Execution of the Example

The synchronous execution of the example network is shown in the table, with time proceeding downwards. The calculation is driven by the 0 initially produced by the pre_0 node.

$w = \mathsf{pre}_0(z)$	$z = 1 + w$	x	$y = x + z$
0	1	x_1	$x_1 + 1$
1	2	x_2	$x_2 + 2$
2	3	x_3	$x_3 + 3$
$\vdots$	$\vdots$	$\vdots$	$\vdots$

1.3. Kahn Semantics

The standard semantics of dataflow programs was defined by Kahn [Kah74]. The outputs of a node, as streams of tokens, are continuous functions of the input streams. The equational description of a network is interpreted as a set of simultaneous equations and solved by taking the least fixed point of an appropriate stream function. The solution yields the streams produced by the entire network from given inputs. The Kahn semantics describes asynchronous execution, but if nodes are modelled by *synchronous* functions (which we will define in Section 3) then the least fixed point solution describes synchronous execution.

1.4. Categorical Semantics

We define a new semantics of dataflow programs, which interprets nodes and networks as morphisms in $\mathcal{SP}roc$, a category of synchronous processes. At each instant, a node imposes a certain relation on its input and output tokens. The behaviour of a network emerges, from composition of these relations at each instant, as a collection of streams which satisfy all the constraints imposed by the nodes and connections.

1.5. Semantic Equivalence

We prove that the categorical semantics is equivalent to the synchronous Kahn semantics, by defining what it means for a synchronous process to compute a function on streams, and showing that the categorical semantics of a dataflow program is a process which computes the stream function given by the Kahn semantics.

This equivalence is subject to the assumption that networks satisfy Wadge's cycle sum condition, guaranteeing deadlock freedom. The result provides alternative ways of understanding the behaviour of a network: abstractly, by composition of stream functions; or more operationally, in terms of step-by-step interactions between processes.

1.6. Contribution

The main contribution of this paper is the definition of the categorical semantics of dataflow and the proof of its equivalence with the Kahn semantics. Identifying the notion of a synchronous stream function and formulating the synchronous Kahn semantics is a secondary contribution, as we are not aware that this has been done explicitly before.

1.7. Applications and Limitations

Dataflow has been used as the basis of at least two industrial-strength programming languages: SIGNAL [BeL91, LGL91] and LUSTRE [HCR91]. These languages are well suited to real-time applications, and can be compiled directly to hardware. One of their key features, which goes beyond the pure model of dataflow programming considered in the present paper, is the introduction of multiple clocks. Rather than synchronising directly with the global clock, each stream has its own clock specifying that tokens are produced at some but not all instants of the global clock. Clocks are streams which in turn have their own clocks, in a hierarchical structure. The categorical semantics of dataflow presented in this paper has been extended to a model of the clock structure of Signal [GaN93, Nag98] and Lustre [Gay95]. However, we are not aware of an extension of the Kahn semantics to clocks, and so our semantic equivalence results only apply to pure dataflow.

1.8. Outline of the Paper

In Section 2 we define an idealised dataflow language $\mathcal{D}$ in which programs are expressions rather than sets of equations. In Section 3 we define the Kahn semantics of $\mathcal{D}$, and prove that the synchronous Kahn semantics is well defined. In Section 4 we define $\mathcal{SP}roc$, the category of synchronous processes, and in Section 5 we define the semantics of $\mathcal{D}$ in $\mathcal{SP}roc$. Section 6 contains the main technical contribution of the paper, and proves a series of results leading to the equivalence of the Kahn and $\mathcal{SP}roc$ semantics. Section 7 concludes.

2. The Idealised Dataflow Language $\mathcal{D}$

When formalising the semantics of dataflow programs, it is convenient for networks to be described by expressions rather than systems of equations. This requires the introduction of an explicit feedback or cycle operator, to replace internal variables (such as z in Fig. 1). We define the idealised dataflow language $\mathcal{D}$, which is a subset of Ştefanescu's [Ste00] calculus of flownomials.

Definition 2.1 Given a collection of ground types $\mathcal{T} = \{A, B, \ldots\}$, the types of $\mathcal{D}$ consist of n-fold tensor products of ground types, $A_1 \otimes \cdots \otimes A_n$, and I, the 0-fold tensor product. Tensor product of n-fold tensor products is defined in the natural way. We sometimes refer to types as general types to emphasise that they need not be ground types.

Definition 2.2 The basic nodes of $\mathcal{D}$ are typed symbols of the form $f : A_1 \otimes \cdots \otimes A_m \to B_1 \otimes \cdots \otimes B_n$, where the A_i and B_j are ground types.

Definition 2.3 The expressions of $\mathcal{D}$ are the typed expressions which can be built from basic nodes, the identity symbols I_m, the transposition symbols ${}^m\mathsf{X}^n$, the fork symbols $\wedge^m$, the sink symbols $\perp^m$, the composition operator ;, the parallel operator $\otimes$, and the feedback operator $\uparrow^m$, by the following typing rules.

- Identities (the A_i are ground types):

  $\mathsf{I}_m : A_1 \otimes \cdots \otimes A_m \to A_1 \otimes \cdots \otimes A_m$

- Transpositions (the A_i and B_j are ground types):

  ${}^m\mathsf{X}^n : A_1 \otimes \cdots \otimes A_m \otimes B_1 \otimes \cdots \otimes B_n \to B_1 \otimes \cdots \otimes B_n \otimes A_1 \otimes \cdots \otimes A_m$

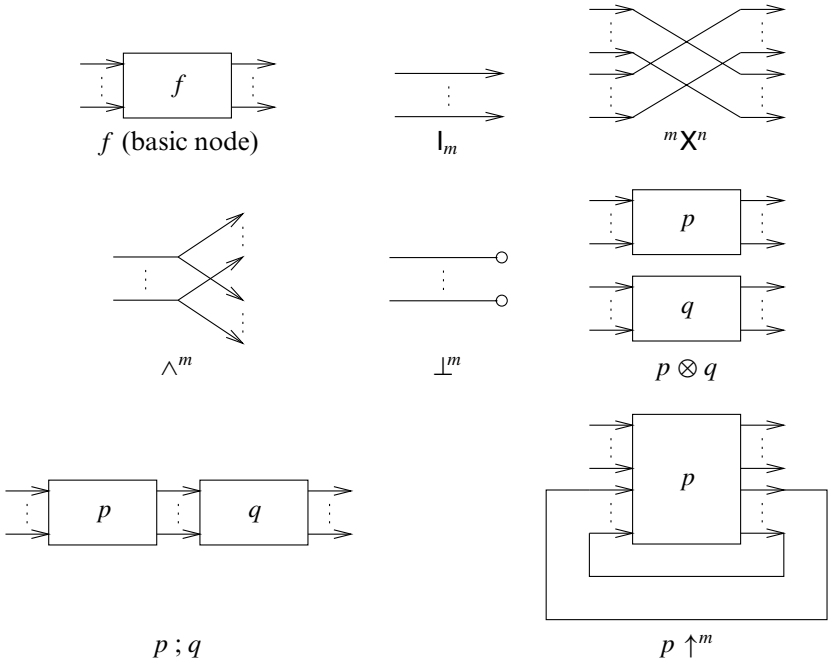


Fig. 2. The correspondence between $\mathcal{D}$ expressions and networks.

- Forks (the A_i are ground types):

  $$\wedge^m : A_1 \otimes \cdots \otimes A_m \to A_1 \otimes \cdots \otimes A_m \otimes A_1 \otimes \cdots \otimes A_m$$

- Sinks (the A_i are ground types):

  $$\perp^m : A_1 \otimes \cdots \otimes A_m \to I$$

- Composition (X, Y, Z are general types):

  $$\frac{p : X \to Y \quad q : Y \to Z}{p \,;\, q : X \to Z}$$

- Parallel (X, Y, Z, W are general types):

  $$\frac{p : X \to Y \quad q : Z \to W}{p \otimes q : X \otimes Z \to Y \otimes W}$$

- Feedback (A and C are general types; the B_i are ground types):

  $$\frac{p : A \otimes B_1 \otimes \cdots \otimes B_m \to C \otimes B_1 \otimes \cdots \otimes B_m}{p \uparrow^m : A \to C}$$

A $\mathcal{D}$ expression $p : A_1 \otimes \cdots \otimes A_m \to B_1 \otimes \cdots \otimes B_n$ corresponds to a network with m inputs and n outputs, as illustrated by Fig. 2.

The network of Fig. 1 corresponds to the $\mathcal{D}$ expression

$$(\mathsf{I}_1 \otimes ((\mathsf{one} \otimes \mathsf{pre}_0) \,;\, \mathsf{plus} \,;\, \wedge^1) \uparrow^1) \,;\, \mathsf{plus} : N \to N$$

where N is the ground type of integers and the basic nodes have been typed as follows:

$$\mathsf{plus} : N \otimes N \to N \qquad \mathsf{one} : I \to N \qquad \mathsf{pre}_0 : N \to N.$$

$$p \otimes (q \otimes r) = (p \otimes q) \otimes r \tag{1}$$
$$\mathsf{I}_0 \otimes p = p \tag{2}$$
$$p \otimes \mathsf{I}_0 = p \tag{3}$$
$$p\,;(q\,;r) = (p\,;q)\,;r \tag{4}$$
$$\mathsf{I}_m\,;p = p \tag{5}$$
$$p\,;\mathsf{I}_n = p \tag{6}$$
$$(p \otimes p')\,;(q \otimes q') = (p\,;q) \otimes (p'\,;q') \tag{7}$$
$$\mathsf{I}_m \otimes \mathsf{I}_n = \mathsf{I}_{m+n} \tag{8}$$
$${}^{m}\mathsf{X}^{n}\,;{}^{n}\mathsf{X}^{m} = \mathsf{I}_{m+n} \tag{9}$$
$${}^{m}\mathsf{X}^{n+p} = ({}^{m}\mathsf{X}^{n} \otimes \mathsf{I}_p)\,;(\mathsf{I}_n \otimes {}^{m}\mathsf{X}^{p}) \tag{10}$$
$$(p \otimes q)\,;{}^{p}\mathsf{X}^{q} = {}^{m}\mathsf{X}^{n}\,;(q \otimes p) \tag{11}$$
$$p\,;(q \uparrow^n)\,;r = ((p \otimes \mathsf{I}_n)\,;q\,;(r \otimes \mathsf{I}_n)) \uparrow^n \tag{12}$$
$$p \otimes (q \uparrow^n) = (p \otimes q) \uparrow^n \tag{13}$$
$$(p\,;(\mathsf{I}_n \otimes q)) \uparrow^p = ((\mathsf{I}_m \otimes q)\,;p) \uparrow^q \tag{14}$$
$$p \uparrow^0 = p \tag{15}$$
$$(p \uparrow^n) \uparrow^m = p \uparrow^{m+n} \tag{16}$$
$$\mathsf{I}_n \uparrow^n = \mathsf{I}_0 \tag{17}$$
$${}^{n}\mathsf{X}^{n} \uparrow^n = \mathsf{I}_n \tag{18}$$
$$\wedge^n\,;(\wedge^n \otimes \mathsf{I}_n) = \wedge^n\,;(\mathsf{I}_n \otimes \wedge^n) \tag{19}$$
$$\wedge^n\,;(\perp^n \otimes \mathsf{I}_n) = \mathsf{I}_n \tag{20}$$
$$\wedge^n\,;{}^{n}\mathsf{X}^{n} = \wedge^n \tag{21}$$
$$\perp^{m+n} = \perp^m \otimes \perp^n \tag{22}$$
$$\wedge^{m+n} = (\wedge^m \otimes \wedge^n)\,;(\mathsf{I}_m \otimes {}^{m}\mathsf{X}^{n} \otimes \mathsf{I}_n) \tag{23}$$
$$\perp^0 = \mathsf{I}_0 \tag{24}$$
$$\wedge^0 = \mathsf{I}_0 \tag{25}$$
$$p\,;\perp^n = \perp^m \tag{26}$$
$$p\,;\wedge^n = \wedge^m\,;(p \otimes p) \tag{27}$$

Fig. 3. Axioms for $\mathcal{D}$.

