SICS/R-90/9012

A Fully Abstract Trace Model for Dataflow and Asynchronous Networks

by

Bengt Jonsson

SICS research report
R90012
ISSN 0283-3638

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 16^{th} Annual ACM Symposium on Principles of Programming Languages, Austin, Texas, January 1989.
This work was supported in part by the Swedish Board for Technical Development (STU) under contract no. 86-4250, and also under contract No. 89-01220P as part of Esprit BRA project SPEC

†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, Hoa85, 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 $\langle 1,2,3\rangle$ and $\langle 1,3,5,7\rangle$, this function produces the sequence $\langle 2,5,8\rangle$.

For nondeterministic dataflow networks, the straight-forward 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 $[\![\cdot]\!]_{\mathcal{D}}$ is fully abstract with respect to a model $[\![\cdot]\!]_{\mathcal{O}}$ if $[\![\cdot]\!]_{\mathcal{D}}$ has added precisely enough information to $[\![\cdot]\!]_{\mathcal{O}}$ 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 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 $\mathcal{L}$ of *labels*, which does not include the *silent label* τ.

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

L is a set of *labels* in $\mathcal{L}$, 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 $l \in (L \cup \tau)$.

$\mathcal{F} \subseteq 2^R$ 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 τ 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 $T_i = \langle L_i, S_i, s_i^0, R_i, \mathcal{F}_i \rangle$ be a transition system. The *parallel composition* of T_1 and T_2, denoted $T_1 \mid T_2$, is the transition system $\langle L, S, s^0, R, \mathcal{F} \rangle$, where

$L = (L_1 \setminus L_2) \cup (L_2 \setminus L_1)$,

$S = S_1 \times S_2$,

$s^0 = \langle s_1^0, s_2^0 \rangle$,

R consists of all triples of form $\langle s_1, s_2 \rangle \xrightarrow{l} \langle s_1', s_2' \rangle$ 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 $\langle s_1, s_2 \rangle \xrightarrow{l} \langle s_1', s_2' \rangle$ onto the 1^{st} component, and that $s_2 \xrightarrow{\tau} s_2$ is a projection onto the 2^{nd} component; in case 2 we define the projections vice versa, and in case 3 we say that $s_i \xrightarrow{l'} s_i'$ is a projection of $\langle s_1, s_2 \rangle \xrightarrow{l} \langle s_1', s_2' \rangle$ onto the i^{th} component for $i = 1, 2$.

$\mathcal{F}$ is obtained as follows: for each fairness set $F_i \in \mathcal{F}_i$ where i is 1 or 2, there is a fairness set $F \in \mathcal{F}$ consisting of the set of all transitions $\langle s_1, s_2 \rangle \xrightarrow{l} \langle s_1', s_2' \rangle$ for which a projection onto the i^{th} component is in F_i. □

Intuitively, the sort of $T_1 \mid T_2$ 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 $\langle L_1, S_1, s_1^0, R_1, \mathcal{F}_1 \rangle$ and $\langle L_2, S_2, s_2^0, R_2, \mathcal{F}_2 \rangle$ are *isomorphic* if there is an isomorphism between the sets of states S_1 and S_2 under which also s_1^0 and s_2^0, R_1 and R_2, and $\mathcal{F}_1$ and $\mathcal{F}_2$ 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:* $T_1 \mid T_2$ *and* $T_2 \mid T_1$ *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,
- $\langle c, v \rangle$ 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 $\langle c, v \rangle$, respectively, for $c \in C$ and $v \in V$.

Definition 2.4 A *node* n is a triple $\langle I_n, O_n, T_n \rangle$ 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 $\langle L, S, s^0, R, \mathcal{F} \rangle$ 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 $F \in \mathcal{F}$ 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 $w \in V$. □

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 $s \xrightarrow{rec(c,v)} s'$ is: "when s is the state of the node, then it may consume the data item v from input channel c and change its state to s'." Analogously, a transition $s \xrightarrow{send(c,v)} s'$ means that "when s is the state of the node, then it may produce the data item v onto output channel c and change its state to s'." 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 *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. *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

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

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

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

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

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

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. □

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 $s \xrightarrow{send(c,v)} s'$ is a transition in R.
3. All transitions in any fairness set in $\mathcal{F}$ are labeled by τ.

