# Semantics of Networks Containing Indeterminate Operators

**Robert M. Keller**

**Prakash Panangaden**

Department of Computer Science
University of Utah
Salt Lake City, UT 84112

**Abstract**: We discuss a denotational semantics for networks containing indeterminate operators. Our approach is based on modelling a network by the set of all its possible behaviors. Our notion of behavior is a sequence of computational actions. The primitive computational action is an *event*: the appearance or consumption of a token on a data path. A sequence of such events is called a *history* and a set of such histories is called an *archive*. We give *composition rules* that allow us to derive an archive for a network from the archive of its constituents. Causal and operational constraints on network behavior are encoded into the definitions of archives. We give a construction that allows us to obtain the denotation of networks containing *loops* by a process of successive approximations. This construction is not carried out in the traditional domain-theoretic setting, but rather resembles the category theoretic notion of limit. By using this construction, we avoid having to impose any *closure conditions* on the set of behaviors, as are typically necessary in powerdomain constructions. The resulting theory is general and compositional, but is also close to operational ideas making it a useful and flexible tool for modelling systems.

## 1. Introduction

The denotational semantics of sequential programming languages is by now well understood. For concurrent programming languages or languages with an indeterminate construct, the situation is more complex. Several variant semantic definitions have been proposed. In this paper we discuss a semantics for networks containing indeterminate operators, one based on the notion of *computational events*. Although we intend to compose program modules in a style reminiscent of stream-based functional programs, the word "module" is used in place of "function" to emphasize that not all modules behave as functions on their input histories, but rather may exhibit time-dependent behavior (*indeterminacy*). The networks we consider are similar in spirit to those of Kahn and MacQueen (as well as many other *data-flow* languages, (*cf.* [Davis 82]), in that our networks consist of operators connected by unidirectional data *arcs*. However, we do not assume that all arcs necessarily behave as unbounded FIFO queues. If desired, such behavior can be modeled by the insertion of unbounded queue modules into otherwise delayless arcs, and we shall informally discuss such networks as if there could be arbitrary delay on arcs, knowing that we can perform the mentioned insertion when precision becomes necessary. However, generality is served, and the basic mathematical description simplified, by not universally assuming unboundedness of arcs. Likewise, the FIFO condition may be relaxed.

We base our semantics on the simple, but elegantly-treatable, notion of sets of sequences of computational *events*, where an event is the production or consumption of a data value by a module on one of its associated arcs. The sequences are termed *histories*, while sets thereof are termed *archives*. The *denotation* of a network is simply its archive. This approach will be seen to be adequate for describing networks of both determinate and indeterminate operators, with or without delay, and with or without cycles.

We present *composition rules* for obtaining archives of networks from the archives of their component parts, either atomic or sub-networks. Our rules are similar to a set of rules stated informally by Pratt [Pratt 82]. The chief difference is that this he uses "partially-ordered multisets" of events instead of sequences. Similar uses of partially-ordered sets of events appeared in the appendix of [Keller 78], and in [Brock 81].

The basic idea of archives does not require that behaviors be effectively given. However, toward this end we also introduce a fixpoint approach to the construction of archives. This approach is based on rules for extending a history, called *extension rules*, in which the causality and other operational constraints are encoded. These rules are presented as functions on sets of partial histories and can thus be easily composed. The theory of limits of iterations of such functions involves a departure from the conventional theory of complete partial orders, due to the nature of extensions of histories. We introduce a new construction in which limits are defined in a style reminiscent of the limit constructions for categories.

## 2. Archives

Programs or systems in our model are networks. A *network* is a directed graph, each node of which is labelled with the name of a *module* and each arc labelled distinctly (using b,c,d,....). The operational behavior of a network consists of nodes passing values to one another in the direction of arcs which connect the nodes. The *production* or *consumption* of a value y on arc b will be called an *event*, and represented either as <+b,y> in the former case or as <-b,y> in the latter case. Whether an event is a production or consumption will be called the *sense* of the event. When we wish to display an event as part of a sequence of events, we will sometimes use the following vertical notation for readability:

$$\begin{array}{cc} + & - \\ b & b \\ y & y \end{array}$$

We also refer to these events as *+b-events* or *-b-events* or just *b-event* when the sense is clear from context. We refer to the value part of a b-event as a *b-value*. When the particular arc, rather than the value, is of central importance, we may omit the latter as if all values were essentially the same. Here it would be the sequence of arcs on which events occur which is important. We hereafter refer to this as the *neutral-value* case. In many instances, the sense is clear from context, and we omit it too. For example, in the case of discussing a module with arcs b and c directed in and d directed out, we may omit the -'s with b and c, and + with with d.

