Towards a Complete Hierarchy of Compositional Dataflow Models

B. Jonsson, J.N. Kok

RUU-CS-91-45
December 1991

Utrecht University

Department of Computer Science

Padualaan 14, P.O. Box 80.089,
3508 TB Utrecht, The Netherlands,
Tel. : ... + 31 - 30 - 531454

Towards a Complete Hierarchy of Compositional Dataflow Models

B. Jonsson, J.N. Kok

Technical Report RUU-CS-91-45
December 1991

Department of Computer Science
Utrecht University
P.O.Box 80.089
3508 TB Utrecht
The Netherlands

ISSN: 0924-3275

Towards a Complete Hierarchy of Compositional Dataflow Models

Bengt Jonsson*
Swedish Institute of Computer Science and Uppsala University
Box 1263
164 28 Kista, Sweden
bengt@sics.se

Joost N. Kok
Dept. of Computer Science
Utrecht University
P.O. Box 80.089
3508 TB Utrecht, the Netherlands
joost@cs.ruu.nl

Abstract

A dataflow network consists of nodes that communicate by passing data over unbounded FIFO channels. For dataflow networks containing only deterministic nodes, Kahn has presented a simple and elegant semantic model. However, the generalization of this model is not compositional for nondeterministic networks. Past work has shown that compositionality can be attained by models based on traces. In the paper, we investigate trace models of dataflow networks, with the aim of characterizing compositional and non-compositional models. We study several compositional trace models, which differ in whether they model liveness, termination or divergence. We relate the models into a hierarchy, according to their capability to distinguish networks. A hierarchy is called *complete* if any gap between two models in the hierarchy contains no compositional models. Our main contribution is to prove that most of the gaps in our hierarchy do not contain compositional models. Several full abstraction results in the literature follow directly from the gaps in our hierarchy. We also show that by restricting the networks to contain less powerful nondeterministic processes, additional models become compositional. This means that additional models are added to the hierarchy.

*Supported in part by the Swedish Board for Technical Development (STU) under contract No. 89–01220P as part of Esprit BRA project SPEC, No. 3096.

1 Introduction

Semantical models of communicating systems has been a topic of intensive study in the last years (e.g. [4, 8, 21, 7, 24]). A purpose of that study is a better understanding of how to describe and reason about the behavior of communicating systems. A semantic model should abstract from the internal activity of a system, describing only its externally observable behavior. It should also be *compositional*, meaning that the denotation of a composed network can be obtained using only the denotations of its components. A compositional model can serve as a basis for modular spefication and verification methods, (e.g. [1, 9, 22, 23, 37]).

This paper is concerned with semantic models of dataflow networks. Dataflow networks are an important model for asynchronous parallel computation, in which streams of data items are exchanged along FIFO channels between executing processes. The interest in semantics of dataflow networks is to a large extent due to the work by Kahn [14], who proposed an elegant semantic model for dataflow networks with only deterministic processes. However, the straight-forward generalization of Kahn's model is not compositional if networks may contain nondeterministic processes [3]. Since the work by Kahn, many compositional models for nondeterministic networks have been proposed [1, 2, 3, 5, 6, 11, 17, 18, 19, 15, 16, 28, 29, 30, 35]. Recently, a number of models have appeared that are in addition fully abstract, i.e., they have to Kahn's model added precisely the information that is necessary for attaining compositionality [11, 18, 31, 34]. Of these models, the ones presented in [11, 18, 34] are isomorphic to each other [13]. However, the model in [31] is different from these in that it does not represent infinite behaviors or liveness properties.

In the literature, there are thus at least two essentially different compositional models of dataflow networks which are also fully abstract. In this paper, we consider the question of investigating which are the compositional models of dataflow networks and which models cannot be compositional. Since the fully abstract models in [11, 31] are both trace models, we start our investigation by a hierarchy of trace-based models. We define several compositional trace models, which differ in whether or not they model safety properties, liveness properties, termination, and divergence. For instance, safety properties can be represented by prefix-closed sets of finite traces (as in [31]), whereas liveness and fairness properties are represented by (possibly infinite) traces of completed executions of a network (as in [11, 18, 34]).

In order to investigate compositional models, we relate the models according to how fine distinctions they make between the modeled networks. If any two networks that are distinguished by model 1 are also distinguished by model 2, then model 2 makes more distinctions than model 1. Our compositional models can thus be organized into a hierarchy. Our main contribution is that using this hierarchy, we prove the nonexistence of certain compositional models: instead of just saying that a model 2

makes more distinctions than a model 1, we also prove that in some cases there is no compositional model which is strictly more distinguishing than model 1 and strictly less distinguishing than model 2. Thus there is a "gap" between model 1 and 2, in which no other compositional model exists. Assume for example that model 1 represents safety properties, and that model 2 represents safety and liveness properties. Then the nonexistence of a model between model 1 and 2 means that liveness properties cannot be represented in less detail than they are in model 2, without sacrificing compositionality (assuming that safety properties are represented). Several full abstraction results, e.g. those by Jonsson [11] and Rabinovich and Trakhtenbrot [31] follow immediately from our hierarchy with associated gaps. Allthough all the models in the hierarchy are trace based models, the results about the gaps also show that there are no other compositional (possibly non trace based) models in the gaps.

In some of our models (the models which represents liveness properties), the denotation of a network contains infinite traces, which correspond to infinite observations. It may be argued that the status of such infinite observations is debatable. Our results show that it is necessary to make distinctions based on infinite traces necessary because of compositionality even if only some finite completed traces can be observed. Of course, we then assume the existence of contexts which are sufficiently powerful to make infinite observations and transform these into finite observations. For the case that such contexts are not at hand, we extend our results to less general classes of nondeterministic networks which do not include fair merge processes and without interrupt capabilities. Panangaden et al [25, 26, 27] have found that there are several inequivalent forms of indeterminacy and fairness in dataflow. For the class of networks without interrupt capabilities, we prove that there exist additional, inequivalent, compositional models, which are not compositional if fair merge primitives are allowed. In this way, we show that under certain circumstances one can use less powerful observation criteria in a semantic model.

The idea of proving gaps between compositional models in a hierarchy has appeared earlier, for I/O-automata ([20]), in the thesis of the first author [10], and in [12]. For synchronously communicating systems, hierarchies of models have been studied by Reed and Roscoe [32, 33], by Olderog and Hoare [24], and by van Glabbeek and Vaandrager [36].

In the next section, we present dataflow networks. In Section 3 we present the general framework for relating models according to information content and for establishing gaps. In Section 4 we define our models of dataflow networks and prove that they are compositional. In Section 5 we relate the models and establish the gaps. Section 6 contains an example of an additional compositional model between the models that represent safety and liveness for the restricted class of dataflow networks. Section 7 contains conclusions.

2 Dataflow Networks

In this section, we give an informal presentation of dataflow networks. A more careful treatment of the subject would require a more thorough definition, e.g. by means of transition systems. Such a definition is given in e.g. [11, 18, 13].

