A Fully Abstract Trace Model for Dataflow Networks

Bengt Jonsson
Swedish Institute of Computer Science
Box 1263, S-164 28 Kista, SWEDEN
bengt@sics.se

Abstract: A dataflow network consists of nodes that communicate over perfect FIFO channels. For dataflow networks containing only deterministic nodes, a simple and elegant semantic model has been presented by Kahn. However, for nondeterministic networks, the straight-forward generalization of Kahn's model is not compositional. We present a compositional model for nondeterministic networks, which is fully abstract, i.e., it has added the least amount of extra information to Kahn's model which is necessary for attaining compositionality. The model is based on traces.

1 Introduction

Dataflow is a paradigm for asynchronous parallel computation, in which data "flows" between nodes that are interconnected by channels into a dataflow network.

We are concerned with semantic models of dataflow networks. A model of dataflow networks is a mathematical description of their behavior. Two desiderata for such a model are: (1) the model should describe only the externally observable behavior of a network, e.g. as manifested by data flowing in to or out from the network, and (2) the model should support modular descriptions of dataflow networks: if a large network is constructed by composing smaller component networks, then the denotation of the composed network should depend only on the denotations of its components. This last property is often called "compositionality".

For dataflow networks with only deterministic processes, Kahn [Kah74] has proposed an elegant semantic model, which satisfies both of the above desiderata. Kahn models a network by a function from sequences of data items on input channels to sequences of data items on output channels.

*This work was supported in part by the Swedish Board for Technical Development (STU) under contract no. 86-4250.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specfic permission.

© ACM 1989 0-89791-294-2/89/0001/0155 $1.50

For nondeterministic networks, the straight-forward generalization of Kahn's model would be to model a network by a relation rather than a function between sequences of data items on its channels. A sequence of data items that appears on a channel is often called a *history*, and we will therefore refer to this model as the *history model*. Unfortunately, for nondeterministic networks the history model fails to satisfy desideratum (2). In other words, the denotations in the history model of two networks, N_1 and N_2, do not contain sufficient information to infer the denotation of the network which is composed of N_1 and N_2. This was shown by Brock and Ackerman [BA81].

To obtain a compositional model for nondeterministic networks, the history model could be refined to a model containing more information about networks. Desideratum (1) suggests that the model should not contain details that are irrelevant for this purpose. A model with this property is called *fully abstract*. Intuitively, a model $\mathcal{D}$ is fully abstract with respect to a model $\mathcal{O}$ if $\mathcal{D}$ has added precisely enough information to $\mathcal{O}$ to attain compositionality. For modular verification methods, where the verification of a network can be split into independent verifications of its components, a fully abstract model indicates what aspects of a network's behavior must be described.

The paper by Brock and Ackerman [BA81] shows why the history model is not compositional for nondeterministic networks. Two nondeterministic networks that are equivalent in the history model may exhibit different behaviors if one introduces constraints on the order in which input is supplied and output appears (e.g. that one input is only supplied after some output appears). Such ordering constraints can be introduced by composing two networks, and therefore the denotation of the larger network is not derivable from the denotations of its components.

In this paper, we present a model of nondeterministic dataflow networks, which is fully abstract with respect to the history model. In order to obtain a compositional model, we must to the history model add information about how a network behaves under different ordering constraints. Our model provides precisely this information. We denote a network by the set of its traces. A trace is a linearly ordered sequence of communication events that may appear on its input and output channels during a computation. A communication event represents the appearance of a data item on a certain channel.

In the literature, many compositional models have been proposed [BM85, Bou82, BA81, Bro83, Bro88, Kok86, Kos78, KP85, KP86, Par83, Pra82, Pra84, SN85], which are not fully abstract. Another fully abstract model has been presented by Kok [Kok87]. In this paper, we include a comparison with the model of Kok. We argue that our model uses less complicated concepts and allows a more natural proof of full abstraction.

Our model is similar to a model presented by Misra and Chandy [Mis84], in our earlier work [Jon85, Jon87], and by Lynch and Tuttle [LT87]. These models are defined for a model of distributed systems, called I/O-automaton in [LT87] and I/O-system in [Sta84, Jon87]. Our formal definition of a dataflow network will in fact be a special case of an I/O-automaton (I/O-system). The results of this paper appear in the context of I/O-systems in the author's thesis [Jon87].

This paper is organized as follows: In the next section, we give the basic definitions of dataflow networks. In section 3, we define our trace model and the history model. In section 4, we prove that our model is compositional. In section 5, we state and prove the main theorem: our model is fully abstract with respect to the history model. Section 6 contains a comparison of our model with the fully abstract model presented by Kok. Section 7 contains a comparison with other related models, and the last section contains conclusions.

2 Dataflow Networks

In this section, we define dataflow networks. In subsection 2.1, we give an informal presentation, which is formalized in section 2.2 by using labeled transition systems.

2.1 Informal Presentation

A *dataflow network* is a set of *nodes* connected by directed *channels*. Each channel is distinctly named. The nodes communicate with each other and with the environment by passing *data items* over the channels. The channels are of three different types:

input channels transmit data items from the environment to a node.

output channels transmit data items from a node to the environment.

internal channels transmit data items from a node to another node of the network.

At any step of its execution, a node can poll its incoming channels for presence of data items, consume data items from incoming channels, perform internal computations (change its state), and produce data items on outgoing channels.

The channels of a network behave like perfect, unbounded FIFO queues. That is, data items sent over a channel are delivered in unchanged order, after a finite unspecified delay. Note that this also applies to the input and output channels of the network.

Larger networks can be built by *composition* of smaller networks. Given networks $N_1, \ldots, N_k$, where each channel name occurs at most once as an input channel and at most once as an output channel, a composite network is obtained by connecting input channels to output channels with the same name. The resulting network can also be viewed as a node whose input and output channels are those that were not connected. Figure 1 shows how two networks, N_1 and N_2, are composed to yield a network with input channel a and output channels b and d.

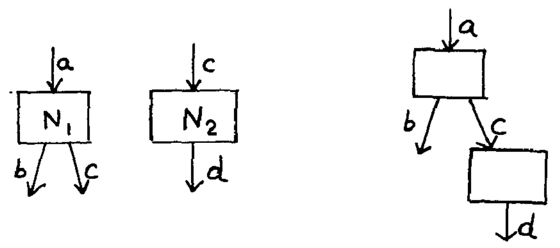

Figure 1: Two networks, N_1 and N_2 (left), and their composition (right).

2.2 Formal Presentation

In this subsection, we give a formal definition of dataflow networks by using labeled transition systems. A labeled transition system has a state, which can be changed by transitions. Transitions can have labels that represent the communication of data values over channels. Labeled transition systems is a general framework, which have often been used for operational descriptions of computing systems, (e.g. [Plo81, MP81]).

