A Simple Generalization of Kahn's Principle to Indeterminate Dataflow Networks

Eugene W. Stark*

Department of Computer Science

State University of New York at Stony Brook

Stony Brook, NY 11794 USA

July 31, 1990

Abstract

Kahn's principle states that if each process in a dataflow network computes a continuous input/output function, then so does the entire network. Moreover, in that case the function computed by the network is the least fixed point of a continuous functional determined by the structure of the network and the functions computed by the individual processes. Previous attempts to generalize this principle in a straightforward way to "indeterminate" networks, in which processes need not compute functions, have been either too complex or have failed to give results consistent with operational semantics. In this paper, we give a simple, direct generalization of Kahn's fixed-point principle to a large class of indeterminate dataflow networks, and we prove that results obtained by the generalized principle are in agreement with a natural operational semantics.

1 Introduction

Dataflow networks are a parallel programming paradigm in which a collection of concurrently and asynchronously executing sequential processes communicate by transmitting sequences or "streams" of "tokens" containing data values over unidirectional FIFO communication channels. Kahn [14, 15] envisioned a simple programming language built on this paradigm, in which the communication primitives available to processes are sufficiently restrictive that only functional processes can be programmed. That is, each process may be viewed as computing a function from the complete history of values received on its input channels to the complete history of values emitted on its output channels. Kahn argued that such processes compute functions that are in fact continuous with respect to a suitable complete partial order (cpo) structure on the sets of input and output histories. Moreover, a network

*Research supported in part by NSF Grants CCR-8702247 and CCR-8902215.

of such processes also computes a continuous function, which can be characterized as the least fixed point of a continuous functional associated with the network. This elegant idea has been called "the Kahn principle," and it has been shown [9, 21, 36] to give results in agreement with a natural "token-pushing" operational semantics.

In practical programming applications of the dataflow idea, the restriction to functional processes is somewhat limiting, because there are useful programs one wants to write that do not describe processes with functional input/output behavior. An example of a class of such processes are the *merge* processes, which shuffle together values arriving on two input channels onto a single output channel. A variety of such processes can be defined, corresponding to various conditions under which arriving inputs are guaranteed to be eventually transmitted to the output [24, 39]. They do not have functional behaviors, because in general there are many possible output interleavings for a single pair of input sequences. We use the term *indeterminate* to refer to dataflow networks in which processes need not have functional behaviors.

There have been many attempts to generalize Kahn's theory to a class of indeterminate networks. The obvious idea of generalizing functions to relations fails to give results consistent with token-pushing semantics. This fact was first noticed by Keller [16], and subsequently became known as the "Brock/Ackerman anomaly," after Brock and Ackerman [7] demonstrated convincingly by some clever examples that no denotational semantics based on set-theoretic input/output relations can give results consistent with token-pushing semantics. Subsequently, a rather large literature has developed on this subject. A variety of sophisticated approaches, such as powerdomains [1, 8], categories [3], "scenarios" [6, 7], sets of "linear traces" [4, 11, 12, 17, 23], "pomsets" [10, 26, 27], multilinear algebra [22], and other ideas [18, 19, 25, 28, 29, 30, 34, 37, 35] have been tried, but up until now none of these approaches has resulted in a truly simple and natural generalization of Kahn's principle that also maintains a clear connection with operational semantics.

In the remainder of this paper, we present a very simple and straightforward generalization of Kahn's principle, and prove that it gives results in agreement with operational semantics.

2 Kahn's Fixed-Point Principle

To give a precise statement of the Kahn principle, we must first formalize the notion of the input/output relation of a dataflow network. We begin by postulating a countably infinite set $\mathcal{V}$, whose elements represent the possible *data values* that can be communicated between processes. Let $\mathcal{V}^\infty$ be the set of all finite and infinite sequences of elements of $\mathcal{V}$, partially ordered by the *prefix relation* $\sqsubseteq$. We use ϵ to denote the empty sequence, which is the least element of $\mathcal{V}^\infty$. If X is a finite or countably infinite set (of *channels*), then an X*-history* is a function from X to $\mathcal{V}^\infty$. We think of such a function as representing the complete history of values communicated on the channels in X during some computation of a dataflow network. Let HX denote the set of all X-histories, then HX is also partially ordered, with $x \sqsubseteq x'$ iff $x(c) \sqsubseteq x'(c)$ for all $c \in X$. The poset HX has the structure of a *Scott domain* (an ω-

algebraic, consistently complete cpo [32]). The least element $\perp$ is the identically ϵ function. The *finite elements* (or *compact elements*, or *isolated elements*) of HX are exactly those histories x with $x(c)$ finite for all channels $c \in X$, and with $x(c) = \epsilon$ for all but finitely many $c \in X$. It is also important for us that HX is *finitary*, which means that each finite element can have at most a finite set of elements below it.

Functions on channel sets induce corresponding functions on channel histories. Formally, if X and Y are sets of channels, then a function $\phi : Y \to X$ induces a function $H\phi : HX \to HY$ satisfying $H\phi(x)(c) = x(\phi(c))$ for all $c \in Y$. The function $H\phi$ is obviously continuous; in fact the mapping H is a contravariant functor from the category of at most countable sets and functions to the category of Scott domains and continuous maps. Moreover, this functor maps the empty set to the one-point domain, and maps coproducts of sets (disjoint union) to products of domains; that is, we have a natural isomorphism $H(X + Y) \simeq HX \times HY$.

In Section 5, we shall continue with this formal development, defining an operational model for dataflow networks in which both a network and its constituent processes are represented as a certain kind of automata having fixed sets of input and output channels. Computation of a dataflow network may be regarded as a token-pushing game played on a graph whose nodes are automata, and whose arcs indicate when an output channel c of one automaton A is connected to an input channel d of another automaton B (we admit the possibility that $A = B$). As execution progresses, the automata change state and tokens containing data values move around on the graph. Communication between automata occurs when a token containing a data value is simultaneously output on channel c by process A and input on channel d by process B. This synchronized communication model looks at first glance to be different from the usual formulations of operational semantics for dataflow networks, in which communication channels are regarded as FIFO buffers distinct from processes. Instead of following the usual approach, we find it more convenient and economical to imagine each process as having, for each of its input channels, a component of its internal state that serves as an input buffer for that channel. We do not actually impose any direct structural requirements on the states of an automaton, but instead merely axiomatize the essential properties of the transition relation that are consequences of this structure. Among other things, the axioms imply that an automaton is always prepared to accept arbitrary input on an input channel (the buffer is never full), and the production of output in a step can depend only on input received in a previous step. In this way, we obtain the effect of a buffered communication model without the notational inconvenience.

Once a formal definition of "token-pushing semantics" has been given, it becomes possible to define the input/output relation of a network. In particular, the input/output relation of a network with input channels X and output channels Y is a subset of $HX \times HY$, whose elements are pairs $\langle x, y \rangle$ representing the history of input and output that could occur in one particular network computation. We are not interested in including in the input/output relation pairs $\langle x, y \rangle$ corresponding to all possible computations, but rather only in those pairs obtained from *completed* computations. Intuitively, a computation of a network is completed if each component process that is capable of performing some non-input step eventually does so. We need this condition to rule out uninteresting computations in which a process fails to

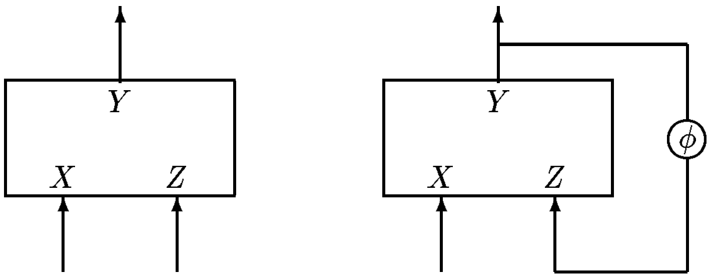

Figure 1: An Open-Loop Network and one with Feedback

process data in its input buffer simply because that process is never scheduled to execute a step. Having defined the input/output relation of a network, we may then classify networks as *determinate* or *indeterminate* according to whether or not their input/output relations are functional (*i.e.* are graphs of functions from HX to HY) or nonfunctional. In [21] it is shown that, for a formal operational model of dataflow networks defined along these lines, if the input/output relation of a network is functional, then in fact it is the graph of a continuous function from HX to HY.

Kahn's principle concerns determinate networks. In its simplest form, the principle gives a relationship between the function computed by a determinate network with a feedback loop and the corresponding "open-loop" network. The principle may be stated as follows:

The Kahn Principle

Suppose a network computes the continuous function $f : HX \times HZ \to HY$. Let $\phi : Z \to Y$ be an injective function, which we regard as designating, for each channel in Z, a corresponding channel in Y to which it is to be connected in a feedback loop (see Figure 1). Define the *feedback functional*

$$\Phi : [HX \to HY] \to [HX \to HY]$$

by

$$\Phi(g)(x) = f(x, H\phi(g(x))).$$

Then Φ is continuous, and the function computed by the closed-loop network is the least fixed-point $\mu\Phi$ of Φ.

3 A Generalization of Kahn's Principle

To motivate the generalized version of Kahn's principle, we reconsider the original version stated above, with a few changes in notation. First of all, instead of representing the behavior of a network with input channels X and output channels Y by a continuous function f :

$HX \to HY$, let us use instead the function $p : HX \to [HY \to HY]$ defined by $p(x)(y) = f(x) \sqcap y$. Here $f(x) \sqcap y$ denotes the greatest lower bound of $f(x)$ and y, which always exists due to the fact that HY is a consistently complete cpo, and is continuous because HX and HY are algebraic. Observe that no information is lost in replacing f by p, because $f(x) = \bigsqcup\{y : y = p(x)(y)\}$. Note also that for each $x \in X$ the function $p(x) : HY \to HY$ has the following properties:

1. $p(x) \circ p(x) = p(x)$.
2. $p(x) \sqsubseteq \mathrm{id}_{HY}$.

That is, $p(x)$ is a *projection* on the domain HY [13]. In the sequel, we use $\mathcal{P}(D)$ to denote the set of all projections on the finitary domain D. It can be shown that $\mathcal{P}(D)$, partially ordered by $\sqsubseteq$, is a domain; in fact it is a complete lattice.

