The Relationship Between Multiway and Two-Way Fair Merges

Vasant Shanbhogue*

TR 90-1146
August 1990

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

*Research supported, in part, by an MSI fellowship.

The relationship between multiway and two-way fair merges

Vasant Shanbhogue*
Computer Science Department
Cornell University

Abstract

In the context of communicating systems of autonomous processes, we study two kinds of processes – a two-way fair merge, that interleaves two possibly infinite sequences into one, and a multiway fair merge, that interleaves more than two possibly infinite sequences into one. We describe two constructions. The first one shows how the effect of any arbitrary number of two-way fair merges in a finite network can be exactly obtained by a single multiway fair merge and some determinate processes. The second one shows how the effect of a single multiway fair merge with any number of input channels can be exactly obtained by a finite network of two-way fair merges.

1 Introduction

The study of indeterminate computing systems is motivated by seeking to understand the theory underlying what is commonly called "systems" programming. A feature that is commonly used in such programs is the "time-out," and a primitive that is usually used to model this phenomenon is *fair merge*.

Our study is carried out in the setting of dataflow networks. A dataflow network is a directed graph with autonomous computing agents at all the nodes, such that communication in the network takes place by message tranmission along the direction of an arc. Each arc behaves like a unidirectional queue of unbounded

*Research supported in part by an MSI fellowship

size, and message transmission takes place in a FIFO (first in first out) manner with an unbounded but finite delay. There may be additional arcs directed into the network or directed out of the network. These are called *input arcs* and *output arcs* respectively. The environment can communicate messages along the input arcs to computing agents in the network as well as receive messages along the output arcs from computing agents in the network. The mode of communication along any arc will be asynchronous — the sender of a message and its receiver do not have to synchronize for a message transmission. We will also refer to the arcs of a dataflow network as *channels*. The interconnection structure of the nodes is fixed throughout execution.

A two-way fair merge primitive has two input channels and one output channel. It reads two sequences of values, one on each input channel, and outputs an interleaving of them. The primitive is guaranteed to read and output both the input sequences of values entirely. A multiway fair merge primitive behaves similarily, except that it has more than two input channels, and it reads sequences of values, one on each input channel, and outputs an interleaving of all these sequences. In this paper, we will show that in any dataflow network, we can get the effect of many two-way fair merges by a single multiway fair merge process. The number of input channels of this multiway fair merge will depend on the number of two-way merges in the network. We will also show that the effect of a single multiway fair merge with any number of input channels can be obtained by many two-way fair merges.

2 Trace set semantics of processes

An operational semantics for dataflow processes and networks has been described by Stark [6]. When used to describe determinate processes, Lynch and Stark showed how this semantics satisfies Kahn's Principle [1,2], which is now universally accepted as a good criterion for the appropriateness of an operational semantics. The trace set semantics arises as an abstraction of this operational semantics. The details of how this abstraction is made may be found in [3,5]. We will describe here the trace set semantics without going into the operational details, but the reader should bear in mind that the operational details underly all the facts and theorems that we claim for traces.

Each process has a set of input channels along which it can receive values, and a set of output channels, along which it can send values. The set of input channels and the set of output channels will be disjoint. We assume that the values that can be communicated are from a fixed set V, that include at least the natural numbers. We will use the notation V^∞ for the set of all finite and infinite sequences of values from V.

Definition 1. An **event** of a process with a set of input channels I and a set of output channels O is a pair (p, v) such that p is a channel in I or O, and v is a value in V. If p is in I, then the event is called an **input event**, and if p is in O, then the event is called an **output event**.

For any pair $r = (p, v)$, we will refer to the value component v by $\mathit{value}(r)$. For any sequence $s = e_1, e_2, \ldots$ of events, we will also refer to the sequence $\mathit{value}(e_1)$, $\mathit{value}(e_2), \ldots$ of value components of events in s by $\mathit{value}(s)$.

We will denote a process by a set of sequences of its events. For this paper, we will assume that a particular set that denotes a process also defines the process. We will call this the **trace set** of the process, and we will represent the trace set of a process P by $\mathit{Trset}(P)$.

Definition 2. A **trace** of a process is a finite or infinite sequence of events of the process, such that this sequence is in the set of sequences defining the process.

If C is a subset of the set of input and output channels of a process and t is a trace of the process, then we will use the notation $\Pi_C(t)$ to represent the subsequence of t consisting of the events at channels in C. If p is a particular input or output channel of the process, then we will use the notation $\Pi_p(t)$ instead of $\Pi_{\{p\}}(t)$.