We first give a formal definition of individual nodes. Thereafter we define a network in terms of its nodes. It should be noted that we use labeled transition systems as a means for obtaining a formal definition of dataflow networks, upon which we can build subsequent definitions and proofs. The details are a formal counterpart of the informal presentation in subsection 2.1.

We assume a set V of *data items*, ranged over by d. Let V^* denote the set of finite sequences of data items in V, ranged over by q. The sequence consisting of the elements $d_1, \ldots, d_n$ is denoted $\langle d_1, \ldots, d_n \rangle$. The empty sequence is denoted by $\langle\rangle$. The concatenation of two sequences q and q' is denoted by $q.q'$. We assume a set of *channels*, ranged over by c, in, and out.

Definition 2.1 A *node* p is a tuple $\langle I_p, O_p, S_p, s_p^0, R_p, \mathcal{F}_p \rangle$ where

I_p is a set of channels, called the set of *incoming channels.*

O_p is a set of channels, called the set of *outgoing channels*, with $I_p \cap O_p = \emptyset$.

S_p is a set, called the set of *states*

s_p^0 is an *initial state*, with $s_p^0 \in S_p$.

R_p is a set of *firings*. A firing is a tuple $\langle s, \chi_{in}, s', \chi_{out} \rangle$ where $s, s' \in S_p$, where χ_{in} is a mapping from I_p to V^*, and where χ_{out} is a mapping from O_p to V^*.

$\mathcal{F}_p \subseteq \mathcal{P}(R_p)$ is a finite collection of *fairness sets*. Each fairness set F is a subset of the set R_p of firings, subject to the following constraint: for each channel $in \in I_p$, if F contains one firing $\langle s, \chi_{in}, s', \chi_{out} \rangle$ which satisfies the property that $\chi_{in}(in) \neq \langle\rangle$, then F contains all firings in R_p with that property.

□

The intuitive meaning of a firing $\langle s, \chi_{in}, s', \chi_{out} \rangle$ is: "when the node is in state s and the contents of each incoming channel in starts with the sequence $\chi_{in}(in)$, then these sequences may be consumed, while the node changes its state to s' and the sequence $\chi_{out}(out)$ is produced on each outgoing channel out". A fairness set represents a set of firings which may not be neglected indefinitely in executions of a network where the node occurs. For instance, if $\mathcal{F}_p = \{R_p\}$, i.e., the set of all firings is one fairness set, then the node will continue to perform firings indefinitely, until no more firings are possible (e.g. for lack of input). The constraint on fairness sets is a technical requirement which excludes certain pathological fairness sets.

We use the following notation for mappings. If $\{x_1, \ldots, x_n\}$ is a set of elements, we use $[x_1 \mapsto e_1, \ldots, x_n \mapsto e_n]$ to denote the mapping from $\{x_1, \ldots, x_n\}$ that maps x_i to e_i for $i = 1, \ldots, n$. If σ is a mapping we use $\sigma[x_1 \mapsto e_1, \ldots, x_n \mapsto e_n]$ to denote the mapping which is equal to σ except that it maps x_i to e_i for $i = 1, \ldots, n$.

Example 2.2 Consider a node *Fairmerge*. Intuitively, it consumes data items from the channels in_1 and in_2 and produces them onto out. The node is "fair", i.e. it never neglects any incoming channel indefinitely. The set of incoming channels is $\{in_1, in_2\}$ and the set of outgoing channels is $\{out\}$. The node has one state, denoted s, which then of course is the initial state. The set of firings is the union of the set

$$R1 = \{\langle s, [in_1 \mapsto \langle d \rangle \;,\; in_2 \mapsto \langle\rangle] \,,\, s \,,\, [out \mapsto \langle d \rangle] \rangle \mid d \in D\}$$

of firings that consume a data item from in_1 and produces it on out, and the set

$$R2 = \{\langle s, [in_1 \mapsto \langle\rangle \;,\; in_2 \mapsto \langle d \rangle] \,,\, s \,,\, [out \mapsto \langle d \rangle] \rangle \mid d \in D\}$$

of firings that consume a data item from in_2 and produces it on out. By postulating two fairness sets, $R1$ and $R2$, we ensure that the merge will be fair. □

We can now define a network N which consists of a set P of nodes, by using labeled transition systems. The definition is a formalization of the intuition that in a network, each node behaves according to its definition, and each channel behaves like an unbounded FIFO channel.

Definition 2.3 A *dataflow network* N consists of a set P_N of nodes, such that in P_N, each channel occurs at most once as an incoming channel and at most once as an outgoing channel.

If N is a dataflow network consisting of the set P_N of nodes, we make the following definitions:

I_N $= (\bigcup_{p \in P} I_p) \setminus (\bigcup_{p \in P} O_p)$ is the set of *input channels* of N.

O_N $= (\bigcup_{p \in P} O_p) \setminus (\bigcup_{p \in P} I_p)$ is the set of *output channels* of N.

E_N $= I_N \cup O_N$ is the set of *external channels* of N.

C_N $= (\bigcup_{p \in P} I_p) \cup (\bigcup_{p \in P} O_p)$ is the set of *channels* of N.

Σ_N is the set of mappings from $P_N \cup C_N$, which map each node p to a state in S_p and each channel in C_N to a sequence in V^*. Such a mapping is called a *state* of N. Intuitively, for a state $\sigma \in \Sigma_N$, the state $\sigma(p)$ gives the current state of node p, and $\sigma(c)$ gives the current content of channel c.

σ_N^0 $\in \Sigma_N$ maps each node p to s_p^0, and maps each channel c to the empty sequence $\langle\rangle$.

A *communication event* of N is a pair $\langle c, d \rangle$, where c is an external channel in E_N, and d is a data item in V.

R_N is a set of *transitions*, i.e., triples of the form $\sigma \xrightarrow{e} \sigma'$, where $\sigma, \sigma' \in \Sigma_N$ and e is either a communication event of N or the silent event τ. The set R_N consists of all transitions that can be generated according to one of the following three alternatives:

1. R_N contains a transition $\sigma \xrightarrow{\tau} \sigma'$ (called an internal transition) if there is a node $p \in P_N$ with a firing $\langle s, \chi_{in}, s', \chi_{out} \rangle$ such that
   - σ satisfies $\sigma(p) = s$ and for each incoming channel $in \in I_p$ to p, the sequence $\chi_{in}(in)$ is a prefix of $\sigma(in)$.
   - σ' agrees with σ on nodes except p and channels not in $I_p \cup O_p$. The state σ' differs from σ exactly in that $\sigma(p) = s'$, that for each $in \in I_p$ the initial sequence $\chi_{in}(in)$ has been removed from the sequence $\sigma(in)$ to get $\sigma'(in)$, and that for each $out \in O_p$ we have $\sigma'(out) = \sigma(out).\chi_{out}(out)$.
2. For each input channel $in \in I_N$ and data item $d \in V$, the set R_N contains a transition of the form $\sigma \xrightarrow{\langle in,d \rangle} \sigma'$ (called an input transition) such that $\sigma' = \sigma[in \mapsto (\sigma(in).d)]$, i.e., σ' is the same state as σ except that d has been added at the end of in.
3. For each output channel $out \in O_N$, R_N contains a transition of the form $\sigma \xrightarrow{\langle out,d \rangle} \sigma'$ (called an output transition) such that $\sigma = \sigma'[out \mapsto (d.\sigma'(out))]$, i.e., d is the first data item in $\sigma(out)$ and σ' is the

same state as σ except that d has been removed from the front of *out*.