Now, let us redefine the feedback functional to be compatible with the new notation. If $p : HX \times HZ \to \mathcal{P}(HY)$ represents the open-loop behavior of a network, and $\phi : Z \to Y$ is an injective function, then the corresponding feedback functional

$$\Psi : [HX \to [HY \to HY]] \to [HX \to [HY \to HY]]$$

is defined by

$$\Psi qxy = p\langle x, H\phi(qxy)\rangle y.$$

Here, and in the rest of the paper, we omit unnecessary parentheses under the convention that application associates to the left, and we use angle brackets to indicate tupling of arguments. It is immediate from the above definition that Ψ is continuous. Moreover, the least fixed point $\mu\Psi$ of Ψ determines the least fixed point $\mu\Phi$ of Φ in the same way as p determines f.

Lemma 3.1 $\mu\Phi x = \bigsqcup\{y : y = \mu\Psi xy\}$.

Proof – Recall that the fundamental fixed-point theorem (see, *e.g.* [32]) in the theory of complete partial orders states that if a functional $\Gamma : D \to D$ is continuous, then $\mu\Gamma = \bigsqcup_i \Gamma^i 0$, where 0 denotes the least element of D, and the *iterates* Γ^i of Γ are defined inductively by: $\Gamma^0 = \mathrm{id}$, $\Gamma^{i+1} = \Gamma \circ \Gamma^i$.

Let $y = \mu\Phi x = \bigsqcup_i \Phi^i 0x$. We first claim that $\Phi^i 0x \sqsubseteq \Psi^i 0xy$ for all $i \geq 0$. This is established by induction on i. The basis case is immediate. For the induction step, observe:

$$\begin{array}{rcl}
\Phi^{i+1}0x & = & \Phi^{i+1}0x \sqcap y \\
& = & f\langle x, H\phi(\Phi^i 0x)\rangle \sqcap y \\
& = & p\langle x, H\phi(\Phi^i 0x)\rangle y \\
& \sqsubseteq & p\langle x, H\phi(\Psi^i 0xy)\rangle y \\
& = & \Psi^{i+1}0xy,
\end{array}$$

where the induction hypothesis was used to obtain the inequality. This completes the induction. Since $\Phi^i 0x \sqsubseteq \Psi^i 0xy$ for all $i \geq 0$, and since $\Psi^i 0xy \sqsubseteq y = \mu\Phi x$ for all i, it follows that $\mu\Phi x = \mu\Psi x(\mu\Phi x)$.

To complete the proof, we now show that if y is any element of HY with $y = \mu\Psi xy$, then $y \sqsubseteq \mu\Phi x$. By induction on i we first show that $\Psi^i 0xy \sqsubseteq \Phi^i 0x$ for all x, y and all $i \geq 0$. The basis case is obvious. For the induction step, observe:

$$\begin{aligned}
\Psi^{i+1}0xy &= p\langle x, H\phi(\Psi^i 0xy)\rangle y \\
&= f\langle x, H\phi(\Psi^i 0xy)\rangle \sqcap y \\
&\sqsubseteq f\langle x, H\phi(\Psi^i 0xy)\rangle \\
&\sqsubseteq f\langle x, H\phi(\Phi^i 0x)\rangle \\
&= \Phi^{i+1}0xy,
\end{aligned}$$

completing the induction. Now, if $y = \mu\Psi xy$, then $y = \bigsqcup_i \Psi^i 0xy \sqsubseteq \bigsqcup_i \Phi^i 0x = \mu\Phi x$. ∎

We thus see that Kahn's principle generalizes in a straightforward way to the new representation of network behaviors. However, the new representation has an important advantage over the old one: it is general enough to permit the specification of some nonfunctional behaviors. We observed above that if $f : HX \to HY$ is continuous, and $p : HX \to \mathcal{P}(HY)$ is defined by $pxy = fx \sqcap y$, then $fx = \bigsqcup\{y : y = pxy\}$; that is to say, fx is the maximal fixed point of the projection $px \in \mathcal{P}(HY)$. Instead of just considering functions $p : HX \to \mathcal{P}(HY)$ of the form $fx \sqcap y$, we may consider the whole class of continuous functions $p : HX \to \mathcal{P}(HY)$. For an arbitrary such function p, the set of fixed points of px need not be directed, hence it may contain many maximal elements. Given $x \in HX$, we regard each maximal fixed point y of px as a possible output on input x, and in this way we may think of the function p as describing an indeterminate network whose input/output relation is the set of all $\langle x, y\rangle$ such that y is a maximal fixed point of px.

As a simple example of an indeterminate behavior, consider a network with no input channels ($X = \emptyset$, hence $HX \simeq \{\perp\}$) and one output channel ($Y = \{*\}$, hence $HY \simeq \mathcal{V}^\infty$), which simply emits an arbitrary infinite sequence of values on its output channel. Such a network is described by the function $p : HX \to \mathcal{P}(HY)$ satisfying $pxy = y$. It is clear that any maximal element y of HY is a maximal fixed point of $p\perp$, hence determines a pair $\langle\perp, y\rangle$ in the input/output relation of the network. This kind of "oracle" network can be used in the construction of a variety of other indeterminate behaviors. For example, it can be shown that a network that implements a kind of merging operation called *infinity-fair merge* [24, 39] can be constructed using functional components and an oracle that outputs an infinite sequence of natural numbers.

Although the generalizations we have made so far permit us to describe some indeterminate networks, the class of networks that can be represented as functions $p : HX \to \mathcal{P}(HY)$ is not yet large enough to admit "hiding" operations (see Section 5.6), by which some output channels of a network are made invisible to its environment. However, we can describe a larger and more interesting class of networks, which does admit hiding, by making one further generalization. The additional generalization is to replace the single map $p : HX \to \mathcal{P}(HY)$

by two maps: $p : HX \to \mathcal{P}(D)$ and $l : D \to HY$, where D is an arbitrary finitary domain. We therefore arrive at the following general definition of an indeterminate behavior:

> An (X, Y)-*behavior* is a triple (D, p, l), where D is a finitary domain, and $p : HX \to \mathcal{P}(D)$ and $l : D \to HY$ are continuous functions.

The *input/output relation* corresponding to an (X, Y)-behavior (D, p, l) is the set of all $\langle x, ld\rangle \in HX \times HY$ such that d is a maximal fixed point of the projection px.

As an example of the kind of indeterminate networks that can be described as behaviors (D, p, l), we consider *angelic merge*. A network that performs angelic merge has two input channels ($X = \{a, b\}$) and one output channel ($Y = \{c\}$), and operates by merging together the sequences of data values on its two input channels into a single output sequence. It is required to satisfy the following liveness condition: In any completed computation, if only a finite number of values arrive on one of the inputs, then eventually all the values from the *other* input will be transmitted to the output.

To describe angelic merge as an (X, Y)-behavior, we first define $\sigma : \{a, b\}^\infty \times HY \to HX$ to be the continuous function that on argument $\langle w, y\rangle$, splits the sequence of values y into two result sequences, where the ith value in y goes to the result sequence for channel a, if a is the ith value in the sequence w, otherwise to the result sequence for channel b. Angelic merge may then be defined to be the behavior $(\{a, b\}^\infty \times HY, p, l)$, where

- $l : \{a, b\}^\infty \times HY \to HY$ is defined by $l\langle w, y\rangle = y$.
- $p : HX \to \mathcal{P}(\{a, b\}^\infty \times HY)$ satisfies the following condition: for each $x \in HX$, $w \in \{a, b\}^\infty$, and $y \in HY$, $px\langle w, y\rangle$ is the greatest $\langle w', y'\rangle \sqsubseteq \langle w, y\rangle$ such that $\sigma\langle w', y'\rangle \sqsubseteq x$.

Many interesting indeterminate networks can be constructed using angelic merge and functional components. However, it should be pointed out that there is yet a more powerful kind of merging operation that cannot be described as a behavior (D, p, l). This is the *fair merge*, which shuffles two input sequences together onto a single output sequence in such a way that every value in both input channels eventually appears in the output channel. The results of this paper do not apply to indeterminate networks having this powerful merging capability.

We now state our generalized version of Kahn's principle.

> **The Generalized Kahn Principle**
>
> Suppose (D, p, l) is an $(X \times Z, Y)$-behavior that corresponds to the open-loop network shown in Figure 1. Let $\phi : Z \to Y$ be an injective function that assigns to each input channel in Z a corresponding output channel in Y. Define the *feedback functional* corresponding to (D, p, l) and ϕ to be the map:
>
> $$\Phi : [HX \to [D \to D]] \to [HX \to [D \to D]]$$
>
> defined by:
>
> $$\Phi qxd = p\langle x, H\phi(l(qxd))\rangle d.$$
>
> Then Φ is continuous, and $(D, \mu\Phi, l)$ is the behavior corresponding to the closed-loop network.

In the remainder of the paper, we justify this principle by showing that it gives results in agreement with token-pushing semantics.

4 Resolution of the Anomalies

In this section, we briefly examine the way in which Keller/Brock/Ackerman-type "anomalies" are avoided by our semantics for networks. We consider a particularly simple example of such an anomaly, due to J. Russell [31]. Suppose P is a process, having one input channel and one output channel, that obeys the following intuitive algorithm: Nondeterministically choose either: (1) read an input value, then output 0 followed by 1, or (2) output 0, read an input value, then output 0. Let Q be a similar process that has the additional possibility: (3) output 0, read an input value, then output 1. The input/output relation of P consists of all pairs of histories of the form $\langle \perp, 0\rangle$, $\langle vx, 01\rangle$, or $\langle vx, 00\rangle$, where v is an arbitrary value in $\mathcal{V}$, and x an arbitrary sequence of elements of $\mathcal{V}$. The input/output relation of Q is identical to that of P.