A *history* of a node is a *sequence* of events which can occur in *one* possible run of the network. For example, one history for a module which produces, on arc d, sums of pairs of values consumed on arcs b and c, might be

```
+ - + - + + - + - + + - + - +
b b c c d c c b b d b b c c d
1 1 3 3 4 2 2 6 6 8 3 3 4 4 7
```

The behavior of this particular module can, of course, be expressed more elegantly as a function, but we are interested in a formalism which encompasses indeterminate operators as well. For example, a *merge* module passes its input values unchanged, rather than adding them, but passes them in some arbitrarily-interleaved order. The following are two possible histories for a merge with the same inputs b, c as above:

```
+ - + + - + + + - + + - + - + + - +
b b d c c d c b c d c c d b d c c d
1 1 1 3 3 3 2 6 2 2 7 7 7 6 6 4 4 4
```

and

```
+ + - + - + + - + + + - + - + + - +
b c b d c d c c d b c b d c d c c d
1 3 1 1 3 3 2 2 2 6 7 6 6 7 7 4 4 4
```

The description of a module can always be given by encapsulating it within a node (which provides labelled arcs with which to describe it), and presenting histories using the labels.

The set of all possible histories is called the *archive* for the module, and characterizes the module, i.e. forms a *denotation* for it. Certain features of potential interest, in particular the "internal state" of the node, may be suppressed in this description of the behavior of a node. For the purposes of the present discussion we assume that the history characterizes all *observable* properties of interest. This notion of observability is admittedly weaker than others that have been used in the CCS framework [Hennessey 82]. In the neutral-value case, a history may be represented as a finite or infinite *arc-sequence*, while an archive may be represented as a *language* (of finite or infinite strings) over the set of arc labels.

Figure 2-1 illustrates the "$\text{fork}_{bcd}$" module, which simply copies its input arc values to both of its output arcs. Its archive,

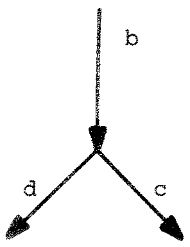

**Figure 2-1:** Fork module

$$\{<-b,v><+c,v><+d,v> \mid v \in V\}^{oo} \cup \{<-b,v><+d,v><+c,v> \mid v \in V\}^{oo}$$

indicates that each consumption of a value v (on arc b) is followed by a production of the same value v on arc c and on arc d. (Throughout this paper, if S is a set of sequences, then $S^{oo}$ designates the set of infinite *concatenations* of members of S.)

The inclusion of both b-c-d and b-d-c orders in the archive for the fork module is indicative of the arbitrariness of the ordering between the production of of c and d events. Here, and in what follows, we follow the customary practice of representing *concurrency* by fine-grain interleaving, shown in the form of *sets* of sequences. In doing so we are assuming that the networks we model are asynchronous.

To describe archives of other modules, some additional notation will be useful. For history x and arc b, $\Pi_b(x)$ is the sub-sequence of b-events in x, and $\Pi_{b=v}(x)$ is the sub-sequence of b-events with value v. Similarly, for a set B of arcs, $\Pi_B(x)$ is the sub-sequence of b-events where b is any element of B. Furthermore, $V_b(x)$ is the sequence of b-values in x, in the order they appear, while $\#_b(x)$ is length of that sequence, i.e. the number of b-events in x, and $\#_{b=v}(x)$ is the length of $\Pi_{b=v}(x)$. Finally, $K_b(x)$ is the sequence of events in x which are *not* b-events, and $K_{bc}(x)$ is the sequence of events in x which are neither b-events nor c-events.

We use $\leq$ to designate both the *prefix ordering* on sequences and the numeric ordering on integers. Some relations recur sufficiently often that we give them special names:

*equal*(b, c, x)

means that $V_b(x) = V_c(x)$, i.e. that the sub-sequence of b and c values in x are the same;

*precedes*(b, c, x)

means that each c-event in x is preceded by a corresponding b-event, i.e. for every prefix $y \leq x$, we have $\#_b(y) \geq \#_c(y)$; and *leads*(b, c, x)

means that each c-event in x is *immediately* preceded by a corresponding b-event. Thus, *leads*(b, c, x) implies *precedes*(b, c, x), but not conversely.

The *leads* and *precedes* relations are used to capture essential *causal* relationships within histories. For example, if each output event on arc c of a module is produced in response to an input event on arc b, we will have *precedes*(-b, +c, x). Note that if we begin with a sequence having the property *leads*(b, c, x), and insert events other than b- and c- events in arbitrary positions, we shall always have a sequence y such that *precedes*(b, c, y). This phenomenon occurs with some of our operations on sequences.

