A Fixpoint Semantics for Nondeterministic Data Flow

JOHN STAPLES AND V. L. NGUYEN

University of Queensland, St. Lucia, Australia

Abstract. Criteria for adequacy of a data flow semantics are discussed and Kahn's successful semantics for functional (deterministic) data flow is reviewed. Problems arising from nondeterminism are introduced and the paper's approach to overcoming them is introduced. The approach is based on generalizing the notion of input–output relation, essentially to a partially ordered multiset of input–output histories. The Brock–Ackerman anomalies concerning the input–output relation model of nondeterministic data flow are reviewed, and it is indicated how the proposed approach avoids them. A new anomaly is introduced to motivate the use of multisets. A formal theory of asynchronous processes is then developed. The main result is that the operation of forming a process from a network of component processes is associative. This result shows that the approach is not subject to anomalies such as that of Brock and Ackerman.

Categories and Subject Descriptors: F.1.0 [**Computation by Abstract Devices**]: General; F.3.2 [**Logics and Meanings of Programs**]: Semantics of Programming Languages–*denotational semantics*

General Terms: Theory

Additional Key Words and Phrases: Asynchronous, data flow, denotational, nondeterminism, semantics

1. *Introduction*

Data flow is a general term for the behavior exhibited by networks of machines (or virtual machines) of the following class.

Data flow machines have designated input ports and output ports. Each port has a designated data type. Their input ports are always ready to receive data, which is stored ("buffered") for later processing. The capacity of such buffering is unbounded. No assumptions are made about the timing of the events that produce output.

Such a machine, or a description of its behaviors, may be called an *asynchronous process.*

The assumption of unbounded buffering capacity is not a restrictive one. Within this class of models, we can define nondeterministic processes which "lose" input data randomly, in such a way as to model bounded or null-buffering capacity. Also, subclasses of processes can be defined that are implicitly tightly synchronized with each other, by the fact that at each transition a datum is read from each input port and written at each output port. Since both these cases are subcases of the general case of unbounded buffering, we do not consider them separately.

Part of the work of the first author was done while on leave at the Computer Science Department, Stanford University. The work of the second author was supported by an Australian Computer Research Board Postgraduate Scholarship.

Authors' present addresses: J. Staples, Department of Computer Science, University of Queensland, St. Lucia, Queensland 4067, Australia; V. L. Nguyen, Computer Science Department, Cornell University, Ithaca, NY 14853.

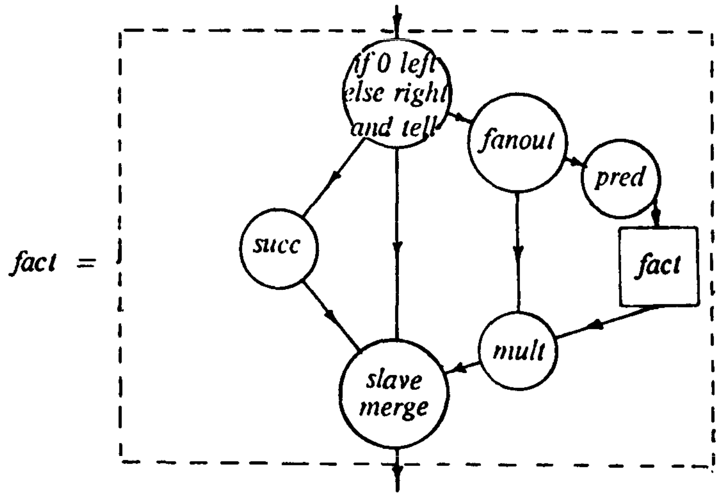

FIG. 1. Recursive definition of a factorial process.

A semantics for data flow is a theoretical model of the possible behaviors of such machines. Mechanistic machine models are possible, but more abstract models are valuable for supporting reasoning about the properties of machines.

From the point of view of this paper, a fixpoint semantics is one in which the behaviors of a network can be characterized as the limit (least upper bound) of a sequence of approximations, each approximation being a finite behavior realizable by a finite execution of the machine in question.

Such a semantics has an intuitive appeal. It also has the technical merit of supporting reasoning about machine behaviors by induction on the sequence of approximations.

It is not a priori clear that some other type of semantics might not have equal or greater benefits. If we enumerate some criteria that a semantics should satisfy, however, then we see that our approach meets them well and compares well with previous approaches.

Here are some such criteria. The purpose of the paper is to show how our approach meets all of them except extensionality. In Section 9, we compare our approach with previous work.

1.1. CRITERIA FOR A DATA FLOW SEMANTICS

1.1.1. *Modularity.* There should be a way to define a single process for each network of machines, from the processes of its components, which refers only to the external behavior of the network.

1.1.2. *Input–Output Correct.* The set of total input–output histories of a machine should be derivable from its process. This set is the set of pairs $(x; y)$, where x is an input history, that is, a sequence of data values for each input port, and y is an output history that describes a total response (over all time) of the machine to the input x.

If the set of all partial and total output histories is derivable from its process, we shall call the process model partially input–output correct.

1.1.3. *Support Definition by Recursion.* For the present, we illustrate this concept informally by means of the diagram in Figure 1, which is a recursive definition of a factorial process. It converts an input history comprising a finite or infinite sequence of natural numbers into an output history comprising their factorials, in order.

Each of the components of this network is deterministic. Its total input–output history is a function; each input history uniquely determines a corresponding total output. The functions for the components of the network may be defined as follows, where we abbreviate "if ··· tell" to "*itet*." We denote by $X.Y$ the concatenation of the sequences X and Y. Likewise for vectors of sequences, $(X, U).(Y, V)$ denotes $(X.Y, U.V)$. The symbol $\perp$ denotes the empty sequence.

$$\begin{aligned}
\mathit{itet}(\perp) &= (\perp,\perp,\perp)\\
\mathit{itet}(0.X) &= (0,0,\perp).\mathit{itet}(X)\\
\mathit{itet}(n.X) &= (\perp,1,n).\mathit{itet}(X), \qquad n>0
\end{aligned}$$

$$\begin{aligned}
\mathit{succ}(\perp) &= \perp\\
\mathit{succ}(n.X) &= (n+1).\mathit{succ}(X)
\end{aligned}$$

$$\begin{aligned}
\mathit{pred}(\perp) &= \perp\\
\mathit{pred}(0.X) &= 0\\
\mathit{pred}((n+1).X) &= n.\mathit{pred}(X)
\end{aligned}$$

$$\begin{aligned}
\mathit{mult}(X,\perp) &= \perp\\
\mathit{mult}(\perp,Y) &= \perp\\
\mathit{mult}(a.X,b.Y) &= (a^*b).\mathit{mult}(X,\ Y)
\end{aligned}$$

$$\mathit{fanout}(X) = (X,\ X)$$

$$\begin{aligned}
\mathit{slavemerge}(X,\perp,Z) &= \perp\\
\mathit{slavemerge}(a.X,0.Y,c.Z) &= a.\mathit{slavemerge}(X,Y,c.Z)\\
\text{For}\quad b\neq 0,\ \mathit{slavemerge}(a.X,b.Y,c.Z) &= c.\mathit{slavemerge}(a.X,Y,Z)
\end{aligned}$$

1.1.4. *Abstract.* If two networks define the *same* process, then so should their substitutions in every context. Intuitively, a context is a network with a hole in it, in which another network may be substituted.

1.1.5. *Extensional.* If two machines define *different* processes, then their substitutions in some context should have different sets of total input–output histories.

1.1.6. *General.* In particular, we seek to model nondeterminism, including the aspect of fairness, as discussed in 3.1.

1.2. Review of Kahn's Model.

All the criteria listed above, except generality, are satisfied by Kahn's semantics for deterministic data flow [4]. In this paper, a new semantics of asynchronous processes is developed. It provides a fixpoint characterization of network processes, extending Kahn's model [4] of deterministic processes.

According to Kahn [4], a deterministic asynchronous process over a given data type D is characterized by its input–output (or *history*) function, which specifies the complete output sequences (or *histories*) at all output ports, given input sequences at all input ports.

For good practical and theoretical reasons, Kahn made some mild assumptions about the nature of these functions, in terms of the following partial order on data sequences.

A sequence A is defined to be less than or equal to another sequence B if and only if A is a prefix (initial subsequence) of B.

With respect to that order, Kahn assumed that history functions are *continuous*. That means they are *monotonic* (future input defines an output that extends the

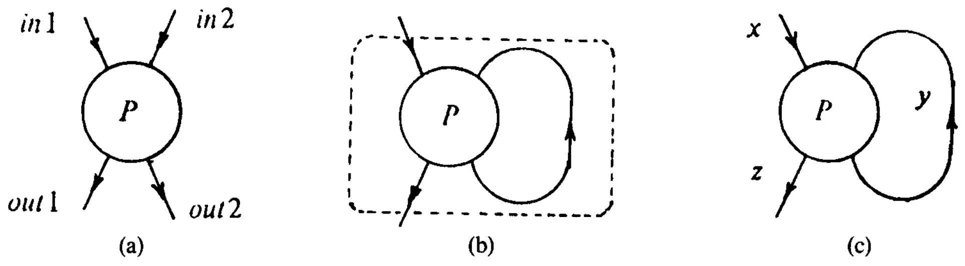

FIG. 2. An example of the use of Kahn's method. (a) P. (b) Network defining Q. (c) Labeled edges of network.

output defined by the current input) and *continuous at limits* (a finite output, which can occur at all, occurs in response to some finite initial part of the input).

Kahn's theory is simple and elegant. A network of processes is characterized as a system of mutually recursive equations for the history on each edge, in terms of the input histories and the history functions of the components. Its behavior for given input sequences can be obtained, and approximations to its behavior computed, by the well-known fixpoint method. The history function of the composite process can be defined in the same way, and it is continuous.

Here is an example of the use of Kahn's method.

Consider a process P as in Figure 2a. The figure gives names to each of the input and output ports of P. The history function of this process has the form

$$f\colon D \times D \rightarrow D \times D.$$

We may write f componentwise as $f = (f_1, f_2)$, where $f_1\colon D \times D \rightarrow D$ and similarly for f_2.

Now suppose that we wish to compute the history function of the process Q obtained from P by connecting *out*2 to *in*2; as in Figure 2b. Kahn's method is to assign history variables to edges (e.g., as in Figure 2c), then to write down the mutually recursive equations in these variables that are defined by the history function, thus:

$$y = f_2(x, y),$$
$$z = f_1(x, y).$$

These equations can be arranged in the form $X = F_x(X)$, as follows:

$$(y, z) = (f_2 \circ (\bar{x}, \mathrm{pr}_1), f_1 \circ (\bar{x}, \mathrm{pr}_1))(y, z),$$

where

pr_1 = The projection of an ordered pair onto its first coordinate;
$\bar{x}$ = The function with constant value x;
$\circ$ = The composition operation on functions.

This function F is continuous with respect to the prefix partial order on histories, in both X and x. The following two properties of that partial order ensure that a well-known theory can be applied to show that, for continuous functions F, such an equation has a *least* solution. Intuitively, the least solution is the solution of practical interest.

(a) There is a least element, $\perp$, in the partial order. (It is just the empty history.)

(b) Increasing sequences of histories have least upper bounds. (The union of an increasing sequence of histories is a history that is the least upper bound of the sequence.)

In general, a partial order that satisfies these two conditions is called an *ω-chain complete* partial order (cpo).

The least solution of $X = F_x(X)$ is the least upper bound (lub) of the increasing sequence

$$\bot,\ F_x(\bot),\ F_x(F_x(\bot)),\ \ldots$$

and is usually denoted

$$\mathrm{lub}_n F_x^n(\bot),$$

or loosely,

$$\mathrm{lub}\ F_x^n(\bot).$$

Recall that our ultimate goal is to solve for the output history of the new process in terms of its input history. Since $X = (y, z)$, the solution for z is just

$$z = \mathrm{pr}_2(\mathrm{lub}\ F_x^n(\bot)).$$

This description of the solution makes it clear that we can compute approximations to the solution, provided F_x is computable. It can also be shown that z is a continuous function of x.

Unfortunately, however, Kahn's method does not extend naively to general asynchronous processes, as shown in [2] and [3], and as reviewed in Section 3.

1.3. A POINT OF NOTATION. In his original paper, Kahn restricted his theory to a proper subset of the continuous functions, as we explain in 2.3.2. He called the smaller class of processes he considered *deterministic*. (This name can cause confusion, since not every such process is definable by a single deterministic automaton. Consider, for example, the process comprising two parallel, not interconnected, copies of the identity process.)

We call the larger class of processes that are within the scope of Kahn's method *functional*; that is, the class that can be described by a continuous history function from inputs to outputs.

2. *Some Basic Ideas and Their Relationship to Kahn's Theory*

2.1. SOME BASIC DEFINITIONS. For simplicity, we assume all data values are from some arbitrary but fixed data type. There is no difficulty in elaborating the approach to deal with a multiplicity of data types.

