A Fully Abstract Trace Model for Dataflow and Asynchronous Networks
by
Bengt Jonsson
A Fully Abstract Trace Model for Dataflow and Asynchronous Networks∗

Bengt Jonsson
Swedish Institute of Computer Science †
and
Dept. of Computer Science, Uppsala University

SICS Research Report 90012
October 1990

Abstract

A dataflow network consists of nodes that communicate over perfect unbounded 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. We also generalize our result, showing that the model is fully abstract also for classes of networks where nodes communicate over other types of asynchronous channels. Examples of such classes are networks with unordered channels, and networks with lossy channels.

∗This Research report is a revised and extended version of a paper that has appeared under the title “A fully abstract trace model for dataflow networks” in the Proceedings of the 16th Annual ACM Symposium on Principles of Programming Languages, Austin, Texas, January 1989.

†Box 1263, S-164 28 Kista, SWEDEN, E-mail:bengt@sics.se
1 Introduction

Semantical models of parallel systems have been a topic of intensive study in the last years (e.g., [BHR84, Hoa83, Mil89, dNH84, OH86]). A purpose of that study is a better understanding of how to describe and reason about the behavior of parallel systems. Two desiderata for a semantic model are: (1) the model should abstract from the internal activity of a system, describing only its externally observable behavior, and (2) the model should be compositional, meaning that the denotation of a composite system can be obtained using only the denotations of its components. A semantic model satisfying these criteria can e.g. serve as the basis for compositional verification methods, where the verification of a composite system can be split into verifications of its components [BM85, Jon85, MC81, NDGO86, Zwi89].

In this paper, we study semantic models of asynchronous networks, i.e., systems in which concurrently executing processes communicate by sending data along asynchronous communication channels. An important class of asynchronous networks is dataflow networks, where the channels are unbounded FIFO channels. For dataflow networks with only deterministic processes, Kahn [Kah74] has proposed an elegant semantic model, which satisfies both desiderata (1) and (2) above. Kahn models a network by a function from sequences of data items on input channels to sequences of data items on output channels. For example, a dataflow network with two input channels and one output channel which adds each pair of integers received on the two input channels and outputs the sum onto the output channel can be modeled by a function. When supplied with e.g. the two sequences \(1, 2, 3\) and \(1, 3, 5, 7\), this function produces the sequence \(2, 5, 8\).

For nondeterministic dataflow networks, the straightforward generalization of Kahn’s model would be 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). This was first shown by Brock and Ackerman [BA81]. They showed that two nondeterministic networks that are equivalent in the history model may exhibit different behaviors if constraints on the order in which input is supplied and output appears are enforced (e.g. that one input is supplied only after some output appears), and that such constraints can be enforced by composing a network with another network.

In order to obtain a compositional model, one must therefore add information about how a dataflow network behaves under different ordering constraints. Desideratum (1) suggests that one should not add more information than is necessary for attaining compositionality. In other words, one should look for a model which is fully abstract. Intuitively, a model \(\llbracket \cdot \rrbracket_D\) is fully abstract with respect to a model \(\llbracket \cdot \rrbracket_O\) if \(\llbracket \cdot \rrbracket_D\) has added precisely enough information to \(\llbracket \cdot \rrbracket_O\) to attain compositionality. Thus, a fully abstract model combines desiderata (1) and (2) in an optimal way. For modular specification methods, where the specification of a network can be split into independent specifications of its components, a fully abstract model indicates what aspects of a network’s behavior must be specified.

In this paper, we present a model of dataflow networks which is fully abstract with respect to the history model. Our model denotes a network by the set of its traces. A trace is a sequence of communication events, each of which represents the transmission or reception of a data item on an external channel of the network. The trace model captures both safety and
liveness properties of a network. Similar models have earlier been used for specification and verification of distributed algorithms and protocols [Jon85, LT87, Jon87a] designed for networks with asynchronous message-passing.

In the paper, we also generalize the results about compositionality and full abstraction to other classes of asynchronous networks, e.g. networks that communicate over unordered channels or over lossy channels. If the channels satisfy certain properties, we show that the trace model is also in this case compositional and fully abstract with respect to the history model. The proof is similar to that for dataflow networks, showing that the full abstraction result is a consequence more of the asynchronicity of channels than of the particular FIFO discipline present in dataflow networks.

In order to prove our results formally, we must first establish a formal definition of asynchronous networks and the special case of dataflow networks. The history model and the trace model consider infinite behaviors, which can be constrained by subtle fairness requirements. We will therefore provide a careful formal definition of asynchronous networks using the framework of labeled transition systems with fairness properties. Transition systems have often been used as a general model for describing the behavior of parallel systems (e.g. [Plo81, MP81]).

For dataflow networks, many compositional models have been proposed in the literature [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]. The trace model presented in this paper is conceptually simpler than that of Kok, and more similar to the semantical models that are commonly used for specification and verification of concurrent systems.

Our model is similar to a model presented by Misra and Chandy [Mis84], in our earlier work [Jon85, Jon87a], and by Lynch and Tuttle [LT87]. These models are defined for a class of communicating systems, called I/O-automata in [LT87] and I/O-systems in [Sta84, Jon87a, Jon87b]. Our formal definition of an asynchronous network will in fact be a special case of an I/O-automaton (I/O-system). Some results of this paper appear in the context of I/O-systems in the author's thesis [Jon87a].

The paper is organized as follows. In the next section, we present the basic definitions of asynchronous networks and dataflow networks. This definition is given in the framework of labeled transition systems with fairness properties, where the fairness properties are necessitated by the fact that our model considers infinite behaviors. The trace model and history model are defined in Section 3. In Section 4, we prove that the trace model is compositional. In Section 5, the compositionality of the trace model is illustrated by an example due to Brock and Ackerman. In Section 6, we prove that for dataflow networks the trace model is fully abstract with respect to the history model. Section 7 contains a generalization of the compositionality and full abstraction results to other classes of asynchronous networks. In Section 8, we review related work, and Section 9 contains conclusions.
2 Dataflow and Asynchronous Networks

In this section, we define asynchronous networks and the special case of dataflow networks. Our model of networks represents infinite behaviors, which can be constrained by fairness requirements. The concept of fairness often leads to subtleties, and we shall therefore provide a careful formal definition of asynchronous networks within the framework of labeled transition systems with fairness properties. This framework is presented in Section 2.1, and the definition of networks is contained in Section 2.2.

2.1 Transition Systems

In this subsection, we present the framework of labeled transition systems, which will be used to give a formal definition of asynchronous networks. Transition systems are often used as a general model for describing the behavior of parallel systems (e.g. [Plo81, MP81, Mil89]). Since our model of networks also considers infinite behavior, we include fairness, as in e.g. [MP81], in the transition systems.

We assume a set $L$ of labels, which does not include the silent label $\tau$.

**Definition 2.1** A transition system $T$ is a tuple $\langle L, S, s^0, R, F \rangle$, where

$L$ is a set of labels in $L$, called the sort of $T$.

$S$ is a set, called the set of states

$s^0 \in S$ is an initial state.

$R \subseteq S \times (L \cup \{\tau\}) \times S$ is a set of transitions. A transition is thus a triple, which we denote $s \xrightarrow{l} s'$, where $s, s' \in S$ and $l \in (L \cup \tau)$.

$F \subseteq 2^R$ is a finite collection of fairness sets. Each fairness set $F$ is a subset of the set $R$ of transitions.

A transition of form $s \xrightarrow{\tau} s$ (with both states the same and a silent label) is called a stuttering transition. For technical reasons, we will in the following for each transition system require that its set $R$ of transitions contains all stuttering transitions of form $s \xrightarrow{\tau} s$ with $s \in S$. A transition $s_1 \xrightarrow{l} s_2$ is enabled in the state $s_1$. A set $F$ of transitions is enabled in a state $s$ if $F$ contains a transition which is enabled in $s$.

Intuitively, the sort of a transition system $T$ can be thought of as a set of ports through which $T$ may interact with its environment. The state of a transition system contains information that is relevant for determining its future behavior. E.g., the state of a transition system representing an asynchronous network could contain the contents of its channels and the information that is stored in its nodes. A transition represents a possible change of state together with the occurrence of a label. A transition with the silent label $\tau$ is called a silent transition. A fairness set represents a set of transitions which may not be neglected indefinitely in executions of the transition system. The rôle of fairness sets will be seen more precisely from the definition of computations in Definition 3.1.

