MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Laboratory for Computer Science
(formerly Project MAC)

Computation Structures Group Note 33-1

An Anomaly in the Specifications of Nondeterminate Packet Systems

by

J. Dean Brock and William B. Ackerman

4 January 1978

A problem relating to the specification of the semantics of nondeterminate packet communication systems has been discovered. It has generally been assumed that the semantics of these computational models can be specified by history functions that map input history vectors into sets of possible output history vectors. However, two systems to be presented here demonstrate the inadequacies of history functions for describing the behavior of *all* packet communication systems.

A history is a sequence of data values received at an input port or transmitted at an output port. A history vector is a tuple consisting of a history for each port of a system. History functions, with various embellishments, have been used by many researchers [1, 2, 4, 6, 8, 9, 10] to specify the semantics of packet communication systems. While these semantic theories differ in many ways, they usually have the following common aspects. A system S is specified by a function $\mathcal{F}_S$ which maps history vectors into sets of history vectors. A system S is said to realize $\mathcal{F}_S$ (that is, $\mathcal{F}_S$ completely describes the behavior of S) if:

1) Whenever history vector X of input symbols is presented to the inputs of S, some output history vector in the set $\mathcal{F}_S(X)$ will eventually appear as the outputs of S. If $\mathcal{F}_S(X)$ contains only one vector for each X, S is said to be *determinate*.
2) S exhibits *causality*, or *finite-delay*, that is, the outputs do not appear until *after* the presentation of the inputs.

Further it is generally accepted that all history functions corresponding to

physically realizable systems are:

1) *monotonic*, that is, the more inputs a system receives the more outputs it will produce, without changing previous outputs. This is equivalent to saying that a system never has to "erase" an output that it has transmitted. For determinate systems, $\mathcal{F}$ is monotonic if $X \sqsubseteq Y \Rightarrow \mathcal{F}(X) \sqsubseteq \mathcal{F}(Y)$, where $\sqsubseteq$ is the prefix ordering on history vectors, or its natural extension to singleton sets of history vectors.
2) *continuous*, that is, a system's response to an infinite input is the "limit" of its response to finite prefixes of that input. For determinate systems, $\mathcal{F}$ is continuous if given an increasing chain $X_0, X_1, \ldots$, $\mathcal{F}(\bigcup X_n) = \bigcup \mathcal{F}(X_n)$.

Several definitions of monotonicity and continuity have been given for nondeterminate computations [10, 11, 13]. These definitions are discussed later in this paper.

Such semantics are generally assumed to obey a criterion of *modularity* (or *substitutibility*) which states that $\mathcal{F}_S$ is a complete specification of the behavior of S. When modules are interconnected, the resultant system is also a module, whose input and output ports are the unconnected ports of the constituent modules. The semantic specification of the resultant system depends only on the specifications of its components and the manner of their interconnection. Hence any module can be treated as a "black box," which can be understood by use of the semantic characterization without examination of its internal structure.

These properties have been proved, in various forms and under various assumptions, for determinate systems [1, 8, 12], for determinate recursive systems [4, 9], and for acyclic systems [6]. However, they appear not to be true for arbitrary cyclic interconnections of non-determinate systems as shown by the following example.[1]

Consider the single input/single output systems S and T illustrated in figures 1 and 2. There are five types of modules common to both of these systems. The 1st module passes its first input token as its only output token. Succeeding input tokens are absorbed and discarded. The 2nd module passes its second input token as its only output token, discarding further tokens. The cons module (similar to that described by Weng [14]) produces as its first output token the first input on its left input port and then produces as its remaining tokens the input tokens from its right input port. The small dots represent modules which copy their input tokens to each of their output ports. All of the preceding modules are determinate. Their history functions may be specified as:

$\mathcal{T}_1 st(\Lambda) = \{\Lambda\}$ $\quad$ Λ = empty history

$\mathcal{T}_1 st(X\alpha) = \{X\}$ $\quad$ X, Y = single tokens

α, β = arbitrary (possibly empty) streams of tokens

1. The "anomaly" discussed in Keller [9] does not demonstrate the inability of history functions to characterize non-determinate operators, but rather the inability of the "incremental approach to semantics" to characterize cyclic interconnections of non-determinate operators.

$\mathcal{T}_{2nd}(\Lambda) = \{\Lambda\}$

$\mathcal{T}_{2nd}(X) = \{\Lambda\}$

$\mathcal{T}_{2nd}(XY\alpha) = \{Y\}$

$\mathcal{T}_{cons}(\Lambda, \beta) = \{\Lambda\}$

$\mathcal{T}_{cons}(X\alpha, \beta) = \{X\beta\}$

The **merge** module [2, 3, 5] is non-determinate. It merges (arbitrates) its two input streams into one output stream.

