**A Fully Abstract Trace Model for
Dataflow Networks**
by
Bengt Jonsson

A Fully Abstract Trace Model for Dataflow Networks*

Bengt Jonsson

Swedish Institute of Computer Science

Box 1263, S-164 28 Kista, SWEDEN

bengt@sics.se

SICS Research Report 88016

November 1988

Abstract

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

*This Research report is a revised version of a paper that will appear under the same title in the Proceedings of the 16th Annual ACM Symposium on Principles of Programming Languages, Austin, Texas, January 1989. Copyright is with ACM.

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

1 Introduction

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

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

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

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

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

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

In this paper, we present a model of nondeterministic dataflow networks, which is fully abstract with respect to the history model. In order to obtain a compositional model, we must to the history model add information about how a network behaves under different ordering constraints. Our model provides precisely this information.

We denote a network by the set of its traces. A trace is a linearly ordered sequence of communication events that may appear on its input and output channels during a computation. A communication event represents the appearance of a data item on a certain channel.

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

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 model of distributed systems, called I/O-automaton in [LT87] and I/O-system in [Sta84,Jon87a,Jon87b]. Our formal definition of a dataflow network will in fact be a special case of an I/O-automaton (I/O-system). The results of this paper appear in the context of I/O-systems in the author's thesis [Jon87a].

This paper is organized as follows: In the next section, we give the basic definitions of dataflow networks. In section 3, we define our trace model and the history model. In section 4, we prove that our model is compositional. In section 5, we illustrate our model by an example due to Brock and Ackerman. In section 6, we state and prove the main theorem: our model is fully abstract with respect to the history model. Section 7 contains a comparison of our model with the fully abstract model presented by Kok. Section 8 contains a comparison with other related models, and the last section contains conclusions.

2 Dataflow Networks

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

2.1 Informal Presentation

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

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

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

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

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

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

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

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

2.2 Formal Presentation

In this subsection, we give a formal definition of dataflow networks by using labeled transition systems. Labeled transition systems is a general framework, which has often been used for operational descriptions of computing systems, (e.g. [Plo81,MP81]). It should be noted that we use labeled transition systems as a means for obtaining a formal definition of dataflow networks, upon which we can build subsequent definitions and proofs. The details are a formal counterpart of the informal presentation in subsection 2.1.

The structure of our definition is the following: First, we define a dataflow network as having a state. The state of a dataflow network represent the state of its nodes and contents of internal channels. It does not represent the contents of input and output channels. The state is is changed by firings in which data items are consumed from input channels and data items are produced on output channels. Second, we define a configuration of a dataflow network as consisting of the state of the network together with contents of input and output channels. The configuration is changed by transitions. Transitions can have labels that represent the reception or transmission of data values over input or output channels.

We assume a set V of *data items*, ranged over by d . Let V^* denote the set of finite sequences of data items in V , ranged over by q . The sequence consisting of the elements $d_1, \dots, d_n$ is denoted $\langle d_1, \dots, d_n \rangle$. The empty sequence is denoted by $\langle \rangle$. The concatenation of two sequences q and q' is denoted by $q \cdot q'$, and $q \leq q'$ denotes that q is an initial prefix of q' . We assume a set of *channel names*, ranged over by c , *in*, and *out*.

Definition 2.1 A *dataflow network* N is a tuple $\langle I_N, O_N, S_N, s_N^0, R_N, \mathcal{F}_N \rangle$ 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 $I_N \cap O_N = \emptyset$.

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

$s_N^0 \in S_N$ is an *initial state*.

R_N is a set of *firings*. A firing is a tuple $\langle s, \chi_{in}, s', \chi_{out} \rangle$ where $s, s' \in S_N$, where χ_{in} is a mapping from I_N to V^* , and where χ_{out} is a mapping from O_N to V^* .

$\mathcal{F}_N \subseteq \mathcal{P}(R_N)$ is a finite collection of *fairness sets*. Each fairness set F is a subset of the set R_N of firings, subject to the following constraints:

1. For each input channel $in \in I_N$, if F contains a firing $\langle s, \chi_{in}, s', \chi_{out} \rangle$ which satisfies the property that $\chi_{in}(in) \neq \langle \rangle$, then F contains all firings in R_N with that property.
2. There is a bound on the length of $\chi_{in}(in)$ for input channels in and firings $\langle s, \chi_{in}, s', \chi_{out} \rangle$ in F .

□

Intuitively, a state in S_N represents the states of the nodes and contents of internal channels (if any) of the network. It does not represent the contents of input and output channels. The intuitive meaning of a firing $\langle s, \chi_{in}, s', \chi_{out} \rangle$ is: “when s is the state of the network and the sequence $\chi_{in}(in)$ is a prefix of the contents of each input channel in , then from each input channel in the sequence $\chi_{in}(in)$ may be consumed, while the network changes its state to s' and the sequence $\chi_{out}(out)$ is produced on each output channel out ”. We shall use ϕ to range over firings. A fairness set represents a set of firings which may not be neglected indefinitely in executions of the network. For instance, if $\mathcal{F}_N = \{R_N\}$, i.e., the set of all firings is one fairness set, then the network will continue to perform firings indefinitely, until no more firings are possible (e.g. for lack of input). The constraint on fairness sets is a technical requirement which excludes certain pathological fairness sets.