A module which introduces arbitrary *delay* can be modeled by the insertion of modules $\delta_{bc}$, which with input arc b, and output arc c, and value set V, have the archive

$\{x \in (\{-b, +c\} \times V)^{\infty} \mid$ *equal*(-b, +c, x) and *precedes*(-b, +c, x) $\}$

## 3. Semantics of Operators and Networks

We now discuss the semantics of various operators and networks. Our discussion centers around rules for composing the archives for operators to obtain the archives of the resulting networks. We consider three types of interconnections between sub-networks, which are sufficient to construct any network: aggregation, serial composition, and loop composition. Of these, the second is a convenience which can be composed from the other two. These rules were first presented in [Keller 83].

The first operation on networks is called **aggregation**. Here we simply lump modules M and N together without connecting them, as shown in Figure 3-1, designating the result by A(M,N).

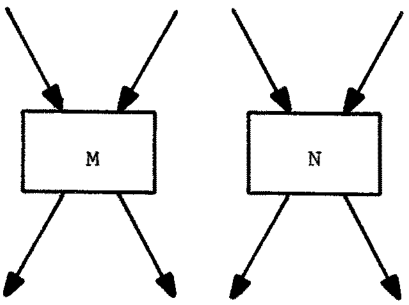

**Figure 3-1:** Aggregation of networks

The archive-composition rule corresponding to **aggregation** is the "shuffle". To describe the shuffle (designated $\Delta$) more precisely, assume initially that x and y are event sequences over *disjoint* sets of arc labels, B and C. Then

$$x \Delta y = \{z \mid \Pi_B(z) = x \text{ and } \Pi_C(z) = y\}$$

That is, the shuffle of two sequences on disjoint sets of arcs is a set of sequences, such that projecting on the respective sets of arcs gives back the original sequences. We extend this in the natural pointwise fashion to sets,

$$S \Delta T = \bigcup \{x \Delta y \mid x \in S, y \in T\}$$

In case that the label sets B,C are *not disjoint*, the definition of shuffle is only slightly more complicated. The simplest definition seems to be to force them to be disjoint by *renaming* one set of labels, shuffling as before, then performing the inverse renaming to get the result.

**Example** To shuffle two events sequences:

$$\begin{array}{cccc} + & + & + & + \\ b & d & b & d \\ 1 & 1 & 2 & 2 \end{array}$$

and

$$\begin{array}{cccc} + & + & + & + \\ c & d & c & d \\ 3 & 3 & 4 & 4 \end{array}$$

We have to rename one of the d's, say that for the second sequence, to d'. Shuffling then gives us, for example,

$$\begin{array}{cccccccc} + & + & + & + & + & + & + & + \\ b & d & c & d' & c & b & d' & d \\ 1 & 1 & 3 & 3 & 4 & 2 & 4 & 2 \end{array}$$

and inverse renaming changes each d' to d.

The next type of interconnection rule is **serial composition**, shown in Figure 3-2, which we designate by $S_{bc}(M,N)$, b being an output arc of M and c being an input arc of N. The archive semantics of serial composition will be more evident after we present the loop interconnection next.

We also need the ability to construct a **loop**, as shown in Figure 3-3, designating the resulting network as $L_{bc}(M)$. It is easy to see that the serial connection is a composition of an aggregation and a loop.

We assume in all the above that arcs of M and N have disjoint sets of labels. Otherwise we can rename them to get this property. It is also clear that connection S can be realized as an aggregation followed by a loop, so if we are treating A and L, we needn't treat S separately.

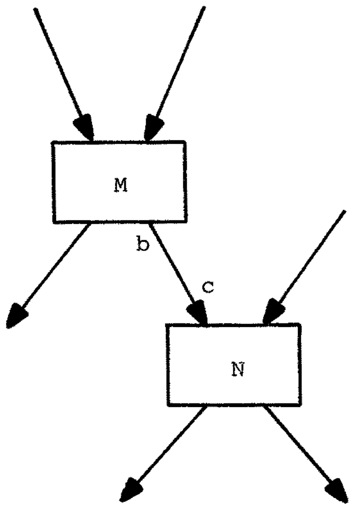

**Figure 3-2:** Serial composition

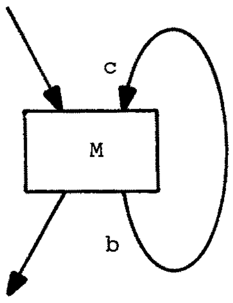

**Figure 3-3:** Network composition with a loop

Next we give the effect of such compositions on archives. If N is a network, with labels for its unconnected arcs, then Archive(N) denotes the corresponding archive.

