Scenarios: A Model of Non-determinate Computation*

J. Dean Brock
William B. Ackerman

Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139
U. S. A.

Recently, much attention has been given to distributed computation by communicating processes interconnected by networks. Data flow is a well suited model of this form of computation. The schema of this model is the data flow graph, a network of autonomous operators which communicate solely by the asynchronous transmission of messages through input and output ports. By the choice of appropriate sets of operators, data flow graphs can represent many forms of parallel computation: by choosing processes as operators, programs of Kahn and MacQueen's [7] parallel programming language are representable; by choosing hardware elements. computer architectures are representable [10]; and by choosing elementary programming operations. such as + and *, programs of applicative languages are representable [1, 11]. Data flow graphs may themselves be the operators of larger graphs. This rather elegant interconnection property permits the modular construction of very large networks. The natural problem of semantics for such networks is to characterize the behavior of an entire network in terms of the behavior of its components, whether these components be subnetworks or elementary operators.

It is known that networks of determinate processes can be adequately characterized by history functions which map each input history tuple to its output history tuple response. Many semantic theories for non-determinate processes have been described, but in none have processes been represented as abstractly as the natural extension of history functions to history relations which map each input history tuple into the set of possible output history tuple responses. Using the data flow model of computation, we demonstrate that history relations are an inadequately detailed characterization of non-determinate behavior. An alternative characterization, incorporating more causality information into history relations, is presented.

1. History Relations

During a network computation, each operator input port receives and each operator output port transmits a *history* (sequence) of values. An operator is *determinate* if, for each input history tuple, it has only one possible output history tuple response. The function mapping its input history tuples into its output history tuple responses is the operator's *history function*. Note that, due to the asynchronous nature of the communication, nothing is said about absolute timing or relative timing of events at different operator ports, though the order of values passing through a single port is, of course, significant. Kahn [6] has shown that a network composed of determinate processes interconnected by channels with unbounded buffering is itself determinate, and he has described a fixpoint theory for deriving its history function from the history functions of its components.

* This research was supported by the National Science Foundation under contract 7915255-MCS.

However, networks for real-time applications, such as operating systems, often require non-determinate processes. One very important, and very basic, non-determinate operator is the *merge* operator used by Dennis [4] in an airline reservation system and by Arvind, Gostelow, and Plouffe [2] in an operating system resource allocator.

The *merge* operator receives two input histories and non-determinately merges them into one output history. The intended implementation of *merge* is a process which waits for a value to appear on either of its input ports and then produces the received value at its output port before returning to its waiting state. If the *merge* were characterized by the obvious extension of history functions to *history relations* mapping each input history tuple into the set of possible resulting output history tuples, it would be defined for *finite input histories* as:

$$\begin{aligned} &merge(X, \varepsilon) = \{X\} \\ &merge(\varepsilon, Y) = \{Y\} \\ &merge(i \cdot X, j \cdot Y) = \{i \cdot Z \mid Z \in merge(X, j \cdot Y)\} \cup \{j \cdot Z \mid Z \in merge(i \cdot X, Y)\} \end{aligned}$$

where ε is the empty history, i and j are single values, X and Y are histories, and $\cdot$ is the history concatenation operator. For now, we decline to define *merge* for infinite input histories, because only finite input histories are required for the examples of this paper and because we thus avoid the controversy of the fair versus the unfair *merge* [13].

2. Semantic Theories for Non-Determinate Networks

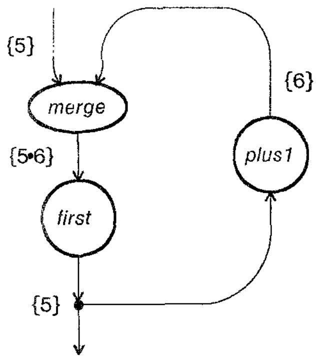
A few semantic theories have been given for networks of non-determinate parallel processes. Kosinski [9] has defined a denotational semantics in which values are "tagged" with the sequence of non-determinate *merge* "decisions" which caused their generation. Networks are then defined as functions mapping tagged input history tuples into sets of tagged output history tuples. The derivation of a network's semantic characterization from those of its components involves the matching of value tags to determine consistent input histories for processes with multiple input ports. This functional characterization contains many details of internal network communication together with the information concerning external communication contained in history relations. For large systems these internal details, most of them irrelevant, could easily overwhelm the useful information.