We define parallel composition on transition systems, which will be used in the definition of asynchronous networks.
Definition 2.2 For \( i = 1, 2 \), let \( T_i = (L_i, S_i, s_0^i, R_i, F_i) \) be a transition system. The \textit{parallel composition} of \( T_1 \) and \( T_2 \), denoted \( T_1 \parallel T_2 \), is the transition system \( (L, S, s_0, R, F) \), where

\[
L = (L_1 \setminus L_2) \cup (L_2 \setminus L_1),
\]

\[
S = S_1 \times S_2,
\]

\[
s_0 = (s_0^1, s_0^2),
\]

\( R \) consists of all triples of form \( (s_1, s_2) \xrightarrow{l} (s_1', s_2') \) in \( S \times (L \cup \{\tau\}) \times S \) such that either

1. \( s_1 \xrightarrow{l} s_1' \in R_1 \) and \( s_2 = s_2' \), and either \( l = \tau \) or \( l \notin L_2 \), or

2. \( s_2 \xrightarrow{l} s_2' \in R_2 \) and \( s_1 = s_1' \), and either \( l = \tau \) or \( l \notin L_1 \), or

3. \( l = \tau \) and there is a label \( l' \neq \tau \) such that \( s_1 \xrightarrow{l'} s_1' \in R_1 \) and \( s_2 \xrightarrow{l'} s_2' \in R_2 \).

In case 1 we say that \( s_1 \xrightarrow{l} s_1' \) is a projection of \( (s_1, s_2) \xrightarrow{l} (s_1', s_2') \) onto the 1\textsuperscript{st} component, and that \( s_2 \xrightarrow{\tau} s_2 \) is a projection onto the 2\textsuperscript{nd} component; in case 2 we define the projections vice versa, and in case 3 we say that \( s_1 \xrightarrow{l'} s_1' \) is a projection of \( (s_1, s_2) \xrightarrow{l} (s_1', s_2') \) onto the \( i \)\textsuperscript{th} component for \( i = 1, 2 \).

\( F \) is obtained as follows: for each fairness set \( F_i \in F_i \) where \( i \) is 1 or 2, there is a fairness set \( F \in F \) consisting of the set of all transitions \( (s_1, s_2) \xrightarrow{l} (s_1', s_2') \) for which a projection onto the \( i \)\textsuperscript{th} component is in \( F_i \).

Intuitively, the sort of \( T_1 \parallel T_2 \) contains the labels that are in exactly one of the component sorts. A state of \( T \) can be written as a tuple with a component from each \( T_i \). A transition in \( R \) corresponds to either (1) a transition of \( T_1 \) with a silent label or with a label not in \( L_2 \), which does not affect the other component, or (2) a transition of \( T_2 \) with a silent label or with a label not in \( L_1 \), which does not affect the other component, or (3) a transition which synchronizes the two component transitions with the same non-silent labels; such a synchronizing transition becomes silent, and can therefore not be used for further synchronization with other components. Thus communication is binary; we have chosen this type of communication since our networks are built from components which communicate by binary synchronizations. This definition of parallel composition is similar to the definition of parallel composition in CCS [Mil89], except for the requirement that a component can perform a non-silent transition in isolation only if the label of the transition is not in the sort of the other component. The definition of projection can be extended to sequences of transitions in the natural way. The fairness sets of the parallel composition are obtained in a natural way from the fairness sets of components.

We say that two transition systems \( (L_1, S_1, s_0^1, R_1, F_1) \) and \( (L_2, S_2, s_0^2, R_2, F_2) \) are \textit{isomorphic} if there is an isomorphism between the sets of states \( S_1 \) and \( S_2 \) under which also \( s_0^1 \) and \( s_0^2 \), \( R_1 \) and \( R_2 \), and \( F_1 \) and \( F_2 \) are obtained from each other. We shall not distinguish isomorphic transition systems from each other, since for our purposes isomorphic transition systems are equivalent.

Proposition 2.3 The following properties hold for the composition operation defined in Definition 2.2:

1. The composition operator is commutative: \( T_1 \parallel T_2 \) and \( T_2 \parallel T_1 \) are isomorphic, and
2. The composition operator is associative in the following sense: if each non-silent label occurs in the sort of at most two of the transition systems \(T_1, T_2,\) and \(T_3,\) then \((T_1 \mid T_2) \mid T_3\) is isomorphic to \(T_1 \mid (T_2 \mid T_3).\)

It is not difficult to find the isomorphisms between the states that prove Proposition 2.3. In the first case the isomorphism reverses the order between the state components and in the second case it changes the parentheses. Proposition 2.3 allows us to write down the parallel composition of \(k\) transition systems \(T_1, \ldots, T_k\) as \(T_1 \mid \cdots \mid T_k\) without parentheses, provided that each non-silent label occurs in the sort of at most two of the transition systems \(T_1, \ldots, T_k.\)

In the following, whenever we write a parallel composition of several components, each non-silent label occurs in the sort of at most two of the components.

### 2.2 Asynchronous Networks

In this section, we give a formal definition of asynchronous networks in terms of transition systems. The structure of our definition is as follows. First nodes and channels are defined separately. Then an asynchronous network is defined as the parallel composition of nodes and of the channels that connect the nodes to each other and to the environment.

We assume a set \(V\) of data items, ranged over by \(v, w.\) We assume a set of channel names, ranged over by \(c, in,\) and \(out.\) We assume a set of labels, consisting of labels of the following three forms:

- \(send(c, v)\) represents the sending of the data item \(v\) onto channel \(c\) from a node having \(c\) as an outgoing channel,
- \(rec(c, v)\) represents the reception of the data item \(v\) from channel \(c\) to a node having \(c\) as an incoming channel,
- \((c, v)\) represents the reception of the data item \(v\) from channel \(c\) by the environment, or the sending of the data item \(v\) onto channel \(c\) from the environment. A label of this form is called a communication event.

If \(C\) is a set of channel names, then \(\rho(C), \sigma(C),\) and \(\kappa(C)\) are the sets of labels of form \(rec(c, v),\) \(send(c, v),\) and \((c, v),\) respectively, for \(c \in C\) and \(v \in V.\)

**Definition 2.4** A node \(n\) is a triple \((I_n, O_n, T_n)\) where

- \(I_n\) is a finite set of channel names, called the set of input channels.
- \(O_n\) is a finite set of channel names, called the set of output channels, with \(I_n \cap O_n = \emptyset.\)
- \(T_n\) is a transition system \(\langle L, S, s^0, R, F \rangle\) with sort \(L = \rho(I_n) \cup \sigma(O_n)\) such that

1. For each input channel \(c \in I_n\) and state \(s \in S\) the following holds: if a transition with a label of form \(rec(c, v)\) for some \(v \in V\) is enabled in \(s,\) then for all \(w \in V\) a transition with label \(rec(c, w)\) is enabled in \(s.\)

2. For each input channel \(c \in I_n\) the following holds: if a fairness set \(F \in F\) contains a transition with a label of form \(rec(c, v)\) for some \(v \in V,\) then \(F\) contains all transitions in \(R\) with a label of form \(rec(c, w)\) for \(w \in V.\)

\(\square\)
Intuitively, a node \( n \) consumes data items from its input channels \( I_n \) and produces data items onto its output channels \( O_n \). The intuitive meaning of a transition \( s \xrightarrow{\text{rec}(c,v)} s' \) is: \( \text{"when } s \text{ is the state of the node, then it may consume the data item } v \text{ from input channel } c \text{ and change its state to } s' \text{."} \) Analogously, a transition \( s \xrightarrow{\text{send}(c,v)} s' \) means that \( \text{"when } s \text{ is the state of the node, then it may produce the data item } v \text{ onto output channel } c \text{ and change its state to } s' \text{."} \) Requirement 1 on \( T_n \) intuitively states that a node cannot inspect a data value before consuming it, and requirement 2 is an analogous requirement for fairness sets. These requirements will be used in the proof that the trace model is compositional.

**Example 2.5** Consider a node \( \text{Fairmerge} \) which consumes data items from the channels \( in_1 \) and \( in_2 \) and copies them onto \( out \). The node is \("fair", i.e., if it is continuously possible to consume a data item from an input channel, then the node will eventually consume a data item from that channel. \( \text{Fairmerge} \) is represented by the tuple \( \langle \{in_1, in_2\}, \{out\}, T \rangle \), where \( T \) is the transition system \( \langle L, S, s^0, R, \mathcal{F} \rangle \), where \( L = \rho(\{in_1, in_2\}) \cup \sigma(\{out\}) \), where \( S = \{s^0\} \cup V \) (i.e., there is one initial state \( s^0 \) and one state \( v \) for each data item \( v \in V \)), and where the set \( R \) of transitions is the union of the set

\[
R1 = \{ s^0 \xrightarrow{\text{rec}(in_1,v)} v \mid v \in V \}
\]

of transitions that consume a data item from \( in_1 \), the set

\[
R2 = \{ s^0 \xrightarrow{\text{rec}(in_2,v)} v \mid v \in V \}
\]

of transitions that consume a data item from \( in_2 \), and the set

\[
R3 = \{ v \xrightarrow{\text{send}(out,v)} s^0 \mid v \in V \}
\]

of transitions that produce a data item on \( out \). By stating that \( \mathcal{F} = \{R1, R2, R3\} \) we ensure that the node will be fair. The fairness set \( R1 \) ensures that input from \( in_1 \) will not be neglected, fairness set \( R2 \) ensures that input from \( in_2 \) will not be neglected, and \( R3 \) ensures that the consumed data values will be produced onto \( out \) so that the node can continue executing. \( \Box \)

**Definition 2.6** Let \( c \) be a channel name. A channel with channel name \( c \) is a transition system \( T_c = \langle L, S, s^0, R, \mathcal{F} \rangle \), such that

1. \( L = \rho(\{c\}) \cup \sigma(\{c\}) \),

2. For each state \( s \in S \) and data item \( v \in V \), there is an \( s' \in S \) such that \( s \xrightarrow{\text{send}(c,v)} s' \) is a transition in \( R \).