Even though P and Q have identical input/output relations, a difference between the two processes can be detected by placing them in a feedback loop. If $P_{\circlearrowleft\phi}$ denotes the network consisting of P with its output fed back to its input, and similarly for $Q_{\circlearrowleft\phi}$, then under token-pushing semantics the set of possible outputs of $P_{\circlearrowleft\phi}$ is $\{\perp, 00\}$, whereas the set of possible outputs of $Q_{\circlearrowleft\phi}$ is $\{\perp, 00, 01\}$. This demonstrates that no semantics for indeterminate networks, in which processes and networks are represented by their input/output relations, can support a definition of the feedback operation that gives results in agreement with token-pushing semantics.

Let us now see how the difficulty is resolved using our definition of behavior. Let $X = \{a\}$ and $Y = \{b\}$, and let $\{1,2\}_\perp$ denote the flat domain generated by $\{1,2\}$. Then the process P can be represented as the (X,Y)-behavior $(\{1,2\}_\perp \times HY, p, l)$, where $l\langle w, y\rangle = y$ and

$$p : HX \to [\{1,2\}_\perp \times HY \to \{1,2\}_\perp \times HY]$$

is defined as follows:

$$\begin{aligned}
px\langle \perp, y\rangle &= \langle \perp, \perp\rangle \\
px\langle 1, y\rangle &= \begin{cases} \langle 1, \perp\rangle, & \text{if } x = \perp \\ \langle 1, 01 \sqcap y\rangle, & \text{otherwise.} \end{cases} \\
px\langle 2, y\rangle &= \begin{cases} \langle 2, 0 \sqcap y\rangle, & \text{if } x = \perp \\ \langle 2, 00 \sqcap y\rangle, & \text{otherwise.} \end{cases}
\end{aligned}$$

Similarly, the process Q can be represented by the (X,Y)-behavior $(\{1,2,3\}_\perp \times HY, q, l)$, where

$$q : HX \to [\{1,2,3\}_\perp \times HY \to \{1,2,3\}_\perp \times HY]$$

is defined by:

$$qx\langle \perp, y\rangle = \langle \perp, \perp\rangle$$

$$
\begin{aligned}
qx\langle 1, y\rangle &= px\langle 1, y\rangle \\
qx\langle 2, y\rangle &= px\langle 2, y\rangle \\
qx\langle 3, y\rangle &= \begin{cases} \langle 3, 0 \sqcap y\rangle, & \text{if } x = \bot \\ \langle 3, 01 \sqcap y\rangle, & \text{otherwise.} \end{cases}
\end{aligned}
$$

Let $\phi : X \to Y$ be the map taking a to b. Applying the generalized Kahn principle, the behavior of $P_{\circlearrowleft\phi}$ is the $(\emptyset, Y)$-behavior $(\{1,2\}_\bot \times HY, p_{\circlearrowleft\phi}, l)$, where

$$p_{\circlearrowleft\phi} : \{\bot\} \to [\{1,2\}_\bot \times HY \to \{1,2\}_\bot \times HY]$$

is defined by:

$$
p\bot\langle w, y\rangle = \begin{cases} \langle \bot, \bot\rangle, & \text{if } w = \bot \\ \langle 1, \bot\rangle, & \text{if } w = 1 \\ \langle 2, 00 \sqcap y\rangle, & \text{if } w = 2. \end{cases}
$$

The maximal fixed points of $p_{\circlearrowleft\phi}$ are $\{\langle 1, \bot\rangle, \langle 2, 00\rangle\}$, hence the input/output relation of $P_{\circlearrowleft\phi}$ is $\{\langle \bot, \bot\rangle, \langle \bot, 00\rangle\}$.

The behavior of $Q_{\circlearrowleft\phi}$ is the $(\emptyset, Y)$-behavior $(\{1,2,3\}_\bot \times HY, q_{\circlearrowleft\phi}, l)$, where

$$q_{\circlearrowleft\phi} : \{\bot\} \to [\{1,2,3\}_\bot \times HY \to \{1,2,3\}_\bot \times HY]$$

is defined by:

$$
q\bot\langle w, y\rangle = \begin{cases} \langle \bot, \bot\rangle, & \text{if } w = \bot \\ \langle 1, \bot\rangle, & \text{if } w = 1 \\ \langle 2, 00 \sqcap y\rangle, & \text{if } w = 2 \\ \langle 3, 01 \sqcap y\rangle, & \text{if } w = 3. \end{cases}
$$

The maximal fixed points of $q_{\circlearrowleft\phi}$ are $\{\langle 1, \bot\rangle, \langle 2, 00\rangle, \langle 3, 01\rangle\}$, hence the input/output relation of $Q_{\circlearrowleft\phi}$ is $\{\langle \bot, \bot\rangle, \langle \bot, 00\rangle, \langle \bot, 01\rangle\}$.

Evidently, the input/output relations of $P_{\circlearrowleft\phi}$ and $Q_{\circlearrowleft\phi}$, obtained by the generalized Kahn principle, are in agreement with token-pushing semantics.

5 Operational Semantics

In this section, we define an operational semantics for indeterminate dataflow networks. In this semantics a network is represented as a certain kind of automaton, called a *monotone automaton* [24, 39]. The construction of larger networks from component automata is modeled by certain operations on automata. In fact, assuming the existence of sufficiently many "basic" determinate automata (in particular, the existence of an automaton that computes function $H\phi : HX \to HY$ for each function $\phi : Y \to X$), then any finite network can be constructed using basic automata and just three network-building operations: *parallel composition*, *hiding*, and *feedback*. Parallel composition corresponds to simply placing two

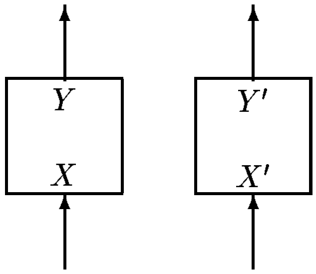

Figure 2: Parallel Composition

automata side-by-side in a network, without any communication (see Figure 2). Hiding makes some of the output channels of an automaton into "internal" channels, which are not available for external communication. Feedback is the operation depicted in Figure 1. Hiding and parallel composition pose no particular semantic difficulties in the framework we have set up here, and we shall discuss them here only briefly.

5.1 Monotone Automata

A *monotone automaton* (henceforth simply "automaton") is a tuple

$$A = (X, W, Y, Q, \mathrm{I}, T)$$

where

- X, W, and Y are pairwise disjoint sets of *channels*, which we assume are at most countable. The elements of X, Y, and W are called *input channels*, *output channels*, and *internal channels*, respectively. Let $E = (X + Y + W) \times \mathcal{V}$, then the elements of E are called *actions* of A. If $e = \langle c, v \rangle$ is an action of A, then we write $\mathrm{chan}(e)$ for the channel component c and $\mathrm{val}(e)$ for the value component v, of e. An action e is an *input action* or a *non-input action*, according to whether $\mathrm{chan}(e) \in X$ or $\mathrm{chan}(e) \notin X$.
- Q is a set of *states*, and $\mathrm{I} \in Q$ is a distinguished *initial state*.
- $T : Q \times E \rightarrow Q$ is a partial *transition function*.

These data are required to satisfy the following conditions:

(Commutativity) For all states q and actions e, e', if $\mathrm{chan}(e) \neq \mathrm{chan}(e')$, $r = T(q, e)$, and $r' = T(q, e')$, then there exists a state s such that $s = T(r', e) = T(r, e')$.

(Receptivity) For all states q and input actions e, there exists a state r such that $r = T(q, e)$.

These automata are closely related to the input/output automata defined by Lynch and Tuttle [21], and also to the automata that have been studied by Bednarczyk [5], Kwiatkowska [20], and Shields [33].

A *transition* of A is a triple $q \xrightarrow{e} r$, where $r = T(q, e)$. We write $t : q \xrightarrow{e} r$, or just $q \xrightarrow{e} r$, to assert the existence of a transition $t = q \xrightarrow{e} r$ of A. Intuitively, a transition $q \xrightarrow{e} r$ represents a potential computation step of A in which action e occurs and the state changes from q to r. We say that action $e \in E$ is *enabled* in state q if there exists a transition $q \xrightarrow{e} r$ in T. If $t : q \xrightarrow{e} r$, then q is called the *domain* $\mathrm{dom}(t)$ of t and r is called the *codomain* $\mathrm{cod}(t)$ of t.

The receptivity condition in the definition of an automaton can be viewed as an abstract statement of the unboundedness of the buffers associated with input channels. The commutativity property can be thought of as saying that actions for different ports are concurrent, hence affect different components of the state. Together, the commutativity and receptivity imply the following *monotonicity* property: If non-input action e is enabled in state q, and e' is an arbitrary input action with $r = T(q, e')$, then e is enabled in state r as well. One can think of this condition as saying that it is possible to test for the *presence* of inputs in an input buffer, but not for their *absence*.

As an example of how we can model a dataflow process as an automaton, consider the case of the angelic merge process aleady discussed. Recall that $X = \{a, b\}$ and $Y = \{c\}$. Let $W = \{d\}$. We then represent the merge process as an automaton

$$A = (\{a, b\}, \{d\}, \{c\}, \mathcal{V}^* \times \mathcal{V}^* \times \mathcal{V}^*, \langle \epsilon, \epsilon, \epsilon \rangle, T),$$

where the set of transitions T contains a transition

$$\langle x_1, x_2, y \rangle \xrightarrow{e} \langle x_1', x_2', y' \rangle$$

iff one of the following conditions holds:

1. $e = \langle a, v \rangle$, $x_1' = x_1 v$, $x_2' = x_2$, and $y' = y$.
2. $e = \langle b, v \rangle$, $x_1' = x_1$, $x_2' = x_2 v$, and $y' = y$.
3. $e = \langle d, 1 \rangle$, $v x_1' = x_1$, $x_2' = x_2$, and $y' = yv$.
4. $e = \langle d, 2 \rangle$, $x_1' = x_1$, $v x_2' = x_2$, and $y' = yv$.
5. $e = \langle c, v \rangle$, $x_1' = x_1$, $x_2' = x_2$, and $v y' = y$.

It is straightforward to check that A satisfies the conditions for an automaton. In cases (3) and (4), the actions $\langle d, 1 \rangle$ and $\langle d, 2 \rangle$ are in E by our assumption that $\mathcal{V}$ contains all the natural numbers. (It is actually not important that the numbers 1 and 2 be used here; they could be replaced by any two distinct elements of $\mathcal{V}$.)