A *dataflow network* consists of 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 are used to transmit data items from the environment to a node.

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

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

Larger networks can be built by *composition* of smaller networks as follows. The networks $N_1, \ldots, N_k$ are *compatible* if each channel name occurs at most once as an input channel and at most once as an output channel. Given compatible networks $N_1, \ldots, N_k$, a composite network is obtained by connecting input channels to output channels with the same name. The resulting network can also, if we disregard its input and output channels, be viewed as a node which can be connected to other nodes. As an example, 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. The composition of the nodes N_1 and N_2 and channel c can also be viewed as a node which receives data items from channel a and transmits data items over channels b and d.

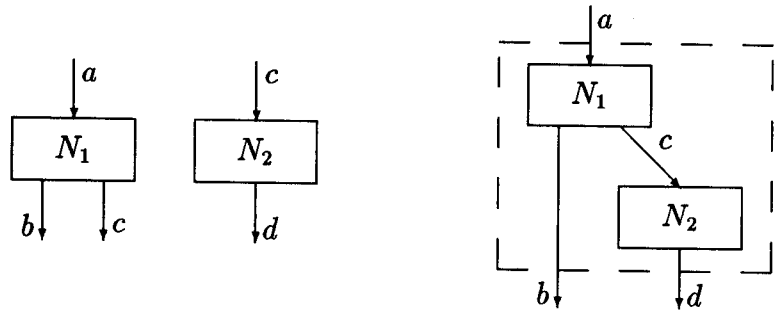

Figure 1: Two networks, N_1 and N_2 (left), and their composition (right).

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.

The nodes have in general an internal state, they can consume data items from incoming channels, produce data items on outgoing channels, and perform internal computations. Some of the states are designated as initial states. With each node of a network is, in addition, associated a set of liveness and fairness requirements. These requirements are specific to each node. Examples of such requirements are: a node may be required eventually to consume a data item from a certain incoming channel if that channel is not empty, a certain operation may not be enabled indefinitely without being executed, etc. For instance, with a fair merge node is associated the requirements that data items on an incoming channel must eventually be consumed and thereafter produced on the outgoing channel.

With each output channel is associated the following liveness requirement: Each data item in the queue of the output channel must eventually be delivered to the environment.

A *communication event* of a dataflow network N is a pair $c(d)$, where c is an input or output channel of N, and d is a data item. A communication event $c(d)$ represents the transmission of data item d from the environment into channel c, or from channel c to the environment, depending on whether c is an input or output channel.

An *execution step* of a network is either

- an operation by a node of the network which is internal and possibly produces, or consumes a data item from some channel or
- the exchange of a data item between a channel and the environment.

An *execution* of a network is a sequence of execution steps that starts in a state where each node is in its initial state and all channels are empty. A *computation* is an execution of the network that satisfies the liveness and fairness requirements associated with the network. A computation can be either finite or infinite. Thus, with each dataflow network is associated a set of computations. A *partial computation* of a network is a finite execution of the network. A partial computation does not have to satisfy any liveness or fairness requirements. A partial computation represents an unfinished execution of the network, which can be continued to a (possibly infinite) computation. A computation is *terminated* if it is finite. A computation is *divergent* if it is infinite, but only performs a finite number of communication events.

3 Relating Models

Before presenting our models of dataflow networks, we shall present the general framework for the relationships between the models that will be established subsequently.

A *model* $[\![\cdot]\!]$ (of dataflow networks) is a mapping that maps dataflow networks to some domain. A model $[\![\cdot]\!]$ is *compositional* iff there is a partial operation $\|_{\mathcal{M}}$ on the domain

of $[\![\cdot]\!]$ such that $[\![\|(N_1, \cdots, N_k)]\!] = \|_{\mathcal{M}}([\![N_1]\!], \cdots, [\![N_k]\!])$ whenever the dataflow networks $N_1, \ldots, N_k$ can be composed.

Examples of rather uninteresting compositional models is the (uninformative) model that maps all dataflow networks to the same single denotation, and the model which maps any dataflow network to itself.

We next define how two models are compared according to their ability to distinguish between dataflow networks.

Definition 3.1 A model $[\![\cdot]\!]_1$ is *more abstract* than another model $[\![\cdot]\!]_2$, denoted $[\![\cdot]\!]_1 \sqsubseteq [\![\cdot]\!]_2$ if $[\![N_1]\!]_2 = [\![N_2]\!]_2$ implies $[\![N_1]\!]_1 = [\![N_2]\!]_1$ for all dataflow networks N_1, N_2. We use $[\![\cdot]\!]_1 \cong [\![\cdot]\!]_2$ to denote that $[\![\cdot]\!]_1 \sqsubseteq [\![\cdot]\!]_2 \sqsubseteq [\![\cdot]\!]_1$, and use $[\![\cdot]\!]_1 \sqsubset [\![\cdot]\!]_2$ to denote that $[\![\cdot]\!]_1 \sqsubseteq [\![\cdot]\!]_2 \not\sqsubseteq [\![\cdot]\!]_1$. □

Intuitively, if $[\![\cdot]\!]_1 \sqsubseteq [\![\cdot]\!]_2$, then the model $[\![\cdot]\!]_2$ distinguishes between all pairs of networks that are distinguished by $[\![\cdot]\!]_1$.

When establishing results about models, we shall instead of Definition 3.1, use the following alternative characterization, which follows immediately from Definition 3.1

Lemma 3.2 Let $[\![\cdot]\!]_1$ and $[\![\cdot]\!]_2$ be models. Then $[\![\cdot]\!]_1 \sqsubset [\![\cdot]\!]_2$ if and only if

1. For all dataflow networks N_1, N_2 we have that $[\![N_1]\!]_2 = [\![N_2]\!]_2$ implies $[\![N_1]\!]_1 = [\![N_2]\!]_1$.
2. There are dataflow networks N_3, N_4 that $[\![N_3]\!]_2 \neq [\![N_4]\!]_2$ and $[\![N_3]\!]_1 = [\![N_4]\!]_1$.

□

The most important concept to be studied in the remainder of the paper is the following, which excludes the existence of compositional models that lie between other models.

Definition 3.3 If $[\![\cdot]\!]_1$ and $[\![\cdot]\!]_2$ are models such that $[\![\cdot]\!]_1 \sqsubset [\![\cdot]\!]_2$, then the relation $[\![\cdot]\!]_1 \sqsubset [\![\cdot]\!]_2$, is a *minimal proper inclusion* (or a *proper gap*) if there is no compositional model $[\![\cdot]\!]_3$ such that $[\![\cdot]\!]_1 \sqsubset [\![\cdot]\!]_3 \sqsubset [\![\cdot]\!]_2$. □

4 Models of Dataflow Networks

In this section, we define the denotational models of dataflow networks that will be studied in this paper.

Let N be a dataflow network.