3. All transitions in any fairness set in \( \mathcal{F} \) are labeled by \( \tau \).

\( \Box \)

Recall that the label \( \text{send}(c,v) \) intuitively represents the insertion of the data item \( v \) into the channel, and that \( \text{rec}(c,v) \) represents the removal of \( v \) from the channel. Requirement 2 states that it is always possible to insert any data item into the channel; this is the most important property of asynchronous channels that we will use. Requirement 3 states that \( T_c \) does not itself impose any fairness restriction on insertions and removals from the channel. The intuitive motivation for this requirement is that insertions and deletions are performed by nodes, and that corresponding fairness requirements should occur in the definition of the corresponding nodes.
Definition 2.7 If $T_c$ is a channel $(L, S, s^0, R, \mathcal{F})$, define

$T_c^c$ as $(L, S, s^0, R, \mathcal{F} \cup \{R_p\})$, where $R_p$ is the set of transitions in $R$ with a rec-label,

$T_c^c$ as $T_c$ but with all labels of form $send(c, v)$ replaced by $(c, v)$,

$T_c^p$ as $T_c^c$ but with all labels of form $rec(c, v)$ replaced by $(c, v)$.

Intuitively, $T_c^c$ is obtained from $T_c$ by adding a fairness requirement that the reception of data items from the channel is not neglected indefinitely. The transition system $T_c^c$ is obtained from $T_c$ by renaming send-labels into communication events. It will be used in situations where the environment can send data items into the channel. Analogously, $T_c^p$ is obtained from $T_c^c$ by renaming rec-labels into communication events. It will be used in situations where the environment can receive data items from the channel.

Definition 2.8 A perfect unbounded FIFO channel with name $c$ is represented by the channel $FIFO_c = (L, S, s^0, R, \mathcal{F})$, where

$L = \rho(\{c\}) \cup \sigma(\{c\})$,

$S = V^*$, the set of finite sequences of data items in $V$,

$s^0 = \langle \rangle$, the empty sequence,

$R$ consists of

- all transition of form $(\nu \cdot \sigma) \xrightarrow{\text{rec}(c, \nu)} \sigma$ for $\sigma \in V^*$ and $\nu \in V$, and

- all transition of form $\sigma \xrightarrow{\text{send}(c, \nu)} (\nu \cdot \sigma)$ for $\sigma \in V^*$ and $\nu \in V$.

$\mathcal{F} = \emptyset$.

Intuitively, the state of $FIFO_c$ contains the sequence of data items that have been sent over the channel but not yet received. Each rec-transition removes a data item from the state, and each send-transition adds a data item to the state.

In the following, we assume that for each channel name $c$, there is a unique channel $T_c$ with channel name $c$. Different classes of asynchronous networks are characterized by different choices of channels for the channel names. The class of dataflow networks is characterized by taking $T_c$ to be $FIFO_c$ for each $c$.

We next define networks and the composition operation on networks.

Definition 2.9 Assume that $n$ is a node $(I_n, O_n, T_n)$ with input channels $\{in_1, \ldots, in_p\}$ and output channels $\{out_1, \ldots, out_q\}$. An atomic network $N$ with node $n$ is the triple $(I_N, O_N, T_N)$, where $I_N = I_n$, and $O_N = O_n$, and $T_N$ is the transition system

$T_n \mid T_{in_1} \mid \cdots \mid T_{in_p} \mid T_{out_1} \mid \cdots \mid T_{out_q}$.

The sets $I_n$ and $O_n$ are called the input and output channels of $N$. If $N$ is the above atomic network, then $N$ without external channels, denoted $U_N$, is defined as $T_n$. 

8
Intuitively, an atomic network is the parallel composition of a node $T_n$ and its incident channels, where for an output channel name $c$ we take $T_c^\phi$ as the channel and for an input channel name $c$ we take $T_c^\sigma$ as the channel. $T_N$ can communicate with other transition systems through communication events on its input and output channels. The transition system $U_N$ is $T_N$ without the external channels, and will subsequently be used in the definition of parallel composition.

**Definition 2.10** The networks $N_1, \ldots, N_k$ are compatible if $I_{N_i} \cap I_{N_j} = \emptyset$ and $O_{N_i} \cap O_{N_j} = \emptyset$ whenever $i \neq j$, i.e., each channel name occurs at most once as an input channel and at most once as an output channel among $N_1, \ldots, N_k$. Let $N_1, \ldots, N_k$ be compatible networks. Let $C_{\text{int}}$ be the set $(\bigcup_{i=1}^k I_{N_i}) \cap (\bigcup_{i=1}^k O_{N_i})$ of channels occurring both as input and as output channels among $N_1, \ldots, N_k$. The composition $N_1 || \cdots || N_k$ of $N_1, \ldots, N_k$ is the tuple $N = (I_N, O_N, T_N)$, where

$$I_N = \left( \bigcup_{i=1}^k I_{N_i} \right) \setminus \left( \bigcup_{i=1}^k O_{N_i} \right),$$

$$O_N = \left( \bigcup_{i=1}^k O_{N_i} \right) \setminus \left( \bigcup_{i=1}^k I_{N_i} \right),$$

and where, assuming that $I_N = \{in_1, \ldots, in_p\}$, that $O_N = \{out_1, \ldots, out_q\}$, and that $C_{\text{int}} = \{c_1, \ldots, c_r\}$, we have

$$U_N = U_{N_1} \mid \cdots \mid U_{N_k} \mid T_{c_1} \mid \cdots \mid T_{c_r},$$

$$T_N = U_N \mid T_{in_1}^\sigma \mid \cdots \mid T_{in_p}^\sigma \mid T_{out_1}^\phi \mid \cdots \mid T_{out_q}^\phi.$$

Intuitively, the composition $N = N_1 || \cdots || N_k$ has as input channels those input channels of the components that are not connected to any output channel. Its output channels are obtained analogously. The transition system $U_N$, i.e., $N$ without external channels, is the parallel composition of $U_{N_1}, \ldots, U_{N_k}$, together with the internal channels $T_{c_1}, \ldots, T_{c_r}$ that connect the components to each other. The transition system $T_N$ is $U_N$ in parallel with the external channels $T_{in_1}^\sigma, \ldots, T_{in_p}^\sigma$ and $T_{out_1}^\phi, \ldots, T_{out_q}^\phi$ that connect the network with its environment. In the following we use $E_N$, the set of external channels of $N$, to denote the set $I_N \cup O_N$ of input and output channels of $N$.

![Diagram](image.png)

Figure 1: Two networks $T_{N_1}$ and $T_{N_2}$ (left), and a network with their composition $T_N$ (right).
Example 2.11 An example of the composition operation is illustrated graphically in Figure 1. To the left are transition systems, \( T_{N_1} \) and \( T_{N_2} \), of two networks, where \( T_{N_1} \) consists of the network without external channels \( U_{N_1} \), together with external channels \( T_0^0 \), \( T_0^1 \), and \( T_0^2 \). To the right is shown the transition system \( T_N \) of the composition \( N = N_1 \parallel N_2 \). The area within dashed lines is intended to illustrate \( U_N \), i.e., \( N \) without its external channels, which consists of the components \( U_{N_1} \), \( U_{N_2} \), and \( T_c \).

3 Models of Dataflow and Asynchronous Networks

In this section, we define the history model and the trace model.

Definition 3.1 A computation of a transition system \( T = (L, S, s^0, R, \mathcal{F}) \) is an infinite sequence

\[
    s^0 \xrightarrow{\ell_1} s^1 \xrightarrow{\ell_2} \ldots \xrightarrow{\ell_n} s^n \xrightarrow{\ell_{n+1}} \ldots
\]

of transitions in \( R \), which starts in the initial state \( s^0 \) of \( T \), and satisfies the following property for each fairness set \( F \in \mathcal{F} \) and \( n \geq 0 \): if the set \( F \) is enabled in all states \( s^i \) with \( i \geq n \), then there must exist a \( m \geq n \) such that \( s^m \xrightarrow{\ell_{m+1}} s^{m+1} \in F \).

The trace of a computation \( \Gamma \) is the sequence of non-empty labels in \( \Gamma \).

Intuitively, a computation is a sequence of transitions from the initial state. The role of a fairness set is that a fairness set is empty continuously from some point on in a computation must eventually be selected for execution. For instance, if \( \mathcal{F} \) is the set of non-stuttering transitions, then the transition system must not perform an infinite sequence of only stuttering transitions if a non-stuttering transition is enabled. Due to the requirement that a transition system must have stuttering transitions, terminating executions can be represented by infinite computations which have an infinite tail of only stuttering transitions.