$\mathcal{F}_N \subseteq \mathcal{P}(R_N)$ is called the set of *fairness sets.* $\mathcal{F}_N$ contains all sets of the following types:

1. For each node p and fairness set F in $\mathcal{F}_p$, there is a fairness set in $\mathcal{F}_N$ consisting of all internal transitions derived from a firing in F,
2. For each output channel $out \in O_N$, the set of transitions of form $\sigma \xrightarrow{\langle out,d\rangle} \sigma'$ in R_N constitute one fairness set.

□

Intuitively, a state in Σ_N gives the state of each node and the contents of each channel. The state can change in transitions, which are either (1) caused by a firing of a node, (2) caused by the arrival of a data item to an input channel, or (3) caused by the output of a data item from an output channel. In the two last cases, the channel and data item in question are recorded as a label on the transition. The fairness sets in $\mathcal{F}_N$ make sure that (1) the fairness requirements of each node are obeyed, and that (2) all data items in output channels are eventually output from the channel in a transition of form (3).

We finally define the *composition* operation, by which large networks can be built from components.

Definition 2.4 The dataflow networks $N_1, \ldots, N_k$ are called *compatible* if in $N_1, \ldots, N_k$ each channel name occurs at most once as an input channel and at most once as an output channel.

Given compatible networks $N_1, \ldots, N_k$, we define their *composition* $N_1 \| \ldots \| N_k$ as the network whose set of processes is the union $\bigcup_i P_{N_i}$ of the processes of $N_1, \ldots, N_k$. □

It follows that the input channels of $N_1 \| \ldots \| N_k$ are those input channels among $N_1, \ldots, N_k$ that to not also occur as output channels, and analogously for the output channels of $N_1 \| \ldots \| N_k$.

Remark: Some authors (e.g. [SN83, KP85, Kok87]) use slightly different operations to form larger networks from components. They define a *tupling* operation which puts networks in parallel without connecting any channels, and a *linking* operation which connects an input channel to an output channel of a network. For the purposes of this paper, this difference is not important. The tupling operation can be regarded as a special case of our composition operator, and the linking operation can be emulated by composing the network with a FIFO buffer that connects the two channels.

3 Models of Dataflow Networks

We can now define the set of traces and histories of a network.

Definition 3.1 A transition $\sigma_1 \xrightarrow{\tau} \sigma_2$ is *enabled* in a state σ if $\sigma = \sigma_1$. □

Definition 3.2 A *computation* of a dataflow network N is a finite or infinite sequence

$$\sigma^0 \xrightarrow{e^1} \sigma^1 \xrightarrow{e^2} \cdots \xrightarrow{e^n} \sigma^n \xrightarrow{e^{n+1}} \cdots$$

of transitions, which satisfies the following conditions:

1. σ^0 is the initial state of N.
2. Each triple $\sigma^{n-1} \xrightarrow{e^n} \sigma^n$ in the sequence is a transition in R_N.
3. For each fairness set $F \in \mathcal{F}_N$, if there is a σ^n in the computation such that a transition in F is enabled in all states σ^m of the computation with $m \geq n$, then there must be a transition $\sigma^m \xrightarrow{e^{m+1}} \sigma^{m+1} \in F$ with $m \geq n$ in the computation. Note that for finite computations this condition is equivalent to the condition that no transition from F is enabled in the last state.

□

Definition 3.3 Let Γ be a computation of the network N.

- the *history function* of Γ is a mapping from E_N which maps each external channel to the sequence of data items transmitted over that channel in Γ.
- the *trace* of Γ is the sequence of communication events in Γ. In general, a trace may be either finite or infinite.

We say that h is a history function of a network N if h is the history function of a computation of N. Similarly, we say that t is a trace of a network N if t is the trace of a computation of N. If N is a dataflow network, we define

H_N as the set of history functions of N

T_N as the set of traces of N.

Definition 3.4 Let N be a dataflow network.

- The *history model* $\mathcal{H}$ is defined as follows: for a dataflow network N, its denotation $\mathcal{H}(N)$ is the triple $\langle I_N, O_N, H_N \rangle$.
- The *trace model* $\mathcal{T}$ is defined as follows: for a dataflow network N, its denotation $\mathcal{T}(N)$ is the triple $\langle I_N, O_N, T_N \rangle$.

□

Intuitively, a computation is a complete run of the network, in which all nodes perform firings according to their definitions, and all channels behave like FIFO channels. A trace of a computation is the sequence of communication events that is exchanged with the environment over input and output channels. A history function of a computation gives for each input and output channel the sequence of data items that have been exchanged over that channel.

An important detail is that in each computation, all data items in an output channel are eventually output from that channel. This follows from the introduction of a fairness set for each output channel in definition 2.3. Without this requirement the trace model would not be compositional. The model would not be able to distinguish between a network that always produces a certain output and a network which sometimes produces this output and sometimes produces nothing (for the network that always produces output, the output may sometimes be left in the output channel).

Example 3.5 Consider a network N which consists of the only node *Fairmerge* with input channels in_1 and in_2 and output channel *out*.

Each history function h of N maps *out* to a sequence $h(out)$ which is obtained by merging the sequences $h(in_1)$ and $h(in_2)$. An example of a history function is the function h for which

$$h(in_1) = \langle 1, 3 \rangle$$

$$h(in_2) = \langle 2 \rangle$$

$$h(out) = \langle 1, 2, 3 \rangle$$

To describe the set of traces of t, let t_{in_1} denote the sequence of data items transmitted over the channel in_1 in t, and similarly for in_2 and *out*. A trace t of N is then a sequence of communication events on in_1, in_2, and *out*, such that

- t_{out} is obtained by merging t_{in_1} and t_{in_2}.
- For each prefix t' of t, the sequence t'_{out} is a prefix of some sequence that is obtained by merging t'_{in_1} and t'_{in_2}.