□

Recall that the label $send(c, v)$ intuitively represents the insertion of the data item v into the channel, and that $rec(c, v)$ 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 $\langle L, S, s^0, R, \mathcal{F}\rangle$, define

T_c^o as $\langle L, S, s^0, R, \mathcal{F} \cup \{R_\rho\}\rangle$, where R_ρ is the set of transitions in R with a *rec*-label,

T_c^σ as T_c but with all labels of form *send*(c, v) replaced by $\langle c, v\rangle$,

T_c^ρ as T_c^o but with all labels of form *rec*(c, v) replaced by $\langle c, v\rangle$.

□

Intuititively, T_c^o 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^σ 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^ρ is obtained from T_c^o 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 $FIFO_c = \langle L, S, s^0, R, \mathcal{F}\rangle$, 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 $(v \cdot \sigma) \xrightarrow{rec(c,v)} \sigma$ for $\sigma \in V^*$ and $v \in V$, and
- all transition of form $\sigma \xrightarrow{send(c,v)} (\sigma \cdot v)$ for $\sigma \in V^*$ and $v \in V$.

$\mathcal{F} = \emptyset$.

□

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 $\langle I_n, O_n, T_n\rangle$ 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 $\langle I_N, O_N, T_N\rangle$, where $I_N = I_n$, and $O_N = O_n$, and T_N is the transition system

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

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. □

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^ρ as the channel and for an input channel name c we take T_c^σ 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_{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* $N_1 \| \ldots \| N_k$ of $N_1, \ldots, N_k$ is the tuple $N = \langle I_N, O_N, T_N \rangle$, where

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

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

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

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

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

□

Intuitively, the composition $N = N_1 \parallel \cdots \parallel N_k$ 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 $U_{N_1}, \cdots, U_{N_k}$, 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 $T_{out_1}^\rho, \ldots, T_{out_q}^\rho$ 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.

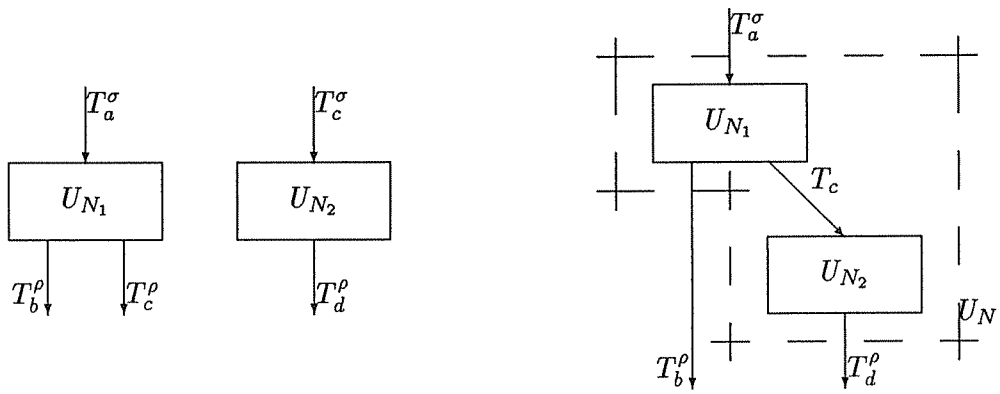

Figure 1: Two networks T_{N_1} and T_{N_2} (left), and a network with their composition T_N (right).

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_a^σ, T_b^ρ, and T_c^ρ. To the right is shown the transition system T_N of the composition $N = N_1 \| N_2$. 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 $T = \langle L, S, s^0, R, \mathcal{F} \rangle$ is an infinite sequence

$$s^0 \xrightarrow{l^1} s^1 \xrightarrow{l^2} \cdots \xrightarrow{l^n} s^n \xrightarrow{l^{n+1}} \cdots$$

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 $s^m \xrightarrow{l^{m+1}} s^{m+1} \in F$.