- A *trace* of N is the sequence of communication events in a computation of N. A trace can be both finite and infinite.
- A *partial trace* of N is the sequence of communication events in a partial computation of N.
- A *terminated trace* of N is the sequence of communication events in a finite computation of N. Note that in the last state of a finite computation, all output channels of the network must be empty, due to the liveness requirement that each data item in an output channel must eventually be delivered to the environment.
- A *divergent trace* of N is a finite sequence of communication events which occurs in an infinite computation of N.

Intuitively, a partial trace records the sequence of communication events that has occured until some moment in an execution. Such a recording can be made within finite time. In contrast, a trace records the sequence of communication events that has occured during a completed computation. A trace can be recorded only after the whole computation, i.e., an infinite stretch of time has passed, because at any point in time during the computation one cannot be sure whether the computation has indeed produced all its communication events. In order to record terminated or divergent traces, one must also assume that one can observe when no more internal execution steps, i.e. execution steps which do not produce communication events, will occur. This cannot be observed from the communication events alone, but one could imagine a lamp which indicates when the system will not perform any more execution steps unless more input is supplied.

As an example, consider a network with only a *copy* node, which copies data items from its single input channel to its single output channel. A partial trace of the network is a finite sequence t of communication events such that for any prefix t' of t, the sequence of data output in t' is a prefix of the sequence of data input in t'. A trace of the network is a (finite or infinite) sequence t of communication events such that all prefixes of t are partial traces of the network, and such that the sequence of data output in t is the same as the sequence of data input in t. A terminated trace of the network is simply a trace of the network which is finite. There are no divergent traces of the network. However, suppose that the network in addition to the copy node contains an internal clock, which performs an infinite sequence of internal execution steps that are unobservable using the communication events performed. In this case, no computation stops after a finite number of steps, all finite traces are divergent, and there are no terminated traces.

Let N be a dataflow network. Define

$I(N)$ as the set of input channels of N

$O(N)$ as the set of output channels of N

$E(N) = I(N) \cup O(N)$, the set of channels of N

$P(N)$ as the set of partial traces of N

$Q(N)$ as the set of terminated traces of N

$D(N)$ as the set of divergent traces of N

$T(N)$ as the set of traces of N

We can now define models of dataflow networks by letting the denotation of a network be a tuple with some of the above defined sets of traces and channels. We shall only consider models that include the sets $I(N)$ and $O(N)$, since these are necessary for determining when networks can be composed. We define the model $[\![\cdot]\!]_{Ev}$ by letting the denotation $[\![N]\!]_{Ev}$ of N be the tuple $\langle I(N), O(N)\rangle$. For the other models, we use the letters P, T, Q, and D as subscripts to indicate which sets of traces are included in addition to the input and output channels. For instance, the model $[\![\cdot]\!]_P$ is defined by letting the denotation $[\![N]\!]_P$ of N be the tuple $\langle I(N), O(N), P(N)\rangle$. The model $[\![N]\!]_{TQ}$ is defined by letting the denotation $[\![N]\!]_{TQ}$ of N be the tuple $\langle I(N), O(N), T(N), Q(N)\rangle$.

By noting that the set $P(N)$ can be obtained from the set $T(N)$, we find that there are twelve inequivalent models that can be obtained in the above way. Of these only 8 are compositional, as we shall see.

Let E be a set of channels and q a sequence of communication events.

$q\lceil_E$ denotes the restriction of q to the channels in E, i.e., the subsequence of q containing only events on channels in E.

$q \setminus E$ denotes the subsequence of q that are not on the channels in E, i.e., the result of deleting the events on channels in E from q.

E^* denotes the set of finite sequences of events on channels in E.

E^ω denotes the set of infinite sequences of events on channels in E.

$E^\dagger$ denotes the set of finite and infinite sequences of events on channels in E.

Theorem 4.1 *All models defined through combinations of P, T, Q, and D are compositional, except for the models $[\![\cdot]\!]_D$, $[\![\cdot]\!]_{PD}$, $[\![\cdot]\!]_{QD}$, and $[\![\cdot]\!]_{PQD}$.* □

Proof: The theorem follows from the following rules for composing the individual sets in the tuples. If $N_1, \ldots, N_k$ can be composed, and N is their composition $N_1 \| \ldots \| N_k$, we have the following facts:

$$I(N) = \bigcup_{i=1}^{k} I(N_i) - \bigcup_{i=1}^{k} O(N_i)$$

$$O(N) = \bigcup_{i=1}^{k} O(N_i) - \bigcup_{i=1}^{k} I(N_i)$$

$$P(N) = \{p\lceil_{(E(N))} : p \in (\bigcup_{i=1}^{k} E(N_i))^* \wedge (\forall i)\ p\lceil_{E(N_i)} \in P(N_i)\}$$

$$T(N) = \{t\lceil_{(E(N))} : t \in (\bigcup_{i=1}^{k} E(N_i))^\dagger \wedge (\forall i)\ t\lceil_{E(N_i)} \in T(N_i)\}$$

$$Q(N) = \{q\lceil_{(E(N))} : q \in (\bigcup_{i=1}^{k} E(N_i))^* \wedge (\forall i)\ q\lceil_{E(N_i)} \in Q(N_i)\}$$

$$D(N) = \{d\lceil_{(E(N))} : d \in (\bigcup_{i=1}^{k} E(N_i))^* \wedge (\exists i)\ [d\lceil_{E(N_i)} \in D(N_i) \wedge (\forall j)\ d\lceil_{E(N_j)} \in T(N_j)]\}$$

The proof of these rules is analogous to other proofs of similar results. See e.g. [11]. □

5 Comparison between models of dataflow networks

We can now state the main result of the paper, which relates the compositional models of dataflow networks that have been defined in Section 4.

$$\begin{array}{ccc}
[\![\cdot]\!]_{TD} & \sqsubset & [\![\cdot]\!]_{TQD} \\
\sqcup & & \sqcup \\
[\![\cdot]\!]_{T} & \sqsubset & [\![\cdot]\!]_{TQ} \\
\sqcup & & \sqcup \\
[\![\cdot]\!]_{P} & \sqsubset & [\![\cdot]\!]_{PQ} \\
\sqcup & & \sqcup \\
[\![\cdot]\!]_{Ev} & \sqsubset & [\![\cdot]\!]_{Q}
\end{array}$$

Figure 2: Relation between the models

Theorem 5.1 *The models defined in Section 4 have relationships as shown in Figure 2. Moreover, all the relations in Figure 2, except for the ones involving divergence and* $[\![\cdot]\!]_Q \sqsubset [\![\cdot]\!]_{PQ}$ *and* $[\![\cdot]\!]_{PQ} \sqsubset [\![\cdot]\!]_{TQ}$*, are minimal proper inclusions.* □

Proof: The first part of the proof, that of showing that the relationships exist, is straightforward, using the obvious mappings between the models. To show that the relations are minimal proper inclusions requires a separate proof using a separate construction for each relation. In the following, we shall devote one section to each relation.