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

Example 2.2 Consider a network *Fairmerge*. Intuitively, it consumes data items from the channels in_1 and in_2 and produces them onto out . The network is “fair”, i.e. each data item on an input channel will eventually be consumed and copied onto an output channel. The set of input channels is $\{in_1, in_2\}$ and the set of output channels is $\{out\}$. The network has one state, denoted s , which then of course is the initial state. The set of firings is the union of the set

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

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

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

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

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

Definition 2.3 The dataflow networks $N_1, \dots, N_k$ 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 $N_1, \dots, N_k$.

Let $N_1, \dots, N_k$ be compatible networks. Let C_{int} be the set $(\bigcup_{1 \leq i \leq k} I_{N_i}) \cap (\bigcup_{1 \leq i \leq k} O_{N_i})$ of channels that are to be connected. The *composition* $N_1 \parallel \dots \parallel N_k$ of $N_1, \dots, N_k$ is the tuple $N = \langle I_N, O_N, S_N, s_N^0, R_N, \mathcal{F}_N \rangle$ where

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

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

S_N is the set of mappings from $\{1, \dots, k\} \cup C_{int}$, which map each index $i \in \{1, \dots, k\}$ to a state of N_i (in S_{N_i}), and map each channel in C_{int} to a sequence in V^* .

$s_N^0 \in S_N$ maps each index i to $s_{N_i}^0$ (the initial state of N_i), and maps each channel $c \in C_{int}$ to the empty sequence $\langle \rangle$.

$R_N = \bigcup_{1 \leq i \leq k} \bigcup_{\phi_i \in R_{N_i}} embed_i(\phi_i)$, where $embed_i$ is defined as follows: For a firing $\phi_i = \langle s_i, \chi_{in}^i, s'_i, \chi_{out}^i \rangle$ of N_i we define $embed_i(\phi_i)$ as the set of tuples $\langle s, \chi_{in}, s', \chi_{out} \rangle$, where $s, s' \in S_N$, and where $\chi_{in} (\chi_{out})$ maps the set $I_N (O_N)$ to V^* , for which

- $s(i) = s_i$ and $s'(i) = s'_i$ and $s(j) = s'(j)$ for $j \neq i$
- If $c \in C_{int}$ then
 - if $c \in I_{N_i}$ then $\chi_{in}^i(c) \leq s(c)$, and $s(c) = \chi_{in}^i(c) \cdot s'(c)$, i.e., the initial sequence $\chi_{in}^i(c)$, which is consumed from c in the firing ϕ_i , has been removed from the sequence $s(c)$ to get $s'(c)$
 - if $c \in O_{N_i}$ then $s'(c) = s(c) \cdot \chi_{out}^i(c)$, i.e., the sequence $\chi_{out}^i(c)$, which is produced on c in the firing ϕ_i , has been appended to the sequence $s(c)$ to get $s'(c)$
 - if $c \notin (I_{N_i} \cup O_{N_i})$ then $s'(c) = s(c)$.

- $\chi_{in}(c) = \chi_{in}^i(c)$ if $c \in (I_{N_i} \cap I_N)$ and $\chi_{in}(c) = \langle \rangle$ otherwise.
- $\chi_{out}(c) = \chi_{out}^i(c)$ if $c \in (O_{N_i} \cap O_N)$ and $\chi_{out}(c) = \langle \rangle$ otherwise.

$\mathcal{F}_N = \bigcup_{1 \leq i \leq k} \{ \text{embed}_i(F) \mid F \in \mathcal{F}_{N_i} \}$, where embed_i here denotes the natural extension of embed_i to sets of firings. $\square$

Intuitively, a state s in S_N gives the state of each node and the contents of each internal channel: $s(i)$ for $i \in \{1, \dots, k\}$ gives the current state of node N_i , and $s(c)$ for $c \in C_{int}$ gives the current contents of the internal channel c . The function embed_i transforms a firing of the subnetwork N_i from its representation as a firing of N_i to its representation as a firing of N . Intuitively, the definition of R_N states that a firing of the network $N_1 \parallel \dots \parallel N_k$ is a firing of a subnetwork N_i . The fairness sets of N is the union of the fairness sets of the subnetworks $N_1, \dots, N_k$, modulo using embed_i to change the representations of the firings in the fairness sets.

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

Definition 2.4 Let N be a dataflow network.