**Aggregation Rule**: Archive(A(M,N)) = Archive(M) $\Delta$ Archive(N), where $\Delta$ is the shuffle operator.

The validity of the above rule is obvious: since the constituent networks do not interact, every interleaving of their two behaviors is a possible behavior of the result, and conversely.

To describe the effect of the **loop** construct L on an archive, we need to introduce a new cperator $M_{bc}$ where, as usual, b and c are arc signed names.

$M_{bc}(S) = \{ K_{bc}(x) \mid x \in S$ and *precedes*(b,c,x)$\}$.

Here $K_{bc}(x)$ discards events on arcs b and c, and preserves others. The purpose of the *precedes* condition is to express causality; if b values are fed to c, then in each archive, each −c−event is preceded by a +b−event with the same value. This rationale is summarized as.

**Loop Rule**: Archive($L_{bc}$(M)) = $M_{bc}$(Archive(M))

$M_{bc}$ is called the *matching* operator, because it selects only those sequences in which each c is preceded by a matching b.

According to preceding discussion, we also have

**Serial Composition Rule**. Archive($S_{bc}$(M, N)) = $M_{bc}$(Archive(M) $\Delta$ Archive(N))

The combination $M_{bc}(S\Delta T)$ occurs sufficiently often that is deserves a special notation,

$$S\Delta_{bc}T$$

This *matching shuffle* operator is similar to the operators used by Riddle [Riddle 79] to manipulate event sequences. In his work, special synchronizing symbols are introduced into the event sequences, and shuffling is performed on substrings occurring between matched synchronizing symbols. We also find it necessary to perform restricted shuffling, but we do not need to introduce synchronizing symbols into our history sequences.

Before further illustrating reasoning about archives using the above composition rules, it will be useful to express a few observations about causality, expressed using the *precedes* primitive:

| | |
|---|---|
| *Prefix* | (*precedes*(b,c,x) and $y \leq x$) imply *precedes*(b,c,y). |
| *Transitivity* | (*precedes*(b,c,x) and *precedes* (c,d,x)) imply *precedes*(b,d,x). |
| *Antisymmetry* | if x contains at least one b or c event, not(*precedes*(b,c,x) and *precedes*(c,b,x)). |

**Example** $L_{cb}$(fork$_{bcd}$) is the network shown in Figure 3−4, with a fork module with the output arc c, connected back to the input arc b. Here we have, assuming the neutral−value case, for any history x, *precedes*(b,c,x), from the definition of the fork archive. But for x to be in $L_{cb}$(Archive(fork$_{bcd}$)), we also require *precedes*(c,b,x). By antisymmetry, this means that only the null history $\Lambda$ can be in the resulting archive, i.e.

$$L_{cb}(\text{fork}_{bcd}) = \{\Lambda\}$$

Note that the fixed−point theory of Kahn would arrive at this result by starting with the null sequence and applying the fork function, which would give the null sequence back, thereby determining the least fixed−point.

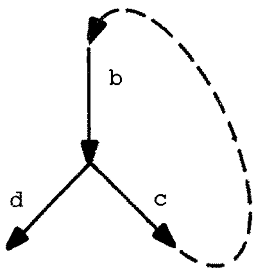

**Figure 3-4:** Example of a loop network

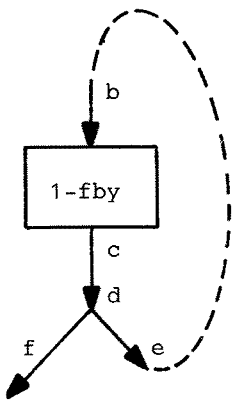

**Figure 3-5:** Another example of loop composition

**Example** Here we use a module *1-fby*, the output of which is a 1, followed by a verbatim copy of its input. The archive, with input b and output c, is given by the language (again assuming the neutral-value case)

$$\{c(bc)^{\infty}\}$$

When we compose 1-fby$_{bc}$ with fork$_{def}$ as shown in Figure 3-5, we have

$\Delta_{cd}$(Archive(1-fby$_{bc}$), Archive(fork$_{def}$)) =

$\Delta_{cd}(c\{bc\}^{\infty}, \{def, dfe\}^{\infty}) = \{ef, fe\}\ \{bef, bfe\}^{\infty}$

When we "close the loop", as shown, we are requiring, in each history x, *leads*(e,b,x), which does not additionally constrain the sequences over the previous composition, so the result, after applying $L_{eb}$, is

$\{f\}^{\infty}$

**Example** The archive for a *merge* module, with inputs b,c and output d, is simply the shuffle of archives for *identities* with arcs b,d and c,d respectively. This rectifies a prior oversimplification, which lead to the so-called "merge anomaly" [Keller 78]. We consider the network shown in Figure 3-6 and show that there are no causal anomalies in any of the histories of the archive. The network has two operators, a merge operator and an operator called g whose action is as follows: The operator g functions so as to map successive input values to output, in such a way that if the value is 1, it outputs 3, otherwise outputs the value itself.