In general the representation of a network as a $\mathcal{D}$ expression is not unique. An alternative representation of our example network is

$$(\mathsf{I}_1 \otimes ((\mathsf{one} \otimes \mathsf{pre}_0)\,;\mathsf{plus}\,;\wedge^1)\,;(\mathsf{plus} \otimes \mathsf{I}_1)) \uparrow^1$$

in which the feedback operator is applied at the top level rather than at the innermost possible level. Even a simple connection such as

$$\mathsf{one}\,;\mathsf{pre}_0 : I \to N$$

can be formed by means of the feedback operator and represented by

$$((\mathsf{one} \otimes \mathsf{pre}_0)\,;{}^{1}\mathsf{X}^{1}) \uparrow^1$$

To restore the exact correspondence between $\mathcal{D}$ expressions and networks, we consider $\mathcal{D}$ expressions up to provable equality in the equational theory generated by the axioms in Fig. 3. These are Ştefanescu's axioms for the calculus of flownomials [Ste00]. We do not make use of the axioms in the present paper, except to note that they are validated by the semantic model which we define in Section 5. Axioms (1)–(18) are the axioms for a symmetric strict monoidal category [Mac71] with a trace [JSV96] (the cycle operator). Together, these axioms are sound and complete for graph isomorphism. For axioms (19)–(27) to be satisfied, it is sufficient but not necessary that the

monoidal product is also a Cartesian product; our category $\mathcal{SP}roc$ satisfies these axioms but is not Cartesian. The additional axioms are sound for graph isomorphism, with the exception of (19) (non-isomorphic branching structures), (26) and (27) (duplication of deletion of p). Jeffrey [Jef97] considers soundness and completeness with respect to a notion of bisimulation on graphs, by adding the following axiom:

if s is any $\mathcal{D}$ expression containing no basic nodes and $(\mathsf{I} \otimes s)\,;\, f = g\,;\,(\mathsf{I} \otimes s)$ then $f \uparrow = g \uparrow$

Jeffrey refers to this axiom as *Plotkin uniformity* and Ştefanescu [Ste00] calls it the *enzymatic axiom*. It is also necessary to consider the relationship between feedback and branching. Ştefanescu adds the axiom

$$\wedge^n \uparrow^n = \top^n$$

where $\top^n$ is a *source* symbol whose axiomatisation also requires the introduction of *joins* $\vee^n$. We do not consider sources or joins in this paper, instead eliminating $\wedge^n \uparrow^n$ by our deadlock-freedom condition.

An equational description of a network can be translated straightforwardly into a $\mathcal{D}$ expression. We will not define the translation here; a similar translation has been formalised elsewhere [Nag98] by the second author. Multiple occurrences of variables must be converted into either compositions or feedbacks, which can be done in various provably equivalent ways.

3. The Kahn Semantics of $\mathcal{D}$

3.1. Notation

If A is a set then we write A^ω for the set of finite and infinite sequences of elements of A, and A^* for the set of finite sequences. If $f : A \to B$ then $f^* : A^* \to B^*$ and $f^\omega : A^\omega \to B^\omega$ are the extensions of f to sequences. We write π_i for the ith projection function from a Cartesian product. If $f : A^\omega \to B^\omega$ then we write $\mathsf{fix}(f)$ for the least fixed point of f in the prefix ordering on A^ω, obtained by iteration from the empty trace: $\mathsf{fix}(f) = \bigsqcup_r [f^r(\varepsilon)]$.

3.2. Basic Definitions

In the Kahn semantics [Kah74] of dataflow networks, a network computes a collection of continuous (with respect to the prefix order) functions of the streams of data forming its inputs. Feedback is interpreted by the least fixed point operator. We now define the Kahn semantics $[\![\cdot]\!]_K$ of $\mathcal{D}$ expressions, which corresponds to Ştefanescu's SPF (stream processing function) model [Ste00] of the calculus of flownomials. The definitions apply to both the synchronous and asynchronous execution models; the difference is in the nature of the functions which represent the basic nodes, as we will see later.

Each ground type is interpreted by a set. An expression

$$p : A_1 \otimes \cdots \otimes A_m \to B_1 \otimes \cdots \otimes B_n$$

is interpreted by a function

$$[\![p]\!]_K : A_1^\omega \times \cdots \times A_m^\omega \to B_1^\omega \times \cdots \times B_n^\omega.$$

For each basic node, an interpreting function must be specified. For other expressions, the semantics is defined compositionally as follows:

- Identities: $[\![\mathsf{I}_m]\!]_K(x_1, \ldots, x_m) = (x_1, \ldots, x_m)$.
- Transpositions: $[\![{}^m\mathsf{X}^n]\!]_K(x_1, \ldots, x_{m+n}) = (x_{m+1}, \ldots, x_{m+n}, x_1, \ldots, x_m)$.
- Forks: $[\![\wedge^m]\!]_K(x_1, \ldots, x_m) = (x_1, \ldots, x_m, x_1, \ldots, x_m)$.
- Sinks: $[\![\perp^m]\!]_K(x_1, \ldots, x_m) = ()$.
- Composition: $[\![p\,;q]\!]_K(x_1, \ldots, x_m) = [\![q]\!]_K([\![p]\!]_K(x_1, \ldots, x_m))$.
- Parallel: $[\![p \otimes q]\!]_K(x_1, \ldots, x_{m+n}) = (y_1, \ldots, y_{r+s})$ where

$$y_i = \begin{cases} \pi_i([\![p]\!]_K(x_1, \ldots, x_m)) & \text{if} \quad 1 \leqslant i \leqslant r \\ \pi_{i-r}([\![q]\!]_K(x_{m+1}, \ldots, x_{m+n})) & \text{if} \quad r < i \leqslant r+s \end{cases}$$

- Feedback: $[\![p \uparrow^1]\!]_K(x_1, \ldots, x_m) = (y_1, \ldots, y_n)$ where

$$y_i = \pi_i([\![p]\!]_K(x_1, \ldots, x_m, z))$$
$$z = \mathsf{fix}(\lambda y.(\pi_{n+1}([\![p]\!]_K(x_1, \ldots, x_m, y)))).$$

  Inductively, $[\![p \uparrow^r]\!]_K = [\![(p \uparrow^{r-1}) \uparrow^1]\!]_K$. It is also possible to define $[\![p \uparrow^r]\!]_K$ directly as a simultaneous fixed point, and it is standard in fixed point theory that the two definitions are equivalent. For the remainder of the paper we will work with single feedbacks.

The semantics of network and basic nodes have the same form. This supports the encapsulation constructs of real dataflow languages, which allow an arbitrary network to be packaged as a new basic node.

3.3. Synchronous Execution, Deadlock Freedom and the Cycle Sum Condition

If a network can be executed synchronously, then the Kahn functions of the basic nodes must be *synchronous*: they cannot consume input without producing output. We now define the notion of synchronous function and the related notion that certain output traces of a network are always longer than certain input traces.

Definition 3.1 A function $f : A_1^\omega \times \cdots \times A_m^\omega \to B_1^\omega \times \cdots \times B_n^\omega$ is *synchronous* if and only if

$$(\forall i.\mathsf{length}(x_i) \geqslant r) \Rightarrow \forall i.\mathsf{length}(\pi_i(f(x_1, \ldots, x_m))) \geqslant r$$

Definition 3.2 Let $J \subseteq \{1, \ldots, m\}, I \subseteq \{1, \ldots, n\}$. A synchronous function $f : A_1^\omega \times \cdots \times A_m^\omega \to B_1^\omega \times \cdots \times B_n^\omega$ *has outputs I longer than inputs J* if and only if

$$\forall a. \quad (\forall i \in J.\mathsf{length}(x_i) \geqslant a) \& (\forall i \notin J.\mathsf{length}(x_i) \geqslant a+1) \Rightarrow \forall i \in I.\mathsf{length}(\pi_i(f(x_1, \ldots, x_n))) \geqslant a+1$$

Wadge [Wad81] identified a sufficient condition for dataflow networks to be deadlock-free, in the sense that their Kahn semantics always produces infinite outputs from infinite inputs. The idea is to calculate a delay for each (*input*, *output*) pair of each basic node, and specify that the sum of the delays around every cycle in a network must be strictly positive. A formal definition of this *cycle sum condition* allows us to prove that the Kahn semantics can be consistently restricted to synchronous functions. From now on we assume that for every basic node f the corresponding semantic function $[\![f]\!]_K$ is synchronous.

We define, for each $\mathcal{D}$ expression $p : A_1 \otimes \cdots \otimes A_m \to B_1 \otimes \cdots \otimes B_n$, a *delay matrix* $(\delta^p_{i,j})_{1 \leqslant i \leqslant m, 1 \leqslant j \leqslant n}$ whose entries are taken from $\mathbb{N} \cup \{\infty\}$. The token received on input i at time t cannot affect output j until time $t + \delta^p_{i,j}$. An entry of ∞ indicates that there is no path between a particular (*input*, *output*) pair.

Definition 3.3 Let $f : A_1 \otimes \cdots \otimes A_m \to B_1 \otimes \cdots \otimes B_n$ be a basic node and assume that $[\![f]\!]_K$ is synchronous. The entries of δ^f are maximal such that if

$$[\![f]\!]_K(x_1, \ldots, x_m) = (y_1, \ldots, y_n)$$

then $\forall j.\mathsf{length}(y_j) \geqslant \mathsf{min}_i(\delta^f_{i,j} + \mathsf{length}(x_i))$.