Milne and Milner [12] have constructed a *flow algebra* of networks in which processes are roughly characterized as a set of possible actions. Each action consists of an instance of process communication followed by process "renewal," that is, assumption of a new set of possible actions. This semantic theory is based on a model of computation which differs fundamentally from that of Kahn's theory. Here, an instance of communication is an unbuffered simultaneous exchange of values. Communication through unbounded buffers can be modeled, but processes which are determinate in Kahn's theory are often non-determinate in this one. Furthermore, because networks are formally characterized as elements of an elaborate powerdomain, some networks with significantly different behaviors are represented by the same element. For example, all networks with even the *possibility* of deadlock, that is, the production of no output, are indistinguishable.

By comparison, history relations are a simple specification of non-determinate computation, but, unfortunately, they lack sufficient detail to describe the behavior of interconnected networks. In the remainder of this paper, we will show this incompleteness and will describe a natural extension of history relations that we believe to be adequate for non-determinate computation. This incompleteness will be demonstrated by exhibiting two data flow graphs S_1 and S_2 with the same history relation, but which cannot be substituted for each other as components of a larger network without changing that network's history relation. Consequently, there can be no "correct" interconnection rules for networks represented by history relations. Formally interpreted within the algebraic terminology employed by Milne and Milner [12], the incompleteness result shows that the function mapping each data flow graph into its history relation cannot be extended to a morphism of the algebra of data flow graphs.

It should be noted that our demonstration of the incompleteness of history relations differs fundamentally from Keller's [8]. His demonstration uses a graph similar that of Figure 1, where *first* is a determinate graph operator that passes only its first input value. Beside each graph arc is written all its possible histories with graph input history 5. When computation begins, the value 5 passes through the leftmost input port of the *merge* and causes 6 to appear on the rightmost input port. Eventually, the *merge* will produce the output history 5 • 6. However, Keller, "pursuing the incremental approach" to semantics, assumes that the output of the *merge* operator could be any element of *merge*(5, 6), including the clearly impossible 6 • 5; given that if "these inputs were presented externally, this would certainly be the case." From this "merge anomaly" he concludes that the history relations must be extended to include causality relations between individual elements of different histories. We accept only that this anomaly demonstrates that history relation interconnection rules which ignore causality fail.[1] It does not show that the necessary causality relations cannot be inferred from history relations by, for example, requiring that later output of a *merge* does not "rewrite" earlier output.[2] Keller shows there are no *easy* interconnection rules for networks characterized by history relations. We show there are *no* interconnection rules.

Figure 1.



Keller's Example (with slight modification)

1. Or, as Pratt [14] states: "The merge anomaly illustrates nothing beyond the fact that an incorrect denotational semantics may well not agree with a correct operational one."

2. There are several subalgebras of data flow graphs which exhibit the "merge anomaly" but which are amenable to analysis by history relations. One consists of the graphs constructed with determinate data flow operators and Park's [13] non-determinate *fairmerge* operator, which differs from our *merge* on finite input histories. Another may be constructed with one generator, *plus1*∘*first*∘*merge*, and one interconnection rule.

3. The Incompleteness of History Relations

Let S_1 and S_2 be the graphs shown in Figure 2. Syntactically, S_1 and S_2 may be written:

$$S_k(X, Y) = P_k(merge(D(X), D(Y)))$$

D, P_1, and P_2 are all determinate processes which produce at most two output values. Process D produces two copies of its first input value. Both P_1 and P_2 pass through their first two input values, but P_1 will produce its first output as soon as it receives its first input, while P_2 will not produce any output until it has received two input values. The history functions for these processes are:

$D(\varepsilon) = \varepsilon$	$P_1(\varepsilon) = \varepsilon$	$P_2(\varepsilon) = \varepsilon$
$D(i \bullet X) = i \bullet i$	$P_1(i) = i$	$P_2(i) = \varepsilon$
	$P_1(i \bullet j \bullet X) = i \bullet j$	$P_2(i \bullet j \bullet X) = i \bullet j$

Despite the difference between P_1 and P_2, networks S_1 and S_2 are represented by the same history relation. Neither network produces any output unless it receives input. If S_k receives one or more input values on either input port, its internal P_k process is guaranteed to receive two or more input values, thus "avoiding" the difference between processes P_1 and P_2. Consequently, S_1 and S_2 have the same history relation:

$$S_k(\varepsilon, \varepsilon) = \{\varepsilon\}$$
$$S_k(i \bullet X, \varepsilon) = \{i \bullet i\}$$
$$S_k(\varepsilon, j \bullet Y) = \{j \bullet j\}$$
$$S_k(i \bullet X, j \bullet Y) = \{i \bullet i, i \bullet j, j \bullet i, j \bullet j\}$$

However, there is a subtle difference in their behaviors: S_2 will not produce its first output until its second output has been determined. These networks can be placed within a larger network which uncovers this difference.

Now let T_1 and T_2 be the networks of Figure 3. T_k is a cyclic network with non-determinate process S_k and determinate process *plus1*. Inputs to T_k are routed to the leftmost input port of S_k. The outputs of S_k are routed to two sources: to the output port of T_k and, through the *plus1* operator, to the rightmost input port of S_k. A possible equational specification of T_k is:

$$T_k(X) = Y \text{ such that } Y = S_k(X, plus1(Y))$$

Suppose T_1 receives the single input value 5. The value 5 passes through the leftmost D, through *merge*, and through P_1. The value 5 then becomes the first output of T_1. The value 5 is also input to the *plus1*

Figure 2.

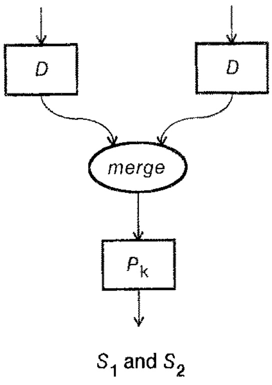
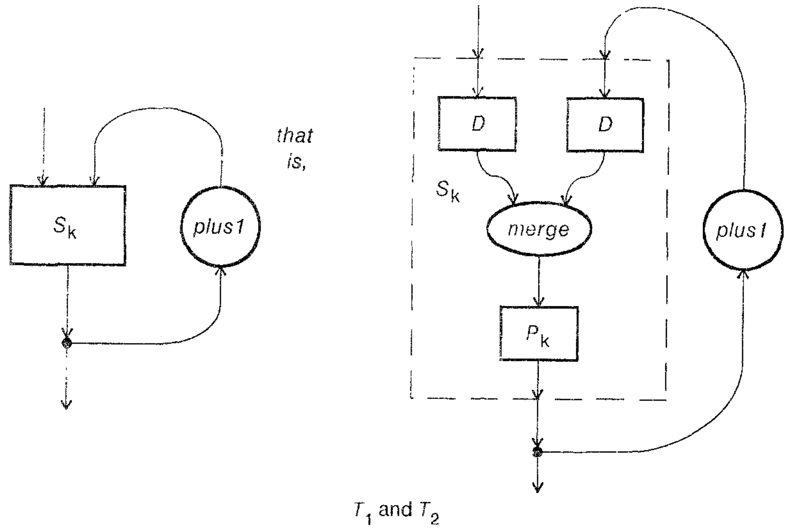


Figure 3.