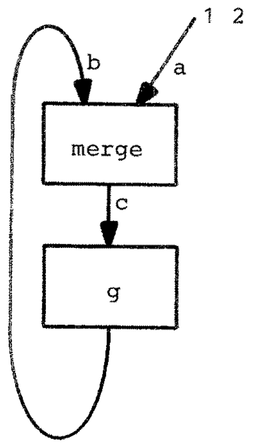

**Figure 3-6:** The Merge Anomaly Network

One may attempt to define the semantics of the merge operator in terms reminiscent of the functional approach via the following axiom:

> If one input arc contains the stream x and the other contains the stream y then the set of possible output streams is $x \Delta y$.

If one uses this axiom within the successive approximation approach of least fixed point theory, then one obtains output streams which can never be obtained operationally. For example, starting with the input stream 1 2 on arc a, after the second approximation using the above axiom, one might be lead to conclude that the set of possible streams on arc c is

$$\{1\ 2\ 3,\quad 1\ 2\ 3,\quad 3\ 1\ 2\}$$

but the last stream is impossible to obtain operationally. Notice that this anomaly arise because during the iteration *causality* is not imposed. It is the requirement that events in the histories on each arc *match* with each other which has been left out.

We shall now show that the archive composition rules permit the correct derivation of the behavior of the network in question. It is easy to verify operationally that if $\Pi_a(h) = 1\ 2$ then for some n,

$\Pi_b(h) = 3^n\ (3\ 2)^{\infty}$ and $\Pi_c(h) = 1\ 3^n 2\ (3\ 2)^{\infty}$

where h is any infinite history from the network archive. We shall now prove that this is indeed true of the network archive, using only properties of the operators in the network and the archival composition rules.

First of all we note that values on the arc c in the network arises from matching events on the output arc of the operator g with the left input arc of the merge operator. The matching requirement is that, in every history, the tokens on the output arc of g precede the corresponding tokens on the input arc of the merge operator. This forces the first c-event to precede the first b-event. This in turn means that the value of the first c-event must appear in response to an a-event. Since we are assuming that the first a-value is 1, it follows that the first c-value is 1. Secondly we observe that since the operator g is determinate, we have

$V_b(h) = G(V_c(h))$

where G is the following function on streams:

```
G(x) = if first(x) = 1
       then 3 ^ G(rest(x))
       else first(x) ^ G(rest(x)).
```

Here "^" is the "followed-by" operator, which builds a sequence by adding an element to the front, and "first" and "rest" comprise the inverse of followed-by. Thus there are no b-values equal to 1. From the archive for merge we conclude that

$V_c(h) = 1\ x\ 2\ y$
where $V_b(h) = x\ y$.

Thus we can inset this expression for $V_c(h)$ into the above equation and obtain

$V_b(h) = 3\ x\ 2\ y$.

We now have the following equation

$$x\ y = 3\ x\ 2\ y.$$

This forces first(x) = 3, and

$$\text{rest}(x\ y) = x\ 2\ y.$$

Peeling off the first value on each side we get first(rest(x)) = 3. Continuing with this peeling process, we conclude that x is of the form $3^n$ for some n. When x is completely peeled off the left hand side, we are left with the equation

$$y = 3\ 2\ y$$

which clearly has the solution $y = (3\ 2)^{\infty}$. Thus we have derived the general form of the histories in the network archive and have shown that they agree with the form expected operationally.

## 4. Extension Rules

In this section, we present a complementary approach to for deriving archives, which corresponds more closely to the traditional [Scott 70] approach to the denotational semantics of sequential programming languages. The key feature of this approach is that the denotations of programming constructs are viewed as functions and the denotations of complex constructs are obtained by ordinary functional composition. We shall view the denotations of operators as functions which act on archives. The actions of these functions on archives is the pointwise extension of their actions on individual histories. The action on a single history inserts new events which correspond to the participation of that operator in the history. Thus these functions are called *extension rules*.