Intuitively, the state of A contains two "input buffers" and one "output buffer." Transitions of type (1) and (2) correspond to arriving input values being placed at the end of the appropriate input buffer. Transitions of type (3) and (4) are internal transitions that correspond to the indeterminate selection of input in one input buffer or the other to be moved to the output buffer. Transitions of type (5) correspond to the transmission of output from the output buffer. Similar constructions can be used to model many other kinds of dataflow processes.

5.2 Computations of Automata

Suppose $A = (X, W, Y, Q, \mathrm{I}, T)$ is an automaton. A *finite computation sequence* for A is a finite sequence γ of transitions of A of the form:

$$q_0 \xrightarrow{e_1} q_1 \xrightarrow{e_2} \dots \xrightarrow{e_n} q_n.$$

An *infinite computation sequence* is an infinite sequence of transitions:

$$q_0 \xrightarrow{e_1} q_1 \xrightarrow{e_2} \dots.$$

The state q_0 is called the *domain* $\mathrm{dom}(\gamma)$ of γ. If γ is finite, then the state q_n is called the *codomain* $\mathrm{cod}(\gamma)$ of γ, and the number n is called the *length* $|\gamma|$ of γ. We call the computation sequence of length 0 from state q the *identity* computation sequence. A computation sequence γ is *initial* if $\mathrm{dom}(\gamma)$ is the distinguished initial state I. Finite computation sequences γ and δ are called *composable* if $\mathrm{dom}(\delta) = \mathrm{cod}(\gamma)$ and in that case we define their *composition* $\gamma\delta$ to be the computation sequence obtained by concatenating them in the obvious way.

Each computation sequence γ of A determines a corresponding *channel history* $\mathcal{H}(\gamma) \in H(X + W + Y)$, where for each channel $c \in X + W + Y$, the sequence $\mathcal{H}(\gamma)(c)$ is the subsequence of all $\mathrm{val}(e_i)$ for those actions e_i with $\mathrm{chan}(e_i) = c$. In view of the natural isomorphism $H(X + W + Y) \simeq HX \times HW \times HY$, we may think of the history $\mathcal{H}(\gamma)$ of a computation sequence γ as a triple $\langle x, w, y\rangle \in HX \times HW \times HY$, and it will generally be convenient to do so. Let $\mathcal{H}(A) \subseteq HX \times HW \times HY$ denote the set of all histories of initial computation sequences γ of A.

Lemma 5.1 *Suppose $A = (X, W, Y, Q, \mathrm{I}, T)$ is an automaton. Then*

1. $\langle \perp, \perp, \perp \rangle \in \mathcal{H}(A)$.

2. *If $\langle x, w, y\rangle \in HX \times HW \times HY$ is finite, $\langle x, w, y\rangle \in \mathcal{H}(A)$, and x' is a finite element of HX with $x \sqsubseteq x'$, then $\langle x', w, y\rangle \in \mathcal{H}(A)$.*

Proof – (1) The identity initial computation sequence has history $\langle \perp, \perp, \perp \rangle$.

(2) Given a finite $\langle x, w, y\rangle \in HX \times HW \times HY$ such that $\langle x, w, y\rangle \in \mathcal{H}(A)$, we may obtain a finite initial computation sequence γ whose history is $\langle x, w, y\rangle$. Given a finite $x' \in HX$ with $x \sqsubseteq x'$, we may then use the receptivity property of A to construct a finite *pure-input extension* γ' of γ, such that γ' has $\langle x', w, y\rangle$ as its history. ∎

To state properly the next results, we need some concepts from domain theory. If D is a domain, then a *subdomain* of D is a subset U of D, which is a domain under the restriction of the ordering on D, such that the inclusion of U in D is continuous. A subdomain U is *normal* if for all $d \in D$, the set $\{u \in U : u \sqsubseteq d\}$ is directed. The following easily proved characterization of normal subdomains will be convenient:

Proposition 5.2 *Suppose U is a subdomain of a domain D. Then U is normal iff the following conditions hold:*

1. $\perp \in U$.

2. *If $u, u' \in U$ are consistent (*i.e. *have an upper bound) in D, then $u \sqcup u' \in U$.*

Suppose U is a subdomain of D, and let $m : U \to D$ be the inclusion map. If U is a normal subdomain of D, then we may define a map $e : D \to U$ by: $ed = \bigsqcup\{u \in U : u \sqsubseteq d\}$. It is then easy to see that $e \circ m = \mathrm{id}_U$ and $m \circ e \sqsubseteq \mathrm{id}_D$. Then $(m \circ e) \circ (m \circ e) = m \circ (e \circ m) \circ e = m \circ e$, so the map $m \circ e : D \to D$ is a projection, and $U = \{d \in D : d = m(ed)\}$. Thus, normal subdomains determine projections. Conversely, if $p : D \to D$ is a projection on a finitary domain D, then $U = p(D) = \{d \in D : d = pd\}$ is a normal subdomain of D. Hence there is a correspondence between projections on D and normal subdomains of D. In fact, the following is easily shown:

Proposition 5.3 *For any finitary[1] domain D, the set of all normal subdomains of D, partially ordered by inclusion, is a complete lattice that is isomorphic to the set of projections on D, partially ordered by $\sqsubseteq$. Moreover, if $\{D_i : i \in I\}$ is a collection of normal subdomains of D which is directed under inclusion order, then its supremum in the lattice of normal subdomains is given by:*

$$(\bigcup\{D_i : i \in I\})^{\mathrm{c}},$$

where $(\ )^{\mathrm{c}}$ denotes closure under directed suprema.

Lemma 5.4 *Suppose U is a normal subdomain of a finitary domain D. Let U° denote the set of elements of U that are finite in D. Then $U = (U^{\circ})^{\mathrm{c}}$.*

Proof – Since D is algebraic, and $U \subseteq D$, every $u \in U$ is the supremum of the set of finite elements $d \in D$ with $d \sqsubseteq u$. Let p denote the projection on D corresponding to the normal subdomain U, then by the continuity of p it follows that u is also the supremum of the set of all pd such that $d \in D$ is finite and $d \sqsubseteq u$. But if $d \in D$ is finite in D, then so is pd, because projections are decreasing and in a finitary domain no infinite element can be below a finite element. Since $U \subseteq D$, an element of u is finite in U if it is finite in D. It follows that u is the supremum of the directed subset $\{pd : d \in D, d \sqsubseteq u, d \text{ finite}\}$ of U°. ∎

An *interval* in a domain D is a pair $[d, d']$ of elements of D such that $d \sqsubseteq d'$. An interval $[d, d']$ is *prime* if there exists no $d'' \in D$ such that $d \sqsubset d'' \sqsubseteq d'$. The result below, proved in [35, 38], gives a great deal of information about the structure of the set $\mathcal{H}(A)$.

[1]The author thanks Carl Gunter for pointing out that for a projection p on an arbitrary domain D, it is not necessarily the case that $p(D)$ is algebraic. However, in case D is finitary, then $p(d)$ is a finite element of $p(D)$ whenever d is a finite element of D, and this is sufficient for algebraicity.

Proposition 5.5 *Suppose $A = (X, W, Y, Q, \mathrm{I}, T)$ is an automaton. Then the set $\mathcal{H}(A)$, partially ordered by $\sqsubseteq$, is a normal subdomain of $HX \times HW \times HY$. Moreover, the inclusion of $\mathcal{H}(A)$ in $HX \times HW \times HY$ preserves prime intervals.*

To give the proof of this proposition would require a fairly substantial digression to develop the necessary techniques. Instead, we merely comment on the method used. We begin by defining two computation sequences γ and γ' to be *equivalent* if $\mathrm{dom}(\gamma) = \mathrm{dom}(\gamma')$ and $\mathcal{H}(\gamma) = \mathcal{H}(\gamma')$. We call equivalence classes of computation sequences *computations.* We then observe that the partial ordering on histories induces a corresponding partial ordering on computations, making the map, taking each computation to the corresponding history, injective and order-preserving. It then remains to show that the partially ordered set of initial computations is a domain, and that the map from computations to histories is an embedding of a normal subdomain that preserves prime intervals. To accomplish this, we introduce the auxiliary notion of the *residual* of one computation sequence "after" another, and obtain a characterization in terms of residuals of the ordering on computations.

More precisely, suppose γ and γ' are finite computation sequences, such that $\mathrm{dom}(\gamma) = \mathrm{dom}(\gamma')$ and such that the histories $\mathcal{H}(\gamma)$ and $\mathcal{H}(\gamma')$ are consistent. Then there exists a pair of finite computation sequences $\gamma \setminus \gamma'$ and $\gamma' \setminus \gamma$ (read "γ *after* γ' and γ' *after* γ), such that the computation sequences $\gamma(\gamma' \setminus \gamma)$ and $\gamma'(\gamma \setminus \gamma')$ are equivalent, and such that if δ and δ' are any finite computation sequences with $\gamma\delta'$ and $\gamma'\delta$ equivalent, then there exists a finite computation sequence ξ, unique up to equivalence, such that $(\gamma \setminus \gamma')\xi$ is equivalent to δ and $(\gamma' \setminus \gamma)\xi$ is equivalent to δ'.

Thus $\setminus$ is a partial binary operation on pairs of computation sequences, where $\gamma \setminus \delta$ is defined exactly when $\mathrm{dom}(\gamma) = \mathrm{dom}(\delta)$ and $\mathcal{H}(\gamma)$ and $\mathcal{H}(\delta)$ are consistent. We then think of γ and δ as potentially being part of the "same concurrent computation," and the computation sequence $\gamma \setminus \delta$ as that obtained by "cancelling" from γ the greatest common prefix of γ and δ, up to equivalence. If $\mathrm{dom}(\gamma) = \mathrm{dom}(\delta)$, but $\mathcal{H}(\gamma)$ and $\mathcal{H}(\delta)$ are not consistent, then $\gamma \setminus \delta$ and $\delta \setminus \gamma$ are undefined. In this case, γ and δ "conflict" in the sense that γ contains some indeterminate choice that is incompatible with a choice made in δ. Observe that we distinguish between two types of choice that may be represented in an automaton: *concurrent* choice, in which actions e, e' for distinct channels are both enabled in the same state q, and *indeterminate* choice, in which actions e and e' for the same channel are both enabled in state q. A formal definition of $\gamma \setminus \delta$ is given by induction on the length of γ and δ, and can be found in [35, 38].