A sequence of data values, finite or infinite, may be called a *port history*.

We assume that each process P has a fixed finite set of input ports and a fixed finite set of output ports.

An association to each port of P of a port history may be called a *history over the ports of P*, or a *process history*. It is convenient to decompose such a history into two parts: the *input history*, which associates a port history to each input port, and the *output history*, which associates a port history to each output port.

We may represent a history over the ports of P by a pair $(u;v)$, where u denotes an input history and v denotes an output history.

For $x = (u;v)$, we may also write $in(x)$ for u and $out(x)$ for v.

Also we may write, for example, $(u;v,w)$, if it is desired to reference the decomposition of the output history into the disjoint parts v and w.

The *prefix* ordering on histories is defined as follows. For histories u and v, $u \leq v$ means that u and v have the same input and output ports, and, for each such port, its u-history is an initial subsequence of its v-history.

For a given set p of ports, we may write **Hist**(p) for the partial order whose elements are all histories over the ports in p, with the prefix order. When the port set is clear from the context, we may abbreviate **Hist**(p) to **Hist**.

The least element, if any, of any partial order may be denoted $\perp$.

2.2. THE CONCEPT OF PROCESS. Given a finite set p of input and output ports, our concept of a process P over p will be that P is a partially ordered set, whose elements are labeled by histories over p (equivalently, a partially ordered multiset of such histories), which satisfies the axioms stated in Section 4.

We write $\sqsubseteq$ to denote the process partial order. Intuitively, for finite elements x and y, $x \sqsubseteq y$ may be interpreted to mean that the least computation of x can be extended to the least computation of y.

A fundamental reason for using multisets is to distinguish between the various minimal computations of a single history, so that each can be regarded as a least computation of some instance of that history.

2.3. EXAMPLE OF FUNCTIONAL PROCESSES. Recall that the defining property of a functional process is that there is a function, f say, from input histories to output histories that is continuous in the initial subsequence ordering and such that, for every input history u, $f(u)$ is the total output history for that input.

2.3.1. *The Modeling of Functional Processes.* We would like to define, from the history function f of an arbitrary functional process, a process $P(f)$ in our sense, as follows.

$$P(f) = \{(u;v): v \leq f(u)\}.$$

We equip $P(f)$ with the prefix ordering.

Notice that nothing is lost by representing f as $P(f)$. We can recover f from $P(f)$. It is the set of elements $h = (u;v)$ of $P(f)$ such that h is the greatest element of $P(f)$ with input u.

In the context of nondeterminism, this approach succeeds for a large class of functional processes, which includes all the deterministic processes originally considered by Kahn. However, there are cases where, to conveniently interface with nondeterministic processes, more care is needed. To illustrate this point, consider the following example.

2.3.2. *The History Function* Detect. We consider a process that has two input ports and one output port. It signals, by emitting a single 0, the arrival of any data at either of its input ports. Formally, we define its history function *Detect* as follows.

$$\begin{aligned} Detect(\perp,\perp) &= \perp, \\ Detect(a.X, Y) &= 0, \\ Detect(X, b.Y) &= 0. \end{aligned}$$

Intuitively, *Detect* has the capacity to mask nondeterministic behavior. For our approach, it is important, when dealing with nondeterministic behavior, to recognize this capacity in functional processes. Since Kahn's method considers functional processes only in the context of other functional processes, this point is not significant for his analysis.

It is interesting that Kahn nevertheless ruled out processes such as *Detect*. His requirement was that processes should be constructible from atomic processes, each of which "is either computing or waiting for input on one of its input lines" (Kahn's underlining).

This requirement implies the following property (as we shall show in 2.4.1).

Property 2.1. For all finite x in $P(f)$, there is a least $z \leq x$ in $P(f)$ such that $out(z) = out(x)$.

It is equivalent to omit the restriction to finite histories, as we show in 2.4.2.

An alternative, equivalent criterion is as follows. Here and later it will be useful to call two elements x and y of a partial order P *consistent* if x and y have a common upper bound in P.

The equivalence will be proved in 2.4.3.

Property 2.2. For all consistent x and y in $P(f)$, the greatest lower bound of x and y in **Hist** is the greatest lower bound of x and y in $P(f)$.

It is clear that *Detect* fails these criteria since, for example, $(\perp, 0;0)$ and $(0, \perp;0)$ are in $P(Detect)$, but $(\perp, \perp;0)$ is not.

2.3.3. *Remark.* Because it is important for our later work, we note here that the property for joins (least upper bounds) corresponding to Property 2.2 is true for all history functions f, as a consequence of the monotonicity of f. Precisely, the property is as follows.

Property 2.3. For all consistent x and y in $P(f)$, the join $x \vee y$ of x and y in **Hist** is also an element of $P(f)$.

To see that, write $x = (u, v)$, $y = (u', v')$, so that $v \leq f(u)$, $v' \leq f(u')$. Now $u \leq u \vee u'$, so $f(u) \leq f(u \vee u')$, and similarly for $f(u')$, so $f(u) \vee f(u') \leq f(u \vee u')$.

Hence, $v \vee v' \leq f(u) \vee f(u') \leq f(u \vee u')$, so $(u \vee u', v \vee v')$ is in $P(f)$.

2.3.4. *The Process* DETECT. As a simple example of the application of our approach, we describe a process *DETECT*, which is a model in our theory of the functional process with history function *Detect*.

Since *DETECT* is based on a multiset of histories, we provide indices to label different instances of the same history, in order to define conveniently their role in the partial order.

In this simple example, there are only three possible indices, which, for definiteness, we take to be the following strings of symbols; the empty string, the string whose single symbol is 0, and the string whose single symbol is 1.

For each element $(a, b;c)$ of $P(Detect)$ and each index i, $(a, b;c)_i$ is an element of *DETECT* if and only if the length of c equals the length of i. In case i is 0 (respectively, 1), then a (respectively, b) is nonempty.

The ordering of *DETECT* is defined by

$$(a, b;c)_i \sqsubseteq (d, e;f)_j$$

just if $(a, b;c) \leq (d, e;f)$ and i is an initial subsequence of j.

For example, $(\perp, 0;0)_1$ and $(0, \perp;0)_0$ are in *DETECT*, but $(\perp, 0;0)_0$ and $(0, \perp;0)_1$ are not. Both $(0, 0;0)_0$ and $(0, 0;0)_1$ are in *DETECT*.

The process *DETECT* models the capacity of *Detect* to mask nondeterminism, intuitively by recognizing its capacity to make a nondeterministic choice of input port. Formally, *DETECT* satisfies the following generalization of Property 2.2.

Property 2.4. Every consistent pair x and y in P has a greatest lower bound in P; which is an instance of the greatest lower bound of x and y in **Hist**.

The process *DETECT* satisfies this condition simply because, although $(\perp,0;0)_1$ and $(0,\perp;0)_0$ are both in *DETECT*, they are not consistent.

2.4. Retracting History Functions. A function f satisfying one of the equivalent properties 2.1 and 2.2 is called *retracting*. So maybe the process $P(f)$, which it defines. For example, all one-input functional processes are retracting.

The name is derived as follows. Given such a function f, a function g may be defined by

$$g(y) \text{ is the least } x \text{ such that } y = f(x).$$

Then, $g \circ f$ is a retraction of the domain of f in the sense that it maps each input history x in the domain of f to the least input history having the output $f(x)$. In particular, it leaves those least input histories invariant.

In this notation we can say: retracting history functions f define $P(f)$'s, which are processes in our sense and which are sets rather than multisets.

Functional processes that have retracting history functions may be called retracting also. In our theory, it is just the retracting functional processes that are representable as partially ordered sets; other processes need the power of the multiset concept.

2.4.1. *Proof That Kahn's Deterministic Processes Are Retracting.* Recall that Kahn's concept of a deterministic process requires that it should be built from a network of atomic processes, each of which "is either computing or waiting for output on one of its input lines."

Here we show that all such processes are retracting. In Lemma 7.4, we show that, for functional processes, retracting is preserved under network construction. Hence, all Kahn's deterministic processes are retracting.

Our argument is in the contrapositive. We consider $P(f)$, which is not retracting, and we show that it does not satisfy Kahn's condition.

Since $P(f)$ is not retracting, it has a finite element $h = (u;v)$, such that there is no least history $h' \leq h$ with the same output history as h.

Now since prefix ordering on histories is *noetherian* (synonyms: wellfounded, all descending chains are finite), then there are two distinct minimal histories below h, say

$$x = (p;v) \qquad \text{and} \qquad y = (q;v)$$

with the same output history as h.

We see as follows that x and y have a greatest lower bound in $P(f)$. From 2.3.3, the set of common lower bounds of x and y is a directed subset of $P(f)$. That set is finite, since x and y are finite. Hence, it has a greatest element, which we denote $x \wedge y$. Consider this meet $z = x \wedge y$ of x and y. Say $z = (p \wedge q;w)$. Since $P(f)$ is not retracting, $w \neq v$. Since x and y both have minimal inputs for output v, it is necessary to increment the input history of z before the output history can be incremented.

Consider arbitrary $z_x = (r_x;w)$ such that $z < z_x < x$, and consider arbitrary $z_y = (r_y;w)$ such that $z < z_y < y$.

Each of z_x and z_y increments input port histories of z. By the definition of z, the two increments have no input events in common. Neither can they be inconsistent.

On the other hand, there are output-producing transitions of the process from z towards x and from z toward y. They occur in response to different input configurations, contradicting Kahn's determinism condition; which concludes the proof. □

2.4.2. *Proof That Property* 2.1 *Is Equivalent to: for All x in $P(f)$, There Is a Least $z \leq x$ in $P(f)$ Such that* $\mathrm{out}(z) = \mathrm{out}(x)$. Clearly this form implies Property 2.1. We consider the converse.

Each element x of $P(f)$ is the limit of an increasing sequence (x_n) in $P(f)$ of finite elements, from the continuity of f. Say (y_n) is a least element below x such that $out(y_n) = out(x_n)$. Then (y_n) is increasing, since $out(x_n \wedge y_{n+1}) = out(x_n)$, so that $y_n \leq x_n \wedge y_{n+1} \leq y_{n+1}$. From the continuity of f, $P(f)$ is ω-complete, so (y_n) has a limit, y say, in $P(f)$.

If $z \leq x$ and $out(z) = out(x)$, then $out(z \wedge y_n) = out(y_n)$, so $y_n \leq z \wedge y_n \leq z$ for all n, hence $y \leq z$. That is, y is the least element w of $P(f)$ below x such that $out(x) = out(w)$. □

2.4.3. *Proof of Equivalence of Property* 2.1 *and Property* 2.2. First, we suppose Property 2.2 and show Property 2.1. Since the prefix ordering on histories is noetherian, then, given $x = (u;v)$, there exists at least one minimal $z = (u';v)$ such that $z \leq x$.

Suppose that z' is another such minimal element. Then $z \wedge z'$ is below z and has output history v also, so $z \wedge z' = z$; thus $z \leq z'$. Symmetrically, $z' \leq z$.

Conversely, suppose now Property 2.1, and consider $x \leq z$, $y \leq z$. By hypothesis, there is a least w below z such that $out(w) = out(x) \wedge out(y)$, $= k$ say.

Note that $out(w \wedge x) = out(w)$, so $w \leq w \wedge x$. That is, $w = w \wedge x$ and so $w \leq x$. Similarly, $w \leq y$.

Hence, w is a lower bound for x and y in $P(f)$. From 2.3.3, the union of w with $(in(x) \wedge in(y);\perp)$ is in $P(f)$. It is the greatest lower bound of x and y in **Hist**, as required. □

3. *Anomalies and Other Motivations*

We describe some basic nondeterministic processes. We review the Brock–Ackerman anomalies, to show how they are resolved in our approach. We also introduce a new anomaly, to motivate our use of multisets.

3.1. NONDETERMINISM AND FAIRNESS: THE FAIR MERGE

3.1.1. *An Unfair Merge Process* UM. Intuitively, we seek a process that can merge two streams of input data into a single output stream. It should be nondeterministic so that it will not wait forever for data from an input which receives none, while there are data available at the other input.

A first try at defining the total input–output history relation r_{UM} of UM might comprise the following recursive definition.

$$\begin{aligned} r_{UM}(\perp,\perp) &= \{\perp\}, \\ r_{UM}(a.X,\perp) &= a.X, \\ r_{UM}(\perp,b.Y) &= b.Y, \\ r_{UM}(a.X,b.Y) &= a.r_{UM}(X,b.Y) \cup b.r_{UM}(a.X,Y). \end{aligned}$$

However, a process with such a relation has the disadvantage of being unfair. Its input–output relation permits it to favor a single infinite stream of input data, forever neglecting some or all of the data available at the other input.

From a practical point of view, such processes are not always appropriate, as the example of 3.1.3 illustrates. Hence, we make the following definition. However, UM does have a role in the theory of data flow, as illustrated in 3.1.4.