Finite networks of processes may be built using two operations – aggregation and feedback. Aggregation involves keeping two networks with disjoint sets of channels "side by side," thus giving us a network with two subnetworks with disjoint sets of channels. Feedback involves identifying an output channel o and an input channel i of a network to obtain a new network with one fewer input channel and one fewer output channel – the identified channels get "hidden" from the external environment. We will make a stipulation here that the channels i and o being identified not be channels of the same process. This does not restrict us in any way, because we can always get the effect of connecting channels of the same process by connecting o to the input channel of an identity process, that simply reads all values on its input channel and outputs all of them, and connecting the output channel of the identity process to i.

The trace set semantics is compositional, and the following fact states how the trace set of a network may be described in terms of the traces of the individual processes in the network. This follows as a theorem from the operational details [3, 5], but we may take it as the definition for trace sets of networks here.

Fact 1. The trace set of a network N with input channels I and output channels O is the set of sequences t in $((I \cup O) \times V)^*$ such that there is a sequence t' such that

- $\Pi_{I \cup O}(t') = t$, and
- for every process P in the network with input and output channels C, $\Pi_C(t')$ is a trace of P.

We can now prove that trace set equality is a congruence with respect to all well-defined network contexts. Informally, a network context is a network with a hole in it, with the hole being associated with two disjoint sets of ports I and O and it is legal to substitute a network into the hole if the network has I as its set of input ports, and O as its set of output ports. In fact,

Theorem 1. For any network context $\mathcal{C}[\,]$, if N_1 and N_2 are two networks with the same set of input channels, the same set of output channels, and the same trace set, and if the networks $\mathcal{C}[N_1]$ and $\mathcal{C}[N_2]$ are well-defined, then $\mathcal{C}[N_1]$ and $\mathcal{C}[N_2]$ have the same trace set.

Proof : Consider $\mathcal{C}[N_1]$ to be a network composed of the following subnetworks : N_1, and the rest of the network context, call it N_c. Suppose t is a trace of $\mathcal{C}[N_1]$. Then, by Fact 1, there is a sequence t' of events on the input and output ports of the subnetworks, such that the restriction of t' to any subnetwork is a trace of that subnetwork, and the restriction of t' to the input and output ports of the whole network is exactly t. Since N_1 and N_2 have the same set of traces by assumption, and we can consider $\mathcal{C}[N_2]$ to be composed of N_2 and N_c, just as we did for $\mathcal{C}[N_1]$, t' is also a sequence of events on the input and output ports of these subnetworks, such that the restriction to every subnetwork is a trace of that subnetwork. Then, using Fact 1 once more, we conclude that the restriction of t' to the input and output ports of $\mathcal{C}[N_2]$ is a trace of $\mathcal{C}[N_2]$. But this restriction is exactly t. Therefore t is a trace of $\mathcal{C}[N_2]$. Hence every trace of $\mathcal{C}[N_1]$ is a trace of $\mathcal{C}[N_2]$, and vice-versa.

3 Fair Merges

We use a description of fair merge, similar to the one used in [4], that says that one must be able to split the output sequence into disjoint subsequences, such that these subsequences are exactly the input sequences.

To describe the traces of the fair merges, we will first need some notation. If s is a finite or infinite sequence, and i is less than or equal to the length of s, then we will refer to the ith element of the sequence s by $s[i]$. Let ω represent the set of all non-negative integers, and ω_+ represent the set of all positive integers. If f is a function from ω_+ to a set S, $a \in S$ and s is a sequence, then let $\Pi_a^f(s)$ be the subsequence of s consisting of all the elements $s[i], i \leq \mathit{length}(s)$ such that $f(i) = a$.

We describe the trace set of a *multiway fair merge* process : If F is a multiway fair merge process with input channels $i_1, \ldots, i_m$ and output channel o, then the trace set of F consists of all sequences t in $(\{i_1, \ldots, i_m, o\} \times V)^\infty$ such that there is a function $f : \omega_+ \to \{1, \ldots m\}$ such that

- for $1 \leq j \leq m$, $\mathit{value}(\Pi_j^f(\Pi_o(t))) = \mathit{value}(\Pi_{i_j}(t))$, and
- for every prefix s of t, and for $1 \leq j \leq m$, $\mathit{value}(\Pi_j^f(\Pi_o(s)))$ is a prefix of $\mathit{value}(\Pi_{i_j}(s))$.