An example of a trace is the sequence

$$\langle \langle in_1, 1 \rangle, \langle in_2, 2 \rangle, \langle out, 1 \rangle, \langle in_1, 3 \rangle, \langle out, 2 \rangle, \langle out, 3 \rangle \rangle$$

□

4 Compositionality

As shown by Brock and Ackerman [BA81], the history model is not compositional for dataflow networks that contain non-deterministic nodes. They presented an example containing two networks with the same denotation in the history model. However, when composing the networks with another network, the composed networks had different denotations in the history model.

In this section, we prove that the trace model is indeed compositional. For a large network is the composition of smaller networks, we present an operation for obtaining the denotation of the large network from the denotations of the smaller networks.

Let us introduce some notation. If C is a set of channels, and t is a sequence of communication events, let $t\lceil_C$ (the restriction of t to C) denote the subsequence of t consisting of those communication events that occur on channels in C. Let $C^\dagger$ denote the set of finite and infinite sequences of communication events on channels in C.

Theorem 4.1 *Let $N_1, \ldots, N_k$ be compatible dataflow networks, and let N be their composition $N_1 \| \ldots \| N_k$. Let C_N denote $\bigcup_i E_{N_i}$. Then the denotation $\mathcal{T}(N)$ is the triple $\langle I_N, O_N, T_N \rangle$ where*

- $I_N = (\cup_i I_{N_i}) \setminus (\cup_i O_{N_i})$
- $O_N = (\cup_i O_{N_i}) \setminus (\cup_i I_{N_i})$
- $T_N = \{ t\lceil_{(I_N \cup O_N)} \mid t \in (C_N)^\dagger \text{ and } t\lceil_{(E_{N_i})} \in T_{N_i} \text{ for } i = 1, \ldots, k \}$

□

In other words, to obtain the traces of $N_1 \| \ldots \| N_k$, first form the "synchronized merges" of the traces of $N_1, \ldots, N_k$ (synchronizing on events on channels that are common to several N_i), then delete the communication events on those channels that are internal channels of $N_1 \| \ldots \| N_k$.

Proof Sketch: The proof for the sets I_N and O_N follows immediately from definition 2.4, by considering how channels are connected.

The proof for the set T_N follows from a related theorem for I/O-automata in [LT87] or I/O-systems in [Jon87]. A dataflow network, as defined in definition 2.3, can be regarded as an I/O-automaton or I/O-system. The composition M of $N_1, \ldots, N_k$ as I/O-automata is similar to their composition $N_1 \| \ldots \| N_k$ as dataflow networks, with the exceptions that (1) M will contain two copies of channels that connect one subnetwork with another, and (2) communication events on channels that connect subnetworks are still observable. For instance, the composition of the networks N_1 and N_2 in the left part of Figure 1 is not exactly the network in the right part of that figure, since it contains two serially connected copies of the channel c, and since communication events on c can be observed. If we define transitions, computations, etc. of M in the natural way, it can be proven that the traces of M are "synchronized merges" of the traces of $N_1, \ldots, N_k$, i.e., they are the sequences t such that $t\lceil_{(E_{N_i})} \in T_{N_i}$ for $i = 1, \ldots, k$. To complete the proof of theorem 4.1, we must prove that t is a trace of M iff $t\lceil_{(I_N \cup O_N)}$ is a trace of $N_1 \| \ldots \| N_k$. This follows from the following two properties

1. Computations of $N_1 \| \ldots \| N_k$ have no observable events on internal channels
2. The set of computations is not changed in an essential way if the two copies in M of channels that connect subnetworks are replaced by the single copy in $N_1 \| \ldots \| N_k$. To prove this property, we need the constraint on fairness sets in 2.1. □

Example 4.2 *An Example by Brock and Ackerman*

To illustrate the composition operator in the trace model, we shall briefly outline how the example of Brock-Ackerman is handled by the trace model. Here we use a version of the example described by Park [Par83].

We consider the networks N_1 and N_2, where N_i has nodes $Merge$, Buf_i, and channels a, b, and c. We shall later compose N_i with the network M which has nodes $Plus1$ and channels c and a. The structure of the networks is shown in Figure 2.

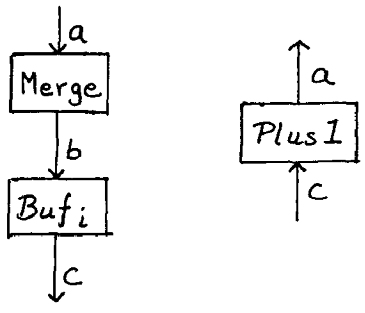

Figure 2: The Example by Brock and Ackerman.

The function of the nodes is intuitively the following

$Merge$ merges the first data item from a with the sequence $\langle 5,5\rangle$ and produces the result on b. Thus, depending on whether some data items arrive on a, three or two data items will be produced on b.

Buf_1 consumes a data item from b and produces it on c. Thereafter Buf_1 repeats this behavior once more before terminating.

Buf_2 first consumes *two* data items from b, produces them on c and thereafter terminates.

$Plus1$ adds one to the first incoming data item from c, produces it on a, and terminates

The networks N_1 and N_2 have the same denotation in the history model. If we for the moment only consider computations in which only the data item 6 arrives on a, then the possible history functions of N_i are those history functions h for which $h(a) = \langle 6\rangle$ and $h(c)$ is one of $\langle 5,5\rangle$, $\langle 5,6\rangle$, or $\langle 6,5\rangle$. However, if we compose N_i and M, and make the channel c external (for example by introducing a fan-out node) then we see that $\langle 5,6\rangle$ is a possible history on c of $N_1\|M$ but not of $N_2\|M$. The reason for this is that when the first data item 5 is produced by $N_2\|M$ on c, then the node Buf_2 has already consumed a second data item. This data item must be a 5, since a 6 has not year appeared on channel a. Therefore the second data item on c must also be a 5.

In the trace model, the networks N_1 and N_2 have different denotations. Again only considering computations in which the data item 6 arrives on a, the possible traces of N_2 are

$$\begin{array}{ll}\langle\langle a,6\rangle,\langle c,5\rangle,\langle c,6\rangle\rangle & \langle\langle c,5\rangle,\langle a,6\rangle,\langle c,5\rangle\rangle\\ \langle\langle a,6\rangle,\langle c,5\rangle,\langle c,5\rangle\rangle & \langle\langle c,5\rangle,\langle c,5\rangle,\langle a,6\rangle\rangle\\ \langle\langle a,6\rangle,\langle c,6\rangle,\langle c,5\rangle\rangle & \end{array}$$

However, the network N_1 has in addition the trace

$$\langle\langle c,5\rangle,\langle a,6\rangle,\langle c,6\rangle\rangle$$

Note that this is not a trace of N_2, since when the communication event $\langle c,5\rangle$ occurs, the node Buf_2 has already consumed a second data item, which must be a 5.