3.1.2. *A Fair Merge Process* MERGE. The merge process to be considered has two input ports, say left and right, and a single output port. Intuitively, for given input histories, the corresponding complete output history is an arbitrary merging of the input histories.

The total input–output history relation r_M of *MERGE* is defined as follows, where "\" denotes the set difference operator.

$$r_M(X, Y) = r_{UM}(X, Y)\setminus \bigcup_{X'} r_{UM}(X', Y)\setminus \bigcup_{Y'} r_{UM}(X, Y'),$$

where X', Y' range over the proper prefixes of X and Y, respectively.

The anomalies discussed in the following paragraphs show that this relation r_M is inadequate for modeling the merge process.

We can however use r_M as a stage in the definition of an adequate merge model. Note that it makes sense to apply the operation $P(\,)$ to r_M, as follows:

$$P(r_M) = \{(x;y)\colon \text{ there is } x' \leq x \text{ and } y' \leq y \text{ such that } y' \text{ is in } r_M(x')\}.$$

We construct a process *MERGE* from the elements of $P(r_M)$.

As in the discussion of *DETECT*, we use indices to name conveniently the multiple instances of histories that are involved.

We take as indices all sequences of zeros and ones, of length n, $n = 1, 2, \ldots \omega$. For $(x, y;z)$ in $P(r_M)$,

$$(x, y;z)_i \text{ is in } MERGE$$

just if the length of i equals the length of z, and if the zeros and ones of i specify a decomposition of z into two disjoint sequences that are initial subsequences of x and y, respectively.

The partial order on the elements of *MERGE* is defined as follows.

$$(x, y;z)_i \sqsubseteq (u, v;w)_j$$

just if $(x, y;z) \leq (u, v;w)$ and i is an initial subsequence of j. It is straightforward to check that *MERGE* satisfies the conditions of Section 4.

3.1.3. *An Example of the Role of Fairness.* Consider a network as in Figure 3, wherein all processes other than *MERGE* are deterministic, as follows. 1^ω outputs an infinite sequence of ones; 0 outputs a single zero. The *null* process outputs nothing. The process *cat* is defined as follows.

$$\begin{aligned} cat(\perp, Y) &= \perp, \\ cat(a.X, Y) &= a.cat(X, Y) \qquad \text{if} \quad a \neq 0, \\ cat(0.X, Y) &= Y. \end{aligned}$$

As *MERGE* is fair, the process defined by this network can generate 1^n for all natural numbers n, but not 1^ω. If we had used *UM* instead, the process would have had the option of outputting 1^ω.

That is actually a matter of some practical importance. For example, one wishes to be able to model, by an elaboration of this example, the capacity to reset suitably configured processes. In the absence of fairness in an asynchronous system, that is impossible.

3.1.4. *An Example of the Role of Unfairness.* Although one might doubt whether a practical machine design would ever call for an unfair merge process, unfair merging can easily arise in the analysis of nondeterministic processes. Here is a contrived example of that.

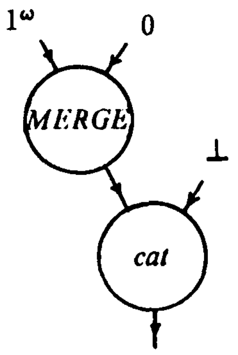

FIG. 3. A network illustrating the role of fairness.

Consider a process with two input ports and one output port. Its behavior is to read integers from its left input and copy them onto its output, until it reads an integer of the form

$$2^x.3^y.5^z.7^n \qquad n > 2,$$

such that

$$x^n + y^n = z^n.$$

After copying that integer to its output, the subsequent behavior is to fairly merge its two inputs.

Does this process fairly merge? That is, is this process $\leq MERGE$, in the ordering of processes to be defined in 4.1.1? It is hard to say! One can say, however, that it is $\leq UM$. It then follows from our later work that a network M in which our process occurs defines a process P, which is $\leq$ the corresponding process Q defined by using UM instead. Thus, if a result of the form $P \leq R$ is required, it is sufficient to show $Q \leq R$.

3.2. THE FIRST ANOMALY OF BROCK AND ACKERMAN. Brock and Ackerman [2] observed that the two networks of Figures 4a and 4b, while having the same total input–output history relation, nevertheless can be distinguished in the context of Figure 4c, by the different responses of the two substituted networks to input "0". The earlier work of [5] in this area was not conclusive.

All the components in all networks of Figure 4, except for *MERGE*, are deterministic. For definiteness, a boolean data type is assumed. The history functions of the deterministic components may be defined as follows:

$$cons^{-1}(\perp) = \perp,$$
$$cons^{-1}(a.X) = (a, X),$$

$$cons(\perp, Y) = \perp,$$
$$cons(a.X, Y) = a.Y,$$

$$wait(X, \perp) = \perp,$$
$$wait(X, b.Y) = X,$$

$$not(\perp) = \perp,$$
$$not(0.X) = 1.not(X),$$
$$not(1.X) = 0.not(X).$$

It may help analysis of the difference between the two networks to consider them in the forms of Figures 5a and 5b, where the deterministic processes *da* and *db*

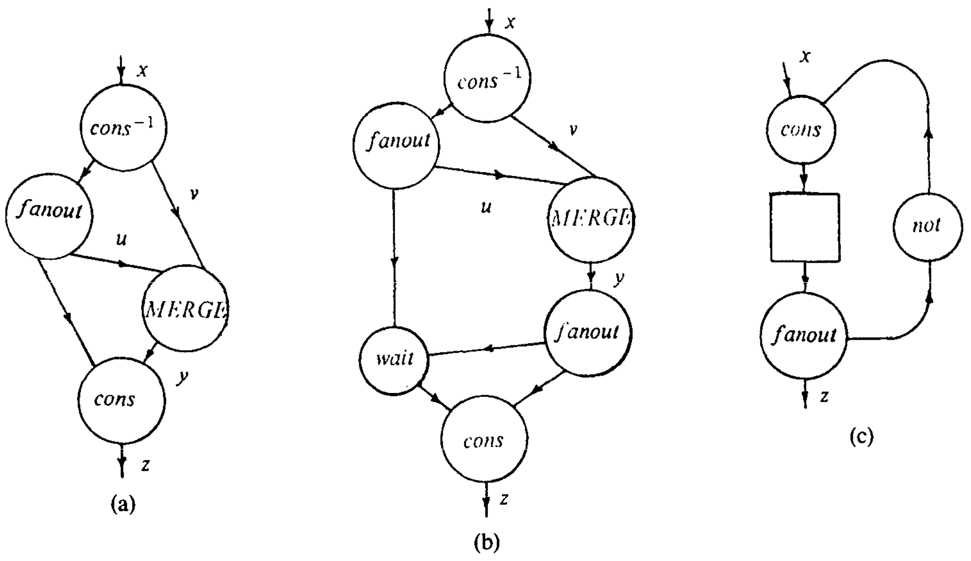

FIG. 4. Networks *MA* and *MB*. (a) *MA*. (b) *MB*. (c) Context.

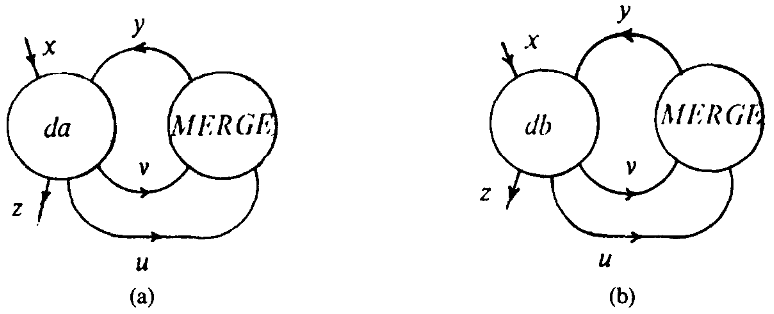

FIG. 5. Different forms of networks *MA* and *MB*. (a) *MA*. (b) *MB*.

have history functions defined as follows:

$$da(x,y) = (cons(head(x),y), head(x), tail(x)),$$
$$db(x,y) = (cons(wait(head(x),y),y), head(x), tail(x)),$$

where

$$head(\bot) = \bot,$$
$$head(a.X) = a,$$

$$tail(\bot) = \bot,$$
$$tail(a.X) = X.$$

The point of the anomaly is that in *MB*, if one inputs "0" and then waits for output before inputting "1," then *MB* outputs a stream of the form "0.0···." However, when *MA* is subjected to the same treatment, the resulting streams may have either of the forms "0.0 ···" or "0.1 ··· ."

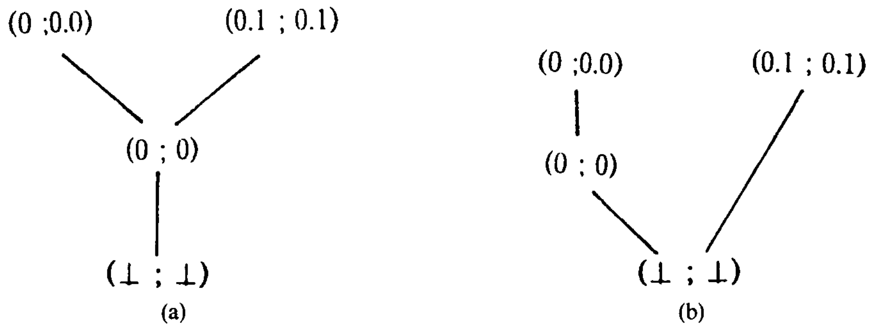

FIG. 6. Parts of the input–output histories. (a) *MA*. (b) *MB*.

This difference is masked in the total input–output history relations by the fact that input "0.1" to *MB* can result in output of the form "0.1 $\cdots$," if one does not wait for the first output datum before inputting the "1."

This difference is not exposed simply by considering partial as well as total input–output histories, since the latter determine the former. However, if we do consider the partial as well as the total input–output histories, we have the opportunity to resolve the anomaly by the use of our concept of order, introduced in 2.2.

For, in *MA* with input "0," the least computation that produces an output "0" is the one in which *MERGE* processes no input—leaving it free to subsequently choose a "1" at the right input. In *MB* with input "0" however, the least computation that produces an output "0" is one in which the *MERGE* process does select a "0" from its left input, and by outputting it ensures that the *MB* output will have the form "0.0 $\cdots$."

The set of partial and total input–output histories, ordered in accordance with the principle of 2.2, is sketched in part in Figure 6. We see that the partial orders are different.

In fact, the method we shall develop does not lead precisely to these orders, but to orders of which these are homomorphic images. However, the distinction remains.

3.3. THE SECOND ANOMALY OF BROCK AND ACKERMAN.

This anomaly was described in [3]. It is similar in essence to the previous one.

Two networks *NA* and *NB*, as sketched in Figures 7a and 7b, have the same total input–output history relation, but are distinguished in the context of Figure 7c by input "0."

The following functions are the history functions of those deterministic processes that occur in these networks, but have not previously been defined.

3.3.1. *Definition.* A duplicating function, *duplicate*, is defined by

$$\begin{aligned} duplicate(\bot) &= \bot, \\ duplicate(a.X) &= a.a.duplicate(X). \end{aligned}$$

3.3.2. *Definition.* A filter A is defined by

$$\begin{aligned} A(\bot) &= \bot, \\ A(a) &= a, \\ A(a.b.X) &= a.b. \end{aligned}$$

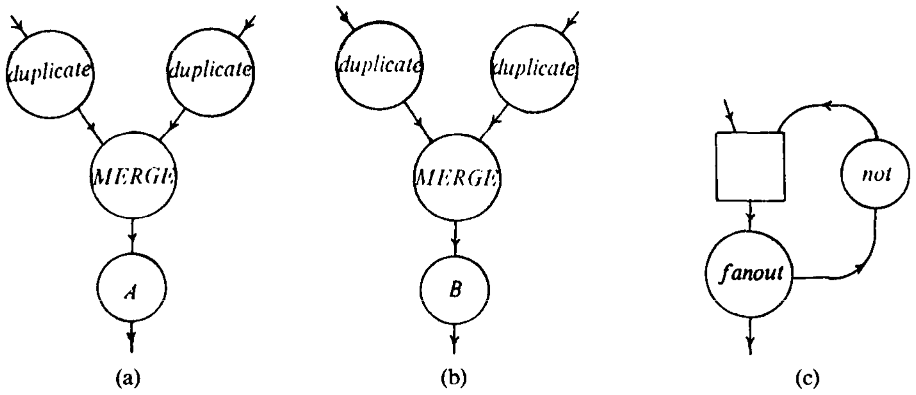

FIG. 7. Networks *NA* and *NB*. (a) *NA*. (b) *NB*. (c) Context.

3.3.3. *Definition.* A filter *B* is defined by

$$B(\bot) = \bot,$$
$$B(a) = \bot,$$
$$B(a.b.X) = a.b.$$

One sees that in the context of Figure 7c, *NA* has the option of an output stream of the form "0.1 ⋯ ," whereas *NB* does not.