The delay matrix constructions for the constructors of $\mathcal{D}$ are as follows:

- Identities: $\delta^{\mathsf{I}_m}_{i,j} = \begin{cases} 0 \text{ if } i = j \\ \infty \text{ otherwise.} \end{cases}$
- Transpositions: $\delta^{m\mathsf{X}n}_{i,j} = \begin{cases} 0 \text{ if } 1 \leqslant i \leqslant m \text{ and } j = i+n \\ 0 \text{ if } m+1 \leqslant i \leqslant m+n \text{ and } j = i-m \\ \infty \text{ otherwise.} \end{cases}$
- Forks: $\delta^{\wedge^m}_{i,j} = \begin{cases} 0 \text{ if } j = i \text{ or } j = i+m \\ \infty \text{ otherwise.} \end{cases}$
- Sinks: $\delta^{\perp^m}$ has no entries.

- Composition: $\delta^{p;q}_{i,j} = \mathsf{min}_k(\delta^p_{i,k} + \delta^q_{k,j})$.
- Parallel: $\delta^{p\otimes q} = \begin{pmatrix} \delta^p & \infty \\ \infty & \delta^q \end{pmatrix}$
- Feedback: $\delta^{p\uparrow^1}_{i,j} = \mathsf{min}(\delta^p_{i,j}, \delta^p_{i,n+1} + \delta^p_{m+1,j}))$ where $p : A_1 \otimes \cdots \otimes A_m \otimes C \to B_1 \otimes \cdots \otimes B_n \otimes C$.

Note that $\delta^{\perp^m}$, and δ^f for any basic node f with no inputs, must be interpreted as trivial matrices with either no rows or no columns and hence no entries. Calculations involving these matrices give the correct results.

With these definitions, the cycle sum condition becomes an additional condition on the typing rule for feedback (here the A_i and B_j are ground types):

$$\frac{p : A_1 \otimes \cdots \otimes A_m \otimes C_1 \otimes \cdots \otimes C_n \to B_1 \otimes \cdots \otimes B_r \otimes C_1 \otimes \cdots \otimes C_n \quad \forall k \in \{1, \ldots, n\}.\delta^p_{m+k,r+k} > 0}{p \uparrow^n : A_1 \otimes \cdots \otimes A_m \to B_1 \otimes \cdots \otimes B_r}$$

Wadge's original statement of the cycle sum condition was that the sum of the delays around every cycle must be strictly positive. In practical dataflow languages, most basic nodes (for example, those which operate as time-independent functions, such as addition) have delays of 0. Non-trivial (i.e. neither 0 nor ∞) delays are restricted to a few basic nodes, the typical example being pre, which has a delay of 1. If cyclic networks are to be constructed, it is essential that there be at least one basic node with a positive delay. In general, negative delays are allowed; for example, the language LUCID [AsW85] has a node called next, whose first output token is the second input token, and this node has a delay of -1. However, in the case of synchronous execution, only positive delays make sense. For synchronous languages the cycle sum condition reduces to the requirement that every cycle contains a node with a non-zero delay.

3.4. Example

For the network of Fig. 1, the delay matrices for the basic nodes are

$$\mathsf{plus} : \begin{pmatrix} 0 \\ 0 \end{pmatrix} \qquad \mathsf{one} : () \qquad \mathsf{pre}_0 : (1).$$

The following derivation shows the calculation of the delay matrix for the network as a whole. Note that the cycle sum condition is satisfied.

$$\dfrac{\mathsf{I}_1 : (0) \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{one} : () \quad \mathsf{pre}_0 : (1)}{\mathsf{one} \otimes \mathsf{pre}_0 : (\infty\ \ 1)} \quad \mathsf{plus} : \begin{pmatrix} 0 \\ 0 \end{pmatrix}}{(\mathsf{one} \otimes \mathsf{pre}_0)\,;\mathsf{plus} : (1)} \qquad \wedge^1 : (0\ \ 0)}{(\mathsf{one} \otimes \mathsf{pre}_0)\,;\mathsf{plus}\,;\wedge^1 : (1\ \ 1)}}{((\mathsf{one} \otimes \mathsf{pre}_0)\,;\mathsf{plus}\,;\wedge^1)\uparrow^1 : ()}}{\mathsf{I}_1 \otimes ((\mathsf{one} \otimes \mathsf{pre}_0)\,;\mathsf{plus}\,;\wedge^1)\uparrow^1 : (0\ \ \infty)} \qquad \mathsf{plus} : \begin{pmatrix} 0 \\ 0 \end{pmatrix}}{(\mathsf{I}_1 \otimes ((\mathsf{one} \otimes \mathsf{pre}_0)\,;\mathsf{plus}\,;\wedge^1)\uparrow^1)\,;\mathsf{plus} : (0)}$$

3.5. Properties of the Synchronous Semantics

The first proposition extends the property of Definition 3.3 from basic nodes to all expressions.

Proposition 3.4 Suppose that every basic node is interpreted by a synchronous function, and that the cycle sum condition is observed. Let $p : A_1 \otimes \cdots \otimes A_m \to B_1 \otimes \cdots \otimes B_n$ be a $\mathcal{D}$ expression. If

$$[\![p]\!]_K(x_1, \ldots, x_m) = (y_1, \ldots, y_n)$$

then $\forall j.\mathsf{length}(y_j) \geqslant \mathsf{min}_i(\delta^p_{i,j} + \mathsf{length}(x_i))$.

Proof. By induction on the structure of p, with a case for each constructor of $\mathcal{D}$. For basic nodes the desired property holds by definition. The cases of identities, transpositions, and forks are straightforward, because the delays are either 0 or ∞ and the semantic functions simply copy inputs to outputs in the positions indicated by the 0 delays. The case of a sink is vacuous. The case of parallel is straightforward because the two parts of the expression operate independently and this fact is mirrored by the definition of the delay matrix.

For the case of composition, consider $p : A_1 \otimes \cdots \otimes A_m \to B_1 \otimes \cdots \otimes B_n$ and $q : B_1 \otimes \cdots \otimes B_n \to C_1 \otimes \cdots \otimes C_r$ with

$$\begin{aligned}
[\![p]\!]_K(x_1, \ldots, x_m) &= (y_1, \ldots, y_n) \\
[\![q]\!]_K(y_1, \ldots, y_n) &= (z_1, \ldots, z_r) \\
[\![p\,;q]\!]_K(x_1, \ldots, x_m) &= (z_1, \ldots, z_r).
\end{aligned}$$

Given, by the induction hypothesis, $\forall j.\mathsf{length}(y_j) \geqslant \mathsf{min}_i(\delta^p_{i,j} + \mathsf{length}(x_i))$ and $\forall j.\mathsf{length}(z_j) \geqslant \mathsf{min}_i(\delta^q_{i,j} + \mathsf{length}(y_i))$ we have, for any j,

$$\begin{aligned}
\mathsf{length}(z_j) &\geqslant \mathsf{min}_i(\delta^q_{i,j} + \mathsf{min}_k(\delta^p_{k,i} + \mathsf{length}(x_k))) \\
&= \mathsf{min}_{i,k}(\delta^q_{i,j} + \delta^p_{k,i} + \mathsf{length}(x_k)) \\
&= \mathsf{min}_k(\mathsf{min}_i(\delta^p_{k,i} + \delta^q_{i,j}) + \mathsf{length}(x_k)) \\
&= \mathsf{min}_k(\delta^{p;q}_{k,j} + \mathsf{length}(x_k))
\end{aligned}$$

as required.

For the case of feedback, consider $p : A_1 \otimes \cdots \otimes A_m \otimes C \to B_1 \otimes \cdots \otimes B_n \otimes C$ with

$$\begin{aligned}
z &= \mathsf{fix}(\lambda y.(\pi_{n+1}([\![p]\!]_K(x_1, \ldots, x_m, y)))) \\
[\![p]\!]_K(x_1, \ldots, x_m, z) &= (y_1, \ldots, y_n, z) \\
[\![p \uparrow^1]\!]_K(x_1, \ldots, x_m) &= (y_1, \ldots, y_n).
\end{aligned}$$

The induction hypothesis gives $\mathsf{length}(z) \geqslant \mathsf{min}(\mathsf{min}_{1 \leqslant i \leqslant m}(\delta^p_{i,n+1} + \mathsf{length}(x_i)), \delta^p_{m+1,n+1} + \mathsf{length}(z))$. The cycle sum condition means that $\delta^p_{m+1,n+1} > 0$, and so $\mathsf{length}(z) \geqslant \delta^p_{m+1,n+1} + \mathsf{length}(z)$ would be contradictory. Therefore $\mathsf{length}(z) \geqslant \mathsf{min}_{1 \leqslant i \leqslant m}(\delta^p_{i,n+1} + \mathsf{length}(x_i))$.

The induction hypothesis also gives, for any j,

$$\mathsf{length}(y_j) \geqslant \mathsf{min}(\mathsf{min}_{1 \leqslant i \leqslant m}(\delta^p_{i,j} + \mathsf{length}(x_i)), \delta^p_{m+1,j} + \mathsf{length}(z))$$

so we have

$$\begin{aligned}
\mathsf{length}(y_j) &\geqslant \mathsf{min}(\mathsf{min}_{1 \leqslant i \leqslant m}(\delta^p_{i,j} + \mathsf{length}(x_i)), \delta^p_{m+1,j} + \mathsf{min}_{1 \leqslant i \leqslant m}(\delta^p_{i,n+1} + \mathsf{length}(x_i))) \\
&= \mathsf{min}(\mathsf{min}_{1 \leqslant i \leqslant m}(\delta^p_{i,j} + \mathsf{length}(x_i)), \mathsf{min}_{1 \leqslant i \leqslant m}(\delta^p_{m+1,j} + \delta^p_{i,n+1} + \mathsf{length}(x_i))) \\
&= \mathsf{min}_{1 \leqslant i \leqslant m}(\mathsf{min}(\delta^p_{i,j}, \delta^p_{i,n+1} + \delta^p_{m+1,j}) + \mathsf{length}(x_i)) \\
&= \mathsf{min}_{1 \leqslant i \leqslant m}(\delta^{p\uparrow^1}_{i,j} + \mathsf{length}(x_i))
\end{aligned}$$

as required. □

Well-definedness of the synchronous semantics is an easy consequence.

Proposition 3.5 Suppose that every basic node of $\mathcal{D}$ is interpreted by a synchronous function, and that the cycle sum condition is observed. Then for every expression p, $[\![p]\!]_K$ is a synchronous function.