The traces of M which match the traces of N_1 and N_2 discussed above are

$$\begin{array}{ll}\langle\langle c,5\rangle,\langle a,6\rangle,\langle c,5\rangle\rangle & \langle\langle c,5\rangle,\langle a,6\rangle,\langle c,6\rangle\rangle\\ \langle\langle c,5\rangle,\langle c,5\rangle,\langle a,6\rangle\rangle & \end{array}$$

If we use the composition operator in the trace model, we see that the only trace of $N_2\|M$ (again considering only those with a 6 on a) is $\langle\langle c,5\rangle,\langle c,5\rangle\rangle$ whereas the network $N_1\|M$ has the traces $\langle\langle c,5\rangle,\langle c,5\rangle\rangle$ and $\langle\langle c,5\rangle,\langle c,6\rangle\rangle$.

5 Full Abstraction

In this section, we present the main result of the paper: the trace model is fully abstract with respect to the history model. In other words, it contains the minimal amount of extra information necessary to attain compositionality.

Let us introduce some terminology. By a model (of dataflow networks) we shall mean a mapping from the set of dataflow networks to some set. By a context $C[\cdot]$ we mean a composition of a set of networks with a "place holder", denoted by a dot $\cdot$ A network N is put into the context by replacing the place holder $\cdot$ by N.

The idea of the definition of full abstraction is that there is a model $\mathcal{O}$, which usually characterizes an observable behavior of a network. In some cases, the model $\mathcal{O}$ is not compositional, and in that case a more refined model $\mathcal{D}$ is defined. The purpose of $\mathcal{D}$ is to add precisely enough information to the model $\mathcal{O}$ to make it compositional.

Definition 5.1 Let $\mathcal{D}$ and $\mathcal{O}$ be two models of dataflow networks. The model $\mathcal{D}$ is said to be *fully abstract* with respect to $\mathcal{O}$ if for all networks N_1 and N_2

$$\mathcal{D}(N_1) = \mathcal{D}(N_2)$$
$$\Longleftrightarrow$$
$$(\forall \text{ contexts } C[\cdot])\ [\mathcal{O}(C[N_1]) = \mathcal{O}(C[N_2])]$$

□

An alternative way to understand this definition is to note that it is equivalent to the conjunction of the following three properties:

1. If $\mathcal{D}(N_1) = \mathcal{D}(N_2)$ then $\mathcal{O}(N_1) = \mathcal{O}(N_2)$, i.e., the model $\mathcal{D}$ is more distinguishing than the model $\mathcal{O}$.
2. For all contexts $C[\cdot]$ we have $\mathcal{D}(N_1) = \mathcal{D}(N_2) \implies \mathcal{D}(C[N_1]) = \mathcal{D}(C[N_2])$, i.e., the model $\mathcal{D}$ is compositional.

3. If $\mathcal{D}(N_1) \neq \mathcal{D}(N_2)$ then there is a context $C[\cdot]$ such that $\mathcal{O}(C[N_1]) \neq \mathcal{O}(C[N_2])$,
   i.e., if two networks are distinguished by $\mathcal{D}$, then there is a context such that the networks can be distinguished by $\mathcal{O}$.

To see that these three properties are equivalent to definition 5.1, note that property 1 follows from the implication $\Longrightarrow$, using the identity context. To derive property 2, note that for a particular context $C'[\cdot]$ and networks N_1 and N_2 such that $\mathcal{D}(N_1) = \mathcal{D}(N_2)$, it follows from the implication $\Longrightarrow$ that for all contexts $C[\cdot]$ we have $\mathcal{O}(C[C'[N_1]]) = \mathcal{O}(C[C'[N_2]])$, since composition of two contexts yields a context. From the implication $\Longleftarrow$ it follows that $\mathcal{D}(C'[N_1]) = \mathcal{D}(C'[N_2])$. Property 3 follows directly from the implication $\Longleftarrow$ as its contrapositive

$$\mathcal{D}(N_1) \neq \mathcal{D}(N_2) \implies (\exists C[\cdot])[\mathcal{O}(C[N_1]) \neq \mathcal{O}(C[N_2])]$$

Conversely, assume that $\mathcal{O}$ and $\mathcal{D}$ satisfy the requirements 1 - 3. The implication $\Longleftarrow$ is equivalent to property 3. The implication $\Longrightarrow$ follows by noting that if $\mathcal{D}(N_1) = \mathcal{D}(N_2)$, then by the fact that $\mathcal{D}$ is compositional (property 2) we infer for all contexts $C[\cdot]$ that $\mathcal{D}(C[N_1]) = \mathcal{D}(C[N_2])$. Finally, by property 1. we have for all context $C[\cdot]$ that $\mathcal{O}(C[N_1]) = \mathcal{O}(C[N_2])$.

In summary, the definition of full abstraction intuitively means that $\mathcal{D}$ is more distinguishing than $\mathcal{O}$, and that $\mathcal{D}$ distinguishes between networks exactly when that distinction is necessary for attaining compositionality.

We now state the main theorem of the paper.

Theorem 5.2 *The trace model is fully abstract with respect to the history model.* □

Proof: To establish the theorem, we shall prove the three properties listed after the theorem. Property 1 follows directly from definition 3.4. Property 2 was proven in theorem 4.1. Property 3 follows from the following lemma 5.3.

Lemma 5.3 If N_1 and N_2 are dataflow networks such that $\mathcal{T}(N_1) \neq \mathcal{T}(N_2)$, then there is a context $C[\cdot]$ such that $\mathcal{H}(C[N_1]) \neq \mathcal{H}(C[N_2])$. □

Proof of lemma 5.3: To prove the lemma, we must for each pair N_1, N_2 of networks find an appropriate context $C[\cdot]$. If $I(N_1) \neq I(N_2)$ or if $O(N_1) \neq O(N_2)$, the lemma follows immediately by taking $C[\cdot]$ as the identity context (i.e., $C[N] = N$).

In the remaining cases we have $T_{N_1} \neq T_{N_2}$. Assume that both N_1 and N_2 have the input channels $in_1, \ldots, in_m$ and the output channels $out_1, \ldots, out_n$. We must find a context which makes it possible to distinguish between N_1 and N_2 in the history model. Intuitively, the history model orders data items that are produced on one channel in a total order, whereas the trace model orders communication events on all channels in a total order. Thus, the sought context must "bring together" the communication events on all external channels of N_i to a single channel. We shall use the context $C[\cdot]$ shown in Figure 3. The idea of the context is to bring together data items on channels $in_1, \ldots, in_m$ and $out_1, \ldots, out_n$ into a totally ordered sequence on channel b. The sequence on b is copied onto c in order to make it observable from outside.