Again, this difference in behavior is masked in the total input–output history relations by the fact that one is unable to distinguish between inputting at the right port before output appears, and after output appears.

Again, the distinction can be made straightforwardly by choosing to order the partial and total input–output histories in accordance with the principle of 2.2. Relevant fragments of these orders are sketched in Figure 8.

3.4. AN ANOMALY TO SHOW THE NECESSITY OF MULTISETS. Since both of the Brock–Ackerman anomalies are resolvable without the use of multisets, it is worthwhile to exhibit an anomaly that is not.

Consider the two networks *LA* and *LB* of Figures 9a and 9b. The deterministic processes not yet defined have the following history functions.

1 emits a single "1."

$$K0(\bot) = \bot,$$
$$K0(a.X) = 0,$$

$$K1(\bot) = \bot,$$
$$K1(a.X) = 1.$$

The processes *LA* and *LB* have the same input–output history relations, but perform differently in the context of Figure 9c with input "0."

Intuitively, if one inputs "0" to the left input port of *LB* and then waits for output before inputting "1" at the right input port, then the output at the left output port must be "0.0." Under the same conditions, *LA* may output either "0.0" or "0.1" at the left output port.

This distinction, however, cannot be described by attributing to *LB* any refinement of the prefix order on the partial and total input–output history relations

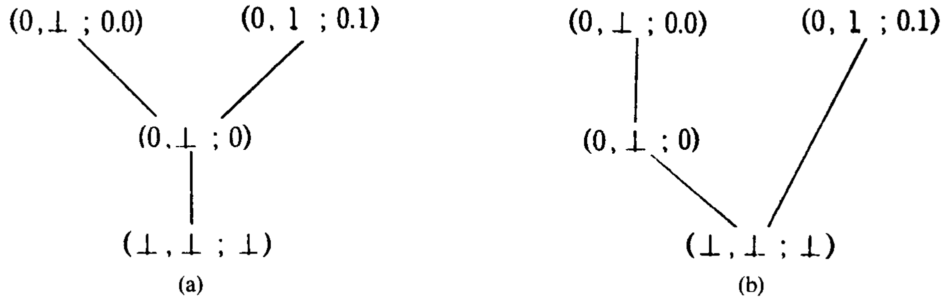

FIG. 8. Relevant fragments of partial or total input–output histories. (a) Part of *NA* order. (b) Part of *NB* order.

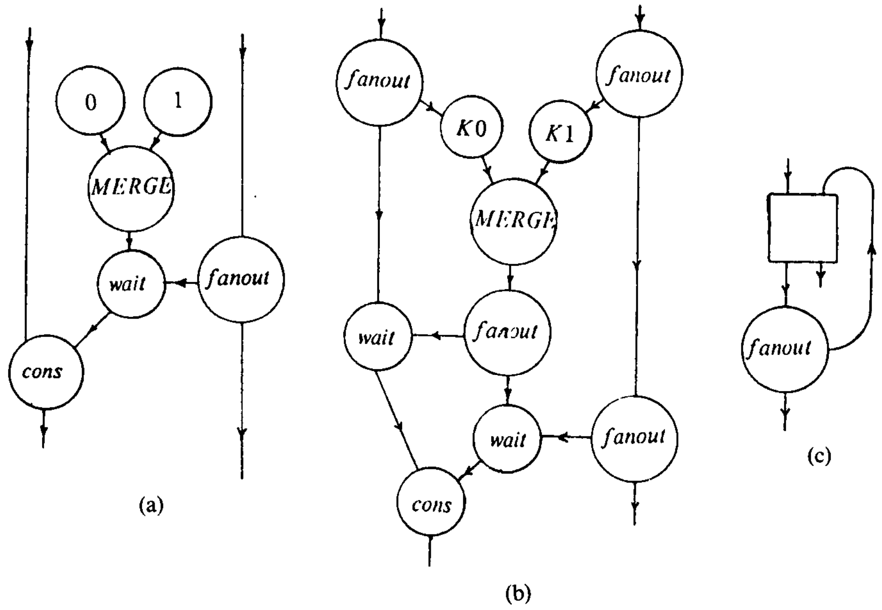

FIG. 9. Networks *LA* and *LB*. (a) *LA*. (b) *LB*. (c) Context.

because any such order should include

$$(0,\bot;0,\bot) \leq (0,1;0,1)$$

and

$$(0,1;0,1) \leq (0,1;0.1,1)$$

but not their transitive closure,

$$(0,\bot;0,\bot) \leq (0,1;0.1,1).$$

The resolution of this anomaly in our approach is that the two appearances of $(0,1;0,1)$ in the desired inequalities are different instances of that history in the multiset order.

The partial ordering principle of 2.2 leads to partially ordered multisets of input–output histories for *LA* and *LB*, which are sketched in part in Figures 10a and 10b.

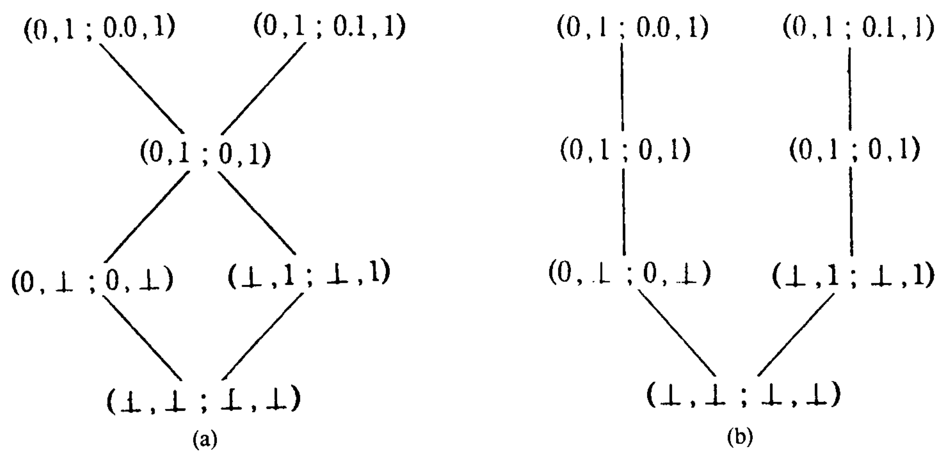

FIG. 10. Parts of the input–output histories for *LA* and *LB*. (a) *LA*. (b) *LB*.

4. *An Axiomatic Definition of Process*

As previously motivated, we define a process P over some fixed, finite set of ports p to be a structure

$$P = (E, \sqsubseteq, \mathbf{h}),$$

which satisfies the conditions stated in the following paragraphs. Here E is the set of elements of P, $\sqsubseteq$ is the partial order relation on E, and $\mathbf{h}$ is a function from E to $\mathbf{Hist}(p)$.

The axioms as stated here are, in some respects, more general than is convenient in applications. For example, they do not require a process to have a defined response to every finite or infinite input stream, nor do they require a process to be *extensional* in the sense that a process is uniquely determined by its response to all its environments (such a process concept has also been called *fully abstract*). We have chosen this weak axiomatization because it is a convenient framework within which to develop the elements of the theory.

Notation

(a) In the following axioms, P denotes an arbitrary process. Objects not specifically defined are implied to be elements of P.
(b) Here and later we may refer to the least upper bound of an increasing sequence in a partial order as its *limit*.
(c) Recall that two elements x and y of a partial order P are called *consistent* if they have a common upper bound in P.
(d) We call an element x of P *finite* if $\mathbf{h}(x)$ is a finite history.
(e) We write $in(x)$ for the restriction of $\mathbf{h}(x)$ to the inports of P, and $out(x)$ for the restriction of $\mathbf{h}(x)$ to the outports of P.

AXIOM 4.1. *Processes are poles.*

That is, there is a least element $\perp$ of P, and $\mathbf{h}(\perp) = \perp$.

AXIOM 4.2. *Principal ideals have prefix order.*

That is, if x and y are consistent and $\mathbf{h}(x) \leq \mathbf{h}(y)$, then $x \sqsubseteq y$.

AXIOM 4.3. *Processes are countably based.*

That is, every element of P is the limit of an increasing sequence of finite elements of P.

AXIOM 4.4. *The function* $\mathbf{h}$: $(E, \sqsubseteq) \to$ **Hist** *is continuous.*

That is,

(a) for all x and y, $x \sqsubseteq y$ implies that $\mathbf{h}(x) \leq \mathbf{h}(y)$;
(b) for every increasing sequence (x_n) in P that has a limit in P, $\mathbf{h}(\lim x_n) = \lim \mathbf{h}(x_n)$.

AXIOM 4.5. *Processes are locally join-closed.*

That is, if x and y are consistent, then x and y have a least upper bound $x \vee y$ in P such that $x \vee y \sqsubseteq z$ and $\mathbf{h}(x \vee y) = \mathbf{h}(x) \vee \mathbf{h}(y)$.

AXIOM 4.6. *Processes are locally meet-closed.*

That is, if x and y are consistent, then they have a greatest lower bound $x \wedge y$ in P such that $\mathbf{h}(x \wedge y) = \mathbf{h}(x) \wedge \mathbf{h}(y)$.

Note that only Axioms 4.1 and 4.3 impose consistency requirements, so the axioms are satisfied by trees that satisfy Axioms 4.1–4.4, inclusive. Some of the following would be simplified by considering *only* trees satisfying Axioms 4.1–4.4; but that would obscure the relationship with Kahn's work.

4.1. THE CPO OF ALL PROCESSES OVER A GIVEN SET OF PORTS. The collection, say **Proc**(p), of all processes over a given port set p is a large one. In set-theoretical terms, it is a class rather than a set, since we have made no restriction on the width of processes. Note Axiom 4.3 restricts their height.

Nevertheless, a simple and useful cpo structure can be defined on this class as follows.

Strictly, the cpo we consider is the cpo of isomorphism classes of processes. We ignore this distinction except where it is useful, but we do state the definition of process isomorphism in 4.2.

The purpose of such a cpo structure on the class of all processes is to support definition by recursion.

For $q \subseteq p$, we identify **Proc**(q) with the subset (in fact, ideal) of **Proc**(q), which comprises processes that have empty sequences at ports of $q \backslash p$.

4.1.1. *Definition of the cpo of All Processes over a Given Set of Ports.* We partially order the class **Proc**(p) of all processes over a given port set p as follows. By

$$P = (E_P, \sqsubseteq_P, \mathbf{h}_P) \leq Q = (E_Q, \sqsubseteq_Q, \mathbf{h}_Q)$$

we mean that (up to isomorphism)

(a) $E_P \subseteq E_Q$;
(b) $\sqsubseteq_P$ is the restriction of $\sqsubseteq_Q$ to E_P;
(c) $\mathbf{h}_P$ is the restriction of $\mathbf{h}_Q$ to E_P;
(d) P is an ideal of Q. That is, E_P is a subset of E_Q, which includes all y in E_Q such that for some x in E_P, $y \sqsubseteq x$.

In fact, we could develop our theory without assuming (d). To do so would be a little easier and shorter, and would generalize the work of this paper. It would not, however, *extend* the work of this paper, since the small extra effort we make shows that our stronger concept of order is preserved by network construction.

Nevertheless, it is useful to define a process P to be a *subprocess* of a process Q if it satisfies (a), (b), and (c) above.

With this definition of partial order, it is clear that the process whose only element is $\perp$ is the least element of $\mathbf{Proc}(p)$. Also, every increasing sequence (P_n) in $\mathbf{Proc}(p)$ has a least upper bound P that is just the union of the terms of the sequence.

More precisely, writing

$$P_n = (E_n, \sqsubseteq_n, \mathbf{h}_n),$$

we note that

(a) the set of elements of P is $\bigcup_n P_n$;
(b) $x \sqsubseteq y$ in P only if $x \sqsubseteq_n y$ in all P_n in which both x and y occur;
(c) $\mathbf{h}(x)$ in P is $\mathbf{h}_n(x)$ for all P_n in which x occurs.

It is straightforward to check that P is a process, that each P_n is an ideal of P (so that P is an upper bound for the sequence) and that for each upper bound Q of the sequence, $P \leq Q$.

4.2. Process Homomorphisms and Isomorphisms. Intuitively, a process homomorphism m: $P \to Q$ is a function that maps the elements of P to (some or all of) the elements of Q, and that preserves the process structure.

The basic structure of a process comprises its order and history function. However, we also wish the lattice structure on principal ideals to be preserved, which accounts for (e). Further, we wish to preserve the ability to interpret $\sqsubseteq$ as stated in 2.2. That is the reason for (f).