The description of a two-way fair merge follows from the above when $m = 2$.

Lemma 1. t is a trace of an m-way fair merge with input channels $i_1, \ldots, i_m$ and output channel o iff $\Pi_o(t)$ can be broken up into m subsequences s_j, such that $\mathit{value}(s_j) = \mathit{value}(\Pi_{i_j}(t))$, and for every $k > 0$, the event $s_j[k]$ is preceded by the event $\Pi_{i_j}(t)[k]$ in t.

Proof : If t is a trace of m-way fair merge, then let f be the function associated with t as in the definition of multiway fair merge. Then we define the jth subsequence s_j of $\Pi_o(t)$ as the subsequence of events $\Pi_o(t)[i]$ such that $f(i) = j$. Since the range of f is $\{1, \ldots m\}$, we obtain m subsequences $s_1, \ldots, s_m$. To prove that $s_j[k]$ is preceded by the event $\Pi_{i_j}(t)[k]$ in t, let $s_j[k]$ be the lth event in t. Then, by the property of multiway fair merge, $\mathit{value}(\Pi_j^f(\Pi_o(t[1..l])))$ is a prefix of $\mathit{value}(\Pi_{i_j}(t[1..l]))$. Since $s_j[k]$ is the last event in $t[1..l]$, this means that $\Pi_{i_j}(t)[k]$ must precede it in $t[1..l]$ and hence in t.

Now if t is a sequence of events on the channels of m-way fair merge, satisfying the conditions in the statement of the lemma, then we prove that t is a trace of m-way fair merge. To do this, we need to define an appropriate f from ω_+ to $\{1, \ldots m\}$.

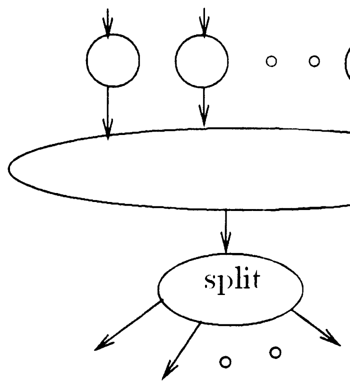

Figure 1: The network N_m

If the ith event in $\Pi_o(t)$ is in s_j, then we define $f(i) = j$. Since every event of $\Pi_o(t)$ is in one of the subsequences $s_1, \ldots, s_m$, we obtain a well-defined function. Then $value(\Pi_j^f(\Pi_o(t))) = value(s_j)$ from which it follows that f satisfies the conditions in the definition of multiway fair merge.

4 Simulation of many two-way fair merges by a single multiway fair merge

In this section, we will describe the main result – namely, that the effect of any number of two-way fair merges in a finite network can be obtained by "replacing" the two-way fair merges by a network containing only a single multiway fair merge and determinate processes. We will formally describe what we mean by "replacement."

Let N be a finite network containing k two-way fair merges. Let the input channels of the jth merge process be i_{2j-1} and i_{2j} and let the output channel be o_j, for $1 \leq j \leq k$. We will assume that all the channel names in $\{i_1, \ldots, i_{2k}, o_1, \ldots, o_k\}$ are distinct. We will consider the case when they are not distinct later.

Let N_2 be the subnetwork of N consisting of the k two-way merge processes, and with the set of input channels $\{i_1, \ldots, i_{2k}\}$ and the set of output channels $\{o_1, \ldots, o_k\}$. We will now describe a network N_m containing determinate processes and one multiway fair merge process with $2k$ input channels such that N_2 and N_m have the same sets of traces. The intuition behind N_m is that we will first "tag" the values on every channel $\{i_1, \ldots i_{2k}\}$ by appropriate identifiers that can identify which pairs of sequences are to be merged, and then we will use a $2k$-way fair merge process to merge these $2k$ tagged sequences. Finally, a determinate process will examine the tags of the elements of the merged sequence and then direct the elements to the appropriate channel among $\{o_1, \ldots o_k\}$ after stripping the tags off.

If s is a sequence $v_1, v_2, \ldots$ and n is an integer, then we will use the notation $\{n\} \times s$ to represent the sequence $(n, v_1), (n, v_2), \ldots$. We will also use the notation $\Pi_j(s)$ for the subsequence of s consisting of the events with value of the form (j, v).