A residual operation can also be defined on finite histories. Formally, if x and x' are consistent histories, then the *residual* of x after x' is the unique history $x \setminus x'$ with the property that $x'(x \setminus x') = x \sqcup x'$. From this definition, one can easily see that $x \sqsubseteq x'$ iff $x \setminus x' = \perp$. A similar relation holds for computations: $[\gamma] \sqsubseteq [\delta]$ iff $\gamma \setminus \delta$ is an identity computation sequence. Thus, the partial ordering on computations has an equivalent characterization in terms of residuals. Proposition 5.5 may then be proved by first using residuals to perform an inductive construction of suprema for directed collections of computations, and also for consistent pairs of computations, and then to show that the map taking each computation to its history preserves residuals (hence is additive and continuous), and reflects consistency.

A map between domains that is strict, additive, continuous, and reflects consistency is an embedding of a normal subdomain.

The same techniques also establish the following result:

Proposition 5.6 *Suppose $A = (X, W, Y, Q, \mathrm{I}, T)$ is an automaton. Let γ be a finite initial computation sequence of A having history $\langle x, w, y\rangle$, and suppose $\langle x', w', y'\rangle$ is a finite element of $HX \times HW \times HY$ with $\langle x, w, y\rangle \sqsubseteq \langle x', w', y'\rangle$. If $\langle x', w', y'\rangle \in \mathcal{H}(A)$, then there exists a finite initial computation sequence δ of A with history $\langle x', w', y'\rangle$, such that γ is a prefix of δ.*

The proof of Proposition 5.6 is accomplished by letting γ' be an arbitrary initial computation sequence of A having history $\langle x', w', y'\rangle$, observing that $\mathcal{H}(\gamma)$ and $\mathcal{H}(\gamma')$ are consistent, hence the residual $\gamma' \setminus \gamma$ is defined, and then taking $\delta = \gamma(\gamma' \setminus \gamma)$.

Lemma 5.7 *Suppose $A = (X, W, Y, Q, \mathrm{I}, T)$ is an automaton. Then the map taking x to the set*

$$U(x) = \{\langle w, y\rangle : \langle x, w, y\rangle \in \mathcal{H}(A)\},$$

is a continuous function from HX to the lattice of normal subdomains of $HW \times HY$.

Proof – We first use Proposition 5.2 to show that for each $x \in HX$, the set $U(x)$ is a normal subdomain of $HW \times HY$. Note that $\langle \perp, \perp\rangle \in U(x)$ for all $x \in HX$ by Lemma 5.1. Also, if $\langle w, y\rangle \in U(x)$ and $\langle w', y'\rangle \in U(x)$, where $\langle w, y\rangle$ and $\langle w', y'\rangle$ are consistent, then $\langle x, w, y\rangle \in \mathcal{H}(A)$ and $\langle x, w', y'\rangle \in \mathcal{H}(A)$, hence $\langle x, w \sqcup w', y \sqcup y'\rangle \in \mathcal{H}(A)$ and $\langle w \sqcup w', y \sqcup y'\rangle \in U(x)$ by the normality of $\mathcal{H}(A)$. To show that $U(x)$ is a subdomain of $HW \times HY$, we show that if $V \subseteq U(x)$ is directed, then $\bigsqcup V \in U(x)$. Now, if $V \subseteq U(x)$ is directed, then $\langle x, w, y\rangle \in \mathcal{H}(A)$ whenever $\langle w, y\rangle \in V$, hence if $\langle w, y\rangle = \bigsqcup V$, then $\langle x, w, y\rangle \in \mathcal{H}(A)$ because $\mathcal{H}(A)$ is a subdomain of $HX \times HW \times HY$. It follows that $\langle w, y\rangle \in U(x)$.

To prove that the map taking x to $U(x)$ is continuous, we show that for all $x \in HX$,

$$U(x) = \left(\bigcup\{U(x') : x' \sqsubseteq x, x' \text{ finite}\}\right)^{\mathrm{c}},$$

where $(\ )^{\mathrm{c}}$ denotes closure under directed suprema. By Lemma 5.4, it suffices to show that if $\langle w, y\rangle \in HW \times HY$ is finite, then $\langle w, y\rangle \in U(x)$ iff $\langle w, y\rangle \in U(x')$ for some finite $x' \sqsubseteq x$. So, suppose $\langle w, y\rangle \in HW \times HY$ is finite. If $\langle w, y\rangle \in U(x)$, then $\langle x, w, y\rangle \in \mathcal{H}(A)$, hence there exists an initial computation sequence γ of A such that the history of γ is $\langle x, w, y\rangle$. Moreover, since $\langle w, y\rangle$ is finite, there must be some finite $x' \in HX$ and some finite prefix γ' of γ whose history is $\langle x', w, y\rangle$. This shows that $\langle w, y\rangle \in U(x')$ for some finite $x' \sqsubseteq x$. Conversely, if $\langle w, y\rangle \in U(x')$ for some finite $x' \sqsubseteq x$, then there exists a finite initial computation sequence γ of A whose history is $\langle x', w, y\rangle$. Then $\langle x, w, y\rangle \in \mathcal{H}(A)$ by Lemma 5.1 and the fact that $\mathcal{H}(A)$ is closed under directed suprema. ∎

5.3 Behaviors of Automata

There is a simple way to obtain an (X, Y)-behavior from an automaton $A = (X, W, Y, Q, \mathrm{I}, T)$. Specifically, let $D = HW \times HY$, and define functions

$$p : HX \to [D \to D] \qquad\qquad l : D \to HY$$

as follows:

$$\begin{aligned} px\langle w, y\rangle &= \bigsqcup\{\langle w', y'\rangle \sqsubseteq \langle w, y\rangle : \langle x, w', y'\rangle \in \mathcal{H}(A)\}. \\ l\langle w, y\rangle &= y. \end{aligned}$$

The definition of p makes sense because by Lemma 5.7, the set on the right-hand side is directed.

Lemma 5.8 *Suppose $A = (X, W, Y, Q, \mathrm{I}, T)$ is an automaton and D, p, l are defined as above. Then (D, p, l) is an (X, Y)-behavior.*

Proof – Obviously D is a domain, and l is continuous. Moreover, from Lemma 5.7 we know that the map taking x to the set

$$U(x) = \{\langle w, y\rangle : \langle x, w, y\rangle \in \mathcal{H}(A)\},$$

is a continuous function from HX to the lattice of normal subdomains of $HW \times HY$. But px is just the projection corresponding to $U(x)$ under the isomorphism, given by Proposition 5.3, between the lattice of normal subdomains of $HW \times HY$ and the lattice of projections on $HW \times HY$. Hence p is a continuous function from HX to $\mathcal{P}(HW \times HY)$. ∎

Recall that we defined the input/output relation of an (X, Y)-behavior (D, p, l) to be the set of all $\langle x, ld\rangle$ such that d is a maximal fixed point of px. A few words are in order here about why this makes sense if (D, p, l) is the behavior of an automaton $A = (X, W, Y, Q, \mathrm{I}, T)$. For such a behavior, a history $d \in D$ is a maximal fixed point of the projection px iff the history $\langle x, d\rangle$ is maximal among all histories of the form $\langle x, d'\rangle \in \mathcal{H}(A)$. In [24], it was shown that for monotone automata that are obtained by composing a collection of "sequential" component automata into networks, a history $\langle x, d\rangle$ is maximal among all histories of the form $\langle x, d'\rangle \in \mathcal{H}(A)$ iff $\langle x, d\rangle$ is the history of a "completed" or "fair" computation sequence. Since we wish all and only the completed computation sequences to contribute pairs to the input/output relation, we regard the coincidence of completedness and maximality as the justification for our definition of the input/output relation associated with a behavior. Of course, it might be that there are automata that are not representable as a network of sequential components, and whose input/output relations thus do not necessarily have any relationship to any concrete, intuitive notion of completed or fair computation sequences. However, we do not regard this potential extra generality of our model as any cause for alarm.

5.4 Automata from Behaviors

For certain (X, Y)-behaviors (D, p, l), it is possible to construct an automaton having (D, p, l) as its behavior up to isomorphism of the domain D. In particular, suppose (D, p, l) is an (X, Y)-behavior that satisfies the following assumptions:

1. $D \simeq HW \times HY$ for some set W.
2. For all $x \in HX$, the inclusion of the normal subdomain

$$\{\langle w, y\rangle \in HW \times HY : \langle w, y\rangle = px\langle w, y\rangle\}$$

   in $HW \times HY$ preserves prime intervals.

Then, define Q to be the set of all finite elements of $HX \times HW \times HY$, and let $\text{I} = \langle \perp, \perp, \perp\rangle$. If $q \in Q$ and $e = \langle c, v\rangle \in (X + W + Y) \times \mathcal{V}$, then let $q; e \in Q$ denote the history such that $(q; e)c = (qc)v$ and $(q; e)c' = qc'$ for $c' \neq c$. Let $r = T(q, e)$ iff $r = q; e$ and one of the following holds:

1. $\text{chan}(e) \in X$.
2. $\text{chan}(e) \in Y + W$, and if $r = \langle x, w, y\rangle$ then $\langle w, y\rangle = px\langle w, y\rangle$.

It is straightforward to check that $A = (X, W, Y, Q, \text{I}, T)$ is an automaton, and that (D, p, l) is its behavior up to isomorphism. Assumption (2) above implies that for all $x \in HX$, every maximal fixed point $\langle w, y\rangle$ of px is reachable from $\langle \perp, \perp\rangle$ by a sequence of prime intervals. This fact is used to prove that for each state $q = \langle x, w, y\rangle$, if $\langle w, y\rangle$ is not a maximal fixed point of px, then there exists some non-input action e that is enabled in state q.