4.2.1. *Definition of Homomorphism.* Writing $P = (E_P, \sqsubseteq_P, \mathbf{h}_P)$, and similarly for Q, a homomorphism m: $P \to Q$ is a function $E_P \to E_Q$ which satisfies the following conditions:

(a) Preservation of least element.

$$m(\perp) = \perp.$$

(b) Monotonicity.

$$x \sqsubseteq_P y \quad \text{implies} \quad m(x) \sqsubseteq_P m(y).$$

(c) Preservation of histories.

$$\mathbf{h}_P(x) = \mathbf{h}_Q(m(x)).$$

(d) Preservation of limits; that is, m is continuous.
(e) Preservation of lattice structure of principal ideals. For all consistent x and y,

$$m(x) \vee m(y) = m(x \vee y),$$
$$m(x) \wedge m(y) = m(x \wedge y).$$

(f) Preservation of computability interpretation. For all y' such that $m(y) = m(y')$, if $m(x) \sqsubseteq m(y')$, then $x \sqsubseteq y$.

4.2.2. *Definition of Isomorphism.* An isomorphism m: $P \to Q$ is a homomorphism such that there is a homomorphism k: $Q \to P$ (necessarily unique) such that $m \circ k$ is the identity function on Q, and $k \circ m$ is the identity function on P.

Lemma 4.7. *A homomorphic image of a process is a process.*

That is, if m: $P \rightarrow Q$ is a homomorphism, then the set $m(P)$ of images of elements of P under m is a process when equipped with order and history function by restriction from Q.

To prove that, one checks straightforwardly each of the conditions of the definition of process.

5. *Networks of Processes*

5.1. INTRODUCTION. The network schemes we consider are the same as those in [7] and [9]. They are as simple as possible, in order to illustrate the basic idea. The method can be extended to deal with more elaborate network concepts, but we shall not do so here.

In such a network scheme there are places for finitely many processes. Static connections are made from output ports to input ports. Each input port can be connected to at most one output port, and vice versa. Connections from output ports to input ports of the same process are permitted.

For simplicity, we assume that each port of each place in a network scheme has a port name. Thus, a network scheme defines a one–one pairing of a set a of input port names of p with a set b of output port names of p. We may denote the pairing loosely by $(a;b)$. Where the ambiguity is not important, we may also use $(a;b)$ to denote the network scheme.

In the process to be defined by substituting processes for the places in the scheme, the connected ports are hidden.

Fanout, and hiding of disconnected input and output ports can be accomplished by use of appropriately defined processes. The example of fanout was treated in 1.1.3.

To construct a process from component processes using a network scheme, we follow [7] and [9] in using two kinds of operation, as described in the following sections.

It is essential to the interest of the work that the concept of network construction should be intuitively satisfying. Accordingly, we discuss the intuitive content of each definition before formalizing it.

5.2. THE DISJOINT UNION OPERATION. To avoid irrelevant details about renaming ports, we shall assume that no two processes whose disjoint union is to be formed have any port names in common.

We call this operation *disjoint union* because it creates from two component processes a composite process that models the grouping together of the components without any interconnections. The implementation of this concept is by means of the elementary notion of cartesian product.

Thus, the disjoint union $P_1 \dot{\cup} \cdots \dot{\cup} P_n$ of processes $P_1, \ldots, P_n$ is the Cartesian product of $P_1, \ldots, P_n$, with the usual product order induced by the orders of P_1, $\ldots, P_n$. See Figure 11.

The partial order and history function of the disjoint union are just the products of those of the components.

The following properties are elementary and will not be discussed in detail.

Property 5.1. Disjoint union is continuous.

That is,

(a) if $P_1 \leq P_2$ and $Q_1 \leq Q_2$, then $P_1 \dot{\cup} Q_1 \leq P_2 \dot{\cup} Q_2$;

(b) for all increasing sequences (P_n) and (Q_n) of processes,

$$\mathrm{lub}_m \mathrm{P}_m \dot{\cup} \mathrm{lub}_n Q_n = \mathrm{lub}_m \mathrm{lub}_n P_m \dot{\cup} Q_n = \mathrm{lub}_n P_n \dot{\cup} Q_n.$$

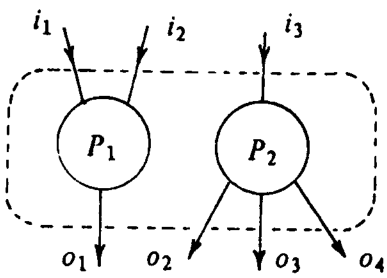

FIG. 11. The disjoint union $P_1 \dot{\cup} P_2$.

Property 5.2. Disjoint union is associative.

That is, up to isomorphism,

$$P_1 \dot{\cup} \cdots \dot{\cup} P_i \dot{\cup} P_{i+1} \dot{\cup} \cdots \dot{\cup} P_n = (P_1 \dot{\cup} \cdots \dot{\cup} P_i) \dot{\cup} P_{i+1} \dot{\cup} \cdots \dot{\cup} P_n.$$

Property 5.3. Disjoint union is commutative.

That is, up to isomorphism,

$$P_1 \dot{\cup} P_2 = P_2 \dot{\cup} P_1.$$

5.3. INTRODUCTION TO LINKING. Given a process P and strings a and b of input port names and output port names, respectively, of the same length, we denote by $\mathbf{Link}_{a;b}(P)$ the process obtained from P as follows. Intuitively, we connect the ports of a to the ports of b, one to one, in order, then hide all the input and output ports so connected. See Figure 2b.

To be more precise, the construction of $\mathbf{Link}_{a;b}(P)$ is as follows. The same approach has previously been used in [7] and [9] for different process models. A more convenient and modular definition of $\mathbf{Link}_{a;b}(P)$, together with a proof that it is a process, will be given in 6.7.

First, we describe the construction completely, then we discuss it.

We write $\mathbf{h}_{a;b}$ for the composition of the history map $\mathbf{h}$ of the process under discussion with that projection on histories, which removes from histories of P the histories of ports in a or b.

5.3.1. *Definition of Linking.* If t is a process history and d is a string of port names, denote by $t(d)$ the restriction of t to the ports of d. Now various sequences (t_n) can be chosen as follows:

Step 1. Choose t_1 to be the least element of P.

Step $h+1$. Choose t_{h+1} in P with finite history such that

$$t_{h+1}(a) \leq t_h(b) \qquad \text{and} \qquad t_h \sqsubseteq t_{h+1}.$$

We call a finite or infinite sequence generated in this way an $(a;b)$-finite sequence. Similarly, the limit of such a sequence may be called $(a;b)$-finite limit.

We also refer to the more general definition of sequence, which is obtained from the above by omitting the condition that t_{h+1} has a finite history. We call such sequences $(a;b)$*-sequences*, and their limits $(a;b)$*-limits.*

We call two $(a;b)$-limits x and y $(a;b)$*-equivalent* if there is z in P such that

$$\mathbf{h}_{a;b}(z) = \mathbf{h}_{a;b}(x) = \mathbf{h}_{a;b}(y) \qquad \text{and} \qquad z \sqsubseteq x,\ z \sqsubseteq y.$$

It is straightforward to check from Axiom 4.6 that this relation is an equivalence relation. Accordingly, we define the elements of $\mathbf{Link}_{a;b}(P)$ to be the set of $(a;b)$-equivalence classes of $(a;b)$-finite limits of P.

The partial order of $\mathbf{Link}_{a;b}(P)$ is that generated by the partial order of P; if $x \sqsubseteq y$ then their equivalence classes are ordered likewise. The history function of $\mathbf{Link}_{a;b}(P)$ is that induced by $\mathbf{h}_{a;b}$.

5.3.2. *Discussion of Linking.* It is intuitively satisfying that we restrict the $(a;b)$-sequences in the definition of linking to have elements of finite history, since it is then clear that all such sequences correspond to operationally realizable behaviors. It is not however essential, as we show in 6.1.

Hence, that restriction is not an essential difference from the iterations that are performed during Kahn's calculation of the history function in the deterministic case. The essential extension of Kahn's method is that we do not *require* of t_{h+1} that its output histories be as great as possible for the given input.

5.4. The Composite Process Defined by a Network. To define the composite process created by the substitution of component processes into a network scheme, we first take the disjoint union of the components and then link that disjoint union as prescribed by the network scheme.

The examples given in our introductory discussion may now be reviewed in order to check that the precise definitions yield the resolutions of the anomalies described. We repeat, however, that the method described here is not as economical as possible in its use of multisets, so that the process calculated by the theory may be homomorphic but unequal to the intuitively expected one. This lack of economy is compensated for by the considerable convenience of Axioms 4.5 and 4.6, and is insignificant in the sense of 7.4.

6. *The Theory of Linking*

Here we establish the technical basis for our main results.

6.1. An Equivalent Definition of Linking. The form of definition of $\mathbf{Link}_{a;b}$ given in 5.3.1 is intuitively appealing, but is not very convenient technically.

Here we establish that it is equivalent to replace 5.3.1 by the following definition, which is the form most used in the remainder of this paper.

6.1.1. *The Alternative Definition.* Instead of basing the definition of $\mathbf{Link}_{a;b}(P)$ on equivalence classes of $(a;b)$-finite limits, the limits we use are the limits of *arbitrary* increasing sequences of the objects t_h defined in 5.3.1.

The equivalence of this definition and 5.3.1 is established by the following lemmas, which also show that the restriction of finiteness of histories may be omitted in either case.

We define two sequences to be consistent if there is a common upper bound for all the elements of both.

We define the *concatenation* of two consistent $(a;b)$-sequences, the first of finite length, say $(x_i : i = 1, \ldots, m)$ and (y_n), to be the sequence (z_n), where

$$\begin{aligned} z_i &= x_i, & 1 \le i \le m, \\ z_{m+i} &= x_m \vee y_{i+1}, & i = 1, 2, \ldots. \end{aligned}$$

Lemma 6.1. *For every $(a;b)$-sequence (t_n),*

$$t_n(a) \le t_n(b).$$

Proof. By induction on n. In brief, $t_1 = \bot$ and for $n = k + 1$,

$$t_{k+1}(a) \le t_k(b), \qquad \text{by definition of } (a;b)\text{-sequence}$$

and

$$t_k(b) \leq t_{k+1}(b), \qquad \text{since} \quad t_k \sqsubseteq t_{k+1}. \qquad \square$$

COROLLARY 6.2. *Every $(a;b)$-sequence of finite length can be extended to an infinite $(a;b)$-sequence with the same limit, simply by repeating the last term indefinitely.*

LEMMA 6.3. *The concatenation of an $(a;b)$-sequence of finite length with a consistent $(a;b)$-sequence is again an $(a;b)$-sequence, which is $(a;b)$-finite if the component sequences are $(a;b)$-finite.*

PROOF. Evident, based upon the fact that, if $t(a) \leq t(b)$ and $t'(a) \leq t'(b)$, then $(t \vee t')(a) \leq (t \vee t')(b)$. $\square$

LEMMA 6.4. *If $x \sqsubseteq y$ and y is finite and $y(a) \leq x(b)$ and x is an $(a;b)$-limit, then so is y.*

(Note that since y is finite, all $(a;b)$-sequences converging to x or y are $(a;b)$-finite. In particular, x and y are $(a;b)$-finite limits.)

PROOF. Say x is the limit of the $(a;b)$-sequence (x_n). As $\mathbf{h}(x_n)$ has limit $\mathbf{h}(x)$ and $\mathbf{h}(y)$ is finite, there is some n such that $y(a) \leq x_n(b)$. Define a suitable (y_n) by:

$$\begin{aligned} y_i &= x_i, \qquad 1 \leq i \leq n, \\ y_j &= y, \qquad j > n. \end{aligned} \qquad \square$$

LEMMA 6.5. (6.1.1 *is equivalent to* 5.3.1.) *If (x_n) is an increasing sequence of $(a;b)$-finite limits that has a limit x in P, then x is an $(a;b)$-finite limit.*

PROOF. Say x_m is the limit of the $(a;b)$-finite sequence $(x_{m,n})$. We define as follows an $(a;b)$-finite sequence (z_n) such that $x_{n,n} \sqsubseteq z_{k(n)} \sqsubseteq x$, where $k(n) = n(n+1)/2$.

$$z_1 = x_{1,1} (= \perp);$$

in order to achieve $z_{k(n+1)} = z_{k(n)} \vee x_{n+1,n+1}$, we define for $k(n) < i \leq k(n+1)$,

$$z_i = z_{k(n)} \vee x_{n+1,i-k(n)}. \qquad \square$$

LEMMA 6.6. *If $x \sqsubseteq y$ and $y(a) \leq x(b)$ and x is an $(a;b)$-finite limit, then so is y.*

PROOF. Say (x_n) is an $(a;b)$-finite sequence with limit x. Say (y_n) is an increasing sequence of finite elements with limit y.

For each fixed m, $\mathbf{h}(y_m)$ is finite, so there is $k(m)$ such that

$$y_m(a) \leq x_{k(m)}(b).$$

Define z_m to be $y_m \vee x_{k(m)}$.

Note that

$$z_m(a) = y_m(a) \vee x_{k(m)}(a) \leq x_{k(m)}(b) \leq x(b),$$

so from Lemma 6.4, z_m is an $(a;b)$-finite limit.