The *trace* of a computation Γ is the sequence of non-empty labels in Γ. □

Intuitively, a computation is a sequence of transitions from the initial state. The rôle of a fairness set is that a fairness set which is enabled 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 Γ be a computation of the transition system T_N.

- the *history function* of Γ 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 Γ.
- The *trace* of Γ is the sequence of non-empty labels in Γ.

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* $[\![\cdot]\!]_H$ is defined as follows: The denotation $[\![N]\!]_H$ of a network N is the triple $\langle I_N, O_N, H_N \rangle$.
- The *trace model* $[\![\cdot]\!]_T$ is defined as follows: The denotation $[\![N]\!]_T$ of a network N is the triple $\langle I_N, O_N, Q_N \rangle$. □

Example 3.4 Assume that we consider dataflow networks, i.e., all channels are perfect unbounded FIFO channels. Consider the atomic network with node *Fairmerge* with input channels in_1 and in_2 and output channel *out*. Each history function h of *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

$h(in_1) = \langle 1, 3 \rangle \qquad h(in_2) = \langle 2 \rangle \qquad h(out) = \langle 1, 2, 3 \rangle$

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 $Fairmerge$ 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

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

□

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).

4 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 $t\lceil_C$ (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.

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

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 $\langle I_N, O_N, Q_N \rangle$ where*

$$I_N = \left(\bigcup_i I_{N_i}\right) \setminus \left(\bigcup_i O_{N_i}\right)$$

$$O_N = \left(\bigcup_i O_{N_i}\right) \setminus \left(\bigcup_i I_{N_i}\right)$$

$$Q_N = \{t\lceil_{(E_N)} \mid t \in (C_N)^\dagger \text{ and } t\lceil_{(E_{N_i})} \in Q_{N_i} \text{ for } i = 1, \ldots, k\}$$

□

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_i} 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 Γ of a transition system T is *rec-persistent* if in all but finitely many states of Γ 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-τ 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 $T_c^\rho | T_c^\sigma$ are equivalent. □

Recall that in $T_c^\rho | T_c^\sigma$ the data items produced by T_c^ρ are consumed by T_c^σ in binary synchronizations. Also recall that T_c^ρ has a fairness set which contains all transitions that transfer a data item to T_c^σ. Thus $T_c^\rho | T_c^\sigma$ 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 Γ_2 of $FIFO_c^\rho | FIFO_c^\sigma$. Each state of Γ_2 is a pair $\langle \sigma^o, \sigma \rangle$, where σ^o and σ are sequences of data items. We obtain an equivalent computation of $FIFO_c$ with the same (*rec*-persistent) trace by replacing each state in Γ_2 of form $\langle \sigma^o, \sigma \rangle$ by the concatenation $\sigma \cdot \sigma^o$ of the two sequences. Conversely, given a computation Γ_1 of $FIFO_c$, we obtain an equivalent computation of $FIFO_c^\rho | FIFO_c^\sigma$ as follows. First replace each state σ in Γ_1 by the state $\langle \sigma, \langle\rangle \rangle$. This gives rise to transitions of form $\langle \sigma, \langle\rangle \rangle \xrightarrow{send(c,v)} \langle \sigma \cdot v, \langle\rangle \rangle$ which are not transitions of $FIFO_c^\rho | FIFO_c^\sigma$, so we replace them by the sequence of transitions $\langle \sigma, \langle\rangle \rangle \xrightarrow{send(c,v)} \langle \sigma, v \rangle \xrightarrow{\emptyset} \langle \sigma \cdot v, \langle\rangle \rangle$, which are indeed transitions of $FIFO_c^\rho | FIFO_c^\sigma$. □

In the following, let C_{int} be the set $(\bigcup_{1 \le i \le k} I_{N_i}) \cap (\bigcup_{1 \le i \le k} O_{N_i})$ of internal channels of $N = N_1 \| \cdots \| N_k$. Assume that C_{int} is $\{c_1, \ldots, c_r\}$, that I_N is $\{in_1, \ldots, in_p\}$, and that O_N is $\{out_1, \ldots, out_q\}$. Let $T_{extchan} = T_{in_1}^\sigma | \cdots | T_{in_p}^\sigma | T_{out_1}^\rho | \cdots | T_{out_q}^\rho$ be the parallel composition of the external channels of N. Define the transition system $\tilde{T_N}$ by $\tilde{T_N} = T_{N_1} | \cdots | T_{N_k}$.