Definition 3.2 Let \( N \) be a node and let \( \Gamma \) be a computation of the transition system \( T_N \).

- the history function of \( \Gamma \) is a mapping from \( E_N \) to the set of (finite and infinite) sequences of data items in \( V \). It maps each channel \( c \in E_N \) to the sequence of data items in labels of form \( \langle c, v \rangle \) in \( \Gamma \).

- The trace of \( \Gamma \) is the sequence of non-empty labels in \( \Gamma \).

We say that \( h \) is a history function of a network \( N \) if \( h \) is the history function of a computation of \( T_N \). Similarly, we say that \( t \) is a trace of a network \( N \) if \( t \) is the trace of a computation of \( T_N \). If \( N \) is a network, we define \( H_N \) as the set of history functions of \( N \) and \( Q_N \) as the set of traces of \( N \).

Definition 3.3 The history model \( \boxplus_H \) is defined as follows: The denotation \( [N]_H \) of a network \( N \) is the triple \( (I_N, O_N, H_N) \).

- The trace model \( \boxplus_T \) is defined as follows: The denotation \( [N]_T \) of a network \( N \) is the triple \( (I_N, O_N, Q_N) \).

Example 3.4 Assume that we consider dataflow networks, i.e., all channels are perfect unbounded FIFO channels. Consider the atomic network with node \( \text{Fairmerge} \) with input channels \( in_1 \) and \( in_2 \) and output channel \( out \). Each history function \( h \) of \( \text{Fairmerge} \) 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
\[ h(in_1) = (1, 3) \quad h(in_2) = (2) \quad h(out) = (1, 2, 3) \]

For a sequence \( t \) of communication events, let \( t_{in_1} \) denote the sequence of data items sent over the channel \( in_1 \) in \( t \), and similarly for \( in_2 \) and \( out \). A trace \( t \) of \textit{Fairmerge} is a finite or infinite sequence of communication events on \( in_1, in_2, \) and \( out \), such that

1. \( t_{out} \) is obtained by merging \( t_{in_1} \) and \( t_{in_2} \), and

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

\[ \langle (in_1, 1), (in_2, 2), (out, 1), (in_1, 3), (out, 2), (out, 3) \rangle \]

\[ \square \]

For many classes of asynchronous networks, among them dataflow networks, it is important that the fairness sets of each output channel require that all data items in an output channel are eventually received by the environment. 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 (since the output of the former network may sometimes be left in the output channel).

\section{Compositionality}

As shown by Brock and Ackerman [BA81], the history model is not compositional for dataflow networks. In this section, we prove that the trace model is indeed compositional for dataflow networks. More precisely, we present an operation for obtaining the denotation of a network, which is the composition of smaller networks, from denotations of the smaller networks, and prove that it is correct with respect to Definition 2.10. Most results of this section will be stated for dataflow networks, but in such a way that they immediately generalize to more general classes of asynchronous networks. This generalization will be considered in Section 7.

If \( C \) is a set of channel names and \( t \) is a sequence of communication events, let \( t[C] \) (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 \( C \).

The following main theorem of this section shows that the trace model is compositional.

\textbf{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 \([N]_T \) is the triple \((I_N, O_N, Q_N)\) where

\begin{align*}
I_N &= \left( \bigcup_i I_{N_i} \right) \setminus \left( \bigcup_i O_{N_i} \right) \\
O_N &= \left( \bigcup_i O_{N_i} \right) \setminus \left( \bigcup_i I_{N_i} \right) \\
Q_N &= \{ t[E_N] : t \in (C_N)^\dagger \text{ and } t[E_{N_i}] \in Q_{N_i} \text{ for } i = 1, \ldots, k \}
\end{align*}

\[ \square \]
In other words, the traces of $N_1 \| \ldots \| N_k$ are obtained by first forming those sequences of communication events on channels in $C_N$ whose projection onto communication events on $E_N$ is a trace of $N_i$ for each $i$, and thereafter deleting the communication events that are not on the external channels of $N_1 \| \ldots \| N_k$.

The proof of Theorem 4.1 consists of three main parts. In the first part, we establish a property of FIFO channels, which intuitively states that the behavior of two channels which are composed in parallel is equivalent to the behavior of a single channel. In the second part, we prove that the traces of $N$ are unchanged if we replace each internal channel of $N$ by two channels composed in parallel. This duplication of channels allows us to split the transition system $T_N$ into smaller networks $T_{N_1}, \ldots, T_{N_k}$. In the third part we can finally use this splitting to prove the relation between the traces of $N$ and those of $N_1, \ldots, N_k$. The third part of the proof is similar to the proof of the rule for composing traces of I/O-automata [Mis84, Jon90, LT87, Sta84].

First we define the property of channels under which Theorem 4.1 holds. A computation $\Gamma$ of a transition system $T$ is rec-persistent if in all but finitely many states of $\Gamma$ a transition with label of form $rec(c, v)$ for some $c, v$ is enabled. A (rec-persistent) trace of $T$ is the sequence of non-$\tau$ labels in a (rec-persistent) computation of $T$. Two transition systems $T_1$ and $T_2$ are equivalent if they have the same sets of traces and the same sets of finite rec-persistent traces.

**Definition 4.2** Let $T_c$ be a channel. We say that $T_c$ is idempotent iff $T_c$ and $T^\circ_c | T^\circ_c$ are equivalent.

Recall that in $T^\circ_c | T^\circ_c$ the data items produced by $T^\circ_c$ are consumed by $T^\circ_c$ in binary synchronizations. Also recall that $T^\circ_c$ has a fairness set which contains all transitions that transfer a data item to $T^\circ_c$. Thus $T^\circ_c | T^\circ_c$ can intuitively be thought of as two copies of the channel $T_c$ connected “in series” with the addition of a fairness constraints which states that transfer of data items between the channels must be treated fairly. Definition 4.2 is thus a formalization of the property that “two channels in a sequence are equivalent to a single channel”.

**Lemma 4.3** For any $c$, the channel FIFO$_c$ is idempotent.

**Proof:** Consider first a computation $\Gamma_2$ of FIFO$^\circ_c | FIFO^\circ_c$. Each state of $\Gamma_2$ is a pair $(\sigma^o, \sigma)$, where $\sigma^o$ and $\sigma$ are sequences of data items. We obtain an equivalent computation of FIFO$^\circ_c$ with the same (rec-persistent) trace by replacing each state in $\Gamma_2$ of form $(\sigma^o, \sigma)$ by the concatenation $\sigma \cdot \sigma^o$ of the two sequences. Conversely, given a computation $\Gamma_1$ of FIFO$^\circ_c$, we obtain an equivalent computation of FIFO$^\circ_c | FIFO^\circ_c$ as follows. First replace each state $\sigma$ in $\Gamma_1$ by the state $\langle \sigma, \langle \rangle \rangle$. This gives rise to transitions of form $\langle \sigma, \langle \rangle \rangle \xrightarrow{\text{send}(c, v) \cdot \sigma \cdot v, \langle \rangle \rangle \theta \rightarrow \langle \sigma \cdot v, \langle \rangle \rangle$, which are indeed transitions of FIFO$^\circ_c | FIFO^\circ_c$.

In the following, let $C_{\text{int}}$ be the set $(\bigcup_{1 \leq i \leq k} I_{N_i}) \cap (\bigcup_{1 \leq i \leq k} O_{N_i})$ of internal channels of $N = N_1 \| \ldots \| N_k$. Assume that $C_{\text{int}}$ is $\{c_1, \ldots, c_t\}$, that $I_N$ is $\{i_{n_1}, \ldots, i_{n_p}\}$, and that $O_N$ is $\{out_1, \ldots, out_q\}$. Let $T_{\text{extchan}} = T_{i_{n_1}} \ldots [T^\circ_{i_{n_p}} | T^\circ_{out_1}] \ldots |T^\circ_{out_q}$ be the parallel composition of the external channels of $N$. Define the transition system $\tilde{T}_N$ by $\tilde{T}_N = T_{N_1} \ldots | T_{N_k}$.

**Lemma 4.4** If for all channel names $c$ the channel $T_c$ is idempotent, then the transition systems $\tilde{T}_N$ and $T_N$ have the same sets of traces.

**Proof:** We first note that $\tilde{T}_N$ can be obtained from $T_N$ by replacing each internal channel $T_{c_i}$ (where $c_i \in C_{\text{int}}$) by the two components $T^\circ_{c_i}$ and $T^\circ_{\tau}$. We shall prove that replacing one internal channel in $C_{\text{int}}$, say $T_{c_1}$, by $T^\circ_{c_1} | T^\circ_{\tau}$ preserves the set of traces. The claim follows by repeating
this proof for each internal channel, and by noting that we can freely use the associativity and commutativity of parallel composition in Proposition 2.3, since each label is in the sort of at most two components.