5.1 Proof that $[\![\cdot]\!]_{Ev} \sqsubset [\![\cdot]\!]_P$ is a minimal proper inclusion

The central ingredient in the proof that $[\![\cdot]\!]_{Ev} \sqsubset [\![\cdot]\!]_P$ is a minimal proper inclusion is a construction of a context, which is given in the following lemma. For a channel c, let the network $0token_c$ be the network with one output channel c and no input channels, which never produces any data items, i.e., $0token_c$ has only one trace, the empty one. Let $01token_c$ be the network with one output channel c and no input channels, which either behaves as $0token_c$ or produces one data item, called *token*, and thereafter does nothing.

Lemma 5.2 Let N_1 and N_2 be dataflow networks for which $[\![N_1]\!]_{Ev} = [\![N_2]\!]_{Ev}$ such that there is a finite sequence t of communication events with $t \notin P(N_1)$ and $t \in P(N_2)$. Let c be a channel which is not a channel of N_1. Then there is a context $\mathcal{C}[\cdot]$, built from deterministic dataflow networks, such that

$$\begin{aligned} [\![\mathcal{C}[N_1]]\!]_P &= [\![0token_c]\!]_P \\ [\![\mathcal{C}[N_2]]\!]_P &= [\![01token_c]\!]_P \end{aligned}$$

□

Proof: The structure of the context $\mathcal{C}[\cdot]$ is shown in Figure 3.

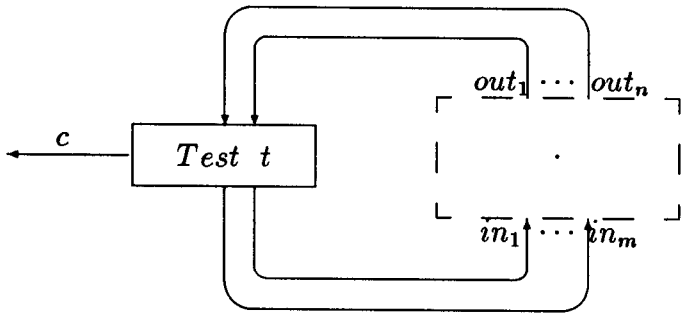

Figure 3: Structure of a Tester Context

The hole of the context is the dashed box, where either N_1 or N_2 will be put. We have assumed that $in_1, \ldots, in_m$ and $out_1, \ldots, out_n$ are the input and output channels of N_1 and N_2. The node $Test\ \ t$ can perform the sequence of events in t and thereafter sends the data item *token* on channel c. If the sequence of data items that arrives to $Test\ \ t$ on the channels $out_1, \ldots, out_n$ does not correspond to the sequence t then $Test\ \ t$ stops and does nothing more. It is easy to see that the context satisfies the conclusion of the lemma. □

For a pair $\langle I, O\rangle$ of disjoint sets of channels, let the network $NIL(I, O)$ be the network which has the denotation $\langle I, O, I^*\rangle$ in the partial trace model. That is, $NIL(I, O)$ has

input channels I, output channels O and as traces all finite sequences of events on channels in I, i.e. no output is produced.

Lemma 5.3 Let N be a dataflow network which does not have the channel c. Then there is a context $\mathcal{C}[\cdot]$ such that

$$\begin{aligned} [\![\mathcal{C}[0token_c]]\!]_P &= [\![NIL(I,O)]\!]_P \\ [\![\mathcal{C}[01token_c]]\!]_P &= [\![N]\!]_P \end{aligned}$$

□

Proof: In the proof, we construct a context $\mathcal{C}[\cdot]$, whose structure is given in Figure 4.

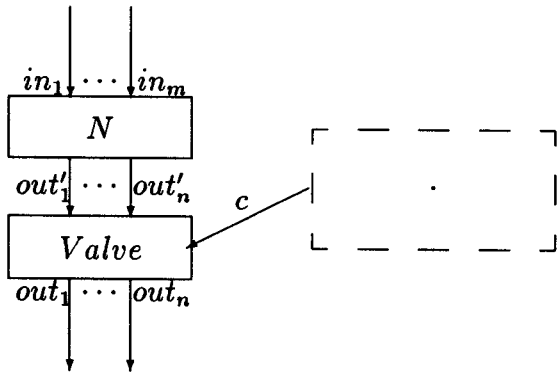

Figure 4: Structure of a Generating Context

The hole of the context is the dashed box, where either $0token_c$ or $01token_c$ will be put. The node $Valve$ initially does nothing. Upon receipt of the data item *token* it starts copying input to its output on all the output channels of N. It is easy to see the the conclusion of the lemma is satisfied. □

Theorem 5.4 *The gap $[\![\cdot]\!]_{Ev} \sqsubset [\![\cdot]\!]_P$ is a minimal proper inclusion for the classes of dataflow networks which includes the deterministic ones.* □

Proof: Assume that there exists a model $[\![\cdot]\!]$ such that $[\![\cdot]\!]_{Ev} \sqsubset [\![\cdot]\!] \sqsubset [\![\cdot]\!]_P$. We shall derive a contradiction. By $[\![\cdot]\!] \sqsubset [\![\cdot]\!]_P$ and Lemma 3.2, there are dataflow networks N_1, N_2 such that

$$[\![N_1]\!] = [\![N_2]\!] \qquad \text{and} \qquad [\![N_1]\!]_P \neq [\![N_2]\!]_P \qquad (1)$$

By $[\![\cdot]\!]_{Ev} \sqsubset [\![\cdot]\!]$ and Lemma 3.2, there are dataflow networks N_3, N_4 such that

$$[\![N_3]\!]_{Ev} = [\![N_4]\!]_{Ev} \qquad \text{and} \qquad [\![N_3]\!] \neq [\![N_4]\!] \qquad (2)$$

By (1) there is a finite sequence t of communication events such that $t \in P(N_2)$ but $t \notin P(N_1)$. By Lemma 5.2 we conclude that there is a context $\mathcal{C}[\cdot]$, such that $[\![\mathcal{C}[N_1]]\!]_P = [\![0token_c]\!]_P$ and $[\![\mathcal{C}[N_2]]\!]_P = [\![01token_c]\!]_P$. By (1) and the fact that $[\![\cdot]\!]$ is compositional we then conclude that $[\![0token_c]\!] = [\![01token_c]\!]$. By lemma 5.3 we then conclude that $[\![NIL(I(N_3), O(N_3))]\!] = [\![N_3]\!]$ and $[\![NIL(I(N_4), O(N_4))]\!] = [\![N_4]\!]$. It follows (because $[\![N_3]\!]_{Ev} = [\![N_4]\!]_{Ev}$ implies $[\![NIL(I(N_3), O(N_3))]\!]_P = [\![NIL(I(N_4), O(N_4))]\!]_P$) that $[\![N_3]\!] = [\![N_4]\!]$ contradicting (2). □

5.2 Proof that $[\![\cdot]\!]_P \sqsubset [\![\cdot]\!]_T$ is a minimal proper inclusion

In this section, we shall first establish a set of constructions of contexts. From these constructions, we can infer that $[\![\cdot]\!]_P \sqsubset [\![\cdot]\!]_T$ is a minimal proper inclusion for the class of nondeterministic dataflow networks. In contrast to the preceding section, it is not enough to use only deterministic process in these contexts, one must also use merge nodes. After the sequence of lemmas, we supply theorems which show that the gap $[\![\cdot]\!]_P \sqsubset [\![\cdot]\!]_T$ is a minimal proper inclusion for the class of all nondeterministic dataflow networks, which includes fair merge nodes. The essential reason for needing fair merge nodes is that we need to use interrupts and timeouts in the contexts.