The network N_m consists of the following processes :

- a multiway fair merge process with input channels $i'_1, \ldots i'_{2k}$ and output channel o, and this set of channel names is disjoint from the channel names in N.

- $2k$ "tag" processes tag-1,... tag-$2k$, where the process tag-j, $1 \leq j \leq 2k$, has one input channel i_j and one output channel i'_j, and the trace set of this process consists of all sequences t in $(\{i_j, i'_j\} \times V)^\infty$ such that $value(\Pi_{i'_j}(t)) = \{(j+1) \text{ div } 2\} \times value(\Pi_{i_j}(t))$, and for every prefix s of t, $value(\Pi_{i'_j}(s))$ is a prefix of $\{(j+1) \text{ div } 2\} \times value(\Pi_{i_j}(s))$.

- a process *split* with input channel o and output channels $o_1, \ldots o_k$, and the trace set of this process consists of all the sequences t in $(\{o, o_1, \ldots o_k\} \times V)^\infty$ such that for $1 \leq j \leq k$, $\{j\} \times value(\Pi_{o_j}(t)) = value(\Pi_j(\Pi_o(t)))$, and for every prefix s of t, $\{j\} \times value(\Pi_{o_j}(s))$ is a prefix of $value(\Pi_j(\Pi_o(s)))$.

The network N_m has input channels $i_1, \ldots i_{2k}$ and output channels $o_1, \ldots o_k$. Let this set of input and output channels be called C. Let us also call the set of channels $\{i'_1, \ldots i'_{2k}, o\} \cup C$, which are all the channels in the processes described above, C'. We will now prove that N_2 and N_m have the same trace sets. First, let us describe the trace sets of N_2 and N_m in terms of its component processes.

Lemma 2. *Trset*(N_2) consists of all possible interleavings of k sequences, such that the jth sequence is a trace of the jth two-way merge in the network.

Proof : Straightforward by fact 1.

Lemma 3. *Trset*(N_m) consists of all sequences t in $(C \times V)^\infty$ such that there is a sequence t' in $(C' \times V)^\infty$ with

(i) $\Pi_{C}(t') = t$,

(ii) for $1 \le j \le 2k$, $\Pi_{\{i_j, i'_j\}}(t')$ is a trace of tag-j.

(iii) $\Pi_{\{i'_1, \dots i'_{2k}, o\}}(t')$ is a trace of the multiway fair merge in N_m, and

(iv) $\Pi_{\{o, o_1, \dots o_k\}}(t')$ is a trace of *split*.

Proof : Follows from fact 1.

Theorem 2. $Trset(N_2) = Trset(N_m)$.

Proof : Let t be a trace of N_2. For every event (i_j, v) in t, we introduce a new event $(i'_j, ((j+1) \text{ div } 2, v))$ immediately after (i_j, v), and we introduce a new event $(o, (j, v))$ immediately before (o_j, v). Let the sequence thus obtained be called t'. We claim that t' satisfies the conditions of Lemma 3, and hence $t = \Pi_{C}(t')$ is a trace of N_m.

Conditions (i),(ii) and (iv) of Lemma 3 clearly follow by the construction of t' from t. We now prove condition (iii) of lemma 3, $t'' = \Pi_{\{i'_1, \dots i'_{2k}, o\}}(t')$ is a trace of a $2k$-way fair merge.

From the definition of *split*, it follows that $\Pi_o(t')$ can be broken up into k subsequences $t_1, \dots t_k$, such that $t_j = value(\Pi_j(\Pi_o(t'))) = \{j\} \times value(\Pi_{o_j}(t'))$. By lemma 1, $\Pi_{o_j}(t')$ can be broken up into subsequences s'_{2j-1} and s'_{2j} such that $value(s'_{2j-1}) = value(\Pi_{i_{2j-1}}(t'))$,and $value(s'_{2j}) = value(\Pi_{i_{2j}}(t'))$. Therefore t_j can also be broken up into two subsequences s_{2j-1} and s_{2j}, such that $value(s_{2j-1}) = \{j\} \times value(s'_{2j-1})$, which is equal to $\{j\} \times value(\Pi_{i_{2j-1}}(t')) = value(\Pi_{i'_{2j-1}}(t'))$, and also $value(s_{2j})$ is equal to $\{j\} \times value(s'_{2j}) = \{j\} \times value(\Pi_{i_{2j}}(t')) = value(\Pi_{i'_{2j}}(t'))$. The other condition on prefixes of t follows similarily. Therefore, from the subsequences $t_1, \dots t_k$, we obtain $2k$ subsequences $s_1, \dots s_{2k}$, satisfying the appropriate conditions.