Evidently, (z_n) is an increasing sequence with limit y, so Lemma 6.5 applies to show that y is $(a;b)$-finite. $\square$

LEMMA 6.7. *Every $(a;b)$-limit is an $(a;b)$-finite limit.*

PROOF. Say (t_n) is an $(a;b)$-sequence with limit t. The argument is by induction on n. Since $t_1 = \perp$, trivially it is an $(a;b)$-finite limit. Then, Lemma 6.6 applies repeatedly to show that each t_i, in turn, is an $(a;b)$-finite limit. Finally, Lemma 6.5 applies to show that the limit t is also. □

6.2. Operators for Making Linking Steps. It will be convenient for the modular development of the theory to introduce an operator that, intuitively speaking, performs in parallel all possible instances of "**Step** $h + 1$" of 6.1.1. We shall also use it to define further operators that can perform an arbitrary finite number of such steps.

We do not contrive a sense in which all of these operators are totally defined, or continuous; but they lead to the definition of continuous operators, as we shall see.

6.2.1. *Definition of* $\mathbf{Step}_{a;b}$. Intuitively, a single step in our linking construction adds to a subprocess P (intuitively, the elements already found) of a process Q, some additional elements from Q.

The function has two variables and is defined as follows:

$$\mathbf{Step}_{a;b}(P, Q) = \{x \text{ in } Q : x \text{ is finite and for some } d \text{ in } P,\ d \sqsubseteq x \ \text{ and } \ d(b) \geq x(a)\}.$$

It is straightforward to show that $\mathbf{Step}_{a;b}(P, Q)$ is a process. It is also straightforward to check that

Lemma 6.8. $\mathbf{Step}_{a;b}$ *commutes with disjoint union.*

More precisely, for each $(a;b)$, none of whose ports are in the port set of Q,

$$\mathbf{Step}_{a;b}(P, Q) \mathbin{\dot\cup} R = \mathbf{Step}_{a;b}(P, Q \mathbin{\dot\cup} R).$$

6.3. Taking a Finite Sequence of Steps: The **Next** and $\mathbf{Step}^{(k)}_{a;b}$ Operators. Given a linking $(a;b)$ on a process P, $\mathbf{Step}_{a;b}(\bot, P)$ is the first step in the construction of the linked process and, for example,

$$\mathbf{Step}_{a;b}(\mathbf{Step}_{a;b}(\bot, P), P)$$

is the second.

To define the general case, it is convenient to introduce an operator

$$\mathbf{Next}_{a;b}\colon \mathbf{Proc} \times \mathbf{Proc} \to \mathbf{Proc} \times \mathbf{Proc}$$

defined by

$$\mathbf{Next}_{a;b}(P, Q) = (\mathbf{Step}_{a;b}(P, Q), Q).$$

We can then define the k-step function $\mathbf{Step}^{(k)}_{a;b}$ by

$$\begin{aligned} \mathbf{Step}^{(0)}_{a;b}(P) &= \bot \\ \mathbf{Step}^{(k+1)}_{a;b}(P) &= \mathbf{Step}_{a;b}(\mathbf{Next}^{k}_{a;b}(\bot, P)). \end{aligned}$$

We note that

Lemma 6.9. *The elements of* $\mathbf{Step}^{(n)}_{a;b}(P) = pr_1(\mathbf{Next}^{n}_{a;b}(\bot, P))$ *are the last terms of* $(a;b)$*-finite sequences of length* n.

The proof is a straightforward induction of n, from the definition of **Next**. □

Lemma 6.10. $\mathbf{Step}^{(k)}_{a;b}(P)$ *is a monotonic function of* k *and* P.

Proof. Say $h \leq k$ and $P \leq Q$. We first observe that

$$\mathbf{Step}^{(h)}_{a;b}(P) \subseteq \mathbf{Step}^{(k)}_{a;b}(Q).$$

That is evident from Lemma 6.9 and the fact that $P \leq Q$.

To show that the left side is an ideal of the right, suppose that x is in the left side, y is in the right, and $y \sqsubseteq x$. Writing $(x_i : i = 1, \ldots, h)$ for an $(a;b)$-finite sequence in P with limit x, we note that $x_i \wedge y$ is in P, since $P \leq Q$, and $(x_i \wedge y: i = 1, \ldots, h)$ is an $(a;b)$-finite sequence in P with limit y. □

LEMMA 6.11. *$\mathbf{Step}^{(k)}_{a;b}$ is continuous.*

PROOF. Say (P_n) is increasing with limit P. In view of Lemma 6.10, we have to show that

$$\mathbf{Step}^{(k)}_{a;b}(P) \subseteq \mathrm{lub}_n \mathbf{Step}^{(k)}_{a;b}(P_n).$$

Each element of $\mathbf{Step}^{(k)}_{a;b}(P)$ is the limit of an $(a;b)$-finite sequence $(x_i : i = 1, \ldots, k)$ in P. But by definition of least upper bound for processes, each x_i is in some $P_{m(i)}$, $i = 1, \ldots, k$, so x is in $\mathbf{Step}^{(k)}_{a;b}(P_m)$, where $m = \max_{i=1,\ldots,k} m(i)$; as required. □

It is not difficult to check from Lemma 6.8 that:

LEMMA 6.12. *$\mathbf{Step}^{(k)}_{a;b}$ commutes with disjoint union.*

That is, if $(a;b)$ is disjoint from the port set of Q, then

$$(\mathbf{Step}^{(n)}_{a;b}(P)) \dot{\cup} Q = \mathbf{Step}^{(n)}_{a;b}(\mathbf{P} \dot{\cup} Q).$$

Here are some further lemmas about $\mathbf{Next}_{a;b}$ and $\mathbf{Step}^{(k)}_{a;b}$, which will be used later. We write $Q \subseteq P$ to denote that Q is a process obtained by restricting P to a subset of its elements, without assuming that Q is an ideal of P.

LEMMA 6.13. *If $a \subseteq c$ are input port sets and $b \subseteq d$ are output port sets, then*

$$\mathbf{Next}_{c;d}(P, Q) \subseteq \mathbf{Next}_{a;b}(P, Q).$$

COROLLARY 6.14. *Under the same conditions,* $\mathbf{Step}^{(k)}_{c;d}(P) \subseteq \mathbf{Step}^{(k)}_{a;b}(P)$.

LEMMA 6.15. *If $a \subseteq c$ and $b \subseteq d$, then* $\mathbf{Step}^{(k)}_{a;b}(\mathbf{Step}^{(k)}_{c;d}(P)) = \mathbf{Step}^{(k)}_{c;d}(P)$.

This is proved by considering the $(c;d)$-iterative sequences of length k. Every $(c;d)$-sequence is an $(a;b)$-sequence.

Hence, we see that

LEMMA 6.16. *If $(a;b)$ and $(c;d)$ are disjoint, then* $\mathbf{Step}^{(k)}_{ac;bd}(P) \subseteq \mathbf{Step}^{(k)}_{a;b}(\mathbf{Step}^{(k)}_{c;d}(P))$.

PROOF. From Lemma 6.13,

$$\mathbf{Step}^{(k)}_{ac;bd}(P) \subseteq \mathbf{Step}^{(k)}_{c;d}(P),$$

so from Lemma 6.10,

$$\mathbf{Step}^{(k)}_{a;b}(\mathbf{Step}^{(k)}_{ac;bd}(P)) \subseteq \mathbf{Step}^{(k)}_{a;b}(\mathbf{Step}^{(k)}_{c;d}(P)).$$

But from Lemma 6.15, the left side is $\mathbf{Step}^{(k)}_{ac;bd}(P)$. □

To obtain an inclusion of the converse type, we use the following lemma.

LEMMA 6.17. *If $(a;b)$ and $(c;d)$ are disjoint, if $(x_i : i = 1, \ldots, m)$ is an $(ac;bd)$-sequence and $(y_i : 1 = 1, \ldots, n)$ is an $(a;b)$-sequence, if $x_m \sqsubseteq y_n$ and if $x_m(d) \geq y_n(c)$, then the concatenation $(z_i : i = 1, \ldots, m + n - 1)$ of (x_i) and (y_i) is an $(ac;bd)$-sequence whose limit is y.*

PROOF. By definition,

$$z_{m+i} = x_m \vee y_{i+1}, \qquad i = 0, \ldots, n-1.$$

Now $y_1 = \bot$, so $z_m(ac) \leq z_{m-1}(bd)$ by hypothesis.
For $1 \leq i \leq m-1$,

$$\begin{aligned} z_{m+i-1}(b) &= (x_m \vee y_i)(b)) = x_m(b) \vee y_i(b), \\ &\geq x_m(a) \vee y_{i+1}(a) \qquad \text{by hypothesis}, \\ &= z_{m+i}(a), \end{aligned}$$

and

$$\begin{aligned} z_{m+i-1}(d) &= x_m(d) \vee y_i(d) = x_m(d) = y_n(c), \\ &\geq x_m(c) \vee y_{i+1}(c) = z_{m+i}(c). \end{aligned}$$ □

LEMMA 6.18. *If $(a;b)$ and $(c;d)$ are disjoint then*

$$\mathbf{Step}^{(n)}_{c;d}(\mathbf{Step}^{(m)}_{a;b}(P)) \subseteq \mathbf{Step}^{(m,n)}_{ac;bd}(P).$$

PROOF. Say $x = x_n$ where $(x_i : i = 1, \ldots, n)$ is a $(c;d)$-finite sequence of $(a;b)$-finite limits of $(a;b)$-sequences of length m. □

One proves that x is the limit of an $(ac;bd)$-finite sequence of length at most m,n by induction on n, using the previous lemma. The base case is trivial since $x_1 = \bot$.

6.4. COMPLETE ITERATIONS: THE **Iter** OPERATOR

6.4.1. *Definition of* **Iter**. We define a continuous function **Iter**: **Proc** → **Proc** by

$$\mathbf{Iter}_{a;b}(P) = \mathrm{lub}_n \mathbf{Step}^{(n)}_{a;b}(P).$$

It follows from Lemma 6.12 that

LEMMA 6.19. **Iter** *commutes with disjoint union.*

That is, if p is a port set that is disjoint from the ports of Q, then

$$(\mathbf{Iter}_{a;b}(P)) \dot\cup Q = \mathbf{Iter}_{a;b}(P \dot\cup Q).$$

LEMMA 6.20. *Disjoint iterations associate (and commute).*

That is, if $(a;b)$ and $(c;d)$ are disjoint, then

$$\mathbf{Iter}_{a;b}(\mathbf{Iter}_{c;d}(P)) = \mathbf{Iter}_{ac;bd}(P)(=\mathbf{Iter}_{c;d}(\mathbf{Iter}_{a;b}(P))).$$

PROOF. We have to show that

$$\mathrm{lub}_m \mathbf{Step}^{(m}_{a;b}(\mathrm{lub}_n \mathbf{Step}^{(n)}_{c;d}(P)) = \mathrm{lub}_n \mathbf{Step}^{(n)}_{ac;bd}(P).$$

Since both are subprocesses of P, it is enough to show that each side is a subset of the other. □

That the right side is included in the left follows from Lemmas 6.10, 6.11, and 6.16. The converse follows from Lemmas 6.10, 6.11, and 6.18.

6.5. A CLOSED ITERATION OPERATOR. We now define, for all processes over the given set of ports and for each linking $(a;b)$, a closed iteration operator $\mathbf{Closit}_{a;b}$.
$\mathbf{Closit}_{a;b}(P)$ is the set of all limits in P of increasing sequences in $\mathbf{Iter}_{a;b}(P)$.

It is straightforward to check that this subset of P defines a subprocess, which we hereafter denote by $\mathbf{Closit}_{a;b}(P)$ also.

We first show

LEMMA 6.21. **Closit** *is monotonic.*

PROOF. Say $P \leq Q$. From Lemma 6.10, $\mathbf{Iter}_{a;b}(P) \leq \mathbf{Iter}_{a;b}(Q)$. It is therefore evident that $\mathbf{Closit}_{a;b}(P)$ is a subset of $\mathbf{Closit}_{a;b}(Q)$.

Suppose then that x, y are limits in P, Q, respectively of increasing sequences (x_n), (y_n) in $\mathbf{Iter}_{a;b}(P)$, $\mathbf{Iter}_{a;b}(Q)$, respectively, and that $y \sqsubseteq x$ in Q.

First note that since $P \leq Q$, then y is in P.