For a channel c, let $1token_c$ be the network with one output channel c and no input channels, which produces one data item, called *token* and thereafter does nothing.

Lemma 5.5 Let N_1 and N_2 be dataflow networks for which $[\![N_1]\!]_P = [\![N_2]\!]_P$ such that there is a finite or infinite sequence t of communication events with $t \notin T(N_1)$ and $t \in T(N_2)$. Let c be a channel which is not a channel of N_1. Then there is a context $\mathcal{C}[\cdot]$, such that

$$\begin{aligned} [\![\mathcal{C}[N_1]]\!]_T &= [\![1token_c]\!]_T \\ [\![\mathcal{C}[N_2]]\!]_T &= [\![01token_c]\!]_T \end{aligned}$$

□

Proof: The structure of the context $\mathcal{C}[\cdot]$ is shown in Figure 5.

The hole of the context is the dashed box, where either N_1 or N_2 will be put. We have assumed that $in_1, \ldots, in_m$ and $out_1, \ldots, out_n$ are the input and output channels of N_1 and N_2.

The intuition behind the node $Test\ \neg t$ is that it will try to perform the sequence of events in t and outputs a token if something is wrong (either timeout and there are communication events to be received or wrong communication events arrive). At the arrival of a timeout token at a channel all the communication events at that channel should have occurred.

If t is infinite then $Test\ \neg t$ tries to perform an infinite sequence of events corresponding to t. If this goes well, then there will be no token on the channel c. If a wrong communication event arrives, then a token is put on the channel c. If timeout arrives on a channel, then it is checked if there are still communication events expected (in

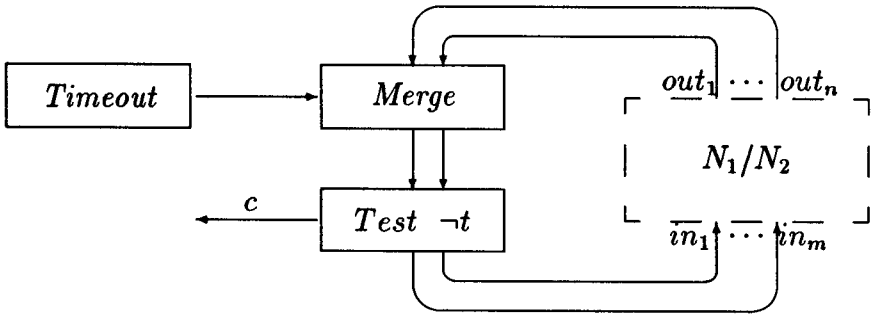

Figure 5: Structure of an Timeout-Tester Context

the rest of the trace t) at that channel. If this is the case, a token is put on the channel c.

If t is finite, then $Test\ \neg t$ will output no token on c if it can perform the sequence t followed by reception of an *timeout* token on each input channel. However, if before the sequence t is completed a token caused by a network in the hole arrives which does not correspond with the sequence t then $Test\ \neg t$ sends the data item *token* on channel c and thereafter does nothing. Again, if timeout arrives on a channel, then it is checked if there are still communication events expected at that channel. If this is the case, a token is put on the channel c. The item *token* will also be sent if t is finite, but more input, which is not the item *timeout*, arrives after t has been performed. The node $Test\ \neg t$ always stops execution after sending *token*.

The node *Timeout* sends a single distinguished data item *timeout* on its output channel. The node *Merge* forwards all incoming data items on channels $out_1, \ldots, out_n$ to the corresponding outgoing channels. It can also receive the data item *timeout* from *Timeout* and forward *timeout* on all its outgoing channels. The node *Merge* is fair in the sense that it must continue forwarding some input as long as some input arrives, but it does not need to be fair to any particular input channel. Thus *Merge* must forward the *timeout* token if no more input arrives on the other incoming channels, but it is also possible for *Merge* to neglect the input from *Timeout* if infinitely many data items arrive on the other incoming channels.

To see that the context satisfies the conclusion of the lemma, note that $Test\ \neg t$ will never perform the sequence t if the hole is filled with N_1 and that if N_1 performs a sequence which is a proper prefix of t then the item *timeout* will arrive to $Test\ \neg t$ and generate the item *token*. Further note that if the hole is filled with N_2 then there is a computation in which $Test\ \neg t$ sees the sequence t, and in which if t has infinite number of output events, the *timeout* is never input by *Merge* and if t is finite, *timeout* arrives to $Test\ \neg t$ after all other data items. □

For a network N let $\lceil T(N) \rceil$ be all finite and infinite sequences of comunication events of $E(N)$ that can be obtained by appending a finite or infinite sequence of input events

to some sequence in $P(N)$. Intuitively, $\lceil T(N) \rceil$ contains the sequences that can be obtained by performing first a part of a computation of N with a sequence of communication events in $P(N)$ and thereafter blocking all further output so that the remaining communication events in the computation are input events. Let $FinOut(N)$ be the network which has the denotation $\langle I(N), O(N), \lceil T(N) \rceil \rangle$. Intuitively, $FinOut(N)$ can be thought of as the network N where output is blocked after some finite time of each computation so that each computation has only finitely many output events. Let $FinInfOut(N)$ be the network which has the denotation $\langle I(N), O(N), \lceil T(N) \rceil \cup T(N) \rangle$. Intuitively, $FinInfOut(N)$ is a network which behaves either like $FinOut(N)$ or like N.