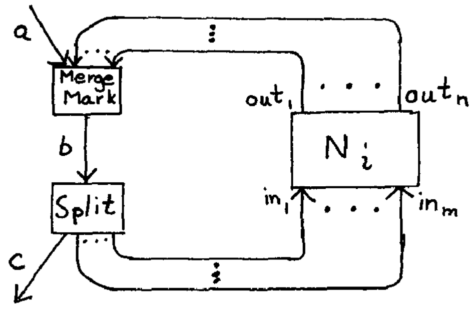

Figure 3: Context.

Over the channels $in_1, \ldots, in_m$ and $out_1, \ldots, out_n$ are transmitted data items in V. Over the remaining channels a, b, and c, are transmitted data items that have the form of communication events of N_i, i.e., they are pairs of the form $\langle in_j, d\rangle$ and $\langle out_l, d\rangle$, where in_j or out_l is one of the channels of N_i, and d is an ordinary data item that can be transmitted over that channel.

The nodes perform the following functions:

MergeMark merges data items from a and $out_1, \ldots, out_n$ onto b, i.e., it reads a data item from either incoming channel and produces it on the outgoing channel. Each data item d from a channel out_l is first transformed into the pair $\langle out_l, d\rangle$, i.e., it is tagged with a channel name, whereas the data items from channel a (which are all of the form $\langle in_j, d\rangle$) are transmitted unchanged. The merge is "fair", i.e., it does not indefinitely neglect any of its incoming channels.

Split copies each incoming data item from b onto c. Moreover, if the incoming data item is of the form $\langle in_j, d\rangle$, then the data item d is transmitted onto the channel in_j in addition to the data item $\langle in_j, d\rangle$ being transmitted over c.

Let N_M be the network consisting of only the node *MergeMark*, and let N_S be the network consisting of only the node *Split*. We can then write the context $C[\cdot]$ as $N_M \| \cdot \| N_S$.

Continuing the proof, recall that we assumed $T_{N_1} \neq T_{N_2}$. From this we conclude that there must exist a trace t such that $t \notin T_{N_1}$ but $t \in T_{N_2}$ (if not, reverse the roles of N_1 and N_2). The trace t is a (finite or infinite) sequence of communication events of the form $\langle in_j, d\rangle$ and $\langle out_l, d\rangle$. Let $in(t)$ be $t\lceil_{\{in_1,\ldots,in_m\}}$, i.e., the subsequence of t consisting of all communication events of the form $\langle in_j, d\rangle$. Define the history function h by $h(a) = in(t)$ and $h(b) = t$. We claim that

$(*)$ $h \in H_{C[N_i]}$ iff $t \in T_{N_i}$.

We first give a sketchy proof of the claim $(*)$. First note that the node *Split* is essentially connected to nodes of N_i via FIFO channels: for each data item of form $\langle in_j, d\rangle$ that are consumed by *Split* the data item d is transmitted to N_i over a FIFO channel, and each data item d transmitted from N_i over out_l reaches *Split* in the form $\langle out_l, d\rangle$ over FIFO channels via *MergeMark*. It follows that tagged data items are consumed by *Split* in the same order as the corresponding communication events would occur in a computation of N_i. Hence the sequence t is transmitted over b (and hence over c) iff t is a trace of N_i.

We next give a more detailed proof of $(*)$. First assume that t is a trace of N_i. We transform t into a sequence t' of communication events on the channels $in_1, \ldots, in_m$, $out_1, \ldots, out_n$, a, b, and c as follows:

- Each communication event of the form $\langle out_l, d\rangle$ in t is replaced by the sequence of communication events

$$\langle\ \langle out_l, d\rangle\ \langle b, \langle out_l, d\rangle\rangle\ \langle c, \langle out_l, d\rangle\rangle\ \rangle$$

- Each communication event of the form $\langle in_j, d\rangle$ in t is replaced by the sequence of communication events

$$\langle\ \langle a, \langle in_j, d\rangle\rangle\ \langle b, \langle in_j, d\rangle\rangle\ \langle c, \langle in_j, d\rangle\rangle\ \langle in_j, d\rangle\ \rangle$$

We now use theorem 4.1 to prove that $t'\lceil_{\{a,c\}}$ is a trace of $C[N_i]$. This follows if we note that (1) $t'\lceil_{\{out_1,\ldots,out_n,a,b\}}$ is a trace of the network N_M, since it is composed of fragments of the form $\langle\ \langle out_l, d\rangle\ \langle b, \langle out_l, d\rangle\rangle\ \rangle$ and of the form $\langle\ \langle a, \langle in_j, d\rangle\rangle\ \langle b, \langle in_j, d\rangle\rangle\ \rangle$, and that (2) $t'\lceil_{\{in_1,\ldots,in_m,b,c\}}$ is a trace of the network N_S, since it is composed of fragments of the form $\langle\ \langle b, \langle out_l, d\rangle\rangle\ \langle c, \langle out_l, d\rangle\rangle\ \rangle$ and of the form $\langle\ \langle b, \langle in_j, d\rangle\rangle\ \langle c, \langle in_j, d\rangle\rangle\ \langle in_j, d\rangle\ \rangle$. The if-part of $(*)$ now follows by observing that the sequence of data items transmitted over c in $t'\lceil_{\{a,c\}}$ is t, ans that the sequence of data items transmitted over a in $t'\lceil_{\{a,c\}}$ is $in(t)$.

To prove the only if-part of $(*)$ it appears necessary to argue about computations rather than about traces. Assume that there is a history function h of $C[N_i]$ such that $h(a) = in(t)$ and $h(c) = t$. Then there is a computation Γ' of $C[N_i]$ in which the sequence of items transmitted over c is t. Hence the sequence of data items produced by *Split* is also t. We shall construct a computation Γ of N_i in which t is the sequence of communication events.

We transform Γ' into a computation of N_i as follows:

1. Remove all transitions with communication events of the form $\langle a, d\rangle$ or $\langle c, d\rangle$, and all transitions resulting from firings of the node *MergeMark*.

2. Each internal transition (derived from a firing of *Split*) which consumes a data item of form $\langle out_l, d\rangle$ from channel b is labeled by $\langle out_l, d\rangle$.

3. Each internal transition (derived from a firing of *Split*) which consumes a data item of form $\langle in_j, d\rangle$ from channel b (and produces d on in_j) is labeled by $\langle in_j, d\rangle$.