Proof. Let $p : A_1 \otimes \cdots \otimes A_m \to B_1 \otimes \cdots \otimes B_n$ with

$$[\![p]\!]_K(x_1, \ldots, x_m) = (y_1, \ldots, y_n)$$

and suppose that $\forall i.\mathsf{length}(x_i) \geqslant r$. By Proposition 3.4 we have

$$\forall j.\mathsf{length}(y_j) \geqslant \mathsf{min}_i(\delta^p_{i,j} + \mathsf{length}(x_i)).$$

Because each $\mathsf{length}(x_i) \geqslant r$ and each $\delta^p_{i,j} \geqslant 0$ we conclude that $\forall i.\mathsf{length}(y_i) \geqslant r$ as required. □

We can now give a semantic counterpart of the cycle sum condition.

Proposition 3.6 If $p : A_1 \otimes \cdots \otimes A_m \otimes C \to B_1 \otimes \cdots \otimes B_n \otimes C$ and $\delta^p_{m+1,n+1} > 0$ then $[\![p]\!]_K$ has output $n+1$ longer than input $m+1$.

Proof. Let $[\![p]\!]_K(x_1, \ldots, x_{m+1}) = (y_1, \ldots, y_{n+1})$ and assume $\mathsf{length}(x_{m+1}) \geqslant a$ and $\forall(1 \leqslant i \leqslant m).\mathsf{length}(x_i) \geqslant a+1$. We need to prove $\forall(1 \leqslant j \leqslant n+1).\mathsf{length}(y_j) \geqslant a+1$.

By Proposition 3.4 we have, for any j, $\mathsf{length}(y_j) \geqslant \mathsf{min}_i(\delta^p_{i,j} + \mathsf{length}(x_i))$. For $1 \leqslant i \leqslant m$ we have $\mathsf{length}(x_i) \geqslant a+1$ and $\delta^p_{i,j} \geqslant 0$. Also $\mathsf{length}(x_{m+1}) \geqslant a$ and $\delta^p_{m+1,n+1} > 0$. Therefore $\mathsf{length}(y_j) \geqslant a+1$. □

The next two results illustrate the way in which non-trivial delays, indicated semantically by outputs longer than inputs, are propagated and preserved by network constructions. A similar result holds for parallel composition.

Proposition 3.7 Let $f : A_1 \otimes \cdots \otimes A_n \to A$ and $g : B_1 \otimes \cdots \otimes B_m \to A_i$ be synchronous and let

$$h : A_1 \otimes \cdots \otimes A_{i-1} \otimes B_1 \otimes \cdots \otimes B_m \otimes A_{i+1} \otimes \cdots \otimes A_n \to A$$

be defined by

$$h(x_1, \ldots, x_{i-1}, y_1, \ldots, y_m, x_{i+1}, \ldots, x_n) = f(x_1, \ldots, x_{i-1}, g(y_1, \ldots, y_m), x_{i+1}, \ldots, x_n).$$

If f has output longer than inputs J then h has output longer than inputs J', where

$$J' = \{j \mid j \in J, j < i\} \cup \{j + m - 1 \mid j \in J, j > i\} \cup K$$

$$K = \begin{cases} \{i, i+1, \ldots, i+m-1\} & \text{if } i \in J \\ \varnothing & \text{if } i \notin J \end{cases}$$

If g has output longer than inputs J then h has output longer than inputs $J' = \{j + i - 1 \mid j \in J\}$.

Proof. Straightforward, by considering the lengths of the arguments of h, the relationship of the argument positions to J', and the properties of f and g. □

Proposition 3.8 If $p : A_1 \otimes \cdots \otimes A_m \otimes C \to B_1 \otimes \cdots \otimes B_n \otimes C$ and $[\![p]\!]_K$ has output $n+1$ longer than input $m+1$ and output i ($i \leqslant m$) longer than inputs $J \subseteq \{1, \ldots, m\}$, then $[\![p \uparrow^1]\!]_K$ has output i longer than inputs J.

Proof. $[\![p \uparrow^1]\!]_K(x_1, \ldots, x_m) = (y_1, \ldots, y_n)$ where

$$\begin{aligned} y_i &= \pi_i([\![p]\!]_K(x_1, \ldots, x_m, z)) \\ z &= \mathsf{fix}(\lambda y.(\pi_{n+1}([\![p]\!]_K(x_1, \ldots, x_m, y)))) \\ &= \bigsqcup_r [(\lambda y.(\pi_{n+1}([\![p]\!]_K(x_1, \ldots, x_m, y))))^r(\varepsilon)] \end{aligned}$$

We need to show that

$$\forall r.((\forall i \in J.\mathsf{length}(x_i) \geqslant r) \& (\forall i \notin J.\mathsf{length}(x_i) \geqslant r+1) \Rightarrow \mathsf{length}(y_i) \geqslant r+1)$$

Because $[\![p]\!]_K$ has output i longer than inputs J, it is sufficient to show that

$$\forall r.((\forall i \in J.\mathsf{length}(x_i) \geqslant r) \& (\forall i \notin J.\mathsf{length}(x_i) \geqslant r+1) \Rightarrow \mathsf{length}(z) \geqslant r)$$

In turn, it is sufficient to prove

$$\forall r.((\forall i \in J.\mathsf{length}(x_i) \geqslant r) \& (\forall i \notin J.\mathsf{length}(x_i) \geqslant r+1) \Rightarrow \mathsf{length}((\lambda y.(\pi_{n+1}([\![p]\!]_K(x_1, \ldots, x_m, y)))^r(\varepsilon))) \geqslant r)$$

which we prove by induction on r.

The base case is trivial. For the inductive step, assume

$$\begin{aligned} &\forall r.((\forall i \in J.\mathsf{length}(x_i) \geqslant r) \& (\forall i \notin J.\mathsf{length}(x_i) \geqslant r+1) \Rightarrow \\ &\qquad \mathsf{length}((\lambda y.(\pi_{n+1}([\![p]\!]_K(x_1, \ldots, x_m, y)))^r(\varepsilon))) \geqslant r) \\ &\forall i \in J.\mathsf{length}(x_i) \geqslant r+1 \\ &\forall i \notin J.\mathsf{length}(x_i) \geqslant r+2 \end{aligned}$$

and consider

$$(\lambda y.(\pi_{n+1}([\![p]\!]_K(x_1, \ldots, x_m, y))))^{r+1}(\varepsilon) = \pi_{n+1}([\![p]\!]_K(x_1, \ldots, x_m, (\lambda y.(\pi_{n+1}([\![p]\!]_K(x_1, \ldots, x_m, y)))))^r(\varepsilon))$$

Because $[\![p]\!]_K$ has output $n+1$ longer than input $m+1$, the induction hypothesis and the assumption that $\forall i.\mathsf{length}(x_i) \geqslant r+1$ imply that $\mathsf{length}((\lambda y.(\pi_{n+1}([\![p]\!]_K(x_1, \ldots, x_m, y))))^{r+1}(\varepsilon)) \geqslant r+1$). □

4. The Category $\mathcal{SP}roc$ of Synchronous Processes

$\mathcal{SP}roc$ is an example of an *interaction category* [AGN95, AGN96, Abr93, AGN99, Gay95, Nag98], a class of categories in which the morphisms are concurrent processes and the objects combine interface descriptions and behavioural specifications. Interaction categories have been used as a semantic framework for the specification and verification of safety and deadlock-freedom properties of concurrent systems. Their $*$-autonomous or compact closed structure means that they are well suited to describing concurrent systems with a fixed communication topology, such as dataflow networks. In the version of $\mathcal{SP}roc$ which we use in the present paper, the morphisms are synchronous processes and the objects specify alphabets of actions. The definitions rely on the notions of *labelled transition system*, *bisimulation*, and *transition rules* [Mil89].

Definition 4.1 The set ST_L of *synchronisation trees* with label-set L is defined by the recursive equation $\mathsf{ST}_L \cong \wp(L \times \mathsf{ST}_L)$.

The simplest way of solving the recursive equation in Definition 4.1 is to use Aczel's theory [Acz88] of non-well-founded sets. In this theory, ST_L is the final co-algebra of the functor $X \mapsto \wp(L \times X)$. The final co-algebra property supports non-well-founded recursive definitions of elements of ST_L, and provides a principle of co-induction. We work with a labelled transition system whose states are the elements of ST_L, and whose transitions are defined by the interpretation of a synchronisation tree as the set of transitions to its descendants: $P = \{(a, Q) \mid P \xrightarrow{a} Q\}$. Equality of synchronisation trees corresponds to bisimulation. We refer to an element of ST_L as a *process with alphabet* L.

Definition 4.2 The functions $\mathsf{traces} : \mathsf{ST}_A \to A^*$, $\mathsf{inftraces} : \mathsf{ST}_A \to A^\omega$ and $\mathsf{alltraces} : \mathsf{ST}_A \to A^\omega$ are defined as follows. Note that the first definition must be interpreted co-inductively so that the infinite traces are included.

$$\mathsf{alltraces}(P) \stackrel{\text{def}}{=} \{\varepsilon\} \cup \{a\sigma \mid P \xrightarrow{a} Q, \sigma \in \mathsf{alltraces}(Q)\}$$

$$\mathsf{traces}(P) \stackrel{\text{def}}{=} \{\sigma \in \mathsf{alltraces}(P) \mid \sigma \text{ is finite}\}$$

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

We now define $\mathcal{SP}roc$, the category of synchronous processes.

Definition 4.3 The objects of $\mathcal{SP}roc$ are sets, thought of as alphabets of actions for processes. If A is an object of $\mathcal{SP}roc$, a *process of type* A is an element of ST_A. If p is a process of type A we write $p : A$. Given objects A and B, the object $A \otimes B$ is defined by $A \otimes B = A \times B$. A morphism $p : A \to B$ of $\mathcal{SP}roc$ is a process p of type $A \otimes B$.

In the usual definition of $\mathcal{SP}roc$, objects incorporate safety specifications which morphisms are required to satisfy. In the present paper we do not make use of safety specifications and therefore we simplify the definitions by omitting them.

Definition 4.4 If $p : A \to B$ and $q : B \to C$ then the composite $p\,;q : A \to C$ is defined by the following transition rule:

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

During each transition, the actions being performed in the common type B are required to match. The processes being composed constrain each other's behaviour, selecting the possibilities which agree in B. Composition thus involves ongoing interaction or communication. The operation of composition corresponds, in standard process calculi, to a combination of parallel composition and restriction or hiding.

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 transition rule would reduce to precisely the definition of relational composition. $\mathcal{SP}roc$ processes can be viewed as *relations extended in time*.