- A *configuration* σ of a dataflow network N is a pair $\langle s, \rho \rangle$, where s is a state in S_N , and ρ is mapping from the external channels E_N of N to sequences in V^* .
- The *initial configuration* σ_N^0 of N is the pair $\langle s_N^0, \rho_N^0 \rangle$, where s_N^0 is the initial state of N , and where ρ_N^0 maps each channel in E_N to $\langle \rangle$.
- A *communication event* of N is a pair $\langle c, d \rangle$, where c is an external channel in E_N , and d is a data item in V . An *event* is either a communication event or the silent event τ . We use e to range over events.
- An *internal transition* of N is a triple of the form $\langle s, \rho \rangle \xrightarrow{\tau} \langle s', \rho' \rangle$, where $\langle s, \rho \rangle$ and $\langle s', \rho' \rangle$ are configurations of N such that there is a firing $\langle s, \chi_{in}, s', \chi_{out} \rangle$ of N for which
 - if $c \in I_N$ then $\chi_{in}(c) \leq \rho(c)$, and $\rho(c) = \chi_{in}(c) \cdot \rho'(c)$, i.e., the initial sequence $\chi_{in}(c)$ has been removed from the sequence $\rho(c)$ to get $\rho'(c)$.
 - If $c \in O_N$ then $\rho'(c) = \rho(c) \cdot \chi_{out}(c)$.
- An *input transition* of N is a triple $\langle s, \rho \rangle \xrightarrow{\langle c, d \rangle} \langle s, \rho' \rangle$ such that $c \in I_N$, and $\rho' = \rho[c \mapsto (\rho(c) \cdot d)]$, i.e., ρ' is the same state as ρ except that d has been added at the end of c .
- An *output transition* of N is a triple $\langle s, \rho \rangle \xrightarrow{\langle c, d \rangle} \langle s, \rho' \rangle$ such that $c \in O_N$, and $\rho = \rho'[c \mapsto (d \cdot \rho'(c))]$, i.e., d is the first data item in $\rho(c)$ and ρ' is the same state

as ρ except that d has been removed from the front of c .

- In the following, the term *fairness set* will also be used to denote the set of internal transitions obtained from the firings in a fairness set of N . Thus a fairness set can either be a set of firings or the corresponding set of internal transitions. $\square$

Intuitively, in a configuration $\langle s, \rho \rangle$ the first component s represents the internal state of the network, and ρ represents the contents of each external channel. A communication event represents the exchange with the environment of a data item on a certain channel. An internal transition represents a firing of the network, in which the configuration is changed accordingly. An input transition represents the reception of a data item on an input channel, in which the data item is added to the end of the contents of that input channel. An output transition represents the transmission of a data item on an output channel, in which the data item is removed from the contents of the output channel in question.

3 Models of Dataflow Networks

3.1 Computations, Histories, and Traces

Definition 3.1 A (internal, input or output) transition $\sigma_1 \xrightarrow{e} \sigma_2$ is *enabled* in a state σ if $\sigma = \sigma_1$. $\square$

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

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

of transitions, which satisfies the following requirements:

1. σ^0 is the initial configuration of N .
2. Each triple $\sigma^{n-1} \xrightarrow{e^n} \sigma^n$ in the sequence is a (internal, input, or output) transition of N .
3. For each fairness set $F \in \mathcal{F}_N$, if there is a configuration σ^n in the computation such that a transition in F is enabled in all subsequent states σ^m with $m \geq n$ of the computation, then there must exist a subsequent transition $\sigma^m \xrightarrow{e^{m+1}} \sigma^{m+1} \in F$ with $m \geq n$ in the computation.
4. For each $c \in O_N$, if $\sigma^n(c) \neq \langle \rangle$ for some state σ^n in the computation, then the computation must contain a subsequent output transition labeled by a communication event on c . $\square$

Intuitively, requirement 3 states that no fairness set may be continuously enabled beyond some point in a computation, without some transition from that fairness set being taken. Note that for finite computations, requirement 3 is equivalent to the condition that no transition from any fairness set of N be enabled in the last state. Requirement 4 intuitively states that each data item in each output channel of N is eventually delivered to the environment.

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

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

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

H_N as the set of history functions of N

T_N as the set of traces of N . □

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

An important detail is that in each computation, all data items in an output channel are eventually output from that channel. 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).

3.2 Models

Definition 3.4 Let N be a dataflow network.

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

Example 3.5 Consider the network *Fairmerge* with input channels in_1 and in_2 and output channel out .

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

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

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

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

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

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

An example of a trace is the sequence

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

□

4 Compositionality

As shown by Brock and Ackerman [BA81], the history model is not compositional for dataflow networks in general. In this section, we prove that the trace model is indeed compositional. We present an operation for obtaining the denotation of a large network, which is the composition of smaller network, from denotations of the smaller networks, and prove that it is correct with respect to definition 2.3.

Let us introduce some notation. If C is a set of channels, 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 .

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

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

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

$$T_N = \{ t|_{(I_N \cup O_N)} \mid t \in (C_N)^\dagger \text{ and } t|_{(E_{N_i})} \in T_{N_i} \text{ for } i = 1, \dots, k \}$$

In other words, to obtain the traces of $N_1 \parallel \dots \parallel N_k$, first form the “synchronized merges” of the traces of $N_1, \dots, N_k$ (synchronizing on events on channels that are common to a pair of subnetworks), then delete the communication events on those channels that are internal channels of $N_1 \parallel \dots \parallel N_k$.

Proof: The proof for the sets I_N and O_N follows immediately from definition 2.3. We therefore concentrate on the set T_N .