4. Replace each state σ' of $C[N_i]$ in Γ' by a state σ of N_i. The state σ agrees with σ' for nodes in N_i, for internal channels of N_i, and for the channels $in_1, \ldots, in_m$. For the channels $out_1, \ldots, out_n$ we obtain $\sigma(out_l)$ as the concatenation $\sigma'(b)\lceil_{\{out_l\}} . \sigma'(out_l)$, where $\sigma'(b)\lceil_{\{out_l\}}$ is the subsequence of $\sigma'(b)$ of data items of form $\langle out_l, d\rangle$.

The result is a sequence Γ. The intuition here is that the channels $in_1, \ldots, in_m$ perform the same sequence of transitions in Γ as in Γ', whereas each channels out_l in Γ simulates the concatenation of the sequence of data items of form $\langle out_l, d\rangle$ in b and the channel out_l in Γ'.

Note that each firing of *Split* in Γ' is replaced by a corresponding transition with the communication event that was produced by *Split*. It follows that the sequence of communication events in Γ is the sequence of events that is produced on c by *Split*, which is exactly t. To see that Γ is indeed a computation of N_i, note that the nodes and internal channels of N_i, and the channels $in_1, \ldots, in_m$ perform exactly the same sequence of transitions in Γ as in Γ'. Also note that the channels $out_1, \ldots, out_n$ participate in the same sequence of transitions in Γ as in Γ', with the difference that the transitions that remove data items from them may occur in a later position but still in the same relative order. □

6 Comparison with Another Fully Abstract Model

Another fully abstract model of dataflow networks. has been presented by Kok [Kok87]. Denoting the set of data items by V, a network is modeled as an element in $((V^*)^\omega)^m \longrightarrow \mathcal{P}(((V^*)^\omega)^n)$, that is, as a function from tuples of infinite sequences of finite words of data items to a set of such tuples of infinite sequences.

Intuitively, we describe the idea behind Kok's model as follows. We use v, w, possibly with subscripts, to range over infinite sequences of finite words. Let $w[i]$ to denote the ith finite word in the sequence w, and let $w \uparrow i$ denote the concatenation $w[1]. \cdots .w[i]$ of the i first words of w, and let $w \uparrow \infty$ denote the concatenation of all words of w. A dataflow network N with input channels numbered from 1 to m and output channels numbered from 1 to n is denoted by a function f_N from $((V^*)^\omega)^m$ to $\mathcal{P}(((V^*)^\omega)^n)$. For f_N we have $\langle w_1, \ldots, w_n\rangle \in f_N(\langle v_1, \ldots, v_m\rangle)$ if there is a computation Γ of the network in which

- for each $i \geq 1$ there is a finite prefix Γ' of Γ such that $v \uparrow i$ is the sequence of data items transmitted over c in Γ', and

- $v \uparrow \infty$ is the sequence of data items transmitted over c in Γ.

We can derive the model of Kok from our model by stating that $\langle w_1, \ldots, w_n\rangle \in f(\langle v_1, \ldots, v_m\rangle)$ precisely if T_N contains the trace

$\langle in_1, v_1[1]\rangle \ldots \langle in_m, v_m[1]\rangle\ \langle out_1, w_1[1]\rangle \ldots \langle out_n, w_n[1]\rangle$
$\langle in_1, v_1[2]\rangle \ldots \langle in_m, v_m[2]\rangle\ \langle out_1, w_1[2]\rangle \ldots$

A proof of this will appear in [JK].

We argue that our trace model uses simpler concepts and gives a more natural proof of full abstraction than the model by Kok. The context in that proof has the property that the set of histories that can be observed on the output channel is exactly the set of traces of the network inside the context.

A trace model is a suitable basis for specification of networks: there is a rather extensive literature on specifying and verifying distributed systems using traces (e.g. [CH81, Jon85, MC82, MCS82, NDGO86]). The model by Kok is in line with some earlier models for dataflow networks (e.g. [Bro86, Par83, Bou82]) in that it uses functions to denote networks.

7 Related Work

In this section, we review other related models of dataflow networks from the point of view of full abstraction.

The seminal paper in this area is by Kahn [Kah74], where a model for deterministic dataflow networks is presented. Subsequently, it was shown by Brock and Ackerman [BA81] that a straight-forward generalization of this model, the history model, is not compositional for nondeterministic networks. Brock and Ackerman showed that in order to attain compositionality, some information about ordering or causality between the appearance of data items on different channels must be introduced.

One way to attain compositionality is to extend the history model by a partial ordering relation between data items on different channels. The partial ordering represents causality or temporal ordering [Kel78, BA81, Pra82, Pra84, SN85]. The introduction of partial ordering information attains compositionality, but not full abstraction. For instance, a network which performs the unrelated output events $\langle out_1, d\rangle$ and $\langle out_2, d\rangle$ is distinguished from a network which either relates $\langle out_1, d\rangle$ before $\langle out_2, d\rangle$ or vice versa.

Keller and Panangaden [KP85] (in [KP86] in a slightly different framework) propose a trace-model related to ours. A difference is that they use input events in traces to represent the consumption of a data item by a node and output events in traces to represent the production of a data item by a node. Hence their model is not fully abstract. For instance, their model distinguishes a network with a single node acting as a one-place buffer from a network with a two-place buffer. But the difference between these networks is "masked" by the input and output channels of the network, and is therefore not observable in any context, Back and Mannila [BM85] model a network by a prefix-closed set of finite sequences, which corresponds to the set of prefixes of our traces. Their model identifies certain networks that are distinguished in the history model.

Several authors represent nondeterminism as determinism with a missing parameter – an "oracle" – which accounts for the nondeterminism. An oracle is an infinite sequence of outcomes of nondeterministic choices. Broy [Bro83, Bro88, Bro86] models a nondeterministic network by a set of deterministic incarnations of it, each corresponding to a particular assignment of oracles to nondeterministic choices. Boussinot [Bou82] and Park [Par83] use oracles and also add "hiatons" to sequences of data items in order to model the passage of time. A network is denoted by a function from oracles and "hiatonized" input sequences to "hiatonized" output sequences. Park also hides the oracles to obtain a model in which a network is denoted by a function from hiatonized input sequences to sets of hiatonized output sequences. However, the resulting model includes too much detail about the number of hiatons in sequences to be fully abstract. Kosinski [Kos78] tags data items by the sequence of internal choices that were made in order to produce them.

A related trace model, which is applicable to both synchronously and asynchronously communicating networks, and hence includes more detail, has been presented by Nguyen et al [NDGO86].

8 Conclusion

We have presented a fully abstract model of dataflow networks, which denotes a network by the set of its traces. As indicated by earlier work (e.g. [Kel78, BA81]), one must add information about how the behavior of a network depends on the ordering of the appearance of data items on different channels. Our model provides precisely this information by the set of traces, giving all possible total orderings of supply of input and appearance of output on the channels of the network.