Definition 4.5 The identity morphism $1_A : A \to A$ is defined by the following transition rule:

$$\frac{a \in A}{1_A \xrightarrow{(a,a)} 1_A}$$

Identity morphisms are synchronous buffers: whatever is received by $1_A : A \to A$ in the left copy of A is instantaneously transmitted to the right copy (and vice versa – there is no real directionality).

Proposition 4.6 *$\mathcal{SP}roc$ is a category.*

Proof. The necessary equations can be verified by bisimulation arguments. Full details can be found elsewhere [AGN95, AGN96, AGN99, Gay95, Nag98]. □

With a few additional definitions, *$\mathcal{SP}roc$* can be given the structure of a $*$-autonomous category [Bar79].

Definition 4.7 If $p : A \to C$ and $q : B \to D$ then the morphism $p \otimes q : A \otimes B \to C \otimes D$ is defined by the following transition rule.

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

The tensor unit object I is defined by $I = \{*\}$.

The contravariant functor $(-)^\perp$ is trivial on objects: $A^\perp \stackrel{\text{def}}{=} A$. If $p : A \to B$ then $p^\perp : B^\perp \to A^\perp$ is defined by the following transition rule.

$$\frac{p \xrightarrow{(a,c)} q}{p^\perp \xrightarrow{(c,a)} q^\perp}$$

The operation $\parr$ is the de Morgan dual of $\otimes$: $A \parr B = (A^\perp \otimes B^\perp)^\perp$. The linear implication $\multimap$ is defined by $A \multimap B = A^\perp \parr B$.

The operation $\otimes$ on processes is synchronous parallel composition. The following notation provides a useful way of defining the rest of the $*$-autonomous structure.

Definition 4.8 If P is a process with alphabet Σ, and $f : \Sigma \rightharpoonup \Sigma'$ is a partial function, then $P[f]$ is the process with alphabet Σ' defined by

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

Definition 4.9 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 $\mathsf{symm}_{A,B} : A \otimes B \to B \otimes A$ are defined as follows, where we use pattern-matching notation to define an appropriate partial function in each case.

$$\begin{aligned}
\mathsf{unitl}_A &= 1_A[(a, a) \mapsto ((*, a), a)] \\
\mathsf{unitr}_A &= 1_A[(a, a) \mapsto ((a, *), a)] \\
\mathsf{assoc}_{A,B,C} &= 1_{A\otimes(B\otimes C)}[((a, (b, c)), (a, (b, c))) \mapsto ((a, (b, c)), ((a, b), c))] \\
\mathsf{symm}_{A,B} &= 1_{A\otimes B}[((a, b), (a, b)) \mapsto ((a, b), (b, a))].
\end{aligned}$$

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

$$\Lambda(f) = f[((a, b), c) \mapsto (a, (b, c))].$$

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

$$\mathsf{Ap}_{A,B} = 1_{A\multimap B}[((a, b), (a, b)) \mapsto (((a, b), a), b)].$$

Proposition 4.10 *$\mathcal{SP}roc$ is a $*$-autonomous category.*

Proof. Because $(-)^{\perp}$ is trivial, it is only necessary to check that the structural isomorphisms (assoc, etc) satisfy the required coherence conditions, and that Λ and Ap satisfy the conditions required for the adjunction between $\otimes$ and $\multimap$ to hold. This is straightforward because all the relevant morphisms are defined by relabelling identity morphisms; essentially, the requirements reduce to facts about associativity of cartesian product on sets. □

For any permutation σ on $\{1, \ldots, n\}$ and any objects $A_1, \ldots, A_n$, there is a canonical isomorphism $A_1 \otimes \ldots \otimes A_n \cong A_{\sigma(1)} \otimes \ldots \otimes A_{\sigma(n)}$.

A *compact closed* category is a $*$-autonomous category in which there are canonical isomorphisms $A \otimes B \cong A \parr B$. $\mathcal{SP}roc$ is compact closed because $(-)^{\perp}$ is defined to be the identity on objects and hence $A \otimes B$ and $A \parr B$ are equal. Every compact closed category has a *trace* [JSV96], which means that given a morphism $p : A \otimes C \to B \otimes C$ there is a morphism $p \uparrow^1 : A \to B$, such that the operation $\uparrow^1$ satisfies the axioms of Section 2. The generic definition of the trace is as follows. Given $p : A \otimes C \to B \otimes C$ we have $\Lambda(p) : A \to C \multimap (B \otimes C)$. Using the definition of $\multimap$ and the isomorphism between $\otimes$ and $\parr$ (twice), we obtain $\Lambda(p) ; \alpha : A \to (C \multimap B) \otimes C$, where α is a canonical isomorphism. Then $p \uparrow^1 = \Lambda(p) ; \alpha ; \mathsf{Ap}_{C,B}$. In the case of $\mathcal{SP}roc$ the generic definition corresponds to:

Definition 4.11 If $p : A \otimes C \to B \otimes C$ in $\mathcal{SP}roc$ then the morphism $p \uparrow^1 : A \to B$ is defined by the following transition rule:

$$\frac{p \xrightarrow{(a,c,b,c)} q}{p \uparrow^1 \xrightarrow{(a,b)} q \uparrow^1}$$

We will use the specific definition later for proofs about $\uparrow^1$, but it will be useful to bear in mind the generic definition; it will emphasise the compositional nature of our semantic definitions.

For our later work, it is useful to define some morphisms which exist in $\mathcal{SP}roc$ but are not part of the $*$-autonomous structure.

Definition 4.12 For each $1 \leqslant i \leqslant n$ and any $A_1, \ldots, A_n$ the *projection morphism* $\pi_i : A_1 \otimes \cdots \otimes A_n \to A_i$ is defined by $\pi_i = 1_{A_1 \otimes \cdots \otimes A_n}[\pi_i]$, where the π_i in the relabelling operation is the set-theoretic projection function.

For any A and any n, the *copying morphism* $\mathsf{fork}_A^{(n)} : A \to A \otimes \cdots \otimes A$ (with n copies of A on the right) is defined by $\mathsf{fork}_A^{(n)} = 1_A[(a, a) \mapsto (a, (a, \ldots, a))]$.

For any A the *sink morphism* $\mathsf{sink}_A : A \to I$ is defined by $\mathsf{sink}_A = 1_A[(a, a) \mapsto (a, *)]$.

The notions of *functionality* and *determinism* of processes are defined coinductively.

Definition 4.13 The set of *functional* processes of type $A \to B$ is the largest set F of processes of type $A \to B$ such that for every $p \in F$ the following conditions hold:

1. If $p \xrightarrow{(a,b)} q$ and $p \xrightarrow{(a,b')} q'$ then $b = b'$.
2. For each $a \in A$ there exists $b \in B$ and q such that $p \xrightarrow{(a,b)} q$ (the *receptivity* condition).
3. If $p \xrightarrow{(a,b)} q$ then $q \in F$.

Definition 4.14 The set of *deterministic* processes of type A is the largest set D of processes of type A such that for every $p \in D$ the following conditions hold.

1. If $p \xrightarrow{a} q$ and $p \xrightarrow{a} q'$ then $q = q'$.
2. If $p \xrightarrow{a} q$ then $q \in D$.

The notion of a process having certain outputs *instantaneously independent* of certain inputs is the $\mathcal{SP}roc$ analogue of the notion that a synchronous function has certain outputs longer than certain inputs. The output values are independent of the input values received during the same step, but may depend on previous values of the inputs. The definition is co-inductive.

Definition 4.15 Let $I \subseteq \{1, \ldots, n\}$ and $J \subseteq \{1, \ldots, m\}$. The set of processes of type $A_1 \otimes \cdots \otimes A_m \to B_1 \otimes \cdots \otimes B_n$ with *outputs I instantaneously independent of inputs J* is the largest set S of processes of type $A_1 \otimes \cdots \otimes A_m \to B_1 \otimes \cdots \otimes B_n$ such that for every $p \in S$ the following conditions hold.

1. If $p \xrightarrow{(a_1,\ldots,a_m,b_1,\ldots,b_n)} q$ and $p \xrightarrow{(a'_1,\ldots,a'_m,b'_1,\ldots,b'_n)} q$ and $\forall j \notin J.a_j = a'_j$ then $\forall i \in I.b_i = b'_i$.
2. If $p \xrightarrow{(a_1,\ldots,a_m,b_1,\ldots,b_n)} q$ then $q \in S$.

Proposition 4.16 Suppose that $p, q : A \to B_1 \otimes \cdots \otimes B_n$ are functional and deterministic, and that $p ; \pi_i = q ; \pi_i$ for each $1 \leqslant i \leqslant n$. Then $p = q$.

Proof. We show that the set $R = \{(p, q) \mid$ satisfying the conditions $\}$ is a bisimulation. Let $(p, q) \in R$ and consider a transition $p \xrightarrow{(a,b_1,\ldots,b_n)} p'$. For each i there is a corresponding transition $p ; \pi_i \xrightarrow{(a,b_i)} p' ; \pi_i$. Hence for each i there is a transition $q ; \pi_i \xrightarrow{(a,b_i)} q'_i ; \pi_i$. These transitions of the $q ; \pi_i$ must be derived from a collection of transitions $q \xrightarrow{(a,b_{i1},\ldots,b_{in})} q'_i$ where for each i, $b_{ii} = b_i$. Because q is functional, the values of the tuples $(b_{i1}, \ldots, b_{in})$ must be independent of i, and therefore there is a collection of transitions $q \xrightarrow{(a,b_1,\ldots,b_n)} q'_i$. Because q is deterministic, all the q'_i are equal, to q' say. Furthermore, $p' ; \pi_i = q' ; \pi_i$ for each i, and p' and q' are functional and deterministic, so $(p', q') \in R$. A symmetrical argument shows that p can match transitions of q, completing the bisimulation proof. □

Proposition 4.17 If $p : A \to B$ and $q : B \to C$ are functional and deterministic, then $p ; q$ is functional and deterministic.

Proof. We show that the set

$$X = \{p ; q \mid p : A \to B \text{ and } q : B \to C \text{ are functional and deterministic}\}$$

satisfies the conditions of Definitions 4.13 and 4.14 and is therefore a subset of the set of functional and deterministic processes.

Suppose $p ; q \xrightarrow{(a,c)} r$ and $p ; q \xrightarrow{(a,c')} r'$. Then there are transitions $p \xrightarrow{(a,b)} p'$, $q \xrightarrow{(b,c)} q'$, $p \xrightarrow{(a,b')} p''$, $q \xrightarrow{(b',c')} q''$ with $r = p' ; q'$ and $r' = p'' ; q''$.