We begin with some definitions. Let C_{ij} be $O_{N_i} \cap I_{N_j}$, i.e., the set of channels that connect the subnetwork N_i to the subnetwork N_j . Let $C_{int} = \bigcup_{i,j} C_{ij}$ be the set of channels that are connected when composing $N_1, \dots, N_k$. A *configuration tuple* is a tuple $\langle \sigma_1, \dots, \sigma_k \rangle$, where σ_i is a configuration of N_i . For a configuration $\langle s, \rho \rangle$ of N , define *split*($\langle s, \rho \rangle$) as the configuration tuple $\langle \langle s_1, \rho_1 \rangle, \dots, \langle s_k, \rho_k \rangle \rangle$ where $s_i = s(i)$, where

$\rho_i(c) = \langle \rangle$ for $c \in \cup_j C_{ij}$, and where $\rho_i(c) = \rho(c)$ for $c \notin \cup_j C_{ij}$. An opposite mapping *combine* from configuration tuples is defined as follows: *combine*($\langle \langle s_1, \rho_1 \rangle, \dots, \langle s_k, \rho_k \rangle \rangle$) is the configuration $\langle s, \rho \rangle$ of N where $s(i) = s_i$ for $1 \leq i \leq k$, where $\rho(c) = \rho_i(c) \cdot \rho_j(c)$ for $c \in C_{ij}$, and where $\rho(c) = \rho_i(c)$ for $c \in E_{N_i} \setminus C_{int}$. Intuitively, if $\langle \sigma_1, \dots, \sigma_k \rangle$ represents a “current” configuration of each subnetwork $N_1, \dots, N_k$ then *combine*($\langle \sigma_1, \dots, \sigma_k \rangle$) represents the corresponding “current” configuration of N , and vice versa for *split*. Here, a complication arises for internal channels in C_{int} since each channel c in C_{ij} occurs both in the configuration of N_i and in that of N_j . In this case, *combine* appends their contents in the natural way, whereas *split* lets c be in the configuration of N_i be empty and lets c in the configuration of N_j have the contents of c in the configuration of N .

A *tuple transition* is a triple $\langle \sigma_1, \dots, \sigma_k \rangle \xrightarrow{e} \langle \sigma'_1, \dots, \sigma'_k \rangle$ where $\sigma_1, \dots, \sigma_k$ and $\sigma'_1, \dots, \sigma'_k$ are configuration tuples, and e is an event of N . A *tuple computation* is a sequence

$$\langle \sigma_1^0, \dots, \sigma_k^0 \rangle \xrightarrow{e_1} \langle \sigma_1^1, \dots, \sigma_k^1 \rangle \xrightarrow{e_2} \dots \xrightarrow{e_n} \langle \sigma_1^n, \dots, \sigma_k^n \rangle$$

of tuple transitions. If the last configuration tuple of the tuple computation γ is the same as the first configuration of the tuple computation γ' then γ and γ' may be appended, denoted $\gamma \cdot \gamma'$, by identifying the common configuration tuple.

Let $\langle d_1 d_2 \dots d_l \rangle$ be a finite sequence of data items in V . Let $c \in C_{ij}$. A tuple computation is said to *transfer* $\langle d_1 d_2 \dots d_l \rangle$ over c if it is of the form

$$\langle \langle s, \rho^0 \rangle_1, \dots, \langle s, \rho^0 \rangle_k \rangle \xrightarrow{\langle c, d_1 \rangle} \langle \langle s, \rho^1 \rangle_1, \dots, \langle s, \rho^1 \rangle_k \rangle \xrightarrow{\langle c, d_2 \rangle} \dots \xrightarrow{\langle c, d_l \rangle} \langle \langle s, \rho^l \rangle_1, \dots, \langle s, \rho^l \rangle_k \rangle$$

and for each $m = 1, \dots, l$, $\langle \rho_1^m, \dots, \rho_k^m \rangle$ is equal to $\langle \rho_1^{m-1}, \dots, \rho_k^{m-1} \rangle$ except that $\rho_j^m(c) = \rho_j^{m-1}(c) \cdot d_m$ and $\rho_i^m(c) = tl(\rho_i^{m-1}(c))$, i.e., d_m has been transferred from ρ_i^{m-1} to ρ_j^m .

An *extension* of an internal transition $\sigma_i \xrightarrow{\tau} \sigma'_i$ of N_i is a tuple transition

$$\langle \sigma_1, \dots, \sigma_i, \dots, \sigma_k \rangle \xrightarrow{\tau} \langle \sigma_1, \dots, \sigma'_i, \dots, \sigma_k \rangle$$

which keeps all components except σ_i unchanged. The term *extension* is extended to computations of N_i with only internal transitions in the natural way.

Let $\sigma \xrightarrow{\tau} \sigma'$ be an internal transition of N which is obtained from some firing of N which is obtained from some firing $\phi_i = \langle s_i, \chi_{in}^i, s'_i, \chi_{out}^i \rangle$ of some N_i . Let $\cup_j C_{ij}$ be the set of channels $\{c_1, \dots, c_n\}$. Define *split*($\sigma \xrightarrow{\tau} \sigma'$) as the tuple computation $\gamma_0 \cdot \gamma_1 \cdot \dots \cdot \gamma_n$ where

- γ_0 is an extension of the transition $\sigma_i \xrightarrow{\tau} \sigma'_i$ which is obtained from ϕ_i , and the first configuration tuple of γ_0 is *split*(σ).
- For $1 \leq h \leq n$, the tuple computation γ_h transfers the sequence $\chi_{out}^i(c_h)$ over c_h .

It follows that the first configuration tuple of *split*($\sigma \xrightarrow{\tau} \sigma'$) is *split*(σ) and that its last configuration tuple is *split*(σ').