$\mathcal{T}_{merge}(\alpha, \beta)$ = {all streams which constitute "shufflings" of α and β, preserving the relative order of symbols within α and within β}

There are various nuances of definition of this module, relating to its response to infinite input histories, which are irrelevant to this paper.

When the system S receives no input tokens, that is, has as its input the empty stream Λ, the system produces no outputs. Hence:

$\mathcal{T}_S(\Lambda) = \{\Lambda\}$

When the system receives a single token X as its input, the value X passes through the left side of the cons operator to be the first element of the output stream. Also, the value X passes through the merge module and the right port of the cons module to be the second element of the output stream. Consequently:

$\mathcal{T}_S(X) = XX$

Finally, when the module receives more than one token, or a stream $XY\alpha$, again

the first input X becomes the first token of the output stream. (As before, X and Y are any single tokens. α denotes an arbitrary stream.) However, now the first and second tokens "race" at the merge operator to become the second element of the output stream. The remaining elements of the input stream are discarded. Hence:

$$\mathcal{F}_S(XY\alpha) = \{XX, XY\}$$

System T is system S with the addition of the determinate wait module as shown in figure 2. The wait module passes all tokens from the top (data) input, but only after at least one token has been presented at the right (enable) input.

$$\mathcal{F}_{\text{wait}}(\alpha, \Lambda) = \{\Lambda\}$$

$$\mathcal{F}_{\text{wait}}(\alpha, \beta) = \{\alpha\} \text{ if } \beta \neq \Lambda$$

System T realizes the same function as system S, that is $\mathcal{F}_S = \mathcal{F}_T$. Observe that the only effect of the wait operator is to delay the system's first input token until the merge operator has produced an output. However, the system's first token will also be an input to the merge operator, thus guaranteeing that the merge will eventually produce an output.

Now consider system S connected to system L (a cons and a boolean not operator) as in figure 3. If a true token is introduced, it passes all the way through system S and is inverted to a false token by the not operator. The false token then becomes the second token to enter S. Both the true and false

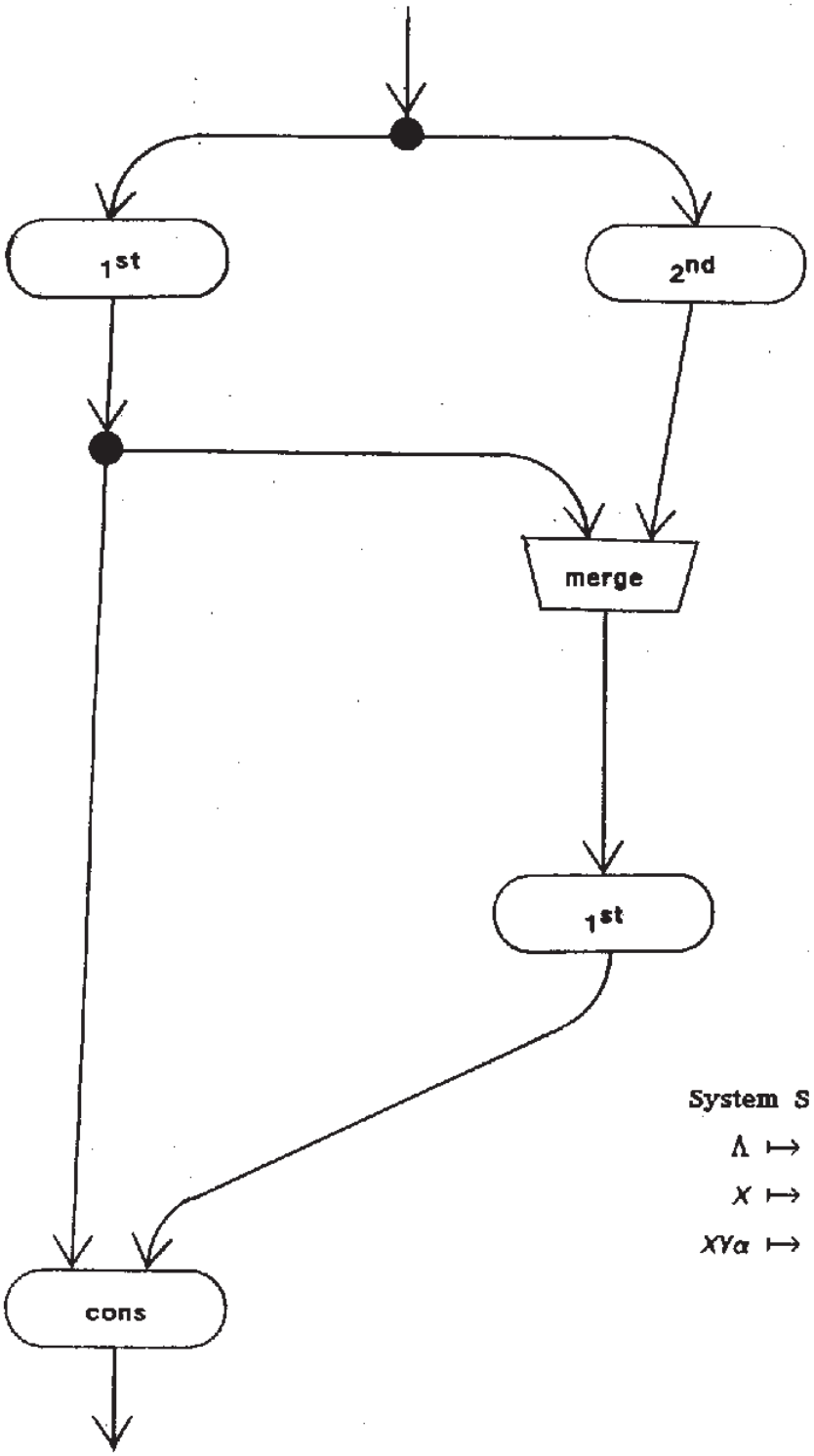
tokens will appear at the **merge** inputs. Since the original **true** token might not have passed through the **merge** by the time the **false** token appears, the **merge** can pass the tokens in either order. Whichever is passed first becomes the second output token of the system. The final output history can therefore be either (**true true**) or (**true false**).