In the above construction, the condition that $D \simeq HW \times HY$ for some W is a strong restriction that can be weakened substantially if we are willing to generalize the definition of automata by relaxing the requirement that internal actions be port/value pairs. In particular, if we merely require that internal actions be elements of a "concurrent alphabet," then we can weaken condition (1) above to the condition that $D \simeq E \times HY$ for some "conflict event domain" E [38]. Further generalizations are possible if we consider automata for which more than one action can appear in a single transition. In this paper, we eschew the extra generality because it makes it more difficult to see the connection with the intuitive token-pushing semantics for dataflow networks.

5.5 The Parallel Composition Operation

The parallel composition operation builds a network from two component automata by placing them next to each other without establishing any communication. Formally, suppose

$$A = (X, W, Y, Q, \text{I}, T), \qquad A' = (X', W', Y', Q', \text{I}', T')$$

are automata, where $X+W+Y$ and $X'+W'+Y'$ are disjoint. Then the *parallel composition* of A and A' is the automaton

$$A\|A' = (X+X', W+W', Y+Y', Q\times Q', \langle \text{I}, \text{I}'\rangle, T''),$$

where $\langle r, r'\rangle = T''(\langle q, q'\rangle, e)$ iff one of the following conditions holds:

1. e is an action of A, $r = T(q,e)$, and $r' = q'$.

2. e is an action of A', $r' = T'(q', e)$ and $r = q$.

There are no particular theoretical difficulties associated with the parallel composition operation. All we shall have to say about it is to state the following result:

Proposition 5.9 *Suppose A has behavior (D, p, l) and A' has behavior (D', p', l'). Then $A\|A'$ has behavior $(D\times D', p'', l\times l')$, where $p''\langle x, x'\rangle\langle d, d'\rangle = \langle pxd, p'x'd'\rangle$.*

5.6 The Hiding Operation

The hiding operation on automata takes some of the output ports and makes them into internal ports. Formally, suppose

$$A = (X, W, V+Y, Q, \text{I}, T)$$

is an automaton. Then the *hiding of V in A* is the automaton

$$A\setminus V = (X, W+V, Y, Q, \text{I}, T).$$

The map from automata to their behaviors also respects hiding operations, as the following result states:

Proposition 5.10 *Suppose the $(X, V+Y)$-automaton A has behavior (D, p, l). Then the (X, Y)-automaton $A\setminus V$ has behavior $(D, p, \rho\circ l)$, where $\rho : H(V+Y)\to HY$ is the evident restriction map.*

5.7 The Feedback Operation

We now formalize the feedback operation depicted schematically in Figure 1 as a construction on automata. Suppose

$$A = (X+Z, W, Y, Q, \text{I}, T)$$

is an automaton, and let $\phi : Z\to Y$ be an injection. Then the *feedback of A by ϕ* is the automaton

$$A_{\circlearrowleft\phi} = (X, W, Y, Q, \text{I}, T_{\circlearrowleft\phi})$$

where $r = T_{\circlearrowleft\phi}(q, e)$ iff one of the following conditions holds:

1. $\mathrm{chan}(e) \notin \phi(Z)$ and $r = T(q, e)$.

2. $e = \langle \phi(c), v \rangle$ for some $c \in Z$ (which is then unique by the injectiveness of ϕ) and if $e' = \langle c, v \rangle$, then there exists a state s with $s = T(q, e)$ and $r = T(s, e')$.

It is easy to check that $A_{\circlearrowleft\phi}$ satisfies the conditions for an automaton.

The intuition behind this construction is as follows: The automaton $A_{\circlearrowleft\phi}$ behaves exactly as A does in the case of actions e with $\mathrm{chan}(e) \notin \phi(Z)$. Such actions correspond either to inputs on channels in X or outputs on channels in Y that are not to be fed back to channels in Z. However, if $e = \langle \phi(c), v \rangle$ for some $c \in Z$, then e corresponds to the production of an output value v that is immediately reapplied as feedback input on channel c.

Lemma 5.11 *Suppose γ is a finite initial computation sequence of $A_{\circlearrowleft\phi}$ having history $\langle x, w, y \rangle$. Then there exists a finite initial computation sequence δ of A having history $\langle x, H\phi y, w, y \rangle$, such that γ and δ end in the same state.*

Proof – Straightforward induction on the length of a finite computation sequence. ∎

Lemma 5.12 *Suppose $\langle x, w, y \rangle$ and $\langle x, w', y' \rangle$ are finite elements of $HX \times HW \times HY$, such that $\langle w, y \rangle \sqsubseteq \langle w', y' \rangle$. If $\langle x, w, y \rangle \in \mathcal{H}(A_{\circlearrowleft\phi})$, and $\langle x, H\phi y, w', y' \rangle \in \mathcal{H}(A)$, then $\langle x, w', y' \rangle \in \mathcal{H}(A_{\circlearrowleft\phi})$.*

Proof – Since $\langle x, w, y \rangle \in \mathcal{H}(A_{\circlearrowleft\phi})$, we may obtain a finite initial computation sequence γ of $A_{\circlearrowleft\phi}$ with history $\langle x, w, y \rangle$. By Lemma 5.11, there exists a finite initial computation sequence δ of A having history $\langle x, H\phi y, w, y \rangle$, such that γ and δ end in the same state q. Since $\langle x, H\phi y, w', y' \rangle \in \mathcal{H}(A)$ by hypothesis, it follows by Proposition 5.6 that there exists a finite computation sequence δ' for A having history $\langle x, H\phi y, w', y' \rangle$, such that δ is a prefix of δ'. Then $\delta' = \delta\tau$, where the history of τ is of the form $\langle \bot, \bot, w'', y'' \rangle$, hence τ contains no actions on ports in Z. A straightforward induction on the length of τ shows that there exists a finite computation sequence σ of $A_{\circlearrowleft\phi}$ from state q, such that exactly the same sequence of actions occurs in σ as in τ. Since σ starts in state q, the computation sequences γ and σ are composable, and the computation sequence $\gamma\sigma$ has history $\langle x, w', y' \rangle$, as required. ∎

6 Correctness of the Generalized Kahn Principle

Suppose $A = (X + Z, W, Y, Q, \iota, T)$ is an automaton, and $\phi : Z \to Y$ is an injection. Let $A_{\circlearrowleft\phi}$ be the feedback of A by ϕ. Let (D, p, l) be the behavior of A, and let $(D_{\circlearrowleft\phi}, p_{\circlearrowleft\phi}, l_{\circlearrowleft\phi})$ be the behavior of $A_{\circlearrowleft\phi}$. Note that $D = D_{\circlearrowleft\phi} = HW \times HY$, and both l and $l_{\circlearrowleft\phi}$ take $\langle w, y \rangle$ to y.

The feedback functional corresponding to (D, p, l) and ϕ is the map:

$$\Phi : [HX \to [D \to D]] \to [HX \to [D \to D]]$$

defined by:

$$\Phi q x d = p\langle x, H\phi(l(qxd))\rangle d.$$

It is obvious from the definition that Φ is continuous. The generalized Kahn principle then states that $p_{\circlearrowleft\phi} = \mu\Phi$.

For all $x \in HX$, $z \in HZ$, and $i \geq 0$, define

$$\begin{aligned}
p_i &= \Phi^i 0 \\
D(x,z) &= \{\langle w,y\rangle : \langle w,y\rangle = p\langle x,z\rangle\langle w,y\rangle\} = \{\langle w,y\rangle : \langle x,z,w,y\rangle \in \mathcal{H}(A)\} \\
D_{\circlearrowleft\phi}(x) &= \{\langle w,y\rangle : \langle w,y\rangle = p_{\circlearrowleft\phi}x\langle w,y\rangle\} = \{\langle w,y\rangle : \langle x,w,y\rangle \in \mathcal{H}(A_{\circlearrowleft\phi})\}. \\
D_i(x) &= \{\langle w,y\rangle : \langle w,y\rangle = p_i x\langle w,y\rangle\}.
\end{aligned}$$

Intuitively, $D(x,z)$ is the set of non-input portions of histories of initial computation sequences of A on input $\langle x,z\rangle$, and $D_{\circlearrowleft\phi}(x)$ is the set of non-input portions of histories of initial computation sequences of $A_{\circlearrowleft\phi}$ on input x. The sets $D_i(x)$ may be thought of as approximations to $D_{\circlearrowleft\phi}(x)$ in which feedback of output to input is limited to at most i cycles.

Lemma 6.1

1. *For all $x \in HX$ and all $i \geq 0$, the map $p_i x : D \to D$ is a projection, hence $D_i(x)$ is a normal subdomain of D.*

2. *For all $x \in HX$, the map $\mu\Phi x : D \to D$ is a projection.*

Proof – The second statement follows immediately from the first by the continuity of composition. To show the first statement, fix an arbitrary $x \in HX$. Clearly, $p_i x \sqsubseteq \mathrm{id}_D$. To complete the proof, we show by induction on i that $(p_i x) \circ (p_i x) = p_i x$ for all $i \geq 0$, hence $p_i x$ is a projection on D. The basis case is obvious. For the induction step, suppose we have shown that $p_i x$ is a projection. Fix an arbitrary $d \in D$, and let $d_i = p_i x d$ and $d_{i+1} = p_{i+1} x d$.

We first claim that $p_i x d_{i+1} = d_i$. To see this, observe that since $p_i x$ is a projection we have

$$\begin{aligned}
p_i x d_{i+1} &= \bigsqcup\{d' : d' \sqsubseteq d_{i+1}, d' = p_i x d'\} \\
&= \bigsqcup\{d' : d' \sqsubseteq d_i, d' = p_i x d'\} \\
&= d_i,
\end{aligned}$$

because $d_{i+1} \sqsubseteq d$ and $d_i = p_i x d$ is the maximal $d' \sqsubseteq d$ with $d' = p_i x d'$.