We can now begin the proof in one direction. First assume that $t \in T_N$. Then t is the trace of a computation Γ of N . In the proof, we first construct a tuple computation Γ' in which the sequence of events is a sequence $t' \in (C_N)^\dagger$ with $t' \upharpoonright_{(I_N \cup O_N)} = t$. From Γ' we extract a computation Γ_i of N_i , the trace of which is $t' \upharpoonright_{E_{N_i}}$.

We perform the following two transformations on Γ :

1. Replace each internal transition $\sigma \xrightarrow{\tau} \sigma'$ in Γ by $split(\sigma \xrightarrow{\tau} \sigma')$ to obtain Γ' .
2. In Γ' , first replace all configuration tuples $\langle \sigma_1, \dots, \sigma_k \rangle$ by their i th component σ_i , and thereafter compress all triples $\sigma_i \xrightarrow{e} \sigma_i$ which are not transitions of N_i to a single configuration σ_i .

It is obvious that the sequence of communication events t' of Γ' satisfies $t' \upharpoonright_{(I_N \cup O_N)} = t$ (since we have added communication events on internal channels of N), and that the trace t_i of Γ_i satisfies $t_i = t' \upharpoonright_{E_{N_i}}$ (since we have deleted all communication events not in E_{N_i}).

We must check that Γ_i is a computation of N_i . From the construction of Γ_i , it is not difficult to prove that Γ_i satisfies the first two requirements of definition 3.2. To verify requirement 3, that of fairness, note that if a fairness set F of internal transitions is enabled continuously in Γ_i , then the corresponding set ($embed_i(F)$) of internal transitions must be enabled continuously in Γ (since the input channels of N_i have at least as many data items in Γ as in Γ_i). Therefore an internal transition from that set must be performed in Γ , and hence by construction in Γ_i . To verify requirement 4, note that if c is an output channel of both N_i and N , then requirement 4 for c must be satisfied in Γ , hence in Γ_i . If $c \in c_{ij}$, then requirement 4 is satisfied by the definition of $split(\sigma \xrightarrow{\tau} \sigma')$ for internal transitions $\sigma \xrightarrow{\tau} \sigma'$ of N : all data items put into $\sigma_i(c)$ are immediately transferred to $\sigma_j(c)$.

To prove the theorem in the other direction, assume that there is a $t' \in (C_N)^\dagger$ such that $t' \upharpoonright_{(E_{N_i})} \in T_{N_i}$ for all $i = 1, \dots, k$. That is, for each i , $t_i = t' \upharpoonright_{(E_{N_i})}$ is the trace of a computation Γ_i of N_i . If $t' = \langle e^1 e^2 e^3 \dots \rangle$ then t_i is a subsequence $\langle e^{i_1} e^{i_2} e^{i_3} \dots \rangle$ of t' .

We first treat the case when t_i is infinite for all $i = 1, \dots, k$. We can then write Γ_i as

$$\gamma_i^0 \xrightarrow{e^{i_1}} \gamma_i^1 \xrightarrow{e^{i_2}} \dots \xrightarrow{e^{i_m}} \gamma_i^m \xrightarrow{e^{i_{m+1}}} \dots$$

where each γ_i^m is a sequence of internal transitions of N_i . We transform Γ_i in two steps:

1. First construct a tuple computation Γ' as

$$\gamma^0 \xrightarrow{e^1} \gamma^1 \xrightarrow{e^2} \dots \xrightarrow{e^n} \gamma^n \xrightarrow{e^{n+1}} \dots$$

where the first configuration tuple of γ^0 is $split(\sigma_N^0)$ (here σ_N^0 is the initial configuration of N), and γ^n is a concatenation of extensions of those (one or two) γ_i^m for which $n + 1$ is equal to i_{m+1} .

2. Second, transform Γ' into Γ by replacing each configuration tuple $\langle \sigma_1, \dots, \sigma_k \rangle$ by $combine(\langle \sigma_1, \dots, \sigma_k \rangle)$, and thereafter compressing each transition which is labeled by a communication event on a channel in C_{int} to a single configuration.

It follows that the sequence of communication events in Γ' is t' , and that the trace of Γ is $t' \upharpoonright_{(E_N)}$.

We must check that Γ is indeed a computation of N . From the construction of Γ , it is not difficult to prove that Γ satisfies requirements 1, 2, and 4 of definition 3.2. To verify requirement 3, assume that a fairness set F of N is continuously enabled beyond

some point in Γ , but that no internal transition from the set is performed beyond that point in Γ . The fairness set F must be obtained from a fairness set F_i of some N_i . It follows that no internal transition in F_i is performed beyond some point in Γ_i . If C_{cons} is the set of channels from which some transition in F (or equivalently F_i) consumes data items, it follows by the first requirement on fairness sets in definition 2.1 that no data item is consumed from a channel in C_{cons} beyond some point both in Γ and in Γ_i . From requirement 4 of definition 3.2 and the second requirement on fairness sets in definition 2.1 it follows that beyond some point in Γ_i , the contents of each $c \in C_{cons}$ becomes sufficiently long so that each firing which is continuously enabled in Γ is also continuously enabled in Γ_i . It follows that also F_i must be enabled beyond some point in Γ_i , which contradicts requirement 3 for Γ_i .