T_1 and T_2

operator, causing the value 6 to be presented to the rightmost port of T_1, where it passes through the rightmost D. Note that *merge* has a "choice" of producing as its second output either its second leftmost input, which is 5, or it first rightmost input, which is 6. Consequently, the second output of T_1 could be either 5 or 6. Therefore:

$T_1(5) = \{5 \bullet 5, 5 \bullet 6\}$

Now suppose T_2 receives the input seqquence 5. The value 5 passes through the leftmost D, through *merge*, and enters P_2. However, P_2 will produce no output until it has received a second input. Eventually, a second value 5 must be produced by the leftmost D and pass through the *merge* to P_2. Then P_2, and consequently S_2 and T_2, will produce the output sequence 5 • 5. Therefore:

$T_2(5) = \{5 \bullet 5\}$

By taking two networks with the same history relation and showing that they are not substitutable as components of a larger network, we have demonstrated that history relations incompletely specify the behavior of non-determinate networks. Although we have derived the history relations of our example networks somewhat intuitively, more formal derivations could be made.

4. Scenarios

The shortcoming of history relations can be overcome by using a different model of non-determinate computation -- *scenario sets*. A history relation is a set of pairs of input and output history tuples. Each pair represents one possible, potentially eternal, computation of a modeled process. A single *scenario* is one of these pairs augmented with a partial ordering on the individual data items of the history tuples showing the causality constraints. Two data items are causally ordered under this relation, if the event of producing one *must* precede the event of producing the other. Items of the history of a single port are totally related by this ordering, since the production of early items must precede the production of later ones. An output item is causally related to those input items which contributed, through a non-indivisible primitive event or through a series of internal process events, to its production. Due to the time-independent nature of data flow computation, the are no causality relations between items of different input ports, between items of different output ports, and from output items to input items. In some ways the ordering of the scenario resembles the "combined order" of Hewitt and Baker [5].

A scenario can be drawn graphically with each input or output history shown as a column of values with arrows drawn down each column and from an input value to an output value wherever a causality constraint exists between the two. The requirement that the scenario represent a partial order is just the requirement that it contain no directed cycles of arrows, that is, that no value "causes" itself. The three possible scenarios for the *merge* operator receiving the input history tuple (5 • 6, 7) are shown below:

$(5 \bullet 6, 7) \rightarrow 5 \bullet 6 \bullet 7$

$$\begin{tikzcd}
5 \arrow[d] \arrow[rr, bend left] & 7 \arrow[ddr] & 5 \arrow[d] \\
6 \arrow[rr] & & 6 \arrow[d] \\
& & 7
\end{tikzcd}$$

$(5 \bullet 6, 7) \rightarrow 5 \bullet 7 \bullet 6$

$$\begin{tikzcd}
5 \arrow[d] \arrow[rr, bend left] & 7 \arrow[dr] & 5 \arrow[d] \\
6 \arrow[drr] & & 7 \arrow[d] \\
& & 6
\end{tikzcd}$$

$(5 \bullet 6, 7) \rightarrow 7 \bullet 5 \bullet 6$

$$\begin{tikzcd}
5 \arrow[d] \arrow[drr] & 7 \arrow[r] & 7 \arrow[d] \\
6 \arrow[drr] & & 5 \arrow[d] \\
& & 6
\end{tikzcd}$$

Every network may be characterized by the set of all its possible scenarios.

The composition rule for two networks, in terms of their sets of scenarios, is:

First, "enumerate" the Cartesian product of the scenario sets for the two networks.

Second, discard all scenario pairs whose data values do not agree on ports that are linked to each other, and merge each pair into one scenario, identifying the columns of linked ports. The remaining pairs are called *port-consistent*.

Third, if a directed cycle exists within a merged port-consistent pair, discard the pair. The remaining pairs are called *causally-consistent*.

Fourth, remove the columns for ports that are linked (since they do not appear externally in the final system), preserving the partial order. The resulting set of scenarios characterizes the joined networks.

The reader may have noticed the similarities between port-consistent pairs and fixpoints and between causally-consistent pairs and least fixpoints.