Lemma 5.6 Let N be a dataflow network which does not have the channel c. Then there is a context $\mathcal{C}[\cdot]$ such that

$$\begin{aligned} [\![\mathcal{C}[1\,token_c]]\!]_T &= [\![FinOut(N)]\!]_T \\ [\![\mathcal{C}[01\,token_c]]\!]_T &= [\![FinInfOut(N)]\!]_T \end{aligned}$$

□

Proof: In the proof, we use a context $\mathcal{C}[\cdot]$ with the same structure as that in Figure 4. The hole of the context is the dashed box, where either $1\,token_c$ or $01\,token_c$ will be put. The node $Valve$ is initially copying input to its output on all the output channels of N. Upon receipt of the datum *token* it stops copying data items and does nothing further. This implies that if *token* is received, then only a finite portion of the output of the network is produced. The conclusion of the lemma follows. □

Lemma 5.7 Let N be a dataflow network which does not have the channel c. Then there is a context $\mathcal{C}[\cdot]$ such that

$$\begin{aligned} [\![\mathcal{C}[1\,token_c]]\!]_T &= [\![N]\!]_T \\ [\![\mathcal{C}[01\,token_c]]\!]_T &= [\![FinInfOut(N)]\!]_T \end{aligned}$$

□

Proof: In the proof, we use a context $\mathcal{C}[\cdot]$ with the same structure as that in Figure 4. The hole of the context is the dashed box, where either $1\,token_c$ or $01\,token_c$ will be put. The difference is that this time the node $Valve$ is initially copying input to its output on all the output channels of N. At some randomly chosen finite time, it stops copying data items and blocks further output and only waits for the datum *token* from the network in the hole of the context. Upon recept of *token* it again starts copying input to its output on all the output channels of N. This implies that if *token* is received, then a normal computation of the network is observed. If *token* is never received, then only a finite portion of the output of the network is produced. The conclusion of the lemma follows. □

Theorem 5.8 *The gap* $[\![\cdot]\!]_P \sqsubset [\![\cdot]\!]_T$ *is a minimal proper inclusion for the class of all dataflow networks.* □

Proof: Assume that there exists a model $[\![\cdot]\!]$ such that $[\![\cdot]\!]_P \sqsubset [\![\cdot]\!] \sqsubset [\![\cdot]\!]_T$. We shall derive a contradiction. By $[\![\cdot]\!] \sqsubset [\![\cdot]\!]_T$ and Lemma 3.2, there are dataflow networks N_1, N_2 such that

$$[\![N_1]\!] = [\![N_2]\!] \qquad \text{and} \qquad [\![N_1]\!]_T \neq [\![N_2]\!]_T \qquad (1)$$

By $[\![\cdot]\!]_P \sqsubset [\![\cdot]\!]$ and Lemma 3.2, there are dataflow networks N_3, N_4 such that

$$[\![N_3]\!]_P = [\![N_4]\!]_P \qquad \text{and} \qquad [\![N_3]\!] \neq [\![N_4]\!] \qquad (2)$$

By (1) there is a finite or infinite sequence t of communication events such that $t \in [\![N_2]\!]_T$ but $t \notin [\![N_1]\!]_T$. By Lemma 5.5 we conclude that there is a context $\mathcal{C}[\cdot]$, such that $[\![\mathcal{C}[N_1]]\!]_T = [\![1\,token_c]\!]_T$ and $[\![\mathcal{C}[N_2]]\!]_T = [\![01\,token_c]\!]_T$. By (1), the assumption $[\![\cdot]\!] \sqsubset [\![\cdot]\!]_T$, and the fact that $[\![\cdot]\!]$ is compositional we then conclude that $[\![1\,token_c]\!] = [\![01\,token_c]\!]$. By Lemma 5.7 we then conclude that $[\![N_3]\!] = [\![FinInfOut(N_3)]\!]$. By lemma 5.6 we conclude that $[\![FinOut(N_3)]\!] = [\![FinInfOut(N_3)]\!]$, which implies that $[\![N_3]\!] = [\![FinOut(N_3)]\!]$. We similarly conclude that $[\![N_4]\!] = [\![FinOut(N_4)]\!]$. By the fact that for all networks N_1 and N_2 we have that $[\![N_1]\!]_P = [\![N_2]\!]_P$ implies $[\![FinOut(N_1)]\!]_T = [\![FinOut(N_2)]\!]_T$ we infer $[\![N_3]\!] = [\![N_4]\!]$ contradicting (2). □

5.3 Proof that $[\![\cdot]\!]_P \sqsubset [\![\cdot]\!]_{PQ}$ is a minimal proper inclusion

Let *DIV* be the network with only the empty partial trace, which always diverges, i.e., it has no terminated trace. let *HALFDIV* the the network with the same partial trace, which sometimes diverges and sometimes terminates, i.e., the empty trace is both terminated and divergent.

Lemma 5.9 Let N_1 and N_2 be dataflow networks for which $[\![N_1]\!]_P = [\![N_2]\!]_P$. Assume that there is a finite sequence of events t such that $t \notin Q(N_1)$ but $t \in Q(N_2)$. Then there is a context $\mathcal{C}[\cdot]$, such that

$$\begin{aligned} [\![\mathcal{C}[N_1]]\!]_{PQ} &= [\![DIV]\!]_{PQ} \\ [\![\mathcal{C}[N_2]]\!]_{PQ} &= [\![HALFDIV]\!]_{PQ} \end{aligned}$$

□

Proof: The proof uses the context in Figure 6. The nodes perform the following functions:

Test t can perform the sequence t of communication events, and thereafter sends the datum *stop* to *Clock*. If any more input appears from the hole in the context, then the datum *start* is sent to *Clock*.

Clock Initially performs internal activity without stopping. Upon receiving the datum *stop* it stops its activity. When receiving the datum *start* it again starts its activity.

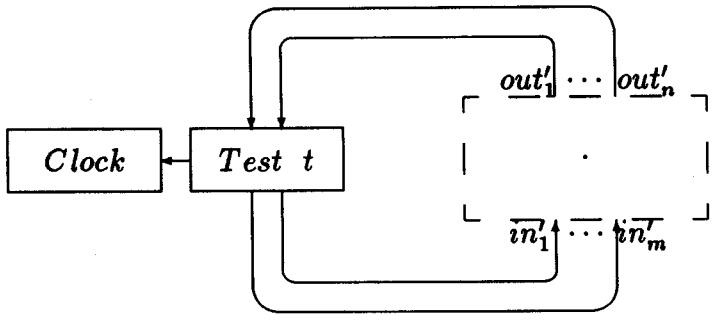

Figure 6: Context in the proof of Lemma 5.9

The consequence of this context is that if in a computation, the sequence of communication events on $E(N_1)$ is not t, the clock is switched on, whereas if the sequence actually is t, then the clock is switched off. It follows that the whole network wil always diverge unless the hole performs t (and nothing more). □

Lemma 5.10 Let N be a dataflow network. Then there is a context such that

$$\begin{aligned} [\![C[HALFDIV]]\!]_{PQ} &= [\![N]\!]_{PQ} \\ [\![C[DIV]]\!]_{PQ} &= [\![DIV \parallel N]\!]_{PQ} \end{aligned}$$

□

Proof: The proof uses the context in Figure 7. The conclusion of the theorem obviously

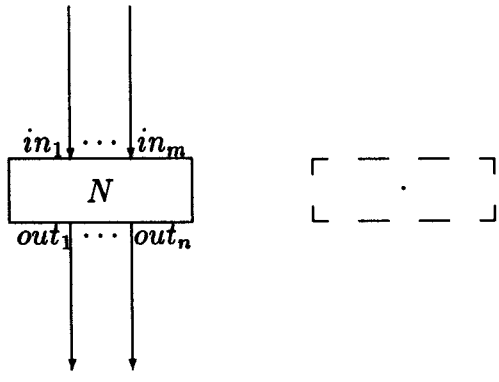

Figure 7: Context in the proof of Lemma 5.10

follows from this context. □

Theorem 5.11 *The gap* $[\![\cdot]\!]_P \sqsubset [\![\cdot]\!]_{PQ}$ *is a minimal proper inclusion for the class of all dataflow networks.* □

Proof: The proof is analogous to the proof of Theorem 5.4: for a compositional model $[\![\cdot]\!]$ in the gap we have that

1. $[\![DIV]\!] = [\![HALFDIV]\!]$,
2. for any N we have $[\![N \parallel HALFDIV]\!] = [\![N]\!]$,
3. $[\![N_3]\!]_P = [\![N_4]\!]_P$ implies $[\![DIV \parallel N_3]\!]_{PQ} = [\![DIV \parallel N_4]\!]_{PQ}$.