So, consider a computation $\Gamma$ of $T_N$. Recall that by Definitions 2.9 and 2.10 and Proposition 2.3, $T_N$ can be written as

$$U_{N_1}|\cdots|U_{N_k}|T_{c_1}|\cdots|T_{c_i}|T_\text{extchan}.$$  \hspace{1cm} (1)

Let $T_N'$ be obtained from (1) by replacing one internal channel, say $T_{c_1}$, by $T_{c_1}^\sigma|T_{c_1}^\tau$. We shall prove that $T_N'$ has a computation $\Gamma'$ with the same trace as $\Gamma$. There is a projection $\Gamma_1$ of $\Gamma$ onto the component $T_{c_1}$ which is a computation of $T_{c_1}$, since according to the third requirement of Definition 2.4 no fairness sets of $T_{c_1}$ involve synchronization with other components. Since $T_{c_1}$ is idempotent, $\Gamma_1$ is equivalent to a computation $\Gamma_2$ of $T_{c_1}^\sigma|T_{c_1}^\tau$. Let $\Gamma'$ be obtained from $\Gamma$ by replacing the projection $\Gamma_1$ onto the component $T_{c_1}$ by $\Gamma_2$. When doing this replacement, it may be necessary to insert stuttering transitions into the involved computations to make the synchronizing transitions correspond.

We shall prove that $\Gamma'$ is a computation of $T_N'$ with the same trace as $\Gamma$. That the traces of $\Gamma$ and $\Gamma'$ are the same follows from the observation that the replacement preserves the label of each transition. That each transition in $\Gamma'$ is indeed a transition of $T_N'$ follows from the fact that the sequence of non-$\tau$ labels of $\Gamma_2$ and $\Gamma_1$ are the same, and therefore synchronize in the same way with the other components. To check that $\Gamma'$ satisfies the fairness requirements of $T_N'$, assume that a fairness set $F'$ of $T_N'$ is enabled in all but finitely many states of $\Gamma'$. In the case that $F'$ is a fairness set of $T_{c_1}^\sigma|T_{c_1}^\tau$, a transition in $F'$ must be executed since $\Gamma_2$ is a computation of $T_{c_1}^\sigma|T_{c_1}^\tau$ and $F'$ contains only silent transitions. In the case where $F'$ is a fairness set of the other components which does not contain any transition labeled by $\text{rec}(c,v)$ for some $v$, it follows, using Requirement 2 in Definition 2.6 and the fact that only the projection onto the component $T_{c_1}$ is replaced, that a transition in $F'$ is enabled in a state of $\Gamma'$ iff it is enabled in the corresponding state of $\Gamma$. Hence a transition in $F'$ is subsequently performed in $\Gamma$ and hence in $\Gamma'$. In the last case where $F'$ contains a transition labeled by $\text{rec}(c,v)$ for some $v$, we divide into two cases. In the first case, when there are infinitely many $\text{rec}(c,v)$-labels in $\Gamma$, then by the second requirement on $T_n$ in Definition 2.4, $\Gamma$ must contain infinitely many occurrences of transitions in $F'$. In the second case, where there are only finitely many $\text{rec}(c,v)$-labels in $\Gamma$, we conclude from the first requirement on $T_n$ in Definition 2.4 that for all $w \in V$ there is a transition in $F'$ labeled by $\text{rec}(c,w)$. This together with the definition of equivalence implies that $F'$ must be enabled continuously from some point on in $\Gamma$, which implies that the fairness set $F'$ is treated unfairly in $\Gamma$. This contradicts the assumption that $\Gamma$ is a computation. \hfill \Box

**Lemma 4.5** A sequence $t_N$ is a trace of $T_N = T_{N_1}|\cdots|T_{N_k}$ iff there is a sequence $t \in (C_N)^*$ such that $t_N = t|\{E_N\}$ and for each $i = 1, \ldots, k$ it is the case that $t|\{E_{N_i}\}$ is a trace of $T_{N_i}$.

**Proof:** First assume that $t_N$ is a trace of a computation $\Gamma$ of $T_N$. Let $\Gamma'$ be obtained from $\Gamma$ by labeling with $\langle c_i, v \rangle$ each silent transition which can be the result of synchronizing transitions of two channels $T_{c_1}^\sigma$ and $T_{c_1}^\tau$ via the label $\langle c_i, v \rangle$ according to case 3 in the definition of $R$ in Definition 2.2. If $t = t|\{E_N\}$ we have $t = t|\{E_N\}$. Let $\Gamma_1$ be the projection of $\Gamma'$ onto $T_{N_1}$. We claim that $\Gamma_1$ is a computation of $T_{N_1}$. By the definition of composition, each transition in $\Gamma_1$ is a transition of $T_{N_1}$. We must only check that the fairness requirements of $T_{N_1}$ are satisfied in $\Gamma_1$. So assume that a fairness set $F_i$ of $T_{N_1}$ is enabled continuously from some point on in $\Gamma_1$. The fairness set $F_i$ induces a fairness set $F$ of $T_{N_1}|\cdots|T_{N_k}$. Any transition in $F_i$ is labelled either by $\tau$ or by $\langle c, v \rangle$ for some $c \in O_{N_i}$. A corresponding transition in $F$ will be enabled continuously in $\Gamma$, since by requirement 2 of Definition 2.6 any other component $T_{N_j}$
whose sort contains \((c, v)\) can (since the only other component that \((c, v)\) can affect is an input channel) always perform a transition labeled by \((c, v)\). Therefore, a corresponding transition from \(F\) will be performed in \(\Gamma\), hence some transition from \(F_i\) will eventually be performed in \(\Gamma_i\). Thus \(\Gamma_i\) satisfies the fairness requirements of \(T_{N_i}\).

To prove the theorem in the other direction, assume that there is a \(t \in (C_N)^i\) such that for each \(i\), the sequence \(t_i = t_i(F_{N_i})\) is the trace of a computation \(\Gamma_i\) of \(T_{N_i}\). Without loss of generality, we assume that each \(t_i\) is infinite (by appending infinitely many \(\tau\)-labels if necessary). If \(t = l_1 \ l_2 \ l_3 \ldots\) then \(t_i\) is a subsequence \(l_i^1 \ l_i^2 \ l_i^3 \ldots\) of \(t\). Since each \(l_i^n\) contains one label (of form \((c, v)\)) and is in the sort of at most two components, there is for each \(n\) exactly one or two pairs \((i, m)\) such that \(n = i_m\). The computation \(\Gamma_i\) can be written as

\[
\gamma_i^0 \xrightarrow{l_i^1} \gamma_i^1 \xrightarrow{l_i^2} \ldots \xrightarrow{l_i^m} \gamma_i^{m+1} \ldots
\]

where each \(\gamma_i^m\) is a sequence of internal transitions of \(T_{N_i}\).

Define an extension of an internal transition \(s_i \xrightarrow{\tau} s_i'\) of \(T_{N_i}\) to be a transition

\[
(s_1, \ldots, s_i, \ldots, s_k) \xrightarrow{\tau} (s_1, \ldots, s_i', \ldots, s_k)
\]

of \(T_{N_1} \cdots | T_{N_k}\), which keeps all components except \(s_i\) unchanged. The term extension is extended to sequences of internal transitions of \(T_{N_i}\) in the natural way.

For each \(n = 0, 1, 2, \ldots\), construct the sequence \(\gamma^n\) of transitions of \(T_N = T_{N_1} \cdots | T_{N_k}\) by creating an extension of each (one or two) \(\gamma_i^m\) for which \(n + 1\) is equal to \(i_{m+1}\), and then (if there are two such extensions) concatenating these extensions. The first state of \(\gamma^{n+1}\) should be the the result of performing the transitions corresponding to the label \(l_i^{n+1}\) from the last state of \(\gamma^n\). The first state of \(\gamma^0\) should be the initial state of \(T_N\). We can now conclude that

\[
\Gamma = \gamma^0 \xrightarrow{l} \gamma^1 \xrightarrow{l} \ldots \xrightarrow{l} \gamma^n \xrightarrow{l^{n+1}} \ldots
\]

is a sequence of transitions of \(T_N\). To check that \(\Gamma\) is indeed a computation of \(T_N\), we need to consider the fairness sets of \(T_N\). Assume that a fairness set \(F\) of \(T_N\) is continuously enabled beyond some point in \(\Gamma\). The fairness set \(F\) is the set of transitions of \(T_N\) that have an \(i^{th}\) projection in some fairness set \(F_i\) of some \(T_{N_i}\). It follows that \(F_i\) is enabled continuously beyond some point in \(\Gamma_i\). Hence some transition from \(F_i\) must be performed in \(\Gamma_i\), hence some transition from \(F\) must be performed in \(\Gamma\).

**Proof of Theorem 4.1:** We can now prove Theorem 4.1. The proof for the sets \(I_N\) and \(O_N\) follows immediately from Definition 2.10. The proof for the set \(Q_N\) follows from Lemmas 4.3, 4.4, and 4.5.

5  An Example by Brock and Ackerman

To illustrate the composition operation on dataflow networks 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]. Note that in this section we consider dataflow networks, i.e., assume that all channels are perfect and unbounded.