As all elements of $\mathbf{Iter}_{a;b}(P)$ and $\mathbf{Iter}_{a;b}(Q)$ are finite, and as $y \sqsubseteq x$, so that $\mathbf{h}(y) \leq \mathbf{h}(x)$, then for all n there is $m(n)$ such that $\mathbf{h}(y_n) \leq \mathbf{h}(x_{m(n)})$. Since both are below x, then $y_n \sqsubseteq x_{m(n)}$, from Axiom 4.2. Thus, y_n is in $\mathbf{Iter}_{a;b}(P)$, so that y is in $\mathbf{Closit}_{a;b}(P)$, as required. □

LEMMA 6.22. **Closit** *is continuous.*

PROOF. In view of Lemma 6.21, it remains to show, for an arbitrary increasing sequence (P_n) of processes with limit P, that $\mathbf{Closit}_{a;b}(P) \leq \text{lub}\ \mathbf{Closit}_{a;b}(P_n)$.

Say x is the limit in P of (x_n) in $\mathbf{Iter}_{a;b}(P)$. Each x_n is in some $P_{m(n)}$, by definition of $\mathbf{Iter}(P)$, as required for the result. □

It follows from Lemma 6.19 that

COROLLARY 6.23. **Closit** *commutes with disjoint union.*

That is, if $(a;b)$ is disjoint from the port set of Q then

$$\mathbf{Closit}_{a;b}(P \mathbin{\dot{\cup}} Q) = (\mathbf{Closit}_{a;b}(P)) \mathbin{\dot{\cup}} Q.$$

It follows from Axiom 6.20 that

COROLLARY 6.24. **Closit** *is associative.*

That is, if $(a;b)$ and $(c;d)$ are disjoint port sets, then

$$\mathbf{Closit}_{a;b}(\mathbf{Closit}_{c;d}(P)) = \mathbf{Closit}_{ac;bd}(P).$$

6.6. A PORT-HIDING OPERATOR. We define an operation $\mathbf{Hide}_{a;b}$, which forms a process of equivalence classes as in 5.3.1. It is convenient, however, to define the operator over all processes.

Thus, for an arbitrary process Q, which includes the ports of $(a;b)$, call x and y in $Q(a;b)$-equivalent, if there is some z in Q such that $z \sqsubseteq x$, $z \sqsubseteq y$ and

$$\mathbf{h}_{a;b}(z) = \mathbf{h}_{a;b}(x) = \mathbf{h}_{a;b}(y).$$

The elements of $\mathbf{Hide}_{a;b}(Q)$ are the $(a;b)$-equivalence classes of Q.

The ordering of $\mathbf{Hide}_{a;b}(Q)$ is defined by

$$x \sqsubseteq y \text{ means that for some } x' \text{ in } x \text{ and } y' \text{ in } y,\ x' \sqsubseteq y'.$$

The history function of $\mathbf{Hide}_{a;b}(Q)$ is induced in the evident way by $\mathbf{h}_{a;b}$ on Q.

First, we check that $\mathbf{Hide}_{a;b}(Q)$ is a process. We use the following lemmas.

LEMMA 6.25. *Every $(a;b)$-equivalence class has a least element.*

PROOF. By definition of $(a;b)$-equivalence, $(a;b)$-equivalence classes are downwards directed. That is, if x and y are $(a;b)$-equivalent, then there is z such that $z \sqsubseteq x$ and $z \sqsubseteq y$ and z is $(a;b)$-equivalent to both x and y. Hence, it is enough to show that $(a;b)$-equivalence classes have minimal elements.

However, that follows from the fact that the prefix ordering of histories is noetherian (descending chains are finite), in view of Axiom 4.2. □

LEMMA 6.26. *If $y \sqsubseteq x$ are $(a;b)$-equivalence classes and y', x' are their least elements, then $y' \sqsubseteq x'$.*

PROOF. Observe that $y \wedge x' \sqsubseteq y$ and $out(y \wedge x') = out(y)$ so that $y' \sqsubseteq y \wedge x' \sqsubseteq x'$, as required. □

LEMMA 6.27. *If x and y are consistent and $\mathbf{h}_{a;b}(x) \le \mathbf{h}_{a;b}(y)$ and x', y' are the least elements of their equivalence classes, then $x' \sqsubseteq y'$.*

PROOF. Say $x \sqsubseteq w$ and $y \sqsubseteq w$. Since

$$\mathbf{h}_{a;b}(x \wedge y) = \mathbf{h}_{a;b}(x) \wedge \mathbf{h}_{a;b}(y) = \mathbf{h}_{a;b}(x),$$

then $x \wedge y$ is $(a;b)$-equivalent to both x and y. Hence, $x' \sqsubseteq x \wedge y$, by definition of x'. Then, by Lemma 6.27, $x' \sqsubseteq y'$, as required. □

COROLLARY 6.28. $\mathbf{Hide}_{a;b}(P)$ *satisfies Axiom* 4.2.

The remainder of the proof that $\mathbf{Hide}_{a;b}(P)$ is a process is straightforward and is omitted.

LEMMA 6.29. *If $P \sqsubseteq Q$, then x, y in P are $(a;b)$-equivalent in P just if they are $(a;b)$-equivalent in Q.*

PROOF. If there is z in Q as in the definition of $(a;b)$ equivalent, then z is in P, as required, because $z \sqsubseteq x$. □

LEMMA 6.30. $\mathbf{Hide}_{a;b}$ *is monotonic.*

That is, if $P \le Q$, then (up to isomorphism) $\mathbf{Hide}_{a;b}(P) \le \mathbf{Hide}_{a;b}(Q)$.

PROOF. From Lemma 6.29, the embedding of P in Q induces an embedding of $\mathbf{Hide}_{a;b}(P)$ in $\mathbf{Hide}_{a;b}(Q)$, in which the equivalence class of x in P is identified with the equivalence class of x in Q (though they need not be equal as sets).

Moreover, if x is in $\mathbf{Hide}_{a;b}(P)$ and y is in $\mathbf{Hide}_{a;b}(Q)$ and $y \sqsubseteq x$ in $\mathbf{Hide}_{a;b}(Q)$, then by definition of $\sqsubseteq$ in $\mathbf{Hide}_{a;b}(Q)$, there is some element y^* of y, which is below some element x^* of x, where we can assume that x^* is in P.

Now $x^* \wedge y^* \sqsubseteq x^*$ and so $x^* \wedge y^*$ is in P. It is an element of y, so y is in $\mathbf{Hide}_{a;b}(P)$, as required for monotonicity. □

LEMMA 6.31. $\mathbf{Hide}_{a;b}$ *is continuous.*

PROOF. In view of the previous result, it remains to show that, if (P_n) is an increasing sequence of processes with limit P, then (up to isomorphism)

$$\mathbf{Hide}_{a;b}(P) = \text{lub}\,\mathbf{Hide}_{a;b}(P_n).$$

The inclusion of the right side in the left follows from monotonicity, so we show the converse inclusion.

Consider x in $\mathbf{Hide}_{a;b}(P)$. It is the equivalence class of some x' in $\bigcup_n P_n$. But that means x' is in some P_n, as required. □

The next two lemmas are straightforward.

LEMMA 6.32. **Hide** *is associative.*

That is, if p and q are disjoint port sets, then

$$\mathbf{Hide}_p(\mathbf{Hide}_q(P)) = \mathbf{Hide}_{p \cup q}(P).$$

LEMMA 6.33. **Hide** *commutes with disjoint union.*

That is, if the port set p is disjoint from the ports of Q, then

$$\mathrm{Hide}_p(P \dot{\cup} Q) = \mathbf{Hide}_p(P) \dot{\cup} Q.$$

We next show

LEMMA 6.34. **Hide** *commutes with* **Step**.

That is, if $(a;b)$ and $(c;d)$ are disjoint and P is a subprocess of Q, then

$$\mathbf{Step}_{a;b}(\mathbf{Hide}_{c;d}(P), \mathbf{Hide}_{c;d}(Q)) = \mathbf{Hide}_{c;d}(\mathbf{Step}_{a;b}(P, Q)).$$

PROOF. Suppose first that x is an element of the left side. Then, x is in $\mathbf{Hide}_{c;d}(Q)$ and there is e in $\mathbf{Hide}_{c;d}(P)$ such that $e \sqsubseteq x$ and $e(b) \geq x(a)$. Thus e, x are $(c;d)$-equivalence classes of elements, e' and x' say, of P and Q, respectively, such that $e' \sqsubseteq x'$ and $e'(b) \geq x'(a)$. Hence, x' is in $\mathbf{Step}_{a;b}(P, Q)$ and so x is in $\mathbf{Hide}_{c;d}(\mathbf{Step}_{a;b}(P, Q))$.

If conversely x is an element of the right side, then x is the $(c;d)$-equivalence class of x' say in Q such that for some e' in P, $e' \sqsubseteq x$ and $e'(b) \geq x'(a)$. It follows that, writing e for the $(c;d)$-equivalence class of e', $e \sqsubseteq x$ and $e(b) \geq x(a)$; so that x is in the left side, as required. □

As a corollary, we have

COROLLARY 6.35. **Hide** *commutes with* **Closit.**

That is, if $(a;b)$ and $(c;d)$ are disjoint, then

$$\mathbf{Hide}_{a;b}(\mathbf{Closit}_{c;d}(P)) = \mathbf{Closit}_{c;d}(\mathbf{Hide}_{a;b}(P)).$$

6.7. THE LINKING OPERATOR. We can now define

$$\mathbf{Link}_{a;b}(P) = \mathbf{Hide}_{a;b}(\mathbf{Closit}_{a;b}(P)).$$

It is immediate from Corollary 6.23 and Lemma 6.33 that

LEMMA 6.36. *Linking commutes with disjoint union.*

That is, if $(a;b)$ is disjoint from the ports of Q, then

$$\mathbf{Link}_{a;b}(P) \dot{\cup} Q = \mathbf{Link}_{a;b}(P \dot{\cup} Q).$$

Likewise, it is immediate from Corollary 6.24, Lemma 6.32, and Corollary 6.35 that

LEMMA 6.37. *Linking is associative.*

That is, if $(a;b)$ and $(c;d)$ are disjoint, then

$$\mathbf{Link}_{a;b}(\mathbf{Link}_{c;d}(P)) = \mathbf{Link}_{ac;bd}(P).$$

7. *Network Construction*

7.1. THE PROCESS DEFINED BY A NETWORK. Recall from 5.1 that a network defines a linking $(a;b)$ of the ports of $P_1 \dot{\cup} \cdots \dot{\cup} P_n$. In our theory, the process defined by that network is

$$\mathbf{Netp}_{a;b}(P_1, \ldots, P_n) = \mathbf{Link}_{a;b}(P_1 \dot{\cup} \cdots \dot{\cup} P_n).$$

7.1.1. *Recursive Definitions of Networks.* The operator **Netp**, being a composition of continuous operators, is continuous. Thus, for a given network scheme, the equation

$$X = \mathbf{Netp}_{a;b}(X, P_2, \ldots, P_n),$$

for example, has a least solution for given $P_2, \ldots, P_n$.

7.2. THE ASSOCIATIVITY PROPERTY. The anomalies discussed in Section 3 can be viewed as failures of an associativity property for the theories in which they occur. For example in those theories, and in the notation of 3.3, the following two methods of computing the process of the substitution of *NB* in the context of Figure 7c do not give the same result.

(a) First form the process defined by *NB*, then use it to form the process defined by the larger context.
(b) Form in one step the process defined by the larger context and *NB*, from the components of *NB* and the components of that context.

To state this example with precision, it is necessary to set up a precise theory in which the anomaly occurs. We omit that theory since it is not central to this work, but we invite the reader to try the exercise, using as much precision as desired.

This type of behavior does not occur in our theory. More precisely we have the following result.

THEOREM 7.1. (ASSOCIATIVITY THEOREM). *If a network defines a linking* $(ac;bd)$ *on the disjoint union* $P_1 \dot{\cup} \cdots \dot{\cup} P_n$, *such that* $(a;b)$ *and* $(c;d)$ *form a disjoint decomposition of* $(ac;bd)$, *and no ports of* $P_{k+1}, \ldots, P_n$ *occur in* $(a;b)$, *then (up to isomorphism)*

$$\mathbf{Netp}_{ac;bd}(P_1, \ldots, P_n) = \mathbf{Netp}_{c;d}(\mathbf{Netp}_{a;b}(P_1, \ldots, P_k), P_{k+1}, \ldots, P_n).$$

In view of the definition of **Netp**, we have to show

$$\mathbf{Link}_{ac;bd}(P \dot{\cup} Q) = \mathbf{Link}_{c;d}(\mathbf{Link}_{a;b}(P) \dot{\cup} Q).$$

But that is immediate from Lemmas 6.36 and 6.37.

7.3. RELATIONSHIP TO KAHN'S THEORY. General discussion of the relationship of our approach to Kahn's has been given already. Hence, we consider here only the key issue of expressing Kahn's method in our approach.

Rather than discussing functional processes by means of the construction $P(f)$, it is more natural in our approach, and more general, to make the following definition.

7.3.1. *Definition of Explicitly Functional Processes.* We call a process *explicitly functional* (meaning, recognizable as functional from the way it is presented) if for all x and y in P such that $in(x)$ and $in(y)$ are consistent histories, x and y are consistent. Note in that case, $out(x)$ and $out(y)$ are consistent.