□

5.4 Proofs that $[\![\cdot]\!]_{Ev} \sqsubset [\![\cdot]\!]_Q$ and $[\![\cdot]\!]_T \sqsubset [\![\cdot]\!]_{TQ}$ are minimal proper inclusions

Follow the same patterns as the proofs in subsection 5.3.

5.5 Other gaps

We conjecture that the gaps $[\![\cdot]\!]_Q \sqsubset [\![\cdot]\!]_{PQ}$ and $[\![\cdot]\!]_{PQ} \sqsubset [\![\cdot]\!]_{TQ}$ contain both one compositional model. Consider networks that only have divergent traces. This class is closed under parallel composition with any other network. In both gaps we can find a model in between by taking for this subclass as denotation the left-model of the gap and for other networks as the denotation the right-model.

Alexander Rabinovich has showed us in a personal communication that the model $[\![\cdot]\!]_{TD}$ is fully abstract with respect to $[\![\cdot]\!]_D$ and that $[\![\cdot]\!]_{TQD}$ is fully abstract with respect to the $[\![\cdot]\!]_{QD}$.

6 An Intermediate Hierarchy for Strict Networks

In Section 5.2, we proved that the gap $[\![\cdot]\!]_P \sqsubset [\![\cdot]\!]_T$ is proper for the class of all dataflow networks. By examining the proofs of Theorem 5.8 and Lemma 5.5, we find that they rely on the existence of a merge node which outputs all data items from one input channel if there arrive finitely many data items on the other input channel. Such a merge is sometimes called an angelic merge node. In this section, we shall look at a less powerful class of networks which does not contain the angelic merge node. This class is characterized by the property that a node may only read from one input channel at a time, and that a read operation does not time out if no input arrives. This implies that if a read operation is performed when there is no data item in the channel, then the node is blocked forever unless data arrives in the channel. Nodes may exhibit nondeterminism, but only if it is the result of internal choices of a node and independent of whether or not data items arrive on input channels. This class of networks does not include the angelic merge node. If we allow unbounded

nondeterministic internal choices, then the class includes the so-called infinity-fair merge. We shall call this the class of *strict networks* because of its unability to produce output if no input appears during a read operation. As a distinguishing property of this class of networks, we shall take the *prefix property* which is defined by Panangaden and Shanbhogue [25] to prove that the infinity-fair merge is strictly less expressive than the angelic merge.

For sequences h_1 and h_2, we use $h_1 \leq h_2$ to denote that h_1 is a prefix of h_2. The prefix relation is extended to tuples of sequences in the natural way. If t_1 and t_2 are sequences of communication events and E is a set of channels, let $t_1 \preceq_E t_2$ denote that

1. for each channel c in E $t_1\lceil_c \leq t_2\lceil_c$,
2. for each channel c not in E $t_1\lceil_c = t_2\lceil_c$, and
3. for all channels c, c' and integers i, j, such that the ith event on c and the jth event on c' are both in t_1, we have that the ith event on c precedes the jth event on c' in t_1 iff it does so in t_2.

We shall use vector notation to denote tuples of channels. If $\overline{c}$ is a tuple of channels, then $t\lceil_{\overline{c}}$ denotes the tuple of histories on the channels in $\overline{c}$.

Definition 6.1 Let N be a network with a tuple $\overline{in}$ of input channels and a tuple $\overline{out}$ of output channels. N is said to have the *prefix property* if whenever $\overline{h}'$ is a tuple and t is a trace of N with $\overline{h}' \leq t\lceil_{\overline{in}}$ then there is a trace t' of N such that $t'\lceil_{\overline{in}} = \overline{h}'$ and $t' \preceq_{E(N)} t$. □

Intuitively, if a certain input produces a certain output, then any prefix of the input produces some prefix of the same output as one possibility. The following theorem, which is a variant of a theorem proven in [25], shows that the prefix property is preserved by composition of networks.

Theorem 6.2 *If N_1 and N_2 are compatible networks that both have the prefix property, then $N_1 \| N_2$ also has the prefix property.* □

Proof: Assume that t is a trace of $N = N_1 \| N_2$. Then there exists a sequence $\hat{t}$ of communication events on the channels $E(N_1) \cup E(N_2)$ such that $\hat{t}\lceil_{E(N)} = t$ and $t_i = \hat{t}\lceil_{E(N_i)}$ is a trace of N_i for $i = 1, 2$. We shall use the following notation for the channels of N_1 and N_2. Let $\overline{in}_1$ be the tuple of input channels of N_1 which are in $I(N)$, let $\overline{out}_1$ be the tuple of output channels of N_1 which are in $O(N)$, and let $\overline{c}_{12}$ be the tuple of channels that are both output channels of N_1 and input channels of N_2. Define $\overline{in}_2$, $\overline{out}_2$, and $\overline{c}_{21}$ analogously. Now let $\overline{h}'_1 \leq t\lceil_{\overline{in}_1}$ and $\overline{h}'_2 \leq t\lceil_{\overline{in}_2}$. We must show that there is a trace t' of N such that $t'\lceil_{\overline{in}_i} = \overline{h}'_i$ for $i = 1, 2$, and $t' \preceq_{E(N)} t$. Since N_1 has the prefix property, there is a trace t^1_1 of N_1 such that $t^1_1\lceil_{\overline{in}_1} = \overline{h}'_1$, and $t^1_1\lceil_{\overline{c}_{12}, \overline{out}_1} \leq t_1\lceil_{\overline{c}_{12}, \overline{out}_1}$, and $t^1_1 \preceq_{E(N_1)} t_1$. Analogously, there is a trace t^1_2 with analogous properties. If now