Consider the same interconnection using system *T*, as in figure 4. The **wait** operator prevents the first **true** token from being transmitted as output by system *T* until the **merge** has already decided to accept its left input ahead of its right input. By the time the **false** token appears at the **merge** input, it must be the second. The final output can therefore only be (**true true**).

We therefore have two interconnections which behave differently, although their respective components are "identical." Thus the complete specification of a system must contain more information than just the history vector function. It must tell how the nondeterminacy is reduced or removed when external causality constraints are placed on the inputs and outputs. In systems *S* and *T*, for example, the response to $XY\alpha$ is $\{XX, XY\}$ without external restrictions. However, if the second input symbol is known not to be introduced until the first output has been transmitted, the response of system *T* to $XY\alpha$ is only $\{XX\}$, whereas the response of system *S* is still $\{XX, XY\}$. The appendix of this note describes a representation of semantics of non-determinate packet communication systems.

Many semantic theories are restricted to the specification of monotonic and continuous functions. Several orderings have been proposed for the powerset of history functions which would allow non-determinate computations to be monotonic and continuous. Since the merge module is not monotonic using the ordering of either Plotkin [13] or Lehman [11], these theories cannot be used to specify non-determinate packet communication systems. The "unfair" merge module is continuous using the ordering of Kosinski [10]. The "unfair" merge module is allowed to ignore tokens on one of its input ports if it is receiving an infinite input history on its other port. Since merge modules are generally considered to be "fair," this theory cannot be used to specify many "interesting" packet communication systems.