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 $c_i \in C_{int}$) by the two components $T_{c_i}^\rho$ and $T_{c_i}^\sigma$. We shall prove that replacing one internal channel in C_{int}, say T_{c_1}, by $T_{c_1}^\rho | T_{c_1}^\sigma$ 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 associtivity 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 Γ of T_N. Recall that by Definitions 2.9 and 2.10 and Proposition 2.3, T_N can be written as

$$U_{N_1}|\cdots|U_{N_k}|T_{c_1}|\cdots|T_{c_r}|T_{extchan} \ . \tag{1}$$

Let T'_N be obtained from (1) by replacing one internal channel, say T_{c_1}, by $T^{\rho}_{c_1}|T^{\sigma}_{c_1}$. We shall prove that T'_N has a computation Γ' with the same trace as Γ. There is a projection Γ_1 of Γ 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, Γ_1 is equivalent to a computation Γ_2 of $T^{\rho}_{c_1}|T^{\sigma}_{c_1}$. Let Γ' be obtained from Γ by replacing the projection Γ_1 onto the component T_{c_1} by Γ_2. When doing this replacement, it may be necessary to insert stuttering transitions into the involved computations to make the synchronizing transitions correpond.

We shall prove that Γ' is a computation of T'_N with the same trace as Γ. That the traces of Γ and Γ' are the same follows from the observation that the replacement preserves the label of each transition. That each transition in Γ' is indeed a transition of T'_N follows from the fact that the sequence of non-τ labels of Γ_2 and Γ_1 are the same, and therefore synchronize in the same way with the other components. To check that Γ' 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 Γ'. In the case that F' is a fairness set of $T^{\rho}_{c_1}|T^{\sigma}_{c_1}$, a transition in F' must be executed since Γ_2 is a computation of $T^{\rho}_{c_1}|T^{\sigma}_{c_1}$ 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 $rec(c,v)$ 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 Γ' iff it is enabled in the corresponding state of Γ. Hence a transition in F' is subsequently performed in Γ and hence in Γ'. In the last case where F' contains a transition labeled by $rec(c,v)$ for some v, we divide into two cases. In the first case, when there are infinitely many $rec(c,v)$-labels in Γ, then by the second requirement on T_n in Definition 2.4, Γ must contain infinitely many occurrences of transitions in F'. In the second case, where there are only finitely many $rec(c,v)$-labels in Γ, 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 $rec(c,w)$. This together with the definition of equivalence implies that F' must be enabled continuously from some point on in Γ, which implies that the fairness set F' is treated unfairly in Γ. This contradicts the assumption that Γ is a computation. □

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

Proof: First assume that t_N is a trace of a computation Γ of $\tilde{T_N}$. Let Γ' be obtained from Γ by labeling with $\langle c_i, v\rangle$ each silent transition which can be the result of synchronizing transitions of two channels $T^{\rho}_{c_i}$ and $T^{\sigma}_{c_i}$ via the label $\langle c_i, v\rangle$ according to case 3 in the definition of R in Definition 2.2. If t is the sequence of labels in Γ' we have $t_n = t\lceil_{E_N}$. Let Γ_i be the projection of Γ' onto T_{N_i}. We claim that Γ_i is a computation of T_{N_i}. By the definition of composition, each transition in Γ_i is a transition of T_{N_i}. We must only check that the fairness requirements of T_{N_i} are satisfied in Γ_i. So assume that a fairness set F_i of T_{N_i} is enabled continuously from some point on in Γ_i. 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 τ or by $\langle c, v\rangle$ for some $c \in O_{N_i}$. A corresponding transition in F will be enabled continuously in Γ, since by requirement 2 of Definition 2.6 any other component T_{N_j}

whose sort contains $\langle c, v\rangle$ can (since the only other component that $\langle c, v\rangle$ can affect is an input channel) always perform a transition labeled by $\langle c, v\rangle$. Therefore, a corresponding transition from F will be performed in Γ, hence some transition from F_i will eventually be performed in Γ_i. Thus Γ_i satisfies the fairness requirements of T_{N_i}.