In the case when t_i is finite, Γ_i ends with a (finite or infinite) sequence of internal transitions. Extensions of these transitions must then be interspersed into Γ' before the two steps of the transformation can be applied.. $\square$

5 An Example by Brock and Ackerman

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

We consider the networks N_1 and N_2 , where N_i has nodes $\langle 5,5 \rangle$ -Merge, $Bu f_i$, and channels a , b , and 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 networks is shown in Figure 2.

Figure 2: The Example by Brock and Ackerman.

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 $N_1 \parallel Plus1 \parallel Fanout$ but not of $N_2 \parallel Plus1 \parallel Fanout$. The reason for this is that when the first data item 5 is produced by $N_2 \parallel Plus1 \parallel Fanout$ 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

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

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

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

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 $N_2 \parallel Plus1 \parallel Fanout$ (again considering only those with a 6 on a) is $\langle \langle e, 5 \rangle, \langle e, 5 \rangle \rangle$ whereas the network $N_1 \parallel Plus1 \parallel Fanout$ 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

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

Let us introduce some terminology. A model (of dataflow networks) is a mapping from the set of dataflow 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. A network N is put into the context by replacing the place holder $\cdot$ by N .

Definition 6.1 Let $\mathcal{D}$ and $\mathcal{O}$ be two models of dataflow networks. The model $\mathcal{D}$ is said to be *fully abstract* with respect to $\mathcal{O}$ if for all networks N_1 and N_2

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

□

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

1. If $\mathcal{D}(N_1) = \mathcal{D}(N_2)$ then $\mathcal{O}(N_1) = \mathcal{O}(N_2)$,
i.e., the model $\mathcal{D}$ is more distinguishing than the model $\mathcal{O}$.
2. For all contexts $C[\cdot]$ we have $\mathcal{D}(N_1) = \mathcal{D}(N_2) \implies \mathcal{D}(C[N_1]) = \mathcal{D}(C[N_2])$,
i.e., the model $\mathcal{D}$ is compositional.
3. If $\mathcal{D}(N_1) \neq \mathcal{D}(N_2)$ then there is a context $C[\cdot]$ such that $\mathcal{O}(C[N_1]) \neq \mathcal{O}(C[N_2])$,
i.e., if two networks are distinguished by $\mathcal{D}$, then there is a context such that the networks can be distinguished by $\mathcal{O}$.

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

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

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

We now state the main theorem of the paper.

Theorem 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 3.4. 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 dataflow networks such that $\mathcal{T}(N_1) \neq \mathcal{T}(N_2)$, then there is a context $C[\cdot]$ such that $\mathcal{H}(C[N_1]) \neq \mathcal{H}(C[N_2])$. □

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

In the remaining cases we have $T_{N_1} \neq T_{N_2}$. Assume that both N_1 and N_2 have the input channels $in_1, \dots, in_m$ and the output channels $out_1, \dots, out_n$. We must find a context which makes it possible to distinguish between N_1 and N_2 in the history model. Intuitively, the history model orders data items that are produced on one channel in a total order, whereas the trace model orders communication events on all channels in a total order. Thus, the sought context must “bring together” the communication events on all external channels of N_i to a single channel. We shall use the context $C[\cdot]$ shown in Figure 3. It is inspired by the context used by Kok [Kok87]. The idea of the context is to bring together data items on channels $in_1, \dots, in_m$ and $out_1, \dots, out_n$ into a totally ordered sequence on channel b . The sequence on b is copied onto c in order to make it observable from outside.

Figure 3: the context C

Over the channels $in_1, \dots, in_m$ and $out_1, \dots, out_n$ 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 $\langle in_j, d \rangle$ and $\langle out_l, d \rangle$, where in_j or out_l is one of the channels of N_i , and d is an ordinary data item that can be transmitted over that channel.

The nodes perform the following functions:

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

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

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

Continuing the proof, recall that we assumed $T_{N_1} \neq T_{N_2}$, i.e., there is a trace t such that $t \notin T_{N_1}$ but $t \in T_{N_2}$ (if not, reverse the roles of N_1 and N_2). The trace t is a (finite or infinite) sequence of communication events of the form $\langle in_j, d \rangle$ and $\langle out_l, d \rangle$. Let $in(t)$ be $t|_{\{in_1, \dots, in_m\}}$, i.e., the subsequence of t consisting of all communication events of the form $\langle in_j, d \rangle$. Define the history function h by $h(a) = in(t)$ and $h(c) = t$. The lemma now follows from the following claim:

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

We first give a sketchy proof of the claim (*). First note that the node *Split* is essentially connected to nodes of N_i via FIFO channels. Intuitively, this follows from the observations that (1) for each data item of form $\langle in_j, d \rangle$ that is consumed by *Split*, the data item d is transmitted to N_i over the FIFO channel in_j , and (2) each data item d transmitted from N_i over out_l reaches *Split* in the form $\langle out_l, d \rangle$ over FIFO channels via *MergeMark*. It follows that tagged data items are consumed by *Split* in the same order as the corresponding communication events would occur in a computation of N_i . Hence the sequence t is transmitted over b (and hence over c) iff t is a trace of N_i .