We consider the networks \(N_1\) and \(N_2\), where \(N_1\) is composed of the nodes \((5, 5)\)-Merge, Buf0, and channels \(a, b, c\). We shall later compose \(N_i\) with the network Plus1 which has channels \(d\) and \(a\), and with the network Fanout with input channel \(c\), and output channels \(d\) and \(e\). The
structure of the nodes and their incident channels is shown in Figure 2. We identify the nodes \textit{Plus1} and \textit{Fanout} with their corresponding atomic networks.

The function of the nodes is intuitively the following:

\(\langle 5,5\rangle\)-\textit{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 \(b\).

\(\text{Buf}_1\) consumes a data item from \(b\) and produces it on \(c\). Thereafter \(\text{Buf}_1\) repeats this behavior once more before terminating.

\(\text{Buf}_2\) first consumes \text{two} data items from \(b\), produces them on \(c\) and thereafter terminates.

\textit{Plus1} adds one to the first incoming data item from \(d\), produces it on \(a\), and terminates.

\textit{Fanout} copies each data that arrives on \(c\) to both \(d\) and \(e\).

The networks \(N_1\) and \(N_2\) have the same denotation in the history model. We shall in the following only consider computations in which only the data item 6 arrives on \(a\). Then \(h\) is a history function of \(N_1\) iff \(h(a) = \langle 6\rangle\) and \(h(c)\) is either \(\langle 5,5\rangle\), \(\langle 5,6\rangle\), or \(\langle 6,5\rangle\). However, if we compose \(N_1\), \textit{Plus1} and \textit{Fanout}, then \(\langle 5,6\rangle\) is a possible history on \(e\) of \(N_1\|\textit{Plus1}\|\textit{Fanout}\) not of \(N_2\|\textit{Plus1}\|\textit{Fanout}\). The reason for this is that when the first data item 5 is produced by \(N_2\|\textit{Plus1}\|\textit{Fanout}\) on \(c\), then the node \(\text{Buf}_2\) has already consumed a second data item. This data item must be a 5, since a 6 has not yet appeared on channel \(a\). Therefore the second data item on \(c\), and hence on \(e\), 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

\(\langle\langle a,6\rangle,\langle c,5\rangle,\langle c,6\rangle\rangle\)
\(\langle\langle a,6\rangle,\langle c,5\rangle,\langle c,5\rangle\rangle\)
\(\langle\langle a,6\rangle,\langle c,6\rangle,\langle c,5\rangle\rangle\)
\(\langle\langle c,5\rangle,\langle a,6\rangle,\langle c,6\rangle\rangle\)
\(\langle\langle c,5\rangle,\langle c,5\rangle,\langle a,6\rangle\rangle\)

The network \(N_1\) has the above traces, and 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 \(\text{Buf}_2\) has already consumed a second data item, which must be a 5.
The traces of Plus1 which match the traces of \( N_1 \) and \( N_2 \) discussed above are

\[
\langle \langle d, 5 \rangle, \langle a, 6 \rangle, \langle d, 5 \rangle \rangle \quad \langle \langle d, 5 \rangle, \langle d, 5 \rangle, \langle a, 6 \rangle \rangle \quad \langle \langle d, 5 \rangle, \langle a, 6 \rangle, \langle d, 6 \rangle \rangle
\]

The traces of Fanout contain the same sequence of data items on the channels \( c, d, \) and \( e \). If we use the composition operator in the trace model, we see that the only trace of \( N_2 || Plus1 || Fanout \) (again considering only those with a 6 on \( a \)) is \( \langle \langle e, 5 \rangle, \langle e, 5 \rangle \rangle \) whereas the network \( N_1 || Plus1 || Fanout \) has the traces \( \langle \langle e, 5 \rangle, \langle e, 5 \rangle \rangle \) and \( \langle \langle e, 5 \rangle, \langle e, 6 \rangle \rangle \).

6 Full Abstraction for Dataflow Networks

In this section, we present the main result of the paper: for dataflow networks 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. In this section we consider only dataflow networks, i.e., assume that all channels are perfect and unbounded.

Let us introduce some terminology. A model (of networks) is a mapping from the set of networks to some set. A context \( C[\cdot] \) is a sequence of applications of the composition operation on a set of networks and a "place holder", denoted by a dot \( \cdot \). A network \( N \) is put into the context by replacing the place holder \( \cdot \) by \( N \).

**Definition 6.1** Let \([\cdot]_D\) and \([\cdot]_O\) be two models of networks. The model \([\cdot]_D\) is said to be fully abstract with respect to \([\cdot]_O\) if for all networks \( N_1 \) and \( N_2 \)

\[
[N_1]_D = [N_2]_D \iff (\forall \text{ contexts } C[\cdot]) \; [C[N_1]]_O = [C[N_2]]_O
\]

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

1. If \([N_1]_D = [N_2]_D\) then \([N_1]_O = [N_2]_O\), i.e., the model \([\cdot]_D\) is more distinguishing than the model \([\cdot]_O\).

2. For all contexts \( C[\cdot] \) we have \([N_1]_D = [N_2]_D \implies [C[N_1]]_D = [C[N_2]]_D\), i.e., the model \([\cdot]_D\) is compositional.

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

To see that these three properties are equivalent to Definition 6.1, note that property 1 follows from the implication \( \implies \), 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 \([N_1]_D = [N_2]_D\), it follows from the implication \( \implies \) that for all contexts \( C[\cdot] \) we have \([C[C[N_1]]]_O = [C[C[N_2]]]_O\), since composition of two contexts yields a context. From the implication \( \iff \) it follows that \([C[N_1]]_D = [C[N_2]]_D\). Property 3 follows directly from the implication \( \iff \) as its contrapositive.

Conversely, assume that \([\cdot]_O\) and \([\cdot]_D\) satisfy the requirements 1 - 3. The implication \( \iff \) is equivalent to property 3. The implication \( \implies \) follows by noting that if \([N_1]_D = [N_2]_D\), then by the fact that \([\cdot]_D\) is compositional (property 2) we infer for all contexts \( C[\cdot] \) that \([C[N_1]]_D = [C[N_2]]_D\). Finally, by property 1, we have for all context \( C[\cdot] \) that \([C[N_1]]_O = [C[N_2]]_O\).
In summary, the definition of full abstraction intuitively means that $[\cdot]_D$ is more distinguishing than $[\cdot]_O$, and that $[\cdot]_D$ distinguishes between networks exactly when that distinction is necessary for attaining compositionality.

We now state the main theorem of the paper.

**Theorem 6.2** 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 Definition 6.1. Property 1 follows directly from Definition 3.3. Property 2 was proven in Theorem 4.1. Property 3 follows from the following Lemma 6.3.

**Lemma 6.3** If $N_1$ and $N_2$ are networks such that $[N_1]_T \neq [N_2]_T$, then there is a context $C[\cdot]$ such that $[[C[N_1]]_H \neq [C[N_2]]_H$.

*Proof:* 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 $Q_{N_1} \neq Q_{N_2}$. Assume that both $N_1$ and $N_2$ have the input channels $in_1, \ldots, in_p$ and the output channels $out_1, \ldots, out_q$. 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 $in_1, \ldots, in_p$ and $out_1, \ldots, out_q$ 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: the context C](image)

Over the channels $in_1, \ldots, in_p$ and $out_1, \ldots, out_q$ are transmitted data items in $V$. Over the 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 $(in_j, v)$ and $(out_i, v)$, where $in_j$ or $out_i$ is one of the channels of $N_i$, and $v$ is an ordinary data item that can be transmitted over that channel. Here we have assumed that the set $V$ of data items is sufficiently rich to allow the representation of such pairs. If $V$ is not sufficiently rich, then instead of transmitting a pair of the form $(in_j, v)$ one can transmit the data item $v$ preceded by a sequence of fixed predetermined length which
encodes the channel \( in_j \). This adjustment only assumes that \( V \) has at least two elements. In the following proof, we assume a sufficiently rich \( V \), but the proof can be adjusted for a smaller \( V \).

The nodes perform the following functions:

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

\textbf{Split} copies each incoming data item from \( b \) onto \( c \). Moreover, if the incoming data item is of the form \( \langle in_j, v \rangle \), then the data item \( v \) is transmitted onto the channel \( in_j \) in addition to the data item \( \langle in_j, v \rangle \) being transmitted over \( c \).

If we identify the nodes \textbf{MergeMark} and \textbf{Split} with their atomic networks, then the context \( C[\cdot] \) can be written as \textbf{MergeMark} \( || \cdot || \textbf{Split} \).