Figure 1



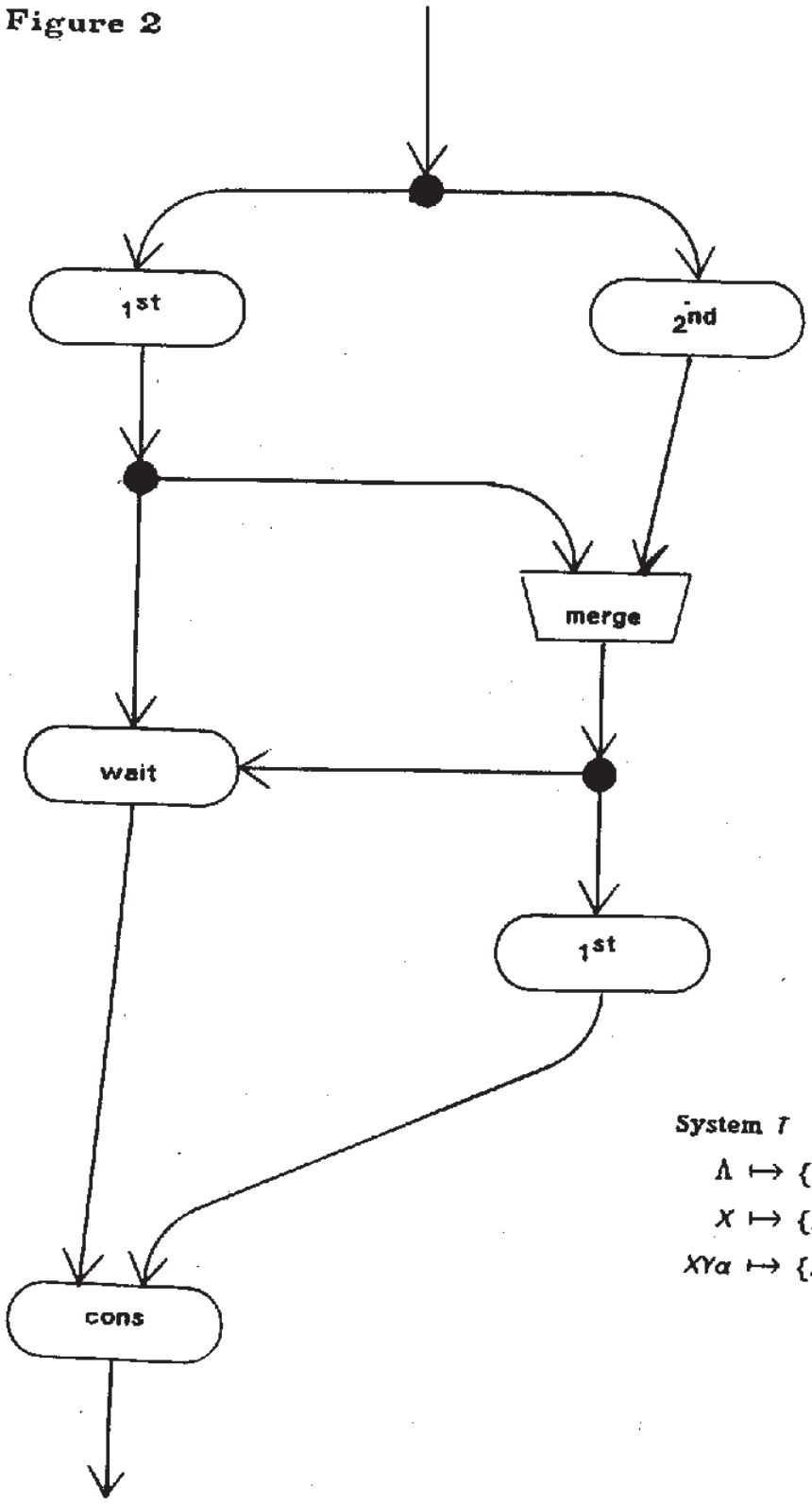
System S

$\Lambda \mapsto \{\Lambda\}$

$X \mapsto \{XX\}$

$XY\alpha \mapsto \{XX, XY\}$

Figure 2



System Γ