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

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

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

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

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

We now use theorem 4.1 to prove that $t'|_{\{a,c\}}$ is a trace of $C[N_i]$. This follows if we note that (1) $t'|_{\{out_1, \dots, out_n, a, b\}}$ is a trace of the network N_M , since it is composed of fragments of the form $\langle \langle out_l, d \rangle \langle b, \langle out_l, d \rangle \rangle \rangle$ and of the form $\langle \langle a, \langle in_j, d \rangle \rangle \langle b, \langle in_j, d \rangle \rangle \rangle$, and that (2) $t'|_{\{in_1, \dots, in_m, b, c\}}$ is a trace of the network N_S , since it is composed of fragments of the form $\langle \langle b, \langle out_l, d \rangle \rangle \langle c, \langle out_l, d \rangle \rangle \rangle$ and of the form $\langle \langle b, \langle in_j, d \rangle \rangle \langle c, \langle in_j, d \rangle \rangle \langle in_j, d \rangle \rangle$. (3) $t'|_{\{in_1, \dots, in_m, out_1, \dots, out_n\}}$ is a trace of the network 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 $in(t)$.

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

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

1. Replace each state σ' of $C[N_i]$ in Γ' by a state σ of N_i . The state σ agrees with σ' for nodes in N_i , for internal channels of N_i , and for the channels $in_1, \dots, in_m$. For the channels $out_1, \dots, out_n$ we obtain $\sigma(out_l)$ as the concatenation of $items(\sigma'(b)|_{\{out_l\}})$

and $\sigma'(out_l)$, where $items(\sigma'(b) \upharpoonright_{\{out_l\}})$ is the sequence of data items in the subsequence of $\sigma'(b)$ of data items of form $\langle out_l, d \rangle$.

2. Compress all transitions with communication events on the channels a and c , and all transitions resulting from firings of the node *MergeMark* into a single state.
3. Each internal transition (derived from a firing of *Split*) which consumes a data item of form $\langle out_l, d \rangle$ from channel b is labeled by $\langle out_l, d \rangle$.
4. Each internal transition (derived from a firing of *Split*) which consumes a data item of form $\langle in_j, d \rangle$ from channel b (and produces d on in_j) is labeled by $\langle in_j, d \rangle$.

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

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

7 Comparison with Another Fully Abstract Model

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

Intuitively, we describe the idea behind Kok's model as follows. We use v, w , possibly with subscripts, to range over infinite sequences of finite words. Let $w[i]$ denote the i^{th} finite word in the sequence w , let $w \upharpoonright i$ denote the concatenation $w[1] \cdot \dots \cdot w[i]$ of the i first words of w , and let $w \upharpoonright \infty$ denote the concatenation of all words of w . A dataflow network N with input channels $in_1, \dots, in_m$ and output channels $out_1, \dots, out_n$ is denoted by a function f_N from $((V^*)^\omega)^m$ to $\mathcal{P}(((V^*)^\omega)^n)$. For f_N we have $\langle w_1, \dots, w_n \rangle \in f_N(\langle v_1, \dots, v_m \rangle)$ if there is a computation Γ of the network in which

- for each $i \leq 1$ there is a finite prefix Γ' of Γ such that $v_j \upharpoonright i$ is the sequence of data items transmitted over in_j in Γ' , and $w_l \upharpoonright i$ is the sequence of data items transmitted over out_l in Γ' .
- $v_j \upharpoonright \infty$ is the sequence of data items transmitted over in_j in Γ , and $w_l \upharpoonright \infty$ is the sequence of data items transmitted over out_l in Γ .

We can relate the model of Kok to our model by stating that $\langle w_1, \dots, w_n \rangle \in f(\langle v_1, \dots, v_m \rangle)$

precisely if T_N contains the trace

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

A proof of this will appear in [JK].

In our opinion, the trace model expresses the temporal ordering between communication events in a straight-forward way. The trace model also allows a natural and direct proof of full abstraction. In our proof of full abstraction, we showed how the context in figure 3 established a relationship between two forms of a sequence: one form was a history and the other form was a trace. The proof was simplified by the fact that the trace model is based on sequences.

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

8 Related Work

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

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

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

Keller and Panangaden [KP85] (in [KP86] in a slightly different framework) propose a trace-model related to ours. A difference is that they use input events in traces to represent the consumption of 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.

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.

Two properties of Kahn’s original model [Kah74] are not shared by our trace model. Kahn showed how the denotation of a network from the denotations of its components can be computed by a fixedpoint construction. It appears difficult to incorporate fixedpoint theory into a model for networks that exhibit nondeterminism and fairness. Approaches to solving this problem appear in e.g. [Bro86,SN83,KP85]. Kahn also showed that his model is equally applicable for recursively defined networks in which a node can expand recursively into a subnetwork.