The archive of a network can be constructed by composing the extension rules of the component operators. Networks which contain loops have extension rules obtained by taking fixed points of the (recursive) composition of the extension rules of the component operators. The sense in which a fixed point is taken will be clarified later in this section. We do not use traditional complete partial order theory but give a construction that is similar to the category theoretic notion of a limit. The use of extension rules will mimic the set-theoretic denotational semantics of the preceding sections quite closely

We emphasize at this point that the functions that we will describe in this section are to viewed in the same spirit as semantic functions for sequential languages [Scott 71] and are not to be construed as "implementations" in any sense. Several of these functions have in fact been programmed and executed, but they are not efficient. The role of these functions is to show how one may express the formalism of archives in *constructive* terms.
We now define an *extension rule* as follows:

**Definition** Let $F$ be an operator and let its archive be $\mathcal{A}_F$. Let $I_F$ be the *input archive*, i.e. the set of all possible input histories to $F$. $I_F$ consists of all sequences which are serializations of possible events on the input arcs to $F$. Then the *extension rule* for $F$, written $E_F$, is a function from archives to archives which satisfies:

$$E_F(I_F) = \mathcal{A}_F.$$

Thus, if $I_F$ is the input archive, then $\Pi_a(E_F(I_F)) = \mathcal{A}_F$ where $\Pi_I$ represents the projection onto the input arcs of $F$.

Typically an extension rule is used to determine the *part* of the archive that corresponds to a *particular* input history.

An extension rule is required to satisfy several "reasonableness" conditions. These conditions will turn out to be essential in constructing extension rules for networks containing *loops*.

To express these conditions we use the following notation:

"e" will stand for a generic extension rule,

"h" will stand for a generic history,

"$h_1 <= h_2$" means that $h_2$ extends $h_1$.

We will also need the following terms:

**Definition**: An event v, in a history h, is said to be *consumable* by an extension rule e, if it is a production event that appears on the input arc of an operator in the network described by e and there is a history in the archive which contains extends h, as well as containing a consumption event corresponding to v.

**Definition**: An event v, is said to be *producible* in a history h, by an extension rule e, if there is a history in the archive of the network which extends h and contains v.

We may use the word *"enabled"* to mean either consumable or producible, and *"fired"* to represent the occurrence of the corresponding production or consumption.

The conditions on extension rules e can now be stated:

**Monotonicity**: The action of e cannot remove an event from a history; in symbols, if $h' \in e(h)$ then $h<=h'$.

**Causality**: An extension rule can only insert an consumption event after the corresponding production event, and a production event can be inserted only after the consumption events which triggered that production event.

**Stability**: If a history h, has no enabled events, then e satisfies $e(h) = \{h\}$.

**Finite Delay**: Within a chain of histories $h_1, h_2, \ldots, h_i, \ldots$ where for each i, $h_{i+1} \in e(h_i)$, no event remains enabled but not fired forever.

Note that monotonicity is not the same as that for partial orders, since embedding is not a partial order, as shall be seen. The finite-delay condition is similar to that of [Karp 69], which is used to show that a suitable notion of taking "limits" of sequences can be defined. It is actually a condition on such limits, rather than a constraint on the *presentation* of the extension rule.

We now present as an example the *extension rule for merge* in which the production of an output event can be delayed indefinitely. It consists of the composition of "consume", an unshown function which inserts the consumption events arbitrarily (consistent with the causality rule) into the history, and "ext", which extends the history

## 4.1. Extension Rules for Networks With Loops

It is easy to see that serial composition of operators corresponds to functional composition of the extension rules of the operators. The more important issue is to establish the connection between loop composition as expressed as an operation on archives and as an operation on extension rules. The extension rule for a network with a loop is easily *expressed* via a system of recursive equations involving the extension rules of the individual operators in the network.

To show that recursively defined expressions have a well-defined meaning involves introducing a suitable mathematical structure on the set of archives and showing that limits of sequences of approximations can be defined. Normally, the mathematical structure used is that of a complete partial order. However, for our purposes, this structure is not appropriate. We would like our structure to reflect the intuitive idea that histories improve by the *insertion* of new events, which are not necessarily added at the *end* of the histories. Thus we would like to express improvement of histories through the *subsequence* relation rather than prefix. This relation is not a partial order; a simple example is provided by the two sequences $(ab)^{\infty}$ and $(aab)^{\infty}$. The anti-symmetric property does not hold, since these sequences are unequal, yet each may be embedded in the other.