$\Lambda \mapsto \{\Lambda\}$

$X \mapsto \{XX\}$

$XY\alpha \mapsto \{XX, XY\}$

Figure 3

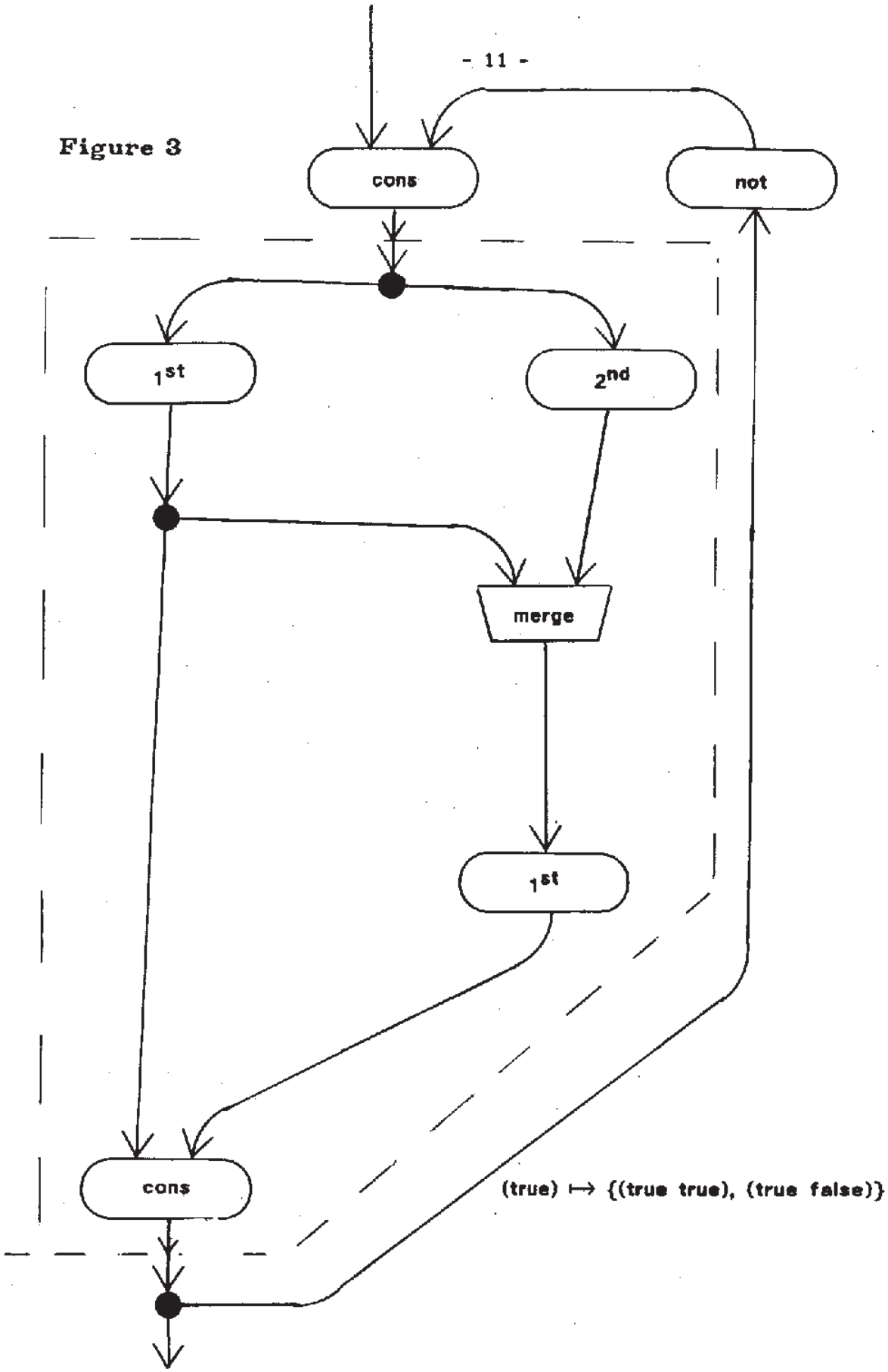
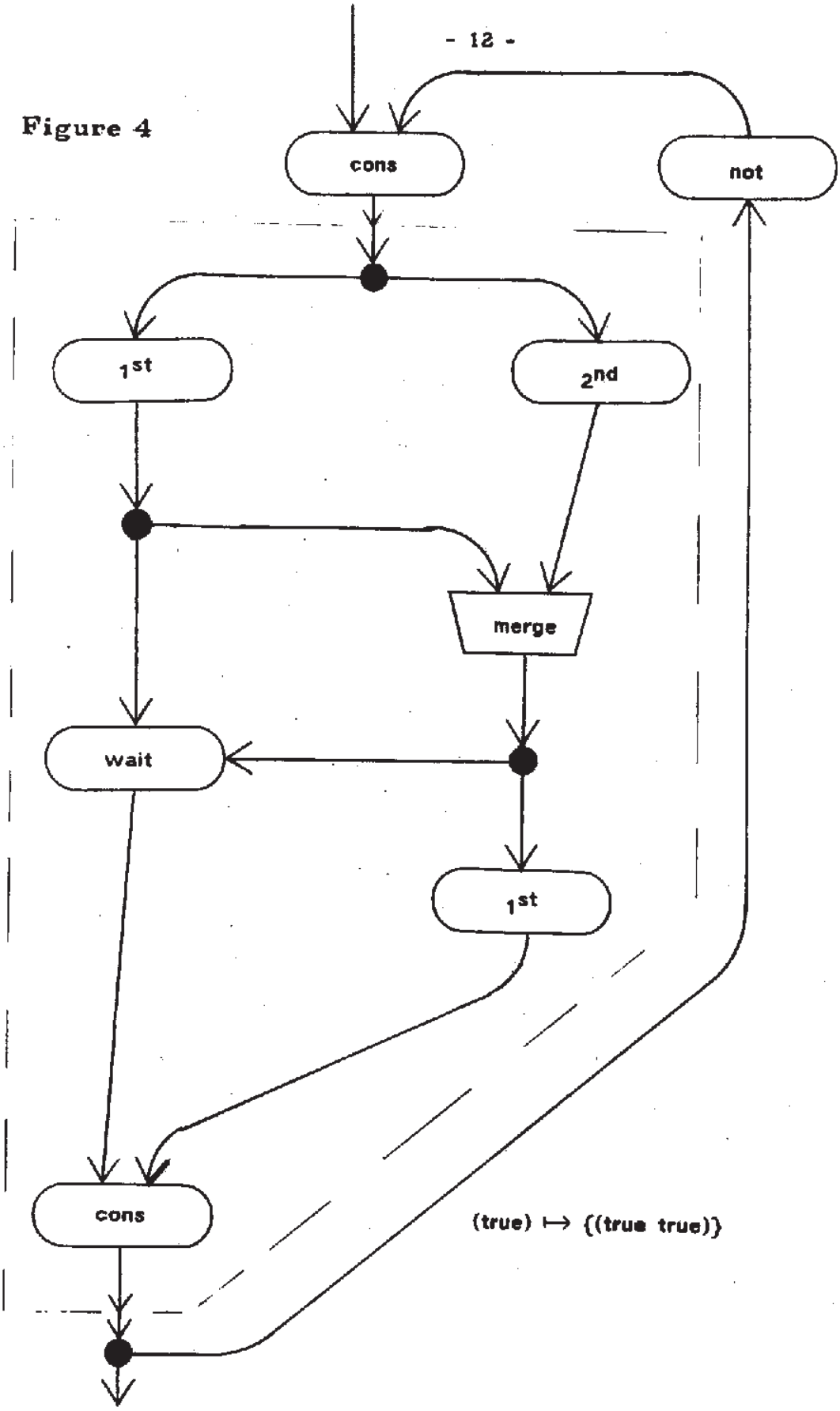