We will now apply the scenario analysis to systems S_k and T_k By application of the preceding composition rule, the scenario sets of S_k may be derived using the scenario sets for D, *merge*, and P_k. Systems S_1 and S_2 have the following scenarios in common:

$(\varepsilon, \varepsilon) \rightarrow \varepsilon$

$$\begin{tikzcd}
\varepsilon & \varepsilon & \varepsilon
\end{tikzcd}$$

$(i, \varepsilon) \rightarrow i \bullet i$

$$\begin{tikzcd}
i \arrow[rr, bend left] & \varepsilon & i \arrow[d] \\
& & i
\end{tikzcd}$$

$(\varepsilon, j) \rightarrow j \bullet j$

$$\begin{tikzcd}
\varepsilon & j \arrow[r] & j \arrow[d] \\
& & j
\end{tikzcd}$$

$(i, j) \rightarrow i \bullet i$

$$\begin{tikzcd}
i \arrow[rr, bend left] & j & i \arrow[d] \\
& & i
\end{tikzcd}$$

$(i, j) \rightarrow j \bullet j$

$$\begin{tikzcd}
i & j \arrow[r] & j \arrow[d] \\
& & j
\end{tikzcd}$$

However, owing to the differences between P_1 and P_2, S_1 also has the scenarios:

$(i, j) \rightarrow (i \bullet j)$

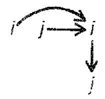
$$\begin{tikzcd}
i \arrow[rr, bend left] & j \arrow[dr] & i \arrow[d] \\
& & j
\end{tikzcd}$$

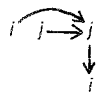
$(i, j) \rightarrow (j \bullet i)$

$$\begin{tikzcd}
i \arrow[rr, bend left] \arrow[drr] & j \arrow[r] & j \arrow[d] \\
& & i
\end{tikzcd}$$

whereas S_2 has the corresponding scenarios:

$(i, j) \rightarrow (i \bullet j)$ $\qquad\qquad$ $(i, j) \rightarrow (j \bullet i)$

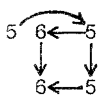


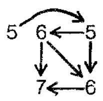


Note how these scenarios indicate the subtle difference between S_1 and S_2: in S_1, one input value causes the first output; in S_2, two input values cause the first output.

Application of the scenario composition rule to the joining of S_k and *plus1* to form T_k, shows that T_1 contains these scenarios:

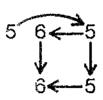
T_1: $5 \rightarrow 5 \bullet 5$ $\qquad\qquad$ T_1: $5 \rightarrow 5 \bullet 6$
S_1: $(5, 6 \bullet 6) \rightarrow 5 \bullet 5$ $\qquad\qquad$ S_1: $(5, 6 \bullet 7) \rightarrow 5 \bullet 6$
plus1: $5 \bullet 5 \rightarrow 6 \bullet 6$ $\qquad\qquad$ *plus1*: $5 \bullet 6 \rightarrow 6 \bullet 7$

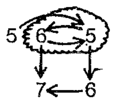




indicating that either $5 \bullet 5$ or $5 \bullet 6$ is a possible response to input history 5. Before scenarios with cycles are discarded, T_2 has the following port-consistent pairs of scenarios:

T_2: $5 \rightarrow 5 \bullet 5$ $\qquad\qquad$ T_2: $5 \rightarrow 5 \bullet 6$
S_2: $(5, 6 \bullet 6) \rightarrow 5 \bullet 5$ $\qquad\qquad$ S_2: $(5, 6 \bullet 7) \rightarrow 5 \bullet 6$
plus1: $5 \bullet 5 \rightarrow 6 \bullet 6$ $\qquad\qquad$ *plus1*: $5 \bullet 6 \rightarrow 6 \bullet 7$





However, the rightmost pair contains a directed cycle and must be discarded since it not causally-consistent. Therefore, $5 \bullet 5$ is the only possible response of T_2 to input history 5.

Please note that this paper is an introduction to the scenario model of computation. By appealing to "common-sense" notions of the role of causality in network communication, we have argued that scenarios are a correct representation of network computation. There is other evidence, perhaps much more satisfying to the theoretically inclined, that could have been presented here. For example, we could have proved that scenario theory agrees with Kahn's [6] theory when restricted to determinate systems. This proof is straightforward, though somewhat tedious.