With *sets* of histories the situation is even worse. Even if one does have a complete partial order, subsets of the complete partial order can only be viewed as a complete partial order if one restricts the subsets to be closed in some appropriate sense [Plotkin 76] [Smythe 78]. We would like to be able to work with arbitrary sets of histories, since otherwise operators with different operational behaviors would have the same denotation.

by adding the produced c events. The function ext is defined as follows:

```
ext(h) = if h = [] then { [] }
         else
            if first(h) is a consumption then
              { first(h) ^ y | x ∊ ext(rest(h)) and y ∊ (produce[out(first(h)),x])}
               else
                 {first(h) ^ y | y ∊ ext(rest(h)) }
```

where *out* produces the output event corresponding to its argument and *produce* performs insertion of an event in all causally acceptable positions. All functions are assumed to be extended to set arguments in the obvious way. The function *produce* must insert the event in all positions upto the next output event. This ensures that the order of input events on a given input arc is preserved on the output arc. The definition of *produce* is

```
produce[e,h] = if h = [] then {e}
                  else
                    if first(h) is an output event then {e ^ h}
                      else
                        {first(h) ^ y | y ∊ produce[e,rest(h)]} ∪ {e ^ h}.
```

The notion of *limit* that we shall define is similar to the category theory notion of limit. Suppose we have an operator *F* which has an input arc c and an output arc b. Let the extension rule for this operator be denoted by E. Now suppose that the arc b is reconnected to the arc c. The extension rule must be modified to identify every reference to a b-event with the corresponding c-event. Let this modified extension rule be denoted by $E_{bc}$. Whenever $E_{bc}$ is used to extend a given history h, there will be new input events to *F* in the extended history $E_{bc}(h)$. The extension rule $E_{bc}$ must therefore be applied again to the result. Given a particular initial history, h, applying the extension rule $E_{bc}$ may result in several histories. Applying $E_{bc}$ to each history in the resulting set of histories will also yield several new histories.

Let us denote the set containing the original history, h, by $S_0$, and the sets containing the subsequent extensions by $S_1$ $S_2$ ... $S_i$ $S_{i+1}$ ... respectively. As we construct each subsequent extension, we define relations, written $R_i$, from $S_{i+1}$ to $S_i$ which express which histories in $S_i$ were extended by $E_{bc}$ to yield particular histories in $S_{i+1}$. Formally, if $h_1 R_i h_2$, then $h_1 \in S_{i+1}$, $h_2 \in S_i$ and $h_1 \in E_{bc}(h_2)$. We refer to such a sequence of sets and relations as the *tower* of $E_{bc}$ *over* h.

The limit of successive iterations of the extension rule $E_{bc}$ applied to h can be defined in terms the *tower of* $E_{bc}$ *over* h. The tower defines a *set of sequences of histories* through the relations $R_i$. Consider a typical such sequence, $h_1$ $h_2$ ... $h_i$ ... with

$h_{i+1}$ $R_i$ $h_i$ for each $i \in N$.

Such sequences of histories satisfy the following property:

**Lemma**: There exists a function f: N ---> N (N the natural numbers), such that, for every k in N, prefix(k,$h_{f(k)}$)) contains no consumable events or enabled events, furthermore this f can be chosen to be monotonic.

Here prefix(k, x) is the length k prefix of x.

**Proof:** Suppose h contains no consumable events or enabled events. Then by the stability condition, $E_{bc}$ will leave h unchanged and no history in the tower will have either an consumable event or an enabled event. Suppose that h does contain an consumable event or enabled event. Then there must be a *unique first* such event. By the finite-delay property, at some *finite* point in the tower, this event must be consumed or produced.

Suppose this consumption or production occurs in step t of the tower and the event that was consumed or produced is the n*th* event in this history. Using causality, we can define f(n) = t. Because an extension rule can insert events after their cause,we see that f is indeed monotonic.

Using this lemma, it is possible to construct the following sequence of embeddings:

$$\mathrm{prefix}(1,h_{f(1)}) <= \mathrm{prefix}(2,h_{f(2)}) <= \mathrm{prefix}(3,h_{f(3)}) <= \ldots \mathrm{prefix}(i,h_{f(i)})$$

However, these embeddings are additionally *prefixes* of one another, so the limit of this sequence does exist and is *unique*. We denote this limit sequence by $h^*$. By construction, none of the members of the sequence has any consumable or enabled events, so neither does $h^*$. Thus $e(h^*) = \{h^*\}$ by the stability condition. The limit of the entire tower is obtained by constructing the limit in the above fashion for every sequence of histories in the tower.

Let us call the limit of the tower of $E_{bc}$ over h $E^*_{bc}(h)$. Then $E^*_{bc}$ defines the extension rule for the network with b connected back into c.