It now follows that

$$\begin{aligned}
p_{i+1} x d_{i+1} &= p\langle x, H\phi(l(p_i x d_{i+1}))\rangle d_{i+1} \\
&= p\langle x, H\phi(l d_i)\rangle d_{i+1} \\
&= p\langle x, H\phi(l d_i)\rangle (p\langle x, H\phi(l d_i)\rangle d) \\
&= p\langle x, H\phi(l d_i)\rangle d \\
&= p_{i+1} x d \\
&= d_{i+1},
\end{aligned}$$

where we have used the idempotence of $p\langle x, H\phi(ld_i)\rangle$. Thus $p_{i+1}x(p_{i+1}xd) = p_{i+1}xd$, completing the induction. ▮

Lemma 6.2 *Given $\langle x, w, y\rangle \in HX \times HW \times HY$, and $i \geq 0$, let $\langle w', y'\rangle = p_i x\langle w, y\rangle$. Then $\langle w, y\rangle \in D_{i+1}(x)$ iff $\langle w, y\rangle \in D(x, H\phi y')$.*

Proof – Simply note that $\langle w, y\rangle \in D_{i+1}(x)$ iff $\langle w, y\rangle = p_{i+1}x\langle w, y\rangle = p\langle x, H\phi y'\rangle\langle w, y\rangle$ iff $\langle w, y\rangle \in D(x, H\phi y')$. ▮

The next result establishes the basic approximation relationship between the domains $D_i(x)$ and the domain $D_{\circlearrowleft\phi}(x)$. Its proof requires an analysis of the relationship between the set of finite initial computation sequences of A and those of $A_{\circlearrowleft\phi}$.

Lemma 6.3 *Suppose $\langle x, w, y\rangle \in HX \times HW \times HY$ is finite. Then $\langle w, y\rangle \in D_{\circlearrowleft\phi}(x)$ iff $\langle w, y\rangle \in D_n(x)$ for some n.*

Proof – Suppose $\langle x, w, y\rangle$ is finite. If $\langle w, y\rangle \in D_{\circlearrowleft\phi}(x)$, then there exists a finite initial computation sequence γ of $A_{\circlearrowleft\phi}$ with $\langle x, w, y\rangle$ as its history. Suppose γ is of length n, and for $0 \leq i \leq n$ let γ_i be the prefix of γ of length i. We claim that for all i with $0 \leq i \leq n$, if $\langle x_i, w_i, y_i\rangle$ is the history of γ_i, then $\langle w_i, y_i\rangle \in D_i(x_i)$. Since $\langle x, w, y\rangle = \langle x_n, w_n, y_n\rangle$, it then follows that $\langle w, y\rangle \in D_n(x)$. The claim is proved by induction on i. The basis case is obvious. For the induction step, suppose we have proved the claim for i, and consider the case of $i+1$. Since γ_{i+1} is exactly one transition longer than γ_i, the history $\langle x_{i+1}, w_{i+1}, y_{i+1}\rangle$ differs from $\langle x_i, w_i, y_i\rangle$ in exactly one of the three components. We consider each case separately.

1. In case $x_i \sqsubset x_{i+1}$, then $\langle w_{i+1}, y_{i+1}\rangle = \langle w_i, y_i\rangle$. Since $\langle w_i, y_i\rangle \in D_i(x_i)$, $D_i(x_i) \subseteq D_i(x_{i+1})$ by Proposition 5.3 and the monotonicity of p_i, and $D_i(x_{i+1}) \subseteq D_{i+1}(x_{i+1})$ by Proposition 5.3 and the fact that $p_i x_{i+1} \sqsubseteq p_{i+1}x_{i+1}$, it follows that $\langle w_{i+1}, y_{i+1}\rangle = \langle w_i, y_i\rangle \in D_i(x_i) \subseteq D_i(x_{i+1}) \subseteq D_{i+1}(x_{i+1})$.

2. Suppose $w_i \sqsubset w_{i+1}$, so that $x_{i+1} = x_i$ and $y_{i+1} = y_i$. Then, $\langle x_{i+1}, w_{i+1}, y_{i+1}\rangle = \langle x_i, w_{i+1}, y_i\rangle \in \mathcal{H}(A_{\circlearrowleft\phi})$, so $\langle x_i, H\phi(y_i), w_{i+1}, y_{i+1}\rangle \in \mathcal{H}(A)$ by Lemma 5.11, and thus $\langle w_{i+1}, y_{i+1}\rangle \in D(x_i, H\phi(y_i))$. Applying Lemma 6.2 and the induction hypothesis, we conclude $\langle w_{i+1}, y_{i+1}\rangle \in D_{i+1}(x_i) = D_{i+1}(x_{i+1})$.

3. If $y_i \sqsubset y_{i+1}$, then the argument is similar to the previous case.

Conversely, suppose $\langle w, y\rangle \in D_n(x)$ for some n. For $0 \leq i \leq n$, let $\langle w_i, y_i\rangle = p_i x\langle w, y\rangle$. We show, by induction on i, that $\langle w_i, y_i\rangle \in D_{\circlearrowleft\phi}(x)$ for all i with $0 \leq i \leq n$. Since $\langle w, y\rangle = \langle w_n, y_n\rangle$, it then follows that $\langle w, y\rangle \in D_{\circlearrowleft\phi}(x)$. The basis case is immediate from Lemma 5.1. For the induction step, suppose we have shown that $\langle w_i, y_i\rangle \in D_{\circlearrowleft\phi}(x)$. Since $\langle w_{i+1}, y_{i+1}\rangle \in D_{i+1}(x)$, and $\langle w_i, y_i\rangle = p_i x\langle w_{i+1}, y_{i+1}\rangle$, it follows that $\langle w_{i+1}, y_{i+1}\rangle = p\langle x, H\phi y_i\rangle\langle w_{i+1}, y_{i+1}\rangle$, hence $\langle w_{i+1}, y_{i+1}\rangle \in D(x, H\phi y_i)$. Then $\langle x, w_i, y_i\rangle \in \mathcal{H}(A_{\circlearrowleft\phi})$ and $\langle x, H\phi y_i, w_{i+1}, y_{i+1}\rangle \in \mathcal{H}(A)$, so $\langle x, w_{i+1}, y_{i+1}\rangle \in \mathcal{H}(A_{\circlearrowleft\phi})$ by Lemma 5.12. But this means that $\langle w_{i+1}, y_{i+1}\rangle \in D_{\circlearrowleft\phi}(x)$, completing the induction step. ▮

Lemma 6.4 $D_{\circlearrowleft\phi}(x) = (\bigcup_i D_i(x))^c$.

Proof – For finite x, the result is a direct consequence of Lemma 6.3, Lemma 5.4, and the fact that $D_{\circlearrowleft\phi}(x)$ is a normal subdomain of $HW \times HY$. To extend to all x, we use the algebraicity of HX and Lemma 5.7 to obtain:

$$D_{\circlearrowleft\phi}(x) = (\bigcup\{D_{\circlearrowleft\phi}(x') : x' \sqsubseteq x, x' \text{ finite}\})^c.$$

Using Lemma 6.1, we obtain a similar characterization of $D_i(x)$:

$$D_i(x) = (\bigcup\{D_i(x') : x' \sqsubseteq x, x' \text{ finite}\})^c.$$

The result is now immediate. ∎

Theorem 1 *$p_{\circlearrowleft\phi} = \mu\Phi$, hence the generalized Kahn principle is correct.*

Proof – First note that for all $x \in HX$ we have $\mu\Phi x = \bigsqcup_i p_i x$. Also, for all $x \in HX$, the projection $p_{\circlearrowleft\phi}x$ is the projection corresponding to the normal subdomain $D_{\circlearrowleft\phi}(x)$ of $HW \times HY$, under the isomorphism between the lattice of normal subdomains of $HW \times HY$ and the lattice of projections on $HW \times HY$. Similarly, $p_i x$ is the projection corresponding to the normal subdomain $D_i(x)$. By the previous lemma, $D_{\circlearrowleft\phi}(x) = (\bigcup_i D_i(x))^c$. But $(\bigcup_i D_i(x))^c$ is the least normal subdomain containing all the $D_i(x)$, hence is the normal subdomain corresponding to the projection $\bigsqcup_i p_i x = \mu\Phi$. ∎

7 Discussion

We have stated a generalized version of Kahn's fixed-point principle for a class of indeterminate networks, and we have proved that it gives results in accordance with token-pushing operational semantics. Our generalized Kahn principle is simple to state, and parallels Kahn's original fixed-point principle in a pleasant way. The class of dataflow networks to which it applies includes at least all networks that can be constructed from functional processes and angelic merge, but not to networks built using fair merge or an equivalent primitive such as "poll" [24]. Although our generalized Kahn principle is easily stated, the proof that it agrees with token-pushing semantics seems to require a rather detailed analysis of a particular operational model, using results the author has been accumulating over the past several years. It remains to be seen whether a simpler proof can be given. Other interesting possibilities for future work are to extend the result to recursively defined networks, and to look at possibilities for treating fair merge.

Abramsky [2] has recently proposed a generalized Kahn Principle that can be viewed as a "pointwise" extension of Kahn's result to those networks whose behaviors can be represented as sets of continuous functions. In Abramsky's work, a network with input channels X and output channels Y would be represented as a set of functions from HX to HY. A precise relationship can be drawn between the generalized Kahn Principle of the present paper and a pointwise version like that of Abramsky. Suppose $B = (D, p, l)$ is an $(X + Z, Y)$-behavior with the following special form: $D \simeq (HX \times HZ) \times D'$, and up to this isomorphism we have $p\langle x, z\rangle\langle\langle x', z'\rangle, d'\rangle = \langle\langle x \sqcap x', z \sqcap z'\rangle, d'\rangle$. Such a behavior models a network in which all indeterminacy arises from the choices made by an "oracle" (represented by D') that operates independently of the network input. These behaviors were called "semi-determinate" in [39]. We may associate with such a semi-determinate behavior the set $F(B)$ of all functions $f_{d'} : HX \times HZ \to HY$ satisfying $f_{d'}\langle x, z\rangle = l(p\langle x, z\rangle\langle\langle x, z\rangle, d'\rangle) = l(\langle\langle x, z\rangle, d'\rangle)$, where d' is a maximal element of D'. Notice that the special form of B ensures that the set $F(B)$ contains enough information to recover the input/output relation $R(B)$ of B, because $R(B) = \{\langle\langle x, z\rangle, f\langle x, z\rangle\rangle : f \in F(B)\}$. Now, if $B_{\circlearrowleft\phi}$ is the (X, Y)-behavior obtained by applying our generalized Kahn Principle to B, and if $f_{\circlearrowleft\phi} : HX \to HY$ denotes the result of applying the original Kahn Principle to a function $f : HX \times HZ \to HY$, then $F(B_{\circlearrowleft\phi}) = \{f_{\circlearrowleft\phi} : f \in F(B)\}$.