To prove the theorem in the other direction, assume that there is a $t \in (C_N)^\dagger$ such that for each i, the sequence $t_i = t\lceil_{(E_{N_i})}$ is the trace of a computation Γ_i of T_{N_i}. Without loss of generality, we assume that each t_i is infinite (by appending infinitely many τ-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^n contains one label (of form $\langle c, v\rangle$) 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 Γ_i can be written as

$$\gamma_i^0 \xrightarrow{l^{i_1}} \gamma_i^1 \xrightarrow{l^{i_2}} \cdots \xrightarrow{l^{i_m}} \gamma_i^m \xrightarrow{l^{i_{m+1}}} \cdots$$

where each γ_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

$$\langle s_1, \ldots, s_i, \ldots, s_k\rangle \xrightarrow{\tau} \langle s_1, \ldots, s_i', \ldots, s_k\rangle$$

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 γ^n of transitions of $\tilde{T_N} = T_{N_1}|\cdots|T_{N_k}$ by creating an extension of each (one or two) γ_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 γ^{n+1} should be the the result of performing the transitions corresponding to the label l^{n+1} from the last state of γ^n. The first state of γ^0 should be the initial state of $\tilde{T_N}$. We can now conclude that

$$\Gamma = \gamma^0 \xrightarrow{l^1} \gamma^1 \xrightarrow{l^2} \cdots \xrightarrow{l^n} \gamma^n \xrightarrow{l^{n+1}} \cdots$$

is a sequence of transitions of $\tilde{T_N}$. To check that Γ is indeed a computation of $\tilde{T_N}$, we need to consider the fairness sets of $\tilde{T_N}$. Assume that a fairness set F of $\tilde{T_N}$ is continuously enabled beyond some point in Γ. The fairness set F is the set of transitions of $\tilde{T_N}$ 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 Γ_i. Hence some transition from F_i must be performed in Γ_i, hence some transition from F must be performed in Γ. □

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_i is composed of the nodes $\langle 5, 5\rangle$-*Merge*, Buf_i, and channels a, b, and c. We shall later compose N_i with the network *Plus*1 which has channels d and a, and with the network *Fanout* with input channel c, and output channels d and e. The

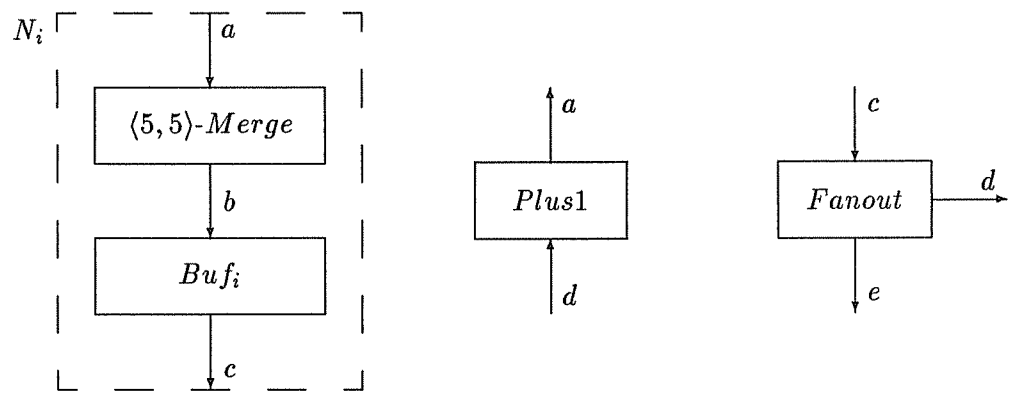

Figure 2: The Example by Brock and Ackerman.

structure of the nodes and their incident channels is shown in Figure 2. We identify the nodes $Plus1$ and $Fanout$ with their corresponding atomic networks.