The asymmetry between the roles of b and c noted in Section 3 is implicitly present here. The new input events present in the N$^{th}$ iteration of the extension rule were generated in the (N−1)$^{th}$ iteration. Thus the b-events have two roles: as output events of the (N−1)$^{th}$ iteration and as input events in the N$^{th}$ iteration. The effect would have been the same as if we had used the original extension rule E with the change that whenever a c-event is inserted, an b-event is also inserted immediately following the c-event. This view of the extension rule shows that it must necessarily construct histories of the form that appear in $L_{bc}(F)$.

## 5. Conclusions

The denotational approach we have outlined in this paper shares some features commonly associated with operational semantics. In particular, causality constraints are explicitly stated in the definitions of most archives. More importantly, the basic concept on which the theory is built is the computational event, which is a very operational concept. The use of events to define suitable domains for denotational semantics has already been studied in great depth by [Winskel 80]. This closeness to operational ideas makes this formalism a convenient and flexible tool for modelling systems of practical interest, while we retain the advantages of a denotational approach.

Our approach is similar to a variety of other semantic theories for networks containing indeterminate operators. These go under the general name of "trace" domains: a domain is built from sequences of computational actions. As far as we are aware, our theory is the only one using the category theoretic style of limit. Abstract semantics have been given by [Plotkin 76], [Broy 81], and [Abramsky 83]. The Plotkin construction does not handle unbounded indeterminacy or non-flat domains. Our construction does allow us to discuss unbounded indeterminacy, manifested by the merge operator, for example. The Broy construction does not allow one to describe arbitrary sets of possible results, as the sets in his domain constructions are required to satisfy a suitable closure condition. Important recent work on semantics of indeterminate constructs is that of Abramsky. His formalism is explicitly category theoretic; using multi- sets of possible results, he can handle arbitrary sets, unbounded indeterminacy, and power constructions over domains which are not flat.

## 6. References

| | |
|---|---|
| [Abramsky 83] | Abramsky, S.<br>Semantic Foundations of Applicative Multiprogramming.<br>In Diaz, J. (editor), *Automata, Languages and Programming*, pages 1-14. Springer-Verlag, July, 1983. |
| [Brock 81] | Brock J.D.,W.B.Ackermann.<br>Scenarios:A Model of Non-Determinate Computation.<br>In J.Diaz, I.Ramos (editor), *Formalization of Programming Concepts, LNCS 107*, pages 252-259. Springer-Verlag, New York, 1981. |
| [Broy 81] | Broy M.<br>A Fixed Point Approach to Applicative Multiprogramming.<br>In *Lectures at the International Summer School on Theoretical Foundations of Programming Methodology.* July, 1981. |
| [Davis 82] | A.L. Davis and R.M. Keller.<br>Dataflow program graphs.<br>*Computer* 15(2):26-41, February, 1982. |
| [Hennessey 82] | Hennessey, M.<br>*Synchronous and Asynchronous Experiments on Processes.*<br>Technical Report, University of Edinburgh, September, 1982. |
| [Karp 69] | Karp R. M., Miller R.<br>Parallel program schemata.<br>*JCSS* , May, 1969. |
| [Keller 78] | Keller R.M.<br>Denotational Models for Parallel Programs With Indeterminate Operators.<br>In E.J.Neuhold (editor), *Formal Descriptions of Programming Concepts*, pages 337-365. North-Holland, Amsterdam, 1978. |
| [Keller 83] | R.M. Keller.<br>Unpublished presentation on archives.<br>June, 1983.<br>Massachusetts Institute Technology, Applicative Languages Workshop, Endicott House. |
| [Plotkin 76] | Plotkin G.<br>A Powerdomain Construction.<br>*SIAM J. of Computing* 5(3), September, 1976. |

[Pratt 82] Pratt V.
On the Composition of Processes.
In *Ninth Annual ACM Symposium on Principles of Programming Languages,* pages 213-223. ACM, January, 1982.

[Riddle 79] Riddle W.E.
An Approach to Software System Behavior Description.
*Computer Languages* 4:29-47, 1979.

[Scott 70] Scott D.S.
Outline of a Mathematical Theory of Computation.
In *Proceedings of the Fourth Annual Princeton Conference on Information Sciences and Systems,* pages 169-176. 1970.

[Scott 71] Scott D.S., Strachey C.
*Towards a Mathematical Semantics for Computer Languages.*
Technical Report PRG-6, University of Oxford, 1971.

[Smythe 78] Smythe M.B.
Power Domains.
*J. CSS* 16:23-36, 1978.

[Winskel 80] Winskel G.
*Events in Computation.*
PhD thesis, University of Edinburgh, 1980.