The comments of the previous paragraph apply only to semi-determinate behaviors. All networks built using functional processes and infinity-fair merge are semi-determinate, however it can be shown [39] that angelic merge is not semi-determinate. Thus, when viewed in this way, it would appear that our generalized Kahn principle applies to a larger class of networks than does the pointwise version of Abramsky. However, B. A. Trakhtenbrot and A. Rabinovich [private communication] have recently reminded the author that there is another way to associate a set of functions with a behavior, and that is to assign to each $(X \times Z, Y)$-behavior $B = (D, p, l)$ the set $G(B)$ of all functions $g_d : HX \times HZ \to D \times HY$ satisfying $g_d\langle x, z\rangle = \langle p\langle x, z\rangle d, l(p\langle x, z\rangle d)\rangle$, where d is an arbitrary element of D. If B is not semi-determinate, then we cannot discard the D component of the codomains of the functions g_d, since to do so would mean that the set $G(B)$ would no longer contain sufficient information to recover the input/output relation of B.[2] As in the previous paragraph, we have that $G(B_{\circlearrowleft\phi}) = \{g_{\circlearrowleft\phi} : g \in G(B)\}$ (this is an easy corollary of Theorem 1), so from this standpoint it appears that the "set of functions" model and the "pointwise lifted Kahn Principle" is just about as good as the behavior model and generalized Kahn Principle presented here. Our generalized Kahn Principle serves to reduce the problem of reasoning about indeterminate networks to that of reasoning about continuous functions, whereas the pointwise version requires reasoning about sets of continuous functions. However, in favor of the behavior model over the set of functions model we advance the argument that the former shows more clearly the underlying mathematical structure, because it makes evident the way in which all the functions in a set must relate to each other, if that set is to represent the behavior of an indeterminate network. It should also be noted that, because of the presence of the domain D, neither of the two models is sufficiently abstract, and therefore further investigation is required to reach the goal of a highly structured, fully abstract model of indeterminate dataflow networks.

[2] If however, as in the work of Trakhtenbrot and Rabinovich to date [29], one does not insist that only "completed" computation sequences contribute to the input/output relation, then it is no problem to throw away the domain D.

References

[1] S. Abramsky. Experiments, powerdomains, and fully abstract models for applicative multiprogramming. In *Foundations of Computation Theory*, pages 1–13, Springer-Verlag. Volume 158 of *Lecture Notes in Computer Science*, 1983.

[2] S. Abramsky. A generalized Kahn principle for abstract asynchronous networks. In *Mathematical Foundations of Program Semantics*, Springer Verlag. *Lecture Notes in Computer Science*, 1990. (to appear).

[3] S. Abramsky. On the semantic foundations for applicative multiprogramming. In *ICALP 83*, Springer Verlag. *Lecture Notes in Computer Science*, 1983.

[4] R. J. Back and N. Mannila. A refinement of Kahn's semantics to handle nondeterminism and communication. In *Proc. ACM Symposium on Principles of Distributed Computing*, pages 111–120, 1982.

[5] M. Bednarczyk. *Categories of Asynchronous Systems.* PhD thesis, University of Sussex, October 1987.

[6] J. D. Brock. *A Formal Model of Non-Determinate Dataflow Computation.* PhD thesis, Massachusetts Institute of Technology, 1983. Available as MIT/LCS/TR-309.

[7] J. D. Brock and W. B. Ackerman. Scenarios: a model of non-determinate computation. In *Formalization of Programming Concepts*, pages 252–259, Springer-Verlag. Volume 107 of *Lecture Notes in Computer Science*, 1981.

[8] M. Broy. Fixed point theory for communication and concurrency. In D. Bjørner, editor, *Formal Description of Programming Concepts II*, pages 125–148, North-Holland. 1983.

[9] A. A. Faustini. An operational semantics for pure dataflow. In *Automata, Languages, and Programming, 9th Colloquium*, pages 212–224, Springer-Verlag. Volume 140 of *Lecture Notes in Computer Science*, 1982.

[10] H. Gaifman and V. Pratt. Partial order models of concurrency and the computation of functions. In *Symposium on Logic in Computer Science*, pages 72–85, Ithaca, NY, June 1987.

[11] B. Jonsson. *Compositional Verification of Distributed Systems.* PhD thesis, Uppsala University, Uppsala, Sweden, 1987.

[12] B. Jonsson. A fully abstract trace model for dataflow networks. In *Sixteenth Annual ACM Symposium on Principles of Programming Languages*, pages 155–165, January 1989.

[13] A. Jung. *Cartesian Closed Categories of Domains.* PhD thesis, University of Darmstadt, 1988.

[14] G. Kahn. The semantics of a simple language for parallel programming. In J. L. Rosenfeld, editor, *Information Processing 74*, pages 471–475, North-Holland, 1974.

[15] G. Kahn and D. B. MacQueen. Coroutines and networks of parallel processes. In B. Gilchrist, editor, *Information Processing 77*, pages 993–998, North-Holland, 1977.

[16] R. M. Keller. Denotational models for parallel programs with indeterminate operators. In E. J. Neuhold, editor, *Formal Description of Programming Concepts*, pages 337–366, North-Holland. 1978.

[17] R. M. Keller and P. Panangaden. Semantics of networks containing indeterminate operators. In S. D. Brookes, A. W. Roscoe, and G. Winskel, editors, *Seminar on Concurrency*, pages 479–496, Springer-Verlag. Volume 197 of *Lecture Notes in Computer Science*, 1984.

[18] J. N. Kok. Denotational semantics of nets with nondeterminism. In *ESOP 86*, pages 237–249, Springer-Verlag. Volume 213 of *Lecture Notes in Computer Science*, March 1986.

[19] J. N. Kok. A fully abstract semantics for data flow nets. pages 351–368, Springer-Verlag. Volume 259 of *Lecture Notes in Computer Science*, 1987.

[20] M. Kwiatkowska. *Fairness for Non-Interleaving Concurrency.* PhD thesis, University of Leicester, May 1989.

[21] N. A. Lynch and E. W. Stark. A proof of the Kahn principle for input/output automata. *Information and Computation*, 82(1):81–92, July 1989.

[22] M. G. Main and D. B. Benson. Functional behavior of nondeterministic and concurrent programs. *Information and Control*, 62:144–189, 1984.

[23] J. Misra. Equational reasoning about nondeterministic processes (preliminary version). In *ACM Symposium on Principles of Distributed Computing*, pages 29–44, 1989.

[24] P. Panangaden and E. W. Stark. Computations, residuals, and the power of indeterminacy. In T. Lepisto and A. Salomaa, editors, *Automata, Languages, and Programming*, pages 439–454, Springer-Verlag. Volume 317 of *Lecture Notes in Computer Science*, 1988.

[25] D. M. R. Park. The “fairness problem” and nondeterministic computing networks. In *Proceedings, 4th Advanced Course on Theoretical Computer Science*, pages 133–161, Mathematisch Centrum, 1982.

[26] V. R. Pratt. On the composition of processes. In *Ninth Annual ACM Symposium on Principles of Programming Languages*, pages 213–223, January 1982.

[27] V. R. Pratt. The pomset model of parallel processes: unifying the temporal and the spatial. In S. D. Brookes, A. W. Roscoe, and G. Winskel, editors, *Seminar on Concurrency*, pages 180–196, Springer-Verlag. Volume 197 of *Lecture Notes in Computer Science*, July 1984.

[28] A. Rabinovich. Pomset semantics is consistent with data flow semantics. *EATCS Bulletin*, 107–117, June 1987.

[29] A. Rabinovich and B. A. Trakhtenbrot. Communication among relations. In *Automata, Languages, and Programming: 17th International Colloquium*, pages 294–307, Springer Verlag. Volume 443 of *Lecture Notes in Computer Science*, 1990.

[30] A. Rabinovich and B. A. Trakhtenbrot. Nets and data flow interpreters. In *Logic in Computer Science*, IEEE, 1989.

[31] J. Russell. Full abstraction for nondeterministic dataflow networks. June 1989. Unpublished manuscript, Cornell University.

[32] D. A. Schmidt. *Denotational Semantics: A Methodology for Language Development.* Allyn and Bacon, 1986.

[33] M. W. Shields. Deterministic asynchronous automata. In E. J. Neuhold and G. Chroust, editors, *Formal Methods in Programming*, pages 317–345, North-Holland. 1985.

[34] J. Staples and V. L. Nguyen. A fixpoint semantics for nondeterministic data flow. *Journal of the ACM*, 32(2):411–444, April 1985.

[35] E. W. Stark. Compositional relational semantics for indeterminate dataflow networks. In *Category Theory and Computer Science*, pages 52–74, Springer-Verlag. Volume 389 of *Lecture Notes in Computer Science*, Manchester, U. K., 1989.

[36] E. W. Stark. Concurrent transition system semantics of process networks. In *Fourteenth ACM Symposium on Principles of Programming Languages*, pages 199–210, January 1987.

[37] E. W. Stark. Concurrent transition systems. *Theoretical Computer Science*, 64:221–269, 1989.

[38] E. W. Stark. Connections between a concrete and abstract model of concurrent systems. In *Fifth Conference on the Mathematical Foundations of Programming Semantics*, Springer-Verlag. *Lecture Notes in Computer Science*, New Orleans, LA, 1990. (to appear).

[39] E. W. Stark. On the relations computed by a class of concurrent automata. In *Seventeenth Annual ACM Symposium on Principles of Programming Languages*, pages 329–340, January 1990.