Now, let t be a trace of N_m, and we would like to prove that t is a trace of N_2. By Lemma 3, there is a sequence $t' \in (C' \times V)^\infty$ satisfying the conditions of the lemma. Then $\Pi_o(t')$ can be broken up into $2k$ subsequences $s_1, \dots s_{2k}$ satisfying $value(s_{2j-1}) = value(\Pi_{i'_{2j-1}}(t'))$, and $value(s_{2j}) = value(\Pi_{i'_{2j}}(t'))$. Since all the events in s_{2j-1} and s_{2j} have values of the form (j, v) and no other events in $\Pi_o(t')$ have values of this form, we can, by the definition of *split*, break up $\Pi_{o_j}(t')$ into two subsequences t_{2j-1} and t_{2j}, such that $\{j\} \times value(t_{2j-1}) = value(s_{2j-1}) = value(\Pi_{i'_{2j}}(t'))$ $= \{j\} \times value(\Pi_{i_{2j-1}}(t'))$. Therefore $value(t_{2j-1}) = value(\Pi_{i_{2j-1}}(t'))$, and similarily,

$value(t_{2j}) = value(\Pi_{i_{2j}}(t'))$. The other condition on prefixes of t follows similarily, and so t can be broken up into k traces of k two-way fair merges, and hence is a trace of N_2.

Theorem 3. For any finite network N containing k two-way fair merge processes and determinate processes, there is a finite network with the same trace set containing only determinate processes and a single multiway fair merge process.

Proof : Let N be a finite network containing k two-way fair merges. Some of the input channels of the merges may also be output channels of the merges. Let the input channels of the jth merge process be i_{2j-1} and i_{2j}, possibly renaming the input channels to be distinct from any of the output channels of the merges, and let the output channel be o_j, for $1 \leq j \leq k$. Let N_2 be the network consisting of these k merges. Then, by previous theorem, there is a finite network N_m with input channels $i_1, \ldots, i_{2k}$ and output channels $o_1, \ldots, o_k$ and with the same trace set as N_2 and containing only determinate processes and a single multiway fair merge process. Now, if some of the input channels of the two-way merges are also output channels of the merges in N, then let us form the appropriate feedback loops in both N_2 and in N_m to obtain the networks N_2' and N_m' respectively. We can define the context $\mathcal{C}[\,]$ to be the network N with the two-way fair merge processes removed from the network, so that if N_2' constitutes the subnetwork of N consisting of the merge processes, then $\mathcal{C}[N_2']$ is N. By the compositionality of trace set semantics (Fact 1), $Trset(\mathcal{C}[N_2']) = Trset(\mathcal{C}[N_m'])$, and therefore $\mathcal{C}[N_m']$ is the finite network as desired.

5 Simulation of a multiway fair merge by many two-way fair merge processes

A multiway fair merge with k input channels can be simulated by a network of $k-1$ two-way fair merges.

We use induction on the number j of input channels to describe the construction and prove that it works. Let N_j represent the network of two-way fair merges that is supposed to simulate a multiway fair merge with j input channels $i_1, \ldots, i_j$ and one output channel o_j. We define N_2 to simply be a two-way fair merge with input channels i_1, i_2 and output channel o_2. Once we have defined N_{k-1}, N_k is defined (see

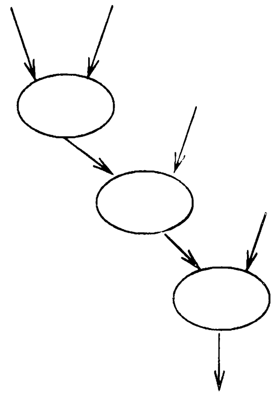

Figure 2: Simulating a four-way fair merge

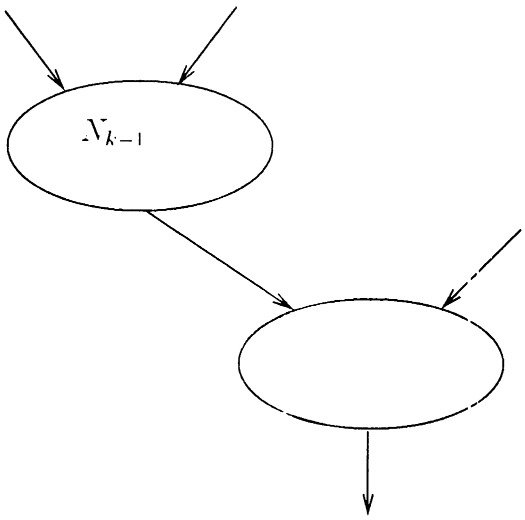

Figure 3: Constructing a k-way fair merge from a $(k-1)$-way fair merge

Figure 3) as the network containing N_{k-1} as a subnetwork, together with a two-way fair merge process with input channels o_{k-1} and i_k and output channel o_k.