Continuing the proof, recall that we assumed \( Q_{N_1} \not= Q_{N_2} \), i.e., there is a trace \( t \) such that \( t \not\in Q_{N_1} \) but \( t \in Q_{N_2} \) (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 \( \langle in_j, v \rangle \) and \( \langle out_i, v \rangle \). Let \( \text{in}(t) \) be \( t[\langle in_1, \ldots, in_p \rangle] \), i.e., the subsequence of \( t \) consisting of all communication events of the form \( \langle in_j, v \rangle \). Define the history function \( h \) by \( h(a) = \text{in}(t) \) and \( h(c) = t \). The lemma now follows from the following claim:

\[(*) \quad h \in H_{C[N_1]} \quad \text{iff} \quad t \in Q_{N_1} \]

We first give a sketchy proof of the claim \((*)\). First note that the node \textbf{Split} is essentially connected to the network \( N_i \) via FIFO channels. Intuitively, this follows from the observations that (1) for each data item of form \( \langle in_j, v \rangle \) that is consumed by \textbf{Split}, the data item \( v \) is transmitted to \( N_i \) over the FIFO channel \( in_j \), and (2) each data item \( v \) transmitted from \( N_i \) over \( out_i \) reaches \textbf{Split} in the form \( \langle out_i, v \rangle \) over FIFO channels via \textbf{MergeMark}. It follows that tagged data items are consumed by \textbf{Split} in the same order as the corresponding communication events would occur in a computation of \( T_{N_i} \). 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 \( in_1, \ldots, in_p, out_1, \ldots, out_q, a, b, \) and \( c \) as follows:

- Each communication event of the form \( \langle out_i, v \rangle \) in \( t \) is replaced by the sequence of communication events

\[
\langle \langle out_i, v \rangle \rangle \langle b, \langle out_i, v \rangle \rangle \langle c, \langle out_i, v \rangle \rangle
\]

- Each communication event of the form \( \langle in_j, v \rangle \) in \( t \) is replaced by the sequence of communication events

\[
\langle \langle a, \langle in_j, v \rangle \rangle \rangle \langle b, \langle in_j, v \rangle \rangle \langle c, \langle in_j, v \rangle \rangle \langle in_j, d \rangle
\]
We now use Theorem 4.1 to prove that $t'_{\{a,c\}}$ is a trace of $C[N_i]$. This follows if we note that

1. $t'_{\{out_1,\ldots, out_p, a, b\}}$ is a trace of $\textit{MergeMark}$, since it is composed of fragments of the form
   \[ \langle \langle \text{out}_i, v \rangle, \langle \text{in}_j, v \rangle \rangle \]
   and of the form
   \[ \langle \langle a, \langle \text{in}_j, v \rangle \rangle, \langle \text{in}_j, v \rangle \rangle \].

2. $t'_{\{\text{in}_1,\ldots, \text{in}_p, \text{out}_1,\ldots, \text{out}_p\}}$ is a trace of $\textit{Split}$, since it is composed of fragments of the forms
   \[ \langle \langle \text{in}_j, v \rangle, \langle \text{out}_i, v \rangle \rangle \]
   and
   \[ \langle \langle b, \langle \text{in}_j, v \rangle \rangle, \langle \text{in}_j, v \rangle, \langle \text{in}_j, v \rangle \rangle \].

3. $t'_{\{\text{in}_1,\ldots, \text{in}_p, \text{out}_1,\ldots, \text{out}_p\}}$ is a trace of $N_i$, since it is equal to $t$. The if-part of (*) now follows by observing that the sequence of data items transmitted over $c$ in $t'_{\{a,c\}}$ is $t$, and that the sequence of data items transmitted over $a$ in $t'_{\{a,c\}}$ is $\text{in}(t)$.

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 $h(a) = \text{in}(t)$ and $h(c) = t$. Then there is a computation $\Gamma'$ of $T_{C[N_i]}$ in which the sequence of items transmitted over $c$ is $t$. Hence the sequence of data items produced by $\textit{Split}$ is also $t$. We shall construct a computation $\Gamma$ of $T_{N_i}$ in which $t$ is the sequence of communication events.

We transform $\Gamma'$ into a computation of $T_{N_i}$ as follows:

1. Each internal transition (derived from a transition of $T_{\textit{Split}}$) which consumes a data item of form $\langle \text{out}_i, v \rangle$ from channel $b$ is labeled by $\{(\text{out}_i, v)\}$.

2. Each internal transition (derived from a transition of $T_{\textit{Split}}$) which consumes a data item of form $\langle \text{in}_j, v \rangle$ from channel $b$ (and produces $v$ on $\text{in}_j$) is labeled by $\{(\text{in}_j, v)\}$.

3. For each $i$, replace the $i$th state of $\Gamma'$ (which is a state of $T_{C[N_i]}$) by a state $\sigma$ of $T_{N_i}$. The state $\sigma$ is obtained by replacing for each $l$ the state component of $\text{out}_l$ by the concatenation of the sequence of data items $v$ in data items of form $\langle \text{out}_l, v \rangle$ in channel $b$ in $\sigma'$ and the sequence of data items in $\text{out}_l$ in $\sigma'$. For the other components (i.e., the network $N_i$ and channels $\text{in}_1, \ldots, \text{in}_p$), $\sigma$ agrees with $\sigma'$.

4. Compress all transitions labeled by communication events on the channels $a$ and $c$, and all transitions resulting from transitions of $T_{\textit{MergeMark}}$ into a single state.

The result is a sequence $\Gamma$. The intuition here is that $N_i$ and the channels $\text{in}_1, \ldots, \text{in}_p$ perform the same sequence of transitions in $\Gamma$ as in $\Gamma'$, whereas each channel $\text{out}_l$ in $\Gamma$ simulates the concatenation of the sequence of data items in items of form $\langle \text{out}_l, v \rangle$ in $b$ and the channel $\text{out}_l$ in $\Gamma'$.

Note that each transition of $T_{\textit{Split}}$ in $\Gamma'$ is replaced by a corresponding transition with the communication event that was produced by $T_{\textit{Split}}$ in $\Gamma'$. It follows that the sequence of communication events in $\Gamma$ is the sequence of events that is produced on $c$ by $T_{\textit{Split}}$, which is exactly $t$. To see that $\Gamma$ is indeed a computation of $T_{N_i}$, note that the component $T_{N_i}$ and the channels $\text{in}_1, \ldots, \text{in}_p$ perform exactly the same sequence of transitions in $\Gamma$ as in $\Gamma'$. Also note that the channels $\text{out}_1, \ldots, \text{out}_4$ participate in the same sequence of transitions in $\Gamma$ as in $\Gamma'$, with the difference that transitions with rec-labels may occur in a later position but still in the same relative order. \(\square\)
7 Asynchronous Networks

In this section, we consider classes of networks that communicate over classes of asynchronous channels which are not necessarily FIFO. We thus remove the restriction that for each channel name $c$ the channel $T_c$ should be an unbounded FIFO channel. Such classes of asynchronous networks could be networks that communicate via unordered channels, via ordered but lossy channels, etc. Under some restrictions on the channels, we prove that the trace model is still compositional – the proof is the same as for dataflow networks. However, when generalizing the full abstraction result, we are no longer sure that there are unbounded FIFO channels to carry out the construction in the proof of Theorem 6.2. We will instead use a construction which allows many other classes of asynchronous channels but requires rather powerful nodes.

7.1 Compositionality

The following theorem generalizes the compositionality result to networks that communicate via asynchronous channels.

**Theorem 7.1** Assume that for each channel name $c$ the channel $T_c$ is idempotent. Then the composition rule of Theorem 4.1 holds.

**Proof:** Follows directly from Lemmas 4.4 and 4.5 which are used to prove Theorem 4.1.

Examples of idempotent channels are: unordered channels, FIFO channels that can lose data items.

7.2 Full Abstraction

When proving full abstraction for more general classes of asynchronous networks, we can no longer use the construction in the proof of Theorem 6.2, since it uses unbounded FIFO channels. Instead we use a different construction which requires a rather weak property of channels but more powerful nodes.

If $\Gamma$ is a computation of a transition system $T_c^o$, let the receive-sequence of $\Gamma$, denoted $\vec{\rho}(\Gamma)$, be the sequence of data items in rec-labels in $\Gamma$, and let the send-sequence of $\Gamma$, denoted $\vec{\sigma}(\Gamma)$, be the sequence of data items in send-labels in $\Gamma$.

**Definition 7.2** A channel $T_c^o$ is **live** if there is a computation $\Gamma$ of $T_c^o$ such that either

1. there is a sequence $\vec{\sigma}_1$ of data items such that $\vec{\sigma}(\Gamma)$ is a proper prefix of $\vec{\sigma}_1$ and no computation of $T_c^o$ has send-sequence $\vec{\sigma}_1$ and receive-sequence $\vec{\rho}(\Gamma)$, or

2. $\vec{\sigma}(\Gamma)$ is infinite and no computation of $T_c^o$ has a send-sequence which is a finite prefix of $\vec{\sigma}(\Gamma)$ and receive-sequence $\vec{\rho}(\Gamma)$.

Intuitively, a channel $T_c^o$ is live if it has a computation $\Gamma$ with certain send- and receive-sequences such that at any moment during the computation, the node that sends data items over the channel can decide to produce a send-sequence which is different from that of $\Gamma$ and be sure that the receive sequence will be different from that of $\Gamma$. In the first case of the definition, if the sending node is sending the sequence $\vec{\sigma}(\Gamma)$, then it can achieve a different receive-sequence by instead extending its sequence of sent data items to $\vec{\sigma}_1$. In the second case of the definition,
if the sending node is sending the infinite sequence $\overline{\sigma}(\Gamma)$, then it can achieve a different receive-sequence by instead deciding to send only a finite prefix of $\overline{\sigma}(\Gamma)$.