Figure 4



Bibliography

[1] Ackerman, W. B., "Interconnections of Determinate Systems," Computation Structures Group (Note 31), Laboratory for Computer Science, MIT, Cambridge, Massachusetts, July 1977.

[2] Ackerman, W. B., *A Structure Memory for Data Flow Computers*, Laboratory for Computer Science (TR-186), MIT, Cambridge, Massachusetts, August 1977.

[3] 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.

[4] Brock, J. D., *Formal Semantics of Data Flow Language*, S. M. Thesis in preparation, Department of Electrical Engineering and Computer Science, MIT, Cambridge, Massachusetts, expected January 1978.

[5] Dennis, J. B., D. P. Misunas, and C. K. C. Leung, "A Highly Parallel Processor Based on a Data Flow Machine Language," Computation Structures Group (Memo 134), Laboratory for Computer Science, MIT, Cambridge, Massachusetts, January 1977.

[6] Ellis, D. J, *Formal Specifications for Packet Communication Systems*, Laboratory for Computer Science (TR-189), MIT, Cambridge, Massachusetts, November 1977.

[7] Hewitt, C. E., and H. Baker, "Actors and Continuous Functionals," *Proceedings of the IFIP Working Conference on the Formal Descriptions of Programming Concepts*, August 1977, 16.1-16.21.

[8] Kahn, G., "The Semantics of a Simple Language for Parallel Programming," *Information Processing 74: Proceedings of the IFIP Congress 74*, August 1974, 471-475.

[9] Keller, R. M., "Denotational Models for Parallel Programs with Indeterminate Operators," *Proceedings of the IFIP Working Conference on Formal Description of Programming Concepts*, August 1977, 15.1-15.27.

[10] Kosinski, P. R., "A Straightforward Denotational Semantics for Non-Determinate Data Flow Programs," to be presented at the *Fifth ACM Symposium on Principles of Programming Languages*, January 1978.

[11] Lehman, D. J., "Categories for Fixpoint Semantics," *Seventeenth Annual Symposium on Foundations of Computer Science*, October 1976, 122-126.

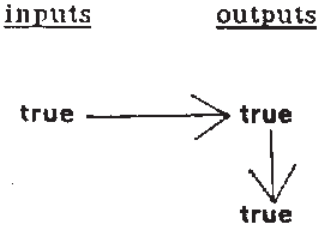
[12] Patil, S. S., "Closure Properties of Interconnected Systems," *Record of the Project MAC Conference on Concurrent Systems and Parallel Computation*, 1970, 107-116.

[13] Plotkin, G. D., "A Powerdomain Construction," *SIAM Journal of Computing* 5, 3(September 1976), 452-487.

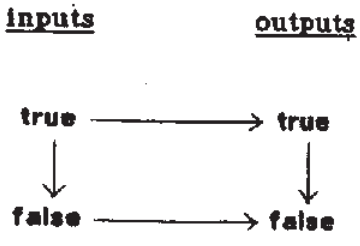
[14] Weng, K.-S., *Stream-Oriented Computation in Recursive Data Flow Schemas*, Laboratory for Computer Science (TM-68), MIT, Cambridge, Massachusetts, October 1975.

Appendix

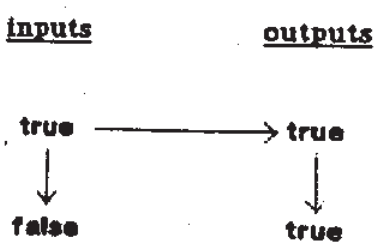
One possible representation of the complete behavior of a system is a collection of partially ordered sets (*posets*). The elements of these posets correspond to events, where an event is the receiving of an input token or the production of an output token. These posets correspond to the externally distinguishable "scenarios" of computations. History functions represent each possible computation by a pair of input history and possible output history. In a scenario the causality relation between the events of these histories is made explicit. (Note that this representation of a computation is similar to that found in actor semantics [6].) For example, the poset representing the response of either *S* or *T* to the single token input history (true) is characterized by the Hasse diagram:



The poset descriptions of systems *S* and *T* differ, however, for the input stream (true false). The characterization of *S* contains these posets:

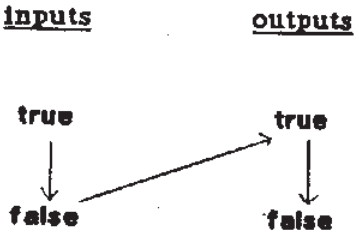


[S1]

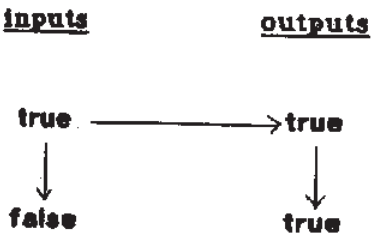


[S2]

while that for *T* contains:



[T1]



[T2]

The events on each port are totally ordered according to time, that is, there are arrows going downward from each event to the next along each column (port history) of the diagram. In a diagram corresponding to a computation, an arrow from an input symbol to an output symbol implies that the receipt of the input symbol must necessarily precede the production of the output symbol during that computation. In diagram T1, this means that the first output true will not be transmitted until after the second input false has been received, *in the scenario represented by that diagram.* If the first output is transmitted before the second input, the first diagram could not apply. Only the second could apply, and so the ultimate output must be (true true).

The composition rule for systems with this characterization is as follows:

(1) To interconnect two systems, enumerate all pairs of diagrams from the characterizations for the systems, that is, generate the Cartesian product of the characterizations.

(2) Discard all pairs whose port histories do not agree on the ports that are linked to each other.

(3) Combine each pair of diagrams into a single diagram, identifying the columns of linked ports. Condition (2) guarantees that all such identified columns are equal.

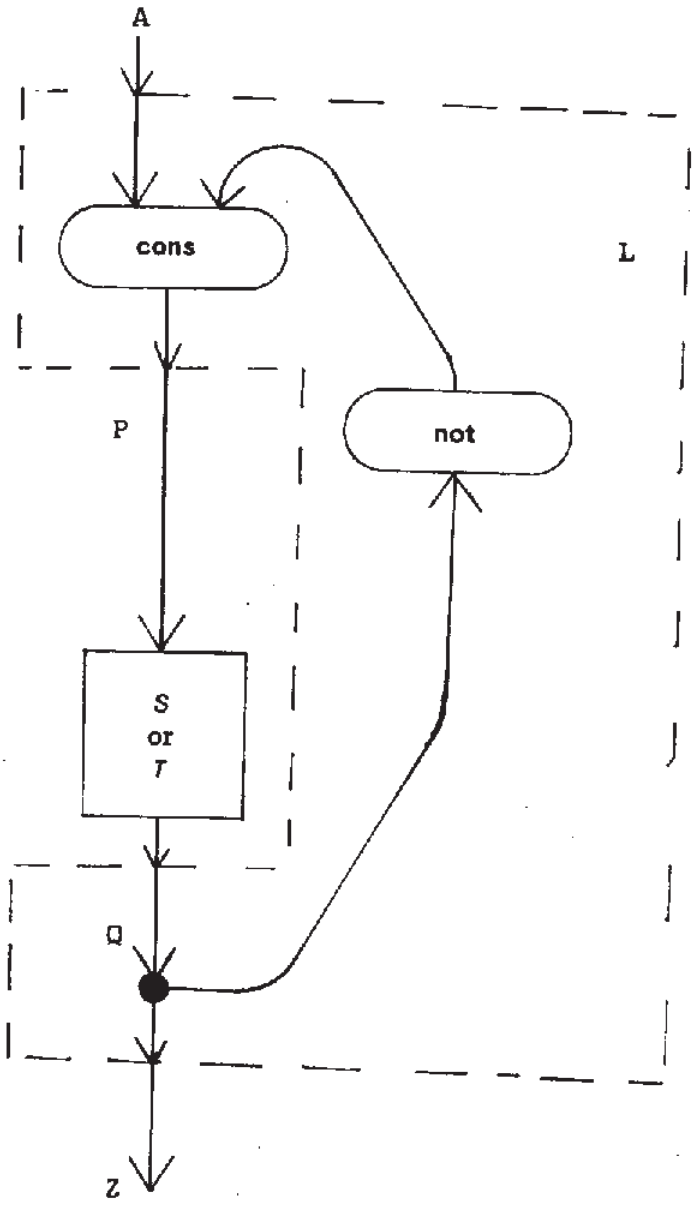
(4) Place the weakest partial order on the combined diagram which contains the component partial orders. If this is not possible (because the resultant order would not be antisymmetric), discard the diagram.