Because p is functional and deterministic, $b' = b$ and $p'' = p'$. Therefore, and because q is functional and deterministic, $c' = c$ and $q'' = q'$. Hence $r' = r$. Also, p' and q' are functional and deterministic, so $p' ; q' \in X$. Receptivity of $p ; q$ follows from receptivity of p and q. Hence, by co-induction, all processes in X are functional and deterministic. □

Proposition 4.18 If $p : A \to B$ and $q : C \to D$ are functional and deterministic, then $p \otimes q$ is functional and deterministic.

Proof. Similar to the proof of Proposition 4.17. □

Proposition 4.19 If $p : A \otimes C \to B \otimes C$ is functional and deterministic and has output C instantaneously independent of input C, then $p \uparrow^1$ is functional and deterministic.

Proof. We use a co-inductive argument similar to the proof of Proposition 4.17. Suppose $p \uparrow^1 \xrightarrow{(a,b)} q$ and $p \uparrow^1 \xrightarrow{(a,b')} q'$. These transitions are derived from transitions $p \xrightarrow{(a,c,b,c)} r$ and $p \xrightarrow{(a,c',b',c')} r'$ for some c and c', with $q = r \uparrow^1$ and $q' = r' \uparrow^1$. Because p has output C instantaneously independent of input C, $c' = c$. Because p is functional and deterministic, $b' = b$ and $r' = r$. For receptivity, let $a \in A, c \in C$. Receptivity of p implies that there exist b, c', q with $p \xrightarrow{(a,c,b,c')} q$. By receptivity again, there exist b', c'', q' such that $p \xrightarrow{(a,c',b',c'')} q'$. Because p has output C instantaneously independent of input C, $c'' = c'$ and therefore $p \xrightarrow{(a,c',b',c')} q'$ which yields a transition $p \uparrow^1 \xrightarrow{(a,b')} q' \uparrow^1$. This completes the coinductive proof. □

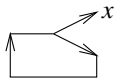


Fig. 4. A trivial network.

5. The $\mathcal{SP}roc$ Semantics of $\mathcal{D}$

We define the semantics $[\![\cdot]\!]_S$ of $\mathcal{D}$ in $\mathcal{SP}roc$. Each ground type is interpreted by an object. An expression $p : A_1 \otimes \cdots \otimes A_m \to B_1 \otimes \cdots \otimes B_n$ is interpreted by a process

$$[\![p]\!]_S : A_1 \otimes \cdots \otimes A_m \to B_1 \otimes \cdots \otimes B_n.$$

Once interpretations have been specified for the basic nodes, the semantics is defined for other expressions as follows:

- Identities: $[\![\mathsf{I}_m]\!]_S$ is an appropriate identity morphism.
- Transpositions: $[\![{}^m\mathsf{X}^n]\!]_S$ is an appropriate permutation morphism.
- Forks: $[\![\wedge^n]\!]_S$ is an appropriate $\mathsf{fork}^{(2)}$ morphism.
- Sinks: $[\![\bot^n]\!]_S$ is an appropriate sink morphism.
- Composition: $[\![p\,;q]\!]_S = [\![p]\!]_S\,;[\![q]\!]_S$.
- Parallel: $[\![p \otimes q]\!]_S = [\![p]\!]_S \otimes [\![q]\!]_S$.
- Feedback: $[\![p\uparrow^1]\!]_S = [\![p]\!]_S\uparrow^1$.

Proposition 5.1 If the interpretations of the basic nodes are functional and deterministic, then for every $\mathcal{D}$ expression p, $[\![p]\!]_S$ is functional and deterministic.

Proof. Follows from Propositions 4.17, 4.18, 4.19 and the straightforward checks that identity, permutation, copying and sink morphisms are functional and deterministic. □

Proposition 5.2 The $\mathcal{SP}roc$ semantics validates the axioms of $\mathcal{D}$, up to associativity and unit isomorphisms.

Proof. Because $\mathcal{SP}roc$ is not strict monoidal, it is necessary to work up to associativity and unit isomorphisms. The axioms are verified by straightforward bisimulation arguments. Note that (26) relies on receptivity (Definition 4.13) of f.

The difference between the Kahn semantics and the $\mathcal{SP}roc$ semantics can be seen in their treatment of feedback loops. The Kahn semantics constructs streams token by token, driven by nodes (if present) which produce output independently of their input. The $\mathcal{SP}roc$ semantics yields the maximal behaviour which is consistent with the constraints imposed by the network connections. This difference can be illustrated by the $\mathcal{D}$ expression $\wedge^1\uparrow^1$, corresponding to the network shown in Fig. 4. This is a feedback loop in which no data is generated; it does not satisfy the cycle sum condition and is not a valid synchronous network. In the Kahn semantics, the only equation determining the output x is $x = x$. Solving by least fixed points gives $x = \varepsilon$, i.e. no output is produced and the network is deadlocked. In the $\mathcal{SP}roc$ semantics the only constraint imposed by the connections is $x = x$, and as a result its semantics non-deterministically generates all possible streams.

It is only in situations like this, which are under-constrained and as a result are deadlocked in the Kahn semantics, that the two semantics differ. In the next section we show that, as long as the cycle sum condition is satisfied, the two semantics are equivalent.

6. Equivalence of the Synchronous Kahn Semantics and the $\mathcal{SP}roc$ Semantics

In principle the $\mathcal{SP}roc$ semantics is more general than the synchronous Kahn semantics, because the processes interpreting the basic nodes need not have functional behaviour. However, interpreting the basic nodes by non-functional processes would yield a semantics in which the behaviour of networks would be radically different from the behaviour specified by the standard Kahn semantics. In this section we define what it means for an $\mathcal{SP}roc$ process to compute a synchronous function, and prove that, given suitable interpretations of the basic nodes, the process resulting from the $\mathcal{SP}roc$ semantics of a network computes the function resulting from its

Kahn semantics. Notation: we write π^* and π^ω to denote the extensions of projection functions to finite and infinite traces respectively.

Definition 6.1 A process $p : A_1 \otimes \cdots \otimes A_m \to A$ *computes* the continuous function $f : A_1^\omega \times \cdots \times A_m^\omega \to A^\omega$ if the following conditions hold:

1. Safety: $\forall \sigma \in \mathsf{traces}(p), \pi^*_{m+1}(\sigma) \sqsubseteq f(\pi^*_1(\sigma), \ldots, \pi^*_m(\sigma))$
2. Liveness: $\forall \sigma \in \mathsf{inftraces}(p), \pi^\omega_{m+1}(\sigma) = f(\pi^\omega_1(\sigma), \ldots, \pi^\omega_m(\sigma))$
3. Totality: $\forall \tau \in A_1^\omega \times \cdots \times A_m^\omega, \exists \sigma \in \mathsf{traces}(p).\langle \pi^\omega_1(\sigma), \ldots, \pi^\omega_m(\sigma)\rangle = \tau$.

In other words, a process computes a function if for any finite input it doesn't output anything which could not be produced by the function (safety); for any infinite input it outputs exactly the value which the function would (liveness); and it can respond to all traces in the function's domain (totality).

Definition 6.2 A process $p : A_1 \otimes \cdots \otimes A_m \to B_1 \otimes \cdots \otimes B_n$ *computes* the continuous function $f : A_1^\omega \times \cdots \times A_m^\omega \to B_1^\omega \times \cdots \times B_n^\omega$ if for each $1 \leqslant i \leqslant n$, the process $p\,;\pi_i$ computes the function $f\,;\pi_i$.

Definition 6.3 For each synchronous function $f : A_1^\omega \times \cdots \times A_m^\omega \to A^\omega$ we define the process $f : A_1 \otimes \cdots \otimes A_m \to B_1 \otimes \cdots \otimes B_n$. The family of processes $[\![f]\!]_S$ for synchronous functions f of this type defined by the transition rule

$$\frac{(a_1, \ldots, a_m) \in A_1 \times \cdots \times A_m}{[\![f]\!]_S \xrightarrow{(a_1,\ldots,a_m,\mathsf{hd}(f(a_1,\ldots,a_m)))} [\![\lambda(x_1, \ldots, x_m).\mathsf{tail}(f(a_1x_1, \ldots, a_mx_m))]\!]_S}$$

Because f is synchronous, $f(a_1, \ldots, a_m) \neq \varepsilon$ and so $\mathsf{hd}(f(a_1, \ldots, a_m))$ is always defined; furthermore, $\lambda(x_1, \ldots, x_m).\mathsf{tail}(f(a_1x_1, \ldots, a_mx_m))$ is also synchronous. We use the notation $[\![\cdot]\!]_S$ both for the $\mathcal{SP}roc$ semantics of $\mathcal{D}$ and for the processes defined here from synchronous functions.

Proposition 6.4 If $f : A_1^\omega \times \cdots \times A_m^\omega \to A^\omega$ is a synchronous function then $[\![f]\!]_S$ is functional and deterministic.

Proof. It is clear from the definition that the initial input to $[\![f]\!]_S$ determines both the initial output and the subsequent state, and that $[\![f]\!]_S$ is receptive; as the subsequent state is $[\![g]\!]_S$ for a particular g, a co-inductive argument completes the proof. □

We now prove that $[\![f]\!]_S$ not only computes f but is uniquely defined by this property.

Lemma 6.5 Let $f : A_1^\omega \times \cdots \times A_m^\omega \to A^\omega$ be a synchronous function. If $[\![f]\!]_S$ performs the trace

$$(a_{11}, \ldots, a_{1m}, b_1)(a_{21}, \ldots, a_{2m}, b_2)\ldots(a_{n1}, \ldots, a_{nm}, b_n)$$

then the resulting process is $[\![g]\!]_S$ where for any $x_1, \ldots, x_m$, g satisfies the equation

$$f(a_{11}\ldots a_{n1}x_1, \ldots, a_{1m}\ldots a_{nm}x_m) = b_1\ldots b_n g(x_1, \ldots, x_m)$$

Proof. By induction on n. The base case ($n = 0$) is trivial, taking $g = f$. For the inductive step, assume the result for traces shorter than n (where $n \geqslant 1$) and all functions of the type of f. By the definition of $[\![f]\!]_S$, the first action $(a_{11}, \ldots, a_{1m}, b_1)$ leads to the state $[\![g]\!]_S$ where

$$g = \lambda(x_1, \ldots, x_m).\mathsf{tail}(f(a_{11}x_1, \ldots, a_{1m}x_m))$$

and we have $b_1 = \mathsf{hd}(f(a_{11}, \ldots, a_{1m}))$. By the induction hypothesis, the next $n-1$ steps lead to the state $[\![h]\!]_S$ where

$$g(a_{21}\ldots a_{n1}x_1, \ldots, a_{2m}\ldots a_{nm}x_m) = b_2\ldots b_n h(x_1, \ldots, x_m)$$

Using the definition of g we get

$$\mathsf{tail}(f(a_{11}\ldots a_{n1}x_1, \ldots, a_{1m}\ldots a_{nm}x_m)) = b_2\ldots b_n h(x_1, \ldots, x_m)$$

which, together with the fact that $b_1 = \mathsf{hd}(f(a_{11}, \ldots, a_{1m}))$, yields the desired result. □

Proposition 6.6 Let $f : A_1^\omega \times \cdots \times A_m^\omega \to A^\omega$ be a synchronous function. Then $[\![f]\!]_S$ computes f.