Theorem 4. *$Trset(N_j)$ is the trace set of a multiway fair merge M with the same input and output channels as N_j.*

Proof : This is obvious for $j = 2$. Suppose we have proved the statement for $j = k - 1$. Let t be a trace of M. For every input event (i_l, v) for $l < k$, we introduce a new event (o_{k-1}, v) immediately after it. We can break $\Pi_{o_k}(t')$ into k subsequences $s_1, \ldots, s_k$, such that $value(s_l) = value(\Pi_{i_l}(t'))$. By construction of t', there is a bijection between events on $i_1, \ldots, i_{k-1}$ and events on o_{k-1}, and so we can break $\Pi_{o_{k-1}}(t')$ into $k-1$ subsequences $t_1, \ldots, t_{k-1}$ satisfying the requirements for $k-1$-way fair merge. Then the subsequence s'_{k-1} of $\Pi_{o_k}(t')$ consisting of all the events in $s_1, \ldots, s_{k-1}$ has the same value sequence as $\Pi_{o_{k-1}}(t')$. It is also clear by construction that every event of s'_{k-1} is preceded by the corresponding event e in $\Pi_{o_{k-1}}(t')$, since it is preceded by the corresponding event e' in one of $\Pi_{i_1}(t'), \ldots \Pi_{i_{k-1}}(t')$, and e, e' are adjacent by construction. Therefore the conditions of Fact 1 are satisfied, and t is a trace of N_k.

Let t be a trace of N_k. Then, by Fact 1, there is a sequence t' of events on the channels of N_{k-1} and the two-way fair merge, such that its projection to N_{k-1} is a trace of N_{k-1} and its projection to the two-way fair merge is a trace of that process. Therefore, we can break up $\Pi_{o_k}(t')$ into two subsequences s'_{k-1} and s_k such that the value sequence in s_k is the same as that in $\Pi_{i_k}(t')$, and the value sequence in s'_{k-1} is the same as that in $\Pi_{o_{k-1}}(t')$, which can be divided into $k-1$ subsequences according to the definition of k-way fair merge. Therefore s'_{k-1} can be broken up into $k-1$ subsequences $s_1, \ldots, s_{k-1}$ exactly as $\Pi_{o_{k-1}}(t')$ was. Then it follows that the k subsequences $s_1, \ldots, s_k$ of $\Pi_{o_k}(t') = \Pi_{o_k}(t)$ satisfy the requirements for t to be a trace of k-way fair merge M.

6 Conclusions

In this paper, we have shown that in the context of static dataflow networks, the power of many two-way fair merges can be obtained by a single multiway fair merge, and vice-versa. This clarifies the expressiveness situation between fair merges with two input channels and fair merges with more than two input channels.

References

[1] G. Kahn. The semantics of a simple language for parallel programming. In *Information Processing 74*, pages 993–998. North-Holland, 1974.

[2] N. A. Lynch and E. W. Stark. A proof of the kahn principle for input/output automata. *Information and Computation*, 1989.

[3] D. McAllester, P. Panangaden, and V. Shanbhogue. Nonexpressibilty of fairness and signaling. In *Proceedings of the 29th Annual Symposium of Foundations of Computer Science*, 1988. Also TR 89-972, Department of Computer Science, Cornell University.

[4] V. Nguyen and R. Strom. Process semantics: global axioms, compositional rules and applications. In *Seventh ACM Symposium on Principles of Distributed Computing*, 1988.

[5] P. Panangaden and V. Shanbhogue. Traces are fully abstract for networks with fair merge. unpublished manuscript, 1989.

[6] E. W. Stark. Connections between a concrete and an abstract model of concurrent systems. In *Proceedings of the 1989 Conference on Mathematical Foundations of Programming Language Semantics*, 1989.