(5) Remove the columns for ports that were identified in step (3), since they are not ports of the interconnection. Preserve the partial order on the rest of the diagram. The resulting collection of diagrams characterizes the interconnection.

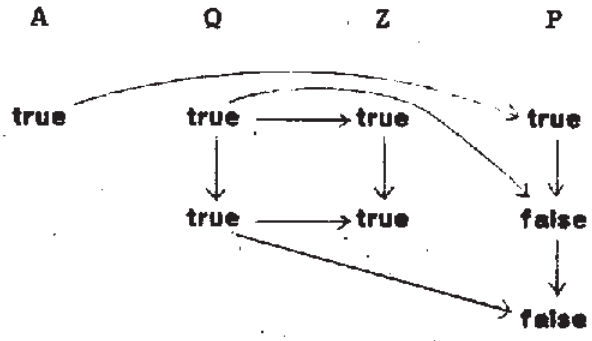
The diagrams given previously for systems *S* and *T* can be derived by application of these rules from the behavior of their elementary components.

Now consider the interconnection of system *S* or *T* with system *L*, as illustrated in figure 5:

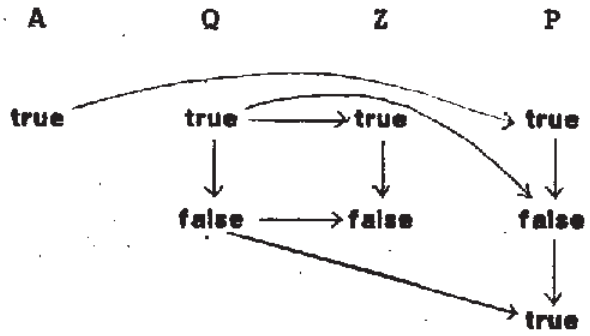
Figure 5



The relevant scenarios for *L* [that is, the scenarios with (true) on port A and (true true) or (true false) on port Q] are:



[L1]

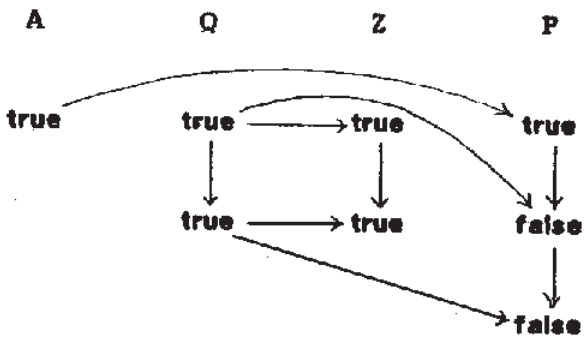


[L2]

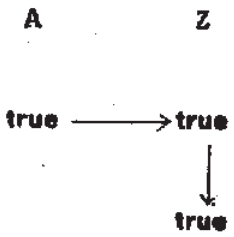
When system *S* or *T* is combined with *L*, the input port of *S* or *T* is identified with port P of *L*, and the output port with port Q.

When *L* is combined with *S*, graph L1 can only be combined with S2 in a consistent manner. [The L1-S1 combination disagrees on port Q.]

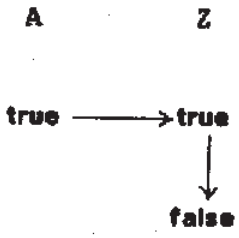
After merging columns, one has:



After removing ports P and Q, one has:

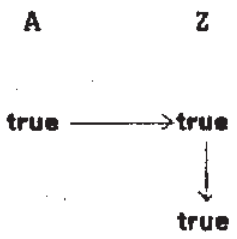


L2 can only be combined with S1. After merging columns and removing ports P and Q, one has:

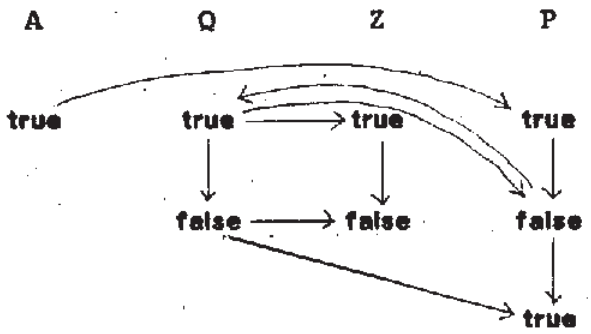


Therefore, the response of the interconnection of S and L to the input true is (true true) or (true false).

When systems T and L are combined, graph L1 can be combined with T2 just as it was combined with S2. Therefore, one scenario for the interconnection of T and L is:



When L2 is combined with T1, one obtains:



There are now arrows in both directions between the first symbol of column Q and the second of column P. This diagram must therefore be discarded, so the only scenario given earlier is the only possible one, and the only response of the interconnection of T and L to the input true is (true true).