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

Acknowledgments

I am grateful to Joost Kok for stimulating discussions and ideas. I also would like to thank Joachim Parrow for critical reading of 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*, pages 252–259, Springer Verlag, 1981.
- [BM85] R. J. R. Back and H. Mannila. On the suitability of trace semantics for modular proofs of communicating processes. *Theoretical Computer Science*, 39(1):47–68, 1985.
- [Bou82] F. Boussinot. Proposition de sémantique dénotationnelle pour des processus avec opérateur de mélange équitable. *Theoretical Computer Science*, 18(2):173–206, 1982.
- [Bro83] M. Broy. Fixed point theory for communication and concurrency. In Björner, editor, *Formal Description of Programming Concepts II*, pages 125–146, North-Holland, Amsterdam, 1983.
- [Bro86] M. Broy. A theory for nondeterminism, parallelism, communication, and concurrency. *Theoretical Computer Science*, 45:1–61, 1986.
- [Bro88] M. Broy. Nondeterministic data flow programs: how to avoid the merge anomaly. *Science of Computer Programming*, 10:65–85, 1988.
- [CH81] Z. C. Chen and C. A. R. Hoare. Partial correctness of communicating sequential processes. In *Proc. International Conference on Distributed Computing*, pages 1–12, Paris, April 1981.
- [JK] B. Jonsson and J. Kok. Comparing dataflow models. In progress.
- [Jon85] B. Jonsson. A model and proof system for asynchronous networks. In *Proc. 4th ACM Symp. on Principles of Distributed Computing*, pages 49–58, Minaki, Canada, 1985.
- [Jon87a] B. Jonsson. *Compositional Verification of Distributed Systems*. PhD thesis, Dept. of Computer Systems, Uppsala University, Sweden, Uppsala, Sweden, 1987. Available as report DoCS 87/09.
- [Jon87b] B. Jonsson. Modular verification of asynchronous networks. In *Proc. 6th ACM Symp. on Principles of Distributed Computing*, pages 152–166, Vancouver, Canada, 1987.
- [Kah74] G. Kahn. The semantics of a simple language for parallel programming. In *IFIP 74*, pages 471–475, North-Holland, 1974.

- [Kel78] R. M. Keller. Denotational models for parallel programs with indeterminate operators. In Neuhold, editor, *Formal Descriptions of Programming Concepts*, pages 337–366, North-Holland, 1978.
- [Kok86] J. N. Kok. Denotational semantics of nets with nondeterminism. In *European Symposium on Programming, Saarbrücken, LNCS 206*, pages 237–249, Springer Verlag, 1986.
- [Kok87] J. N. Kok. A fully abstract semantics for data flow nets. In *Proc. PARLE, LNCS 259*, pages 351–368, Springer Verlag, 1987.
- [Kos78] P. R. Kosinski. A straight-forward denotational semantics for nondeterminate data flow programs. In *Proc. 5th ACM Symp. on Principles of Programming Languages*, pages 214–219, 1978.
- [KP85] R. M. Keller and P. Panangaden. Semantics of networks containing indeterminate operators. In Brookes, Roscoe, and Winskel, editors, *Seminar on Concurrency 1984, LNCS 197*, pages 479–496, 1985.
- [KP86] R. M. Keller and P. Panangaden. Semantics of networks containing indeterminate operators. *Distributed Computing*, 1:235–245, 1986.
- [LT87] N. A. Lynch and M. R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proc. 6th ACM Symp. on Principles of Distributed Computing*, pages 137–151, 1987.
- [MC82] J. Misra and K. M. Chandy. Proofs of networks of processes. *IEEE Transactions on Software Engineering*, SE-7(4):417–426, July 1982.
- [MCS82] J. Misra, K. M. Chandy, and T. Smith. Proving safety and liveness of communicating processes with examples. In *Proc. ACM SIGACT-SIGOPS Symp. on Principles of Distributed Computing*, pages 201–208, 1982.
- [Mis84] J. Misra. Reasoning about networks of communicating processes. In *INRIA Advanced Nato Study Institute on Logics and Models for Verification and Specification of Concurrent Systems*, La Colle sur Loupe, France, 1984.
- [MP81] Z. Manna and A. Pnueli. The temporal framework for concurrent programs. In Boyer and Moore, editors, *The Correctness Problem in Computer Science*, pages 215–274, Academic Press, 1981.
- [NDGO86] V. Nguyen, A. Demers, D. Gries, and S. Owicki. A model and temporal proof system for networks of processes. *Distributed Computing*, 1(1):7–25, 1986.
- [Par83] D. Park. The ‘fairness’ problem and nondeterministic computing networks. In de Bakker and van Leeuwen, editors, *Foundations of Computer Science IV, Part 2*, pages 133–161, Amsterdam, 1983. Mathematical Centre Tracts 159.
- [Plo81] G. Plotkin. *A Structural Approach to Operational Semantics*. Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, 1981.

- [Pra82] V. R. Pratt. On the composition of processes. In *Proc. 9th ACM Symp. in Principles of Programming Languages*, pages 213–223, 1982.
- [Pra84] V. R. Pratt. The pomset model of parallel processes: unifying the temporal and the spatial. In Brookes, Roscoe, and Winskel, editors, *Proc. Seminar on Concurrency, LNCS 197*, pages 180–196, Springer Verlag, 1984.
- [SN83] J. Staples and V. L. Nguyen. Computing the behaviour of asynchronous processes. *Theoretical Computer Science*, 26(3):343–353, 1983.
- [SN85] J. Staples and V. L. Nguyen. A fixpoint semantics for nondeterministic data flow. *ACM Journal*, 32(2):411–444, April 1985.
- [Sta84] E. W. Stark. *Foundations of a Theory of Specification for Distributed Systems*. PhD thesis, Massachusetts Inst. of Technology, 1984. Available as Report No. MIT/LCS/TR-342.