Proof. Consider the three conditions in turn:

1. Safety follows from Lemma 6.5 by taking $x_1 = \ldots = x_m = \varepsilon$.
2. Liveness: let x be an infinite trace of $[\![f]\!]_S$. For each $1 \leqslant i \leqslant m$ let $x_i = \pi_i^*(x)$, and let $y = \pi_{m+1}^*(x)$. For each i, let (x_{ij}) be the chain of finite prefixes of x_i, so that $x_i = \bigsqcup_j x_{ij}$. Similarly let (y_j) be the chain of finite prefixes of y. By continuity $f(x_1, \ldots, x_m) = \bigsqcup_j f(x_{1j}, \ldots, x_{mj})$, and each $f(x_{1j}, \ldots, x_{mj}) \sqsubseteq y$. So $f(x_1, \ldots, x_m) \sqsubseteq y$. By safety, for each j, $y_j \sqsubseteq f(x_{1j}, \ldots, x_{mj})$, hence $y = \bigsqcup_j y_j \sqsubseteq \bigsqcup_j f(x_{1j}, \ldots, x_{mj}) = f(x_1, \ldots, x_m)$. Therefore $y = f(x_1, \ldots, x_m)$.
3. Totality follows from the fact that in the definition of $[\![f]\!]_S$, the input actions at the first step range over the whole of $A_1 \times \cdots \times A_m$. □

Proposition 6.7 Let $f : A_1^\omega \times \cdots \times A_m^\omega \to A^\omega$ be a synchronous function. Then $[\![f]\!]_S$ is the unique process which computes f.

Proof. We prove that the set $R = \{(p, [\![g]\!]_S) \mid p \text{ computes } g\}$ is a bisimulation, where g ranges over all synchronous functions with the same types as f, and p ranges over all processes with the corresponding $\mathcal{SP}roc$ type. This shows that any process which computes f is equal to $[\![f]\!]_S$.

Let $(p, [\![g]\!]_S) \in R$ and consider a transition $p \xrightarrow{(a_1, \ldots, a_m, b)} p'$. Because p computes g, safety implies that $b = \mathsf{hd}(g(a_1, \ldots, a_m))$. By the definition of $[\![g]\!]_S$ there is a matching transition

$$[\![g]\!]_S \xrightarrow{(a_1, \ldots, a_m, b)} [\![\lambda(x_1, \ldots, x_m).\mathsf{tail}(g(a_1 x_1, \ldots, a_m x_m))]\!]_S$$

Conversely, any transition by $[\![g]\!]_S$ is of this form, and totality implies that there is a transition $p \xrightarrow{(a_1, \ldots, a_m, c)} p''$ for some c; safety implies that $c = b$.

It remains to show that if p computes g and $p \xrightarrow{(a_1, \ldots, a_m, b)} q$ then q computes

$$\lambda(x_1, \ldots, x_m).\mathsf{tail}(g(a_1 x_1, \ldots, a_m x_m))$$

We consider the three conditions:

1. Safety: if x is a finite trace of q then $(a_1, \ldots, a_m, b)x$ is a trace of p. By safety of p,

$$\begin{aligned} b\pi_{m+1}^*(x) &\sqsubseteq g(a_1\pi_1^*(x), \ldots, a_m\pi_m^*(x)) \\ &= b\mathsf{tail}(g(a_1\pi_1^*(x), \ldots, a_m\pi_m^*(x))) \end{aligned}$$

   and so

$$\pi_{m+1}^*(x) \sqsubseteq (\lambda(x_1, \ldots, x_m).\mathsf{tail}(g(a_1 x_1, \ldots, a_m x_m)))(\pi_1^*(x), \ldots, \pi_m^*(x))$$

2. Liveness follows from the same argument applied to an infinite trace of p.
3. Totality follows from totality of p. □

We now extend Definition 6.3 and Proposition 6.7 to functions whose outputs have arbitrary product types. The nullary case, $\bot$, causes a complication. According to Definition 6.2 with $n = 0$, any process $A_1 \otimes \cdots \otimes A_m \to I$ computes $\bot^m$. To obtain a general uniqueness result, we must restrict attention to functional and deterministic processes.

Definition 6.8 If $f : A_1^\omega \times \cdots \times A_m^\omega \to B_1^\omega \times \cdots \times B_n^\omega$ is a synchronous function, then the process $[\![f]\!]_S : A_1 \otimes \cdots \otimes A_m \to B_1 \otimes \cdots \otimes B_n$ is defined by

$$[\![f]\!]_S = \mathsf{fork}^{(n)}_{A_1 \otimes \cdots \otimes A_m} ; ([\![f ; \pi_1]\!]_S \otimes \cdots \otimes [\![f ; \pi_n]\!]_S)$$

If $n = 1$, Definition 6.8 reduces to Definition 6.3.

Proposition 6.9 The process $[\![f]\!]_S$ is the unique functional and deterministic process computing the function $f : A_1^\omega \times \cdots \times A_m^\omega \to B_1^\omega \times \cdots \times B_n^\omega$.

Proof. If $n = 0$ then $\mathsf{fork}^{(0)} = \mathsf{sink}$ and $[\![f ; \pi_1]\!]_S \otimes \cdots \otimes [\![f ; \pi_n]\!]_S = 1_I$; it is easy to see that $\mathsf{sink}_{A_1 \otimes \cdots \otimes A_m}$ is the unique functional and deterministic process of its type.

Now assume that $n > 0$. By Definition 6.8, to show that $[\![f]\!]_S$ computes f we must show that for each i, $[\![f]\!]_S ; \pi_i$ computes $f ; \pi_i$. The definitions of $\otimes$, $\mathsf{fork}^{(n)}$ and π_i in $\mathcal{SP}roc$ imply that $[\![f]\!]_S ; \pi_i = [\![f ; \pi_i]\!]_S$, and by Proposition 6.6, this process computes $f ; \pi_i$.

Propositions 4.17 and 4.18, and the definitions of copying and projection morphisms, imply that $[\![f]\!]_S$ is functional and deterministic.

For uniqueness, suppose that p is functional and deterministic and also computes f. By Proposition 4.16, to show that $p = [\![f]\!]_S$ it is sufficient to show that $p ; \pi_i = [\![f]\!]_S ; \pi_i$ for each i. As above, $[\![f]\!]_S ; \pi_i = [\![f ; \pi_i]\!]_S$, which computes $f ; \pi_i$. By Definition 6.8, $p ; \pi_i$ also computes $f ; \pi_i$. Hence by Proposition 6.7, $p ; \pi_i = [\![f]\!]_S ; \pi_i$ as required. □

The next two lemmas lead to the proof of equivalence of the $\mathcal{SP}roc$ and Kahn semantics of $\mathcal{D}$. Lemma 6.10 is straightforward, but Lemma 6.11 is the key to the equivalence result, as it relates the differing ways in which the two semantics model feedback.

Lemma 6.10 Let $f : A_1^\omega \times \cdots \times A_m^\omega \to B_1^\omega \times \cdots \times B_n^\omega$ and $g : B_1^\omega \times \cdots \times B_n^\omega \to C_1^\omega \times \cdots \times C_r^\omega$ be synchronous functions. Then $[\![f ; g]\!]_S = [\![f]\!]_S ; [\![g]\!]_S$.

Proof. We prove by co-induction that the set

$$R = \{([\![f]\!]_S ; [\![g]\!]_S, [\![f ; g]\!]_S) \mid \text{ satisfying the conditions } \}$$

is a bisimulation. An initial action of $[\![f]\!]_S ; [\![g]\!]_S$ is of the form $(a_1, \ldots, a_m, c_1, \ldots, c_r)$ where

$$c_i = \mathsf{hd}(\pi_i(g(\mathsf{hd}(\pi_1(f(a_1, \ldots, a_m))), \ldots, \mathsf{hd}(\pi_n(f(a_1, \ldots, a_m)))))).$$

An initial action of $[\![f ; g]\!]_S$ is of the form $(a_1, \ldots, a_m, c'_1, \ldots, c'_r)$ where

$$c'_i = \mathsf{hd}(\pi_i(g(f(a_1, \ldots, a_m))))$$

Given $a_1, \ldots, a_m$, continuity of g means that $c_i = c'_i$ for each i. Together with totality of the processes resulting from $[\![\cdot]\!]_S$, this means that $[\![f]\!]_S ; [\![g]\!]_S$ and $[\![f ; g]\!]_S$ can match each other's actions; from the definition of $[\![\cdot]\!]_S$, the pair of subsequent states is again in R. □

Lemma 6.11 Let $p : A_1 \otimes \cdots \otimes A_m \otimes C \to B_1 \otimes \cdots \otimes B_n \otimes C$ be a $\mathcal{D}$ expression such that $[\![p]\!]_K$ has output $n+1$ longer than input $m+1$, and assume that $[\![p]\!]_S = [\![[\![p]\!]_K]\!]_S$. Then $[\![p \uparrow^1]\!]_S = [\![[\![p \uparrow^1]\!]_K]\!]_S$.

Proof. From the definition of $[\![\cdot]\!]_K$, we have $[\![p \uparrow^1]\!]_K(x_1, \ldots, x_m) = (y_1, \ldots, y_n)$ where

$$\begin{aligned} y_i &= \pi_i([\![p]\!]_K(x_1, \ldots, x_m, z)) \\ z &= \mathsf{fix}(\lambda y.(\pi_{n+1}([\![p]\!]_K(x_1, \ldots, x_m, y)))) \end{aligned}$$

To show that $[\![p \uparrow^1]\!]_S = [\![[\![p \uparrow^1]\!]_K]\!]_S$ it is sufficient (by Proposition 6.9) to show that $[\![p \uparrow^1]\!]_S$ computes $[\![p \uparrow^1]\!]_K$. According to Definition 6.8, this means showing that for each i, $[\![p \uparrow^1]\!]_S ; \pi_i$ computes $[\![p \uparrow^1]\!]_K ; \pi_i$.