Two properties of Kahn's original model [Kah74] are missing in our trace model. Kahn showed how to compute the denotation of a network from the denotations of its components by a fixedpoint construction. It appears difficult to incorporate this property into a model for networks that exhibit nondeterminism and fairness. Approaches to solving this problem appear in e.g. [Bro86, SN83, KP85]. Kahn also showed that his model is equally applicable for recursively defined networks in which a node can expand recursively into a subnetwork.

We have not investigated either of these properties in the context of our model. Since we have primarily aimed at investigating *what* must be described rather than *how*, our model does not have the first property. It would be interesting to see whether our results can be extended to recursive networks.

Acknowledgments

I am grateful to Joost Kok for stimulating discussions and ideas. I also would like to thank Joachim Parrow for critical reading of earlier versions of the paper and for many fruitful discussions and comments.

References

[BA81] J. D. Brock and W. B. Ackerman. Scenarios: a model of non-determinate computation. In Diaz and Ramos, editors, *Formalization of Programming Concepts, LNCS 107*, pages 252–259, Springer Verlag, 1981.

[BM85] R. J. R. Back and H. Mannila. On the suitability of trace semantics for modular proofs of communicating processes. *Theoretical Computer Science*, 39(1):47–68, 1985.

[Bou82] F. Boussinot. Proposition de semantique denotationelle pour des processus avec operateur de melange equitable. *Theoretical Computer Science*, 18(2):173-206, 1982.

[Bro83] M. Broy. Fixed point theory for communication and concurrency. In Bjoerner, editor, *Formal Description of Programming Concepts II*, pages 125–146, North-Holland, Amsterdam, 1983.

[Bro86] M. Broy. A theory for nondeterminism, parallelism, communication, and concurrency. *Theoretical Computer Science*, 45:1–61, 1986.

[Bro88] M. Broy. Nondeterministic data flow programs: how to avoid the merge anomaly. *Science of Computer Programming*, 10:65–85, 1988.

[CH81] Z. C. Chen and C. A. R. Hoare. Partial correctness of communicating sequential processes. In *Proc. International Conference on Distributed Computing*, pages 1–12, Paris, April 1981.

[JK] B. Jonsson and J. Kok. Comparing dataflow models. In progress.

[Jon85] B. Jonsson. A model and proof system for asynchronous networks. In *Proc. 4:th ACM Symp. on Principles of Distributed Computing*, pages 49–58, Minaki, Canada, 1985.

[Jon87a] B. Jonsson. *Compositional Verification of Distributed Systems.* PhD thesis, Dept. of Computer Systems, Uppsala University, Sweden, Uppsala, Sweden, 1987. Available as report DoCS 87/09.

[Jon87b] B. Jonsson. Modular verification of asynchronous networks. In *Proc. 6th ACM Symp. on Principles of Distributed Computing*, pages 152–166, Vancouver, Canada, 1987.

[Kah74] G. Kahn. The semantics of a simple language for parallel programming. In *IFIP 74*, pages 471–475, North-Holland, 1974.

[Kel78] R. M. Keller. Denotational models for parallel programs with indeterminate operators. In Neuhold, editor, *Formal Descriptions of Programming Concepts*, pages 337–366, North-Holland, 1978.

[Kok86] J. N. Kok. Denotational semantics of nets with nondeterminism. In *European Symposium on Programming, Saarbrücken, LNCS 206*, pages 237–249, Springer Verlag, 1986.

[Kok87] J. N. Kok. A fully abstract semantics for data flow nets. In *Proc. PARLE, LNCS 259*, pages 351–368, Springer Verlag, 1987.

[Kos78] P. R. Kosinski. A straight-forward denotational semantics for nondeterminate data flow programs. In *Proc. 5th ACM Symp. on Principles of Programming Languages*, pages 214–219, 1978.

[KP85] R. M. Keller and P. Panangaden. Semantics of networks containing indeterminate operators. In Brookes, Roscoe, and Winskel, editors, *Seminar on Concurrency 1984, LNCS 197*, pages 479–496, 1985.

[KP86] R. M. Keller and P. Panangaden. Semantics of networks containing indeterminate operators. *Distributed Computing*, 1:235–245, 1986.

[Lam83] L. Lamport. Specifying concurrent program modules. *ACM TOPLAS*, 5(2):190–222, 1983.

[LT87] N. A. Lynch and M. R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proc. 6th ACM Symp. on Principles of Distributed Computing*, pages 137–151, 1987.

[MC82] J. Misra and K. M. Chandy. Proofs of networks of processes. *IEEE Transactions on Software Engineering*, SE-7(4):417–426, July 1982.

[MCS82] J. Misra, K. M. Chandy, and T. Smith. Proving safety and liveness of communicating processes with examples. In *Proc. ACM SIGACT-SIGOPS Symp. on Principles of Distributed Computing*, pages 201-208, 1982.

[Mis84] J. Misra. Reasoning about networks of communicating processes. In *INRIA Advanced Nato Study Institute on Logics and Models for Verification and Specification of Concurrent Systems*, La Colle sur Loupe, France, 1984.

[MP81] Z. Manna and A. Pnueli. The temporal framework for concurrent programs. In Boyer and Moore, editors, *The Correctness Problem in Computer Science*, pages 215–274, Academic Press, 1981.

[NDGO86] V. Nguyen, A. Demers, D. Gries, and S. Owicki. A model and temporal proof system for networks of processes. *Distributed Computing*, 1(1):7–25, 1986.

[Par83] D. Park. The 'fairness' problem and nondeterministic computing networks. In de Bakker and van Leeuwen, editors, *Foundations of Computer Science IV, Part 2*, pages 133–161, Amsterdam, 1983. Mathematical Centre Tracts 159.

[Plo81} G. Plotkin. *A Structural Approach to Operational Semantics.* Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, 1981.

[Pra82] V. R. Pratt. On the composition of processes. In *Proc. 9th ACM Symp. in Principles of Programming Languages*, pages 213–223, 1982.

[Pra84] V. R. Pratt. The pomset model of parallel processes: unifying the temporal and the spatial. In Brookes, Roscoe, and Winskel, editors, *Proc. Seminar on Concurrency, LNCS 197*, pages 180–196, Springer Verlag, 1984.

[SN83] J. Staples and V. L. Nguyen. Computing the behaviour of asynchronous processes. *Theoretical Computer Science*, 26(3):343–353, 1983.

[SN85] J. Staples and V. L. Nguyen. A fixpoint semantics for nondeterministic data flow. *ACM Journal*, 32(2):411–444, April 1985.

[Sta84] E. W. Stark. *Foundations of a Theory of Specification for Distributed Systems.* PhD thesis, Massachussetts Inst. of Technology, 1984. Available as Report No. MIT/LCS/TR-342.