The function of the nodes is intuitively the following:

$\langle 5,5\rangle$-$Merge$ 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 d, produces it on a, and terminates

$Fanout$ 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_i 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_i, $Plus1$ and $Fanout$, then $\langle 5,6\rangle$ is a possible history on e of $N_1||Plus1||Fanout$ but not of $N_2||Plus1||Fanout$. The reason for this is that when the first data item 5 is produced by $N_2||Plus1||Fanout$ 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 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

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

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 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

$$\langle\langle d,5\rangle,\langle a,6\rangle,\langle d,5\rangle\rangle \qquad \langle\langle d,5\rangle,\langle d,5\rangle,\langle a,6\rangle\rangle \qquad \langle\langle d,5\rangle,\langle a,6\rangle,\langle d,6\rangle\rangle$$

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 $[\![\cdot]\!]_{\mathcal{D}}$ and $[\![\cdot]\!]_{\mathcal{O}}$ be two models of networks. The model $[\![\cdot]\!]_{\mathcal{D}}$ is said to be *fully abstract* with respect to $[\![\cdot]\!]_{\mathcal{O}}$ if for all networks N_1 and N_2

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

□

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

1. If $[\![N_1]\!]_{\mathcal{D}} = [\![N_2]\!]_{\mathcal{D}}$ then $[\![N_1]\!]_{\mathcal{O}} = [\![N_2]\!]_{\mathcal{O}}$,
   i.e., the model $[\![\cdot]\!]_{\mathcal{D}}$ is more distinguishing than the model $[\![\cdot]\!]_{\mathcal{O}}$.

2. For all contexts $C[\cdot]$ we have $[\![N_1]\!]_{\mathcal{D}} = [\![N_2]\!]_{\mathcal{D}} \implies [\![C[N_1]]\!]_{\mathcal{D}} = [\![C[N_2]]\!]_{\mathcal{D}}$,
   i.e., the model $[\![\cdot]\!]_{\mathcal{D}}$ is compositional.

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

To see that these three properties are equivalent to Definition 6.1, 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 $[\![N_1]\!]_{\mathcal{D}} = [\![N_2]\!]_{\mathcal{D}}$, it follows from the implication $\Longrightarrow$ that for all contexts $C[\cdot]$ we have $[\![C[C'[N_1]]]\!]_{\mathcal{O}} = [\![C[C'[N_2]]]\!]_{\mathcal{O}}$, since composition of two contexts yields a context. From the implication $\Longleftarrow$ it follows that $[\![C'[N_1]]\!]_{\mathcal{D}} = [\![C'[N_2]]\!]_{\mathcal{D}}$. Property 3 follows directly from the implication $\Longleftarrow$ as its contrapositive.

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

In summary, the definition of full abstraction intuitively means that $[\![\cdot]\!]_{\mathcal{D}}$ is more distinguishing than $[\![\cdot]\!]_{\mathcal{O}}$, and that $[\![\cdot]\!]_{\mathcal{D}}$ 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.

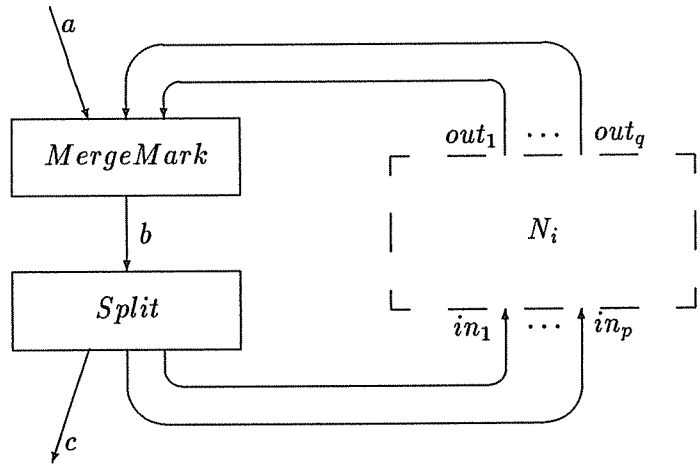

Figure 3: the context C

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 $\langle in_j, v\rangle$ and $\langle out_l, v\rangle$, where in_j or out_l 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 $\langle in_j, v\rangle$ 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:

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_l is first transformed into the pair $\langle out_l, v\rangle$, 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.