Clearly, all retracting functional processes of the form $P(f)$ are explicitly functional. It is also clear that:

LEMMA 7.2. *Explicitly functional processes are functional.*

That is, for a given input history u there is a greatest output history v such that $(u;v)$ is a process history.

Furthermore, it is clear that the disjoint union of explicitly functional processes is explicitly functional.

The following lemma shows that explicit functionality is also preserved by linking, and hence by network construction.

LEMMA 7.3. *Linking preserves explicit functionality.*

For that we have to prove:

If (x_n) and (y_n) are $(a;b)$-sequences of an explicitly functional process P, with limits x and y such that $in(\mathbf{h}_{a;b}(x))$ and $in(\mathbf{h}_{a;b}(y))$ are consistent, then x and y are consistent.

PROOF. By induction on n, as follows, show that x_n and y_n are consistent. It then follows from the definition of explicit functionality that their limits are consistent.

The assertion is trivial for $x_1 = y_1 = \bot$. Suppose then that x_n and y_n are consistent. Then, $x_n(b)$ and $y_n(b)$ are consistent and, by hypothesis, the histories at input ports not in a of x_{n+1} and y_{n+1} are pairwise consistent; hence, the inputs of x_{n+1} and y_{n+1} are consistent. Since P is explicitly functional, it follows that x_{n+1} and y_{n+1} are consistent. $\square$

It follows that

THEOREM 7.4. *Our modeling of Kahn's deterministic processes commutes with network construction.*

That is, for retracting functions $f_1, \ldots, f_n$ and a linking $(a;b)$,

$$\mathbf{Netp}_{a;b}(P(f_1), \ldots, P(f_n)) = P(f),$$

where f is a retracting function that is the function computed by Kahn's method.

COROLLARY 7.5. *All of Kahn's deterministic functions are retracting.*

7.4. PROCESS HOMOMORPHISMS COMMUTE WITH NETWORK CONSTRUCTION. We mentioned in 3.2 that our method is not as economical as seems possible in its use of multisets, but that the processes we compute have homomorphic images that are as depicted in that section.

Naturally, one would hope that a lack of economy in describing a process would not affect the correctness of the description. That is confirmed by the following result, which is proved straightforwardly by following through the steps of the network construction definition.

If m_i: $P_i \rightarrow Q_i$, $i = 1, \ldots, n$ are process homomorphisms and $(a;b)$ is a linking defining the process $\mathbf{Netp}_{a;b}(P_1, \ldots, P_n)$, then a homomorphism

$$m\colon \mathbf{Netp}_{a;b}(P_1, \ldots, P_n) \rightarrow \mathbf{Netp}_{a;b}(Q_1, \ldots, Q_n)$$

is induced such that

$$m(\mathbf{Netp}_{a;b}(P_1, \ldots, P_n)) = \mathbf{Netp}_{a;b}(m_1(P_1), \ldots, m_n(P_n)).$$

8. *Total Behaviors of Networks*

8.1. INTRODUCTION. Our approach includes both partial and total input-output histories within a single partially ordered multiset. Hence, it is not immediately clear that, in the notation of 1.1.2, our approach can be more than partially correct.

For, the question of whether a given response of a nondeterministic machine is total cannot generally be resolved by inspecting the input-output histories. Consider, for example, a machine with no inputs and one output, which merely decides whether or not to output a single "0," then implements its decision. In the approach as outlined so far, we would model this machine's behavior by the same process as for the deterministic variant that always outputs the zero.

Hence, we refine the approach slightly, as follows.

8.2. A TOTALITY PREDICATE. We extend our concept of process P to include also a predicate T on (or, a distinguished subset T of) the set of elements of P.

Intuitively, $\mathrm{T}(x)$, for x in P, means that in some executions x is a total behavior for input $in(x)$.

We allow, but shall not require, $\mathrm{T}(x)$ for elements x, which are *necessarily* total in the sense that there is no y in P with $x \sqsubseteq y$, $x \neq y$, and $in(x) = in(y)$. It is simpler to allow necessarily total elements to "speak for themselves." The only duty of the T predicate is to speak for those elements that may be chosen to be total behaviors but that are not necessarily total.

8.3. AXIOMS FOR A TOTALITY PREDICATE? In view of the policy just stated, to allow necessarily total elements to speak for themselves, we introduce no axioms about the totality predicate.

That does not mean that introducing the predicate is vacuous. Having done so, we have to specify, for example, how such predicates are constructed for processes defined by networks. Our proofs (of associativity, for example) have to be refined so as to establish assertions about the totality predicates associated with the networks involved.

8.4. NOTES ON INCORPORATING THE TOTALITY PREDICATE. Here is a brief indication of how the preceding material is refined to incorporate a totality predicate. We may write, for example, T_P for the totality predicate of P.

We add to the definition of the partial ordering of all processes the requirement

$$\mathrm{T}_P(x) \text{ implies } \mathrm{T}_Q(x), \qquad \text{for all} \quad x \text{ in } P.$$

and

$$\mathrm{T}_Q(m(x)) \text{ implies there is } y \text{ in } P \text{ such that } m(x) = m(y) \text{ and } \mathrm{T}_P(y).$$

Note that this is weaker than requiring that $\mathrm{T}_P(x)$ be the restriction of T_Q to elements of P.

In the definition of homomorphism $m: P \rightarrow Q$, we add the clause

$$\mathrm{T}_P(x) \text{ implies } \mathrm{T}_Q(m(x)), \qquad \text{for all} \quad x \text{ in } P.$$

The totality predicate of a disjoint union is the conjunction of the totality predicates of its components.

To be intuitively correct, the totality definition for $\mathbf{Step}_{a;b}(P, Q)$ should take into account that output from ports of b must reach the linked ports of a and be processed.

For the sake of generality, we have not required that a process P include elements with all possible input histories. It would be natural to require the following additional axiom, which would slightly simplify our treatment of totality.

AXIOM 8.1. *Process input is asynchronous.*

For all $y \sqsubseteq x$ and h such that $in(y) \leq h \leq in(x)$, there is z such that $y \sqsubseteq z \sqsubseteq x$, $in(z) = h$ and $out(z) = out(y)$.

If such an axiom were assumed, then a natural definition of totality for $\mathbf{Step}_{a;b}(P, Q)$ would be as follows:

Definition. T(x) in $\mathbf{Step}_{a;b}(P, Q)$ means T(x) in Q, and there is d in P such that $x(a) = d(b)$.

Since, however, we are not assuming the above additional axiom, we make the following definitions.

Definition. T(x) in $\mathbf{Step}_{a;b}(P, Q)$ means $\mathrm{T}_Q(x)$, and there is d in P such that x is a maximal y in Q such that $d \sqsubseteq y$ and $y(a) \leq d(b)$.

Definition. T(x) in $\mathbf{Closit}_{a;b}(P, Q)$ means T(x) in Q and x is a maximal y in Q such that $y(a) \leq y(b)$.

Definition. T(x) in $\mathbf{Hide}_{a;b}(P)$ means $\mathrm{T}(x')_P$, for some x' in x.

Definition. For retracting history functions f, the totality predicate of $P(f)$ is always false.

With these definitions, our preceding work refines without substantial alteration so as to give the same results for processes with totality predicates.

9. *Relationship to Previous Work*

Since Kahn's successful development of a fixpoint theory for deterministic asynchronous processes [4], several attempts [1–3, 5–9] have been made to provide a similar theory for asynchronous processes in general. The approach in [5] is erroneous, however, as indicated in [9].

We shall comment only on previous work specific to the study of asynchronous processes; except to note that the work of [1] concerns nondeterminism in synchronous processes. It does not bring to that problem a generalization of Kahn's method, but rather observes that its trace-combining method is consistent with Kahn's when a model of Kahn's asynchronous deterministic processes is set up within its theory.

None of the work discussed meets the extensionality criterion, so we shall not mention it again.

Previous work on this problem has been in two main groups. Park [6] is a recent example of the work of one group. It fails the modularity and generality criteria of 1.1 and relies on the extraneous concept of oracle. Park's approach fails the generality criterion because its nondeterminism arises through the use of fair oracles, and so fails to capture unfairness, which we have argued has theoretical value. It fails the modularity criterion because oracle inputs of network components never get hidden.

The other group includes [3], [7], [8], and [9]. It may be divided into two subgroups; those that consider only finite input–output behaviors and those that consider also, or only, infinite input–output behaviors.

The former subgroup fails the generality criterion because finite histories cannot distinguish between fair and unfair merges. It also fails the input–output correctness criterion because it is unable to distinguish between arbitrarily long finite histories and infinite histories.

The latter subgroup avoids that problem and seems essentially equivalent, in one of two senses, to the work of this paper. We developed our approach because it was not clear previously that processes can be defined iteratively. That is valuable both for computation of approximations and theoretical reasoning by induction.

An exhaustive comparison of our approach with those of [3] and [7] has not been made, and would be better based on a formulation of our approach, which is more general with respect to types of networks considered and data types (both of which we have kept as simple as possible). Nevertheless, we offer the following general comments.

The equivalence of our approach and those of [3] and [7] is in different senses in each case. For, whereas both papers model processes as partially ordered multisets of events, different intuitions motivate the choices of partial order, so that apparently different concepts of network construction are used. The intuitions are as follows:

(a) Partial ordering of events is temporal ordering. This concept is due to Pratt in an unpublished draft of [8], and is also used in [7].
(b) Partial ordering of events is causality ordering. This approach is due to Brock and Ackerman [2], and is also used in [9].

To discuss these views from the perspective of this paper, we assume that each process has a fixed, finite set of ports at which all events occur, and that events at a single port are totally ordered.

Consider (a). Each trace has an associated history. The traces may be ordered by: $t_1 \sqsubseteq t_2$ means, $t_1 \subseteq t_2$ and the restriction of the t_2 ordering to elements of t_1 gives t_1. In this way, a set of traces defines a partially ordered multiset of histories. Such a partial order may not satisfy our axioms, but apparently could fail to do so only for inessential reasons. Assuming there is no difficulty arising from a lack of traces to specify behavior in various situations, then we would expect the partial order of traces so obtained to be the homomorphic image of a partial order that satisfies our axioms. The concept of network construction has the same effect as ours, although the definition is quite different.

The concept (b) may be reduced to that of (a) by regarding each trace of type (b) (*scenario*, in Brock and Ackerman's terminology) as a scheme of traces of type (a). That is, a *scenario* stands for all the traces that can be obtained from it by extensions of its partial order.

ACKNOWLEDGMENTS. Thanks to David Park, Ross Paterson, Vaughan Pratt, and Bill Wadge for helpful criticism at various stages. Thanks also to Zohar Manna for hospitality to the first author during January–March 1983 at the Computer Science Department, Stanford.

REFERENCES

1. BACK, R. J. R., AND MANNILA, H. A refinement of Kahn's semantics to handle nondeterminism and communication. In *Proceedings of the ACM Symposium on Distributed Computing* (Ottawa, Ontario, Canada, Aug, 18–20). ACM, New York, 1982, pp. 111–120.
2. BROCK, J. B., AND ACKERMAN, W. B. An anomaly in the specifications of nondeterministic packet systems. Computation Structures Group Note CSG-33, MIT-LCS, Massachusetts Institute of Technology, Cambridge, Mass., November 1977.
3. BROCK, J. D., AND ACKERMAN, W. B. Scenarios: A model of nondeterminate computation. In *Formalisation of Programming Concepts*, J. Diaz, and I. Ramos, Eds. Lecture Notes in Computer Science, vol. 107. Springer-Verlag, New York, 1981, pp. 252–259.
4. KAHN, G. The semantics of a simple language for parallel programming. In *Information Processing 74: Proceeding of the IFIP Congress 77*, J. L. Rosenfeld, Ed. North-Holland, New York, 1974, pp. 471–475.
5. KOSINSKI, P. R. A straightforward denotational semantics for non-determinant data flow programs. In *Conference Record of the Fifth Annual ACM Symposium on Principles of Programming Languages* (Tucson, Arizona, Jan. 23–25). ACM, New York, 1978, pp. 214–221.
6. PARK, D. The "fairness" problem and nondeterministic computing networks. Duplicated notes. Warwick University, 1982.

7. PATERSON, R., AND STAPLES, J. An algebra of processes with a finite basis. Tech. Report No. 29, Dept. Computer Science, Univ. of Queensland, Queensland, Australia, 1981.
8. PRATT, V.R. On the composition of processes. In *Conference Record of the 9th Annual ACM Symposium on Principles of Programming Languages* (Albuquerque, N.M., Jan 25–27). ACM, New York, 1982, pp. 213–223.
9. STAPLES, J., AND NGUYEN, V.L. Computing the behaviour of nondeterministic processes. *Theor. Comput. Sci. 26* (1983) 343–353.

RECEIVED JULY 1982; REVISED OCTOBER 1984; ACCEPTED NOVEMBER 1984