There are several difficult questions, not addressed here, that can only be answered through a thorough examination of the theory of scenarios. One question is whether or not the scenario set derived for a network is total, that is, contains a scenario for every possible input tuple. Insuring totality requires the imposition of conditions, similar to monotonicity and continuity, on scenario sets. Another question is whether or not the theory of scenarios could or should be related to existing fixpoint theories [9, 12] for non-determinate computation. These issues are being addressed in the thesis research of one of the authors [3].

5. Conclusion

Many researchers have sought a semantic theory that extends the history function model of determinate networks to a history relation model of non-determinate networks. By demonstrating that history relations are a fundamentally incomplete characterization of non-determinate computation, we have shown that no such extension exists. This demonstration was accomplished by constructing two networks which, though indistinguishable by history relations, cannot be substituted for each other in a larger network without changing the history relation of the larger network.

A few alternative theories for non-determinate networks have been proposed. Unfortunately, these theories offer complicated semantic models revealing many details of network computation, far more than is often relevant for an interface specification. We have introduced a relatively simple network model, the set of scenarios. This model is a straightforward extension of history relations, for a scenario is merely an element of a history relation whose constituent values are partially ordered by a relation representing causality; but it can reveal subtle, though important, distinctions unseen in history relations.

6. References

[1] Ackerman, W. B., "Data Flow Languages", *Proceedings of the 1979 National Computer Conference, AFIPS Conference Proceedings 48*, June 1979, 1087-1095.

[2] Arvind, K. P. Gostelow, and W. Plouffe, "Indeterminacy, Monitors and Dataflow", *Proceedings of the Sixth ACM Symposium on Operating Systems Principles, Operating Systems Review 11*, 5(November 1977), 159-169.

[3] Brock, J. D., *Formal Properties of a Non-Determinate Data Flow Language*, Ph. D. thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts, expected September 1981.

[4] Dennis, J. B., "A Language Design for Structured Concurrency", *Design and Implementation of Programming Languages: Proceedings of a DoD Sponsored Workshop* (J. H. Williams and D. A. Fisher, Eds.), *Lecture Notes in Computer Science 54*, October 1976, 231-242.

[5] Hewitt, C. E., and H. Baker, "Actors and Continuous Functionals," *Formal Description of Programming Concepts*, (E. J. Neuhold, Ed.), August 1977, 367-390.

[6] Kahn, G., "The Semantics of a Simple Language for Parallel Programming", *Information Processing 74: Proceeding of the IFIP Congress 74* (J. L. Rosenfeld, Ed.), 1974, 471-475.

[7] Kahn, G., and D. MacQueen, "Coroutines and Networks of Parallel Processes", *Information Processing 77: Proceedings of IFIP Congress 77* (B. Gilchrist, Ed.), August 1977, 993-998.

[8] Keller, R. M., "Denotational Models for Parallel Programs with Indeterminate Operators", *Formal Description of Programming Concepts* (E. J. Neuhold, Ed.), August 1977, North-Holland Publishing Company, New York, New York, 337-366.

[9] Kosinski, P. R., "A Straightforward Denotational Semantics for Non-Determinate Data Flow Programs", *Conference Record of the Fifth ACM Symposium on Principles of Programming Languages*, January 1978, 214-221.

[10] Leung, C. K. C., "ADL: An Architecture Description Language for Packet Communication Systems", *Proceedings of the 4th International Symposium on Computer Hardware Description Languages*, October 1979, 6-13.

[11] McGraw, J. R., "Data Flow Computing -- Software Development", *IEEE Transactions on Computers C-29*, 12(December 1980), 1095-1103.

[12] Milne, G., and R. Milner, "Concurrent Processes and Their Syntax", *Journal of the ACM 26*, 2(April 1979), 302-321.

[13] Park, D., "On the Semantics of Fair Parallelism", *Abstract Software Specifications: 1979 Copenhagen Winter School Proceedings* (D. Bjørner, Ed.), *Lecture Notes in Computer Science 86*, February 1979, 504-526.

[14] Pratt, V. R., comment in published discussion on "Denotational Models with Indeterminate Operators" by R. M. Keller, *Formal Description of Programming Concepts* (E. J. Neuhold, Ed.), August 1977, North-Holland Publishing Company, New York, New York, 366.