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 *MergeMark* and *Split* with their atomic networks, then the context $C[\cdot]$ can be written as *MergeMark* $\| \cdot \|$ *Split*.

Continuing the proof, recall that we assumed $Q_{N_1} \neq Q_{N_2}$, i.e., there is a trace t such that $t \notin Q_{N_1}$ 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 $\langle out_l, v\rangle$. Let $in(t)$ be $t\lceil_{\{in_1,\ldots,in_p\}}$, 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 $h(a) = in(t)$ and $h(c) = t$. The lemma now follows from the following claim:

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

We first give a sketchy proof of the claim $(*)$. First note that the node *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 *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_l reaches *Split* in the form $\langle out_l, v\rangle$ 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 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 $\langle out_l, v\rangle$ in t is replaced by the sequence of communication events

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

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

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

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. $t'\lceil_{\{out_1,\ldots,out_q,a,b\}}$ is a trace of *MergeMark*, since it is composed of fragments of the form $\langle\ \langle out_l, d\rangle\ \langle b, \langle out_l, v\rangle\rangle\ \rangle$ and of the form $\langle\ \langle a, \langle in_j, v\rangle\rangle\ \langle b, \langle in_j, v\rangle\rangle\ \rangle$,

2. $t'\lceil_{\{in_1,\ldots,in_p,b,c\}}$ is a trace of *Split*, since it is composed of fragments of the forms $\langle\ \langle b, \langle out_l, v\rangle\rangle\ \langle c, \langle out_l, v\rangle\rangle\ \rangle$ and $\langle\ \langle b, \langle in_j, v\rangle\rangle\ \langle c, \langle in_j, v\rangle\rangle\ \langle in_j, d\rangle\ \rangle$,

3. $t'\lceil_{\{in_1,\ldots,in_p,out_1,\ldots,out_q\}}$ 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 $t'\lceil_{\{a,c\}}$ is t, and 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 $T_{C[N_i]}$ 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 T_{N_i} in which t is the sequence of communication events.

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

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

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

3. For each i, replace the i^{th} state of Γ' (which is a state of $T_{C[N_i]}$) by a state σ of T_{N_i}. The state σ is obtained by replacing for each l the state component of out_l by the concatenation of the sequence of data items v in data items of form $\langle out_l, v\rangle$ in channel b in σ' and the sequence of data items in out_l in σ'. For the other components (i.e., the network N_i and channels $in_1, \ldots, in_p$), σ agrees with σ'.

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

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

Note that each transition of T_{Split} in Γ' is replaced by a corresponding transition with the communication event that was produced by T_{Split} in Γ'. It follows that the sequence of communication events in Γ is the sequence of events that is produced on c by T_{Split}, which is exactly t. To see that Γ is indeed a computation of T_{N_i}, note that the component T_{N_i} and the channels $in_1, \ldots, in_p$ perform exactly the same sequence of transitions in Γ as in Γ'. Also note that the channels $out_1, \ldots, out_q$ participate in the same sequence of transitions in Γ as in Γ', with the difference that transitions with *rec*-labels may occur in a later position but still in the same relative order. □

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

Definition 7.2 A channel T_c^o is *live* if there is a computation Γ 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 Γ 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 Γ and be sure that the receive sequence will be different from that of Γ. In the first case of the definition, if the sending node is sending the sequence $\vec{\sigma}(\Gamma)$, then it can acheive 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 equence $\vec{\sigma}(\Gamma)$, then it can acheive a different receive-sequence by instead deciding to send only a finite prefix of $\vec{\sigma}(\Gamma)$.