An example of a live channel is a channel which does not deliver data items if no data items are transmitted, and which delivers at least one data item whenever an infinite sequence of data items are transmitted. For such a channel we can in Definition 7.2 take $\Gamma$ to be a computation with no send- or rec-labels, and $\overline{\sigma}_1$ to be an infinite sequence of data items. Another example of a live channel is a channel which nondeterministically loses of delivers any data item, but does not create new data items. For such a channel one can take $\Gamma$ to be a computation with an infinite send-sequence and an infinite receive-sequence (no messages lost); then any finite send-sequence will result in a finite receive-sequence.

**Theorem 7.3** Assume that for each channel name $c$, $T_c^\circ$ is live and $T_c$ is idempotent. Then the trace model is fully abstract with respect to the history model. □

**Proof:** In the proof we must, just as in the proof of Theorem 6.2, find a context by which two nodes, $N_1$ and $N_2$, which are distinguished in the trace model can be distinguished in the history model. Such a context is shown in Figure 4. Since $N_1$ and $N_2$ have different denotations in the trace model, there must be a trace $t$ such that $t \notin Q_{N_1}$ but $t \in Q_{N_2}$ (or vice versa). Let $\Gamma'$ be as in Definition 7.2 for $T_c^\circ$. The node Detect can send and receive a sequence of data items over the internal channels that corresponds to the sequence $t$. On $a$, Detect initially intends to send the sequence $\overline{\sigma}(\Gamma')$ of data items. The node Detect will thereafter change its intention, and change the sequence sent over $a$ to $\overline{\sigma}_1$ in case 1, and terminating the sequence in in case 2 of Definition 7.2 if one of the following conditions occur.

1. Detect receives a data item which is not the next expected one according to $t$.
2. For a sufficiently long time, Detect waits to receive a data item which does not appear.

The first condition is easy to detect. The second can be formally represented by fairness sets in Detect: one fairness set changes the state of Detect from a state when an odd-length proper prefix of $t$ has been recorded into a state where the sequence sent over $a$ is changed, and one fairness set does the same for even-length prefixes of $t$. The context will guarantee that if the sequence $\overline{\sigma}(\Gamma')$ is sent over $a$ then the sequence of data items recorded on internal channels is $t$. 

![Figure 4: Context in the Proof of Full Abstraction](image-url)
and further ensure the existence of a computation where the sequence of data items recorded on internal channels is \( t \) and the sequence \( \vec{\sigma}(\Gamma) \) is sent over \( a \).

In summary, if the sequence \( \vec{\rho}(\Gamma) \) is the history of channel \( a \) in some computation, then since \( T_0^a \) is live the sequence \( \vec{\sigma}(\Gamma) \) is sent over \( a \) and hence the sequence of data items recorded on internal channels is \( t \). On the other hand there is a computation where the sequence \( \vec{\rho}(\Gamma) \) is the history of channel \( a \), where the sequence \( \vec{\sigma}(\Gamma) \) is sent over \( a \), and where the sequence of data items recorded on internal channels is \( t \). It follows that the sequence of data items \( \vec{\rho}(\Gamma) \) is a history of \( C[N_i] \) iff \( t \in Q_{N_i} \), which concludes the proof.

In the proof of Theorem 7.3, we note that if the channel \( a \) is live according to the second criterion of Definition 7.2 then the node \( \text{Detect} \) can be constructed to be deterministic. Namely, it will after each successful transmission or reception on the internal channels send a portion of the infinite sequence \( \vec{\sigma}(\Gamma) \). If the sequence observed on the internal channels is not \( t \), then the transmission on \( a \) will terminate automatically both if the wrong data items arrives or if an expected data item does not arrive (in the second case we have a deadlock situation). In the case that \( t \) is finite, \( \text{Detect} \) will send the remaining part of \( \vec{\sigma}(\Gamma) \) after the last item in \( t \).

We also make a remark in the case that the channels of the networks under consideration are well-behaved enough that when a sequence of data items is transmitted over a channel then there is one scenario when the sequence can be reconstructed at the other end and that in all scenarios the receiving end knows when it has received the correct sequence. This case occurs for instance with unbounded FIFO channels and with lossy but sometimes correct FIFO channels by using unbounded sequence numbers to tag the data items. In this case \( \text{Detect} \) does not have to store \( t \) and \( \vec{\sigma}(\Gamma) \) since they can be transmitted to \( \text{Detect} \) over external input channels. Thus \( \text{Detect} \) can be realized with a small number of states.

The last two observations are essentially those that are used for dataflow networks by Russell [Rus89] to prove that the full abstraction result holds for dataflow networks, even when the class of nodes is restricted to any class which includes the deterministic nodes.

**Remark:** The full abstraction proof in Theorem 7.3 can in some sense be said to subsume the proof of Theorem 6.2. However, the node \( \text{Detect} \) is rather powerful in the sense that it must "know" a trace \( t \) which distinguishes between the sets of traces of two networks. Our definition of networks puts no restrictions on the sets of states and transitions of a node. We therefore do not know whether it is always the case that one can find a distinguishing trace that can be represented using a finite amount of memory.

On the other hand, the context in the proof of Theorem 6.2 can be constructed from simple components that are often referred to in the literature on dataflow. The node \( \text{MergeMark} \) can be constructed from a Fairmerge node with several input channels and nodes that perform simple transformations on data items, and the node \( \text{Split} \) is simple to construct. We regard it as important to point out that for dataflow networks, which is the most considered class of asynchronous networks, there is a context which is intuitively simple to construct, and that the proof of Theorem 6.2 is therefore of independent interest.
8 Related Work

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

A 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 non-deterministic 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 \( \langle \text{out}_1, d \rangle \) and \( \langle \text{out}_2, d \rangle \) is distinguished from a network which either relates \( \langle \text{out}_1, d \rangle \) before \( \langle \text{out}_2, d \rangle \) 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 data items from input channels in firings, and output events in traces to represent the production of data items onto output channels. 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.

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 \( ((V^*)^\omega)^m \to \mathcal{P}((V^*)^n) \), 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. The model by Kok is isomorphic to the trace model. More precisely, \( (w_1, \ldots, w_n) \) is an element in Kok's model precisely if \( Q_N \) contains the trace \( \langle \text{in}_1, v_1[1] \rangle \ldots \langle \text{in}_m, v_m[1] \rangle \langle \text{out}_1, w_1[1] \rangle \ldots \langle \text{out}_n, w_n[1] \rangle \langle \text{in}_1, v_1[2] \rangle \ldots \langle \text{in}_m, v_m[2] \rangle \ldots \)

A more elaborate comparison between these models appears in [JK89].
Rabinowitch and Trakhtenbrot [RT89] prove an analogous full abstraction result for a different model, which denotes a node by the set of finite prefixes of its traces. Their model is fully abstract with respect to a model which contains finite prefixes of the history model considered in this paper.

In our proof of full abstraction for dataflow networks, we make use of a fair merge node, which is non-deterministic and needs fairness. Russell [Rus89] has presented a different construction of a context which only adds deterministic nodes, thus proving that the full abstraction result is still valid when one considers an arbitrary subset of nodes which includes the deterministic nodes.

Our trace model is similar to a model presented by Misra and Chandy [Mis84], in our earlier work [Jon85, Jon87a], and by Lynch and Tuttle [LT87]. These models are defined for a model of distributed systems, called I/O-automata in [LT87] and I/O-systems in [Sta84, Jon87a, Jon87b]. Our formal definition of a network can in fact be regarded as a special case of an I/O-automaton. 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].

9 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. We have also generalized the compositionality and full abstraction results to a wider class of networks that communicate over asynchronous channels.

Our full abstraction results for dataflow networks and asynchronous networks indicate that traces is the appropriate basis for reasoning about the behavior of asynchronous networks, e.g. in a method for specification and verification. The trace model captures both safety and liveness properties of a network. The full abstraction result shows that the trace model in an optimal way provides both abstraction from internal details of the behavior of a network and capability to reason about the network's behavior in a context. Properties of traces are indeed used as a semantical criterion of correctness in proofs of distributed algorithms and protocols in e.g. [Mis84, Jon85, Jon87a, LT87, Sta84].

A property of Kahn's original model [Kah74] which is not shared by our trace model, is that the denotation of a network can be computed from the denotations of its components by a iteration to a fixedpoint. It appears difficult to incorporate such constructions into a model for networks that exhibit nondeterminism and fairness. Approaches to solving this problem appear in e.g. [Bro86, SN83, KP85, Rus90].

Acknowledgments

I am grateful to Joost Kok for stimulating discussions and ideas. I also would like to thank Lars-Åke Fredlund and Joachim Parrow for critical reading of the manuscript and for many fruitful discussions and comments.
References