Safety: let σ be a finite trace of $[\![p \uparrow^1]\!]_S ; \pi_i$ with $\mathsf{length}(\sigma) = r$. The trace σ is derived from a trace τ of $[\![p]\!]_S$, also of length r, such that $\pi^*_{m+1}(\tau) = \pi^*_{m+n+2}(\tau)$ (i.e. the projections of τ onto the C input and the C output are equal). Let $\gamma = c_1 \ldots c_r = \pi^*_{m+1}(\tau)$, let $\beta = b_1 \ldots b_r = \pi^*_{m+i+1}$ be the projection of τ onto the B_i output, and for each j, let $\alpha_j = a_{j1} \ldots a_{jr} = \pi^*_j(\tau)$ be the projection of τ onto the A_j input. We need to show that $\beta \sqsubseteq \pi_i([\![p \uparrow^1]\!]_K(\alpha_1, \ldots, \alpha_m))$.

Given $x_1, \ldots, x_m$, define the function s by

$$s(x) = \pi_{n+1}([\![p]\!]_K(x_1, \ldots, x_m, x))$$

so that $[\![p \uparrow^1]\!]_K(x_1, \ldots, x_m) = [\![p]\!]_K(x_1, \ldots, x_m, \mathsf{fix}(s))$. We prove that for $t \leqslant r$, $c_1 \ldots c_t \sqsubseteq s^t(\varepsilon)$, by induction on t. The base case is trivial. For the inductive step, we have

$$c_1 \ldots c_{t+1} \sqsubseteq \pi_{n+1}([\![p]\!]_K(a_{11} \ldots a_{1(t+1)}, \ldots, a_{m1} \ldots a_{m(t+1)}, c_1 \ldots c_{t+1}))$$

by the safety clause of the fact that $[\![p]\!]_S ; \pi_{n+1}$ computes $[\![p]\!]_K ; \pi_{n+1}$. Because $[\![p]\!]_K$ has output $n+1$ longer than input $m+1$, an output of length $t+1$ in position $n+1$ is generated by supplying only $c_1 \ldots c_t$ as input $m+1$. Hence

$$\begin{aligned} c_1 \ldots c_{t+1} &\sqsubseteq \pi_{n+1}([\![p]\!]_K(a_{11} \ldots a_{1(t+1)}, \ldots, a_{m1} \ldots a_{m(t+1)}, c_1 \ldots c_t)) \\ &\sqsubseteq \pi_{n+1}([\![p]\!]_K(\alpha_1, \ldots, \alpha_m, c_1 \ldots c_t)) && \text{by continuity} \\ &\sqsubseteq \pi_{n+1}([\![p]\!]_K(\alpha_1, \ldots, \alpha_m, s^t(\varepsilon))) && \text{by the induction hypothesis} \\ &= s^{t+1}(\varepsilon). \end{aligned}$$

Therefore $\gamma \sqsubseteq \mathsf{fix}(s)$. Hence

$$\begin{array}{lll} \beta & \sqsubseteq & \pi_i([\![p]\!]_K(\alpha_1, \ldots, \alpha_m, \gamma)) \qquad \text{by safety for } [\![p]\!]_S ; \pi_i \\ & \sqsubseteq & \pi_i([\![p]\!]_K(\alpha_1, \ldots, \alpha_m, \mathsf{fix}(s)) \qquad \text{by continuity} \\ & = & \pi_i([\![p \uparrow^1]\!]_K(\alpha_1, \ldots, \alpha_m)) \end{array}$$

as required.

Liveness: consider traces and their projections as for safety, but now the traces are infinite. We need to show that $\beta = \pi_i([\![p \uparrow^1]\!]_K(\alpha_1, \ldots, \alpha_m))$. By liveness for $[\![p]\!]_S ; \pi_{n+1}$ we have $\gamma = \pi_{n+1}([\![p]\!]_K(\alpha_1, \ldots, \alpha_m, \gamma))$ and so $\mathsf{fix}(s) \sqsubseteq \gamma$. Because $[\![p]\!]_K$ has output $n+1$ longer than input $m+1$, it is easy to show (by induction on r) that for all r, $\mathsf{length}(s^r) \geqslant r$. Hence $\mathsf{fix}(s)$ is infinite, and so $\mathsf{fix}(s) = \gamma$. Liveness for $[\![p]\!]_S ; \pi_i$ then implies that $\beta = \pi_i([\![p]\!]_K(\alpha_1, \ldots, \alpha_m, \gamma)) = \pi_i([\![p \uparrow^1]\!]_K(\alpha_1, \ldots, \alpha_m))$.

Totality: let σ be an infinite trace in the type $A_1 \otimes \cdots \otimes A_m$, with projections α_j in the types A_j. Define the infinite trace $\gamma = c_1 c_2 \ldots$ in the type C: take c_i to be the ith element of $\pi_{n+1}([\![p]\!]_K(\alpha_1, \ldots, \alpha_m, c_1 \ldots c_{i-1}))$ (and $c_1 = \mathsf{hd}(\pi_{n+1}([\![p]\!]_K(\alpha_1, \ldots, \alpha_m, \varepsilon)))$), which is well defined because $[\![p]\!]_K$ has output $n+1$ longer than input $m+1$. Let the infinite trace τ in the type $A_1 \otimes \cdots \otimes A_m \otimes C$ have projections σ in $A_1 \otimes \cdots \otimes A_m$ and γ in C. By totality, $[\![p]\!]_S$ has a trace whose projection in the input types is τ and which has equal projections in the input and output C types. This trace yields a trace of $[\![p \uparrow^1]\!]_S$ whose projection in the input types is σ. □

Theorem 6.12 Consider the synchronous Kahn semantics with a particular interpretation $[\![f]\!]_K$ of each basic node f, and the $\mathcal{SP}roc$ semantics with each basic node f interpreted by $[\![[\![f]\!]_K]\!]_S$. Then for every $\mathcal{D}$ expression p which obeys the cycle condition, $[\![p]\!]_S = [\![[\![p]\!]_K]\!]_S$.

Proof. By induction on the structure of p, the base case (basic nodes) follows directly from the hypothesis. The inductive cases are as follows.

- Identities, transpositions, forks, sinks: straightforward from the definitions.
- Composition, parallel: straightforward, using Lemma 6.10 for composition.
- Feedback: follows from Lemma 6.11. □

7. Conclusions

We have defined two semantic models of an idealised synchronous dataflow language. The first semantics is a synchronous version of the classical Kahn semantics. The second semantics, which is new, uses the category $\mathcal{SP}roc$ of synchronous processes. The essential difference between the two semantic models is the way in which connections are treated. In the Kahn semantics, non-cyclic connections are modelled by functional composition, and cyclic connections are modelled by a least fixed point construction. In the $\mathcal{SP}roc$ semantics, all connections are modelled by constructing the maximal behaviour which is consistent with the constraints represented by the connections.

Our semantic models are based on two very different views of the meaning of network connections. To establish the relationship between these views, we have proved, for networks which obey Wadge's cycle sum condition, that the models yield equivalent network behaviours, in a sense which we make precise. The $\mathcal{SP}roc$ semantics therefore provides a new way of understanding the behaviour of dataflow programs, by describing the behaviour of cyclic networks in a different way, but is consistent with the classical model.

Our model is related to but quite different from the synchronous dataflow model of Lee and Messerschmitt [LeM87]. It would be interesting to see how the two models compare. The programming languages SIGNAL and LUSTRE are based on synchronous dataflow, but go beyond our idealised language in their use of complex clock structures to describe the relative timing behaviour of different data streams. Although it is possible to define a semantics in $\mathcal{SP}roc$ of clocked dataflow languages, extending our semantic equivalence results to this case is a topic for future work.

Acknowledgements

Simon Gay was partially supported by the EPSRC project 'Novel Type Systems for Concurrent Programming Languages' (grants GR/L75177 and GR/M39494). We thank the anonymous referees for their comments.

References

[Abr93] Abramsky, S.: 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*, Workshops in Computer Science, Springer, 1993, pp. 57–70.

[Acz88] Aczel, P.: *Non-Well-Founded Sets*. CSLI Lecture Notes 14. Center for the Study of Language and Information, 1988.

[AGN95] Abramsky, S., Gay, S. J. and Nagarajan, R.: Interaction categories and the 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, 1995.

[AGN96] Abramsky, S., Gay, S. J. and Nagarajan, R.: 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, 1996.

[AGN99] Abramsky, S., Gay, S. J. and Nagarajan, R.: A specification structure for deadlock-freedom of synchronous processes. *Theoretical Computer Science*, 222(1–2):1–53, 1999.

[AsW85] Ashcroft, E. A. and Wadge, W. W.: *Lucid, the data-flow programming language*. Academic Press, New York, 1985.

[Bar79] Barr, M.: $*$-*Autonomous Categories*, volume 752 of *Lecture Notes in Mathematics*. Springer, 1979.

[BeL91] Benveniste, A. and Le Guernic, P.: Synchronous programming with events and relations: the SIGNAL language and its semantics. *Science of Computer Programming*, 16(2):103–149, 1991.

[Gay95] Gay, S. J.: *Linear Types for Communicating Processes*. PhD thesis, University of London, 1995.

[GaN93] Gay, S. J. and Nagarajan, R.: Modelling SIGNAL in Interaction Categories. 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*. Workshops in Computer Science, Springer, 1993.

[HCR91] Halbwachs, N., Caspi, P., Raymond, P. and Pilaud, D.: The synchronous data flow programming language LUSTRE. *Proceedings of the IEEE*, 79(9):1305–1320, 1991.

[Jef97] Jeffrey, A.: Premonoidal categories and a graphical view of programs. Electronic publication: `http://klee.cs.depaul.edu/premon/paper.html`, 1997.

[JSV96] Joyal, A., Street, R. and Verity, D.: Traced monoidal categories. *Mathematical Proceedings of the Cambridge Philosophical Society*, 119:446–468, 1996.

[Kah74] Kahn, G.: The semantics of a simple language for parallel programming. In J. L. Rosenfeld, editor, *Proceedings of Information Processing '74*, 1974, pp. 471–475.

[LGL91] Le Guernic, P., Gautier, T., Le Borgne, M. and Le Maire, C.: Programming real-time applications with SIGNAL. *Proceedings of the IEEE*, 79(9):1321–1336, 1991.

[LeM87] Lee, E. A. and Messerschmidt, D. G.: Synchronous dataflow. *Proceedings of the IEEE*, 75(9):1235–1245, 1987.

[Mac71] Mac Lane, S.: *Categories for the Working Mathematician*. Springer, 1971.

[Mil89] Milner, R.: *Communication and Concurrency*. Prentice-Hall, 1989.

[Nag98] Nagarajan, R.: *Typed Concurrent Programs: Specification and Verification*. PhD thesis, University of London, 1998.

[Ste00] Ştefanescu, G.: *Network Algebra*. Springer, 2000.

[Wad81] Wadge, W. W.: An extensional treatment of dataflow deadlock. *Theoretical Computer Science*, 13:3–15, 1981.

Received October 2002
Accepted in revised form June 2003