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 Γ to be a computation with no *send-* or *rec*-labels, and $\vec{\sigma}_1$ 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 Γ 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^o 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

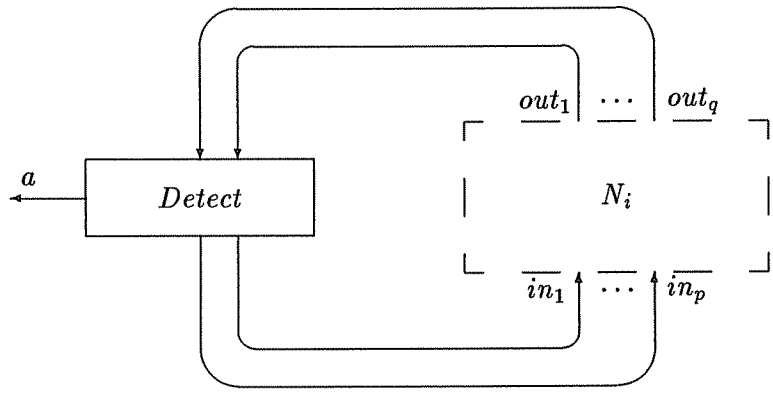

Figure 4: Context in the Proof of Full Abstraction

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 Γ be as in Definition 7.2 for T_a^o. The node *Detect* can sends 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 $\vec{\sigma}(\Gamma)$ of data items. The node *Detect* will thereafter change its intention, and change the sequence sent over a to to $\vec{\sigma}_1$ 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 $\vec{\sigma}(\Gamma)$ is sent over a then the sequence of data items recorded on internal channels is t,

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_a^o 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 *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, *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 *Detect* does not have to store t and $\vec{\sigma}(\Gamma)$ since they can be transmitted to *Detect* over external input channels. Thus *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 *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 *MergeMark* can be constructed from a Fairmerge node with several input channnels and nodes that perform simple transformations on data items, and the node *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 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 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 $((V^*)^\omega)^m \longrightarrow \mathcal{P}(((V^*)^\omega)^n)$, 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. The model by Kok is isomorphic to the trace model. More precisely, $\langle w_1, \ldots, w_n\rangle$ is an element in Kok's model precisely if Q_N contains the trace

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

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

[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*, volume 107 of *Lecture Notes in Computer Science*, pages 252–259. Springer Verlag, 1981.

[BHR84] S.D. Brookes, C.A.R. Hoare, and A.W. Roscoe. A theory of communicating sequential processes. *Journal of the ACM*, 31(3):560–599, 1984.

[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 pur 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, Amsterdam, 1983. North-Holland.

[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.

[dNH84] R. de Nicola and M. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

[JK89] B. Jonsson and J. Kok. Comparing two fully absttract dataflow models. In *Proc. PARLE 89*, volume 365 of *Lecture Notes in Computer Science*, pages 217–234. Springer Verlag, 1989.

[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.

[Jon90] B. Jonsson. A hiearchy of compositional models of I/O-automata. In Rovan, editor, *Proc. Mathematical Foundations of Computer Science*, volume 452 of *Lecture Notes in Computer Science*, pages 347–354. Springer Verlag, 1990.

[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*, volume 206 of *Lecture Notes in Computer Science*, pages 237–249. Springer Verlag, 1986.

[Kok87] J.N. Kok. A fully abstract semantics for data flow nets. In *Proc. PARLE, LNCS 259*, volume 259 of *Lecture Notes in Computer Science*, 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.

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

[MC81] J. Misra and K. M. Chandy. Proofs of networks of processes. *IEEE Trans. on Software Engineering*, SE-7(4):417–426, July 1981.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice-Hall, 1989.

[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.

[OH86] E.R. Olderog and C.A.R. Hoare. Specification-oriented semantics for communicating processes. *Acta Informatica*, 23(1):9–66, 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.

[RT89] A. Rabinovich and B.A. Trakhtenbrot. Nets of processes and data flow. In de Bakker, de Roever, and Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, volume 354 of *Lecture Notes in Computer Science*, pages 574–602. Springer Verlag, 1989.

[Rus89] J.R. Russell. Full abstraction for nondeterministic dataflow networks. In *Proc. 30th Annual Symp. Foundations of Computer Science*, 1989.

[Rus90] J.R. Russell. On oraclizable networks and Kahn's principle. In *Proc. 17th ACM Symp. on Principles of Programming Languages*, 1990.

[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.

[Zwi89] J. Zwiers. *Compositionality, Concurrency and Partial Correctness*, volume 321 of *Lecture Notes in Computer Science.* Springer Verlag, 1989.