t_1^1 and t_2^1 match, i.e. $t_1^1\lceil_{\overline{c}_{12},\overline{c}_{21}} = t_2^1\lceil_{\overline{c}_{12},\overline{c}_{21}}$, then we are done. If not, we must have an inequality in either $t_1^1\lceil_{\overline{c}_{12}} \leq t_2^1\lceil_{\overline{c}_{12}}$ or in $t_2^1\lceil_{\overline{c}_{21}} \leq t_1^1\lceil_{\overline{c}_{21}}$. Since N_1 again has the prefix property, there is a trace t_1^2 of N_1 such that $t_1^2\lceil_{\overline{c}_{21}} = t_2^1\lceil_{\overline{c}_{21}}$, and $t_1^2\lceil_{\overline{c}_{12},\overline{out}_1} \leq t_1^1\lceil_{\overline{c}_{12},\overline{out}_1}$, and $t_1^2 \preceq_{E(N_1)} t_1^1$. Analogously, there is a trace t_2^2 with analogous properties. If now t_1^2 and t_2^2 match, then we are done. Otherwise we continue the process until we find a matching pair t_1^m and t_2^m of traces. The process must terminate wich such a pair, since $\leq$ is a well-founded ordering, and when we hit the empty sequences on $\overline{c}_{12}$ and $\overline{c}_{21}$ they certainly match. The pair t_1^m and t_2^m satisfies $t_1^m\lceil_{\overline{in}_1} = \overline{h}_1'$ and $t_2^m\lceil_{\overline{in}_2} = \overline{h}_2'$, and also $t_1^m\lceil_{\overline{out}_1} \leq t_1\lceil_{\overline{out}_1}$ and $t_2^m\lceil_{\overline{out}_2} \leq t_2\lceil_{\overline{out}_2}$. Together with the fact that t_1^m and t_2^m match, this proves the theorem. □

We shall now define a new compositional model for strict networks. For a network N, define a *shortest trace* of N to be a trace t such that there is no trace t' with $t' \preceq_{O(N)} t$. Intuitively, a shortest trace is a minimal result of a computation where the input of the trace is supplied in the particular way described by the trace. Given any trace t of N, there is always a shortest trace of N which can obtained by removing communication events on some output channels. For a network N let $S(N)$ denote the set of shortest traces of N. As an example, consider a node *copy*123 with one input channel *in* and one output channel *out*, which upon receiving a data item decides internally to produce either one, two, or three copies of the data item onto channel *out*. The shortest traces of *copy*123 are those in which only one copy of each data item is produced, i.e., *copy*123 has the same shortest traces as the simple node *copy* described in Section 4, which just copies without duplication.

Using the set $S(N)$, new models of dataflow networks can defined, analogously to the preceding ones. The following theorem shows that the model $[\![\cdot]\!]_S$ is indeed compositional for the class of strict networks.

Theorem 6.3 *Let N_1 and N_2 be two compatible strict dataflow networks and let N be their composition. Let $C(N) = E(N_1) \cup E(N_2)$. Assume that $t \in S(N)$. Then there is a sequence $t' \in C(N)^\dagger$ such that $t'\lceil_{E(N_i)} \in S(N_i)$ for $i = 1,2$ and $t'\lceil_{E(N)} = t$.* □

Proof: By Theorem 4.1 there is a sequence $\hat{t} \in C(N)^\dagger$ such that $t_i = \hat{t}\lceil_{E(N_i)} \in T(N_i)$ for $i = 1,2$ and $\hat{t}\lceil_{E(N)} = t$. If $t_i \in S(N_i)$ for $i = 1,2$ then we are done. Otherwise, define $\overline{in}_1$, $\overline{out}_1$, $\overline{c}_{12}$, $\overline{in}_2$, $\overline{out}_2$, and $\overline{c}_{21}$ as in the proof of Theorem 6.2. Then there exist shortest traces $t_1^1 \in S(N_1)$ and $t_2^1 \in S(N_2)$ with $t_1^1 \preceq_{O(N_1)} t_1$ and $t_2^1 \preceq_{O(N_2)} t_2$. If now t_1^1 and t_2^1 match, i.e., $t_1^1\lceil_{\overline{c}_{12},\overline{c}_{21}} = t_2^1\lceil_{\overline{c}_{12},\overline{c}_{21}}$, then we are done. If not, we must have an inequality in either $t_1^1\lceil_{\overline{c}_{12}} \leq t_2^1\lceil_{\overline{c}_{12}}$ or in $t_2^1\lceil_{\overline{c}_{21}} \leq t_1^1\lceil_{\overline{c}_{21}}$. Since N_1 again has the prefix property, there is a shortest trace $t_1^2 \in S(N_1)$ with $t_1^2\lceil_{\overline{c}_{21}} = t_2^1\lceil_{\overline{c}_{21}}$, and $t_1^2\lceil_{\overline{c}_{12},\overline{out}_1} \leq t_1^1\lceil_{\overline{c}_{12},\overline{out}_1}$, and $t_1^2 \preceq_{E(N_1)} t_1^1$. Analogously, there is a shortest trace t_2^2 with analogous properties. If now t_1^2 and t_2^2 match, then we are done. Otherwise we continue the process until we find a matching pair t_1^m and t_2^m of shortest traces. The process must terminate wich such a pair, since $\leq$ is a well-founded ordering, and when we hit the empty sequences on $\overline{c}_{12}$ and $\overline{c}_{21}$ they certainly match. The pair t_1^m and t_2^m satisfies $t_1^m\lceil_{\overline{in}_1} = t_1\lceil_{\overline{in}_1}$ and $t_2^m\lceil_{\overline{in}_2} = t_2\lceil_{\overline{in}_2}$, and also $t_1^m\lceil_{\overline{out}_1} = t_1\lceil_{\overline{out}_1}$ and $t_2^m\lceil_{\overline{out}_2} = t_2\lceil_{\overline{out}_2}$ since

because t is a shortest trace of $N_1 \| N_2$ we cannot further shorten the output histories in the composition of t_1^m and t_2^m. The fact that t_1^m and t_2^m match proves the theorem by taking t' such that $t_i^m = t' \lceil_{E(N_i)} \in T(N_i)$ for $i = 1, 2$. □

Using Theorem 6.3 it is easy to see that the shortest-trace model is compositional for strict networks. In the composition, simply compose the shortest traces of the components to yield the shortest traces of the composition. One may in the result have to delete some traces, which have shorter output sequences, but the last theorem shows that we only need the shortest traces of the components to find the shortest traces of the composition. As a remark, the model $[\![\cdot]\!]_{PS}$ is also compositional.

7 Conclusions and Further Work

We have presented a hierarchy of compositional trace models for dataflow networks, and investigated their relationship. The models have been related according to their ability to distinguish networks, and as a major contribution, results about proper gaps have been established.

A conclusion of the work is that in compositional models, the concepts of fairness as represented by infinite traces, and termination represent "indivisible units of description", in the sense that there is no model which can describe a part of the information provided by e.g. the termination component in a compositional way.

We showed that when we restrict the class of dataflow networks, we can add models to the hierarchy. We would like to explore further the hierarchy of compositional models for these networks. It would be interesting to see whether similar ideas can be applied to models of synchronously communicating systems, e.g. CSP, thus extending the work of e.g. [24] and Reed and Roscoe [32, 33].

References

[1] 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.

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

[3] J. Brock and W. Ackerman. Scenarios: a model of non-determinate computation. In *Formalization of Programming Concepts, LNCS 107*, pages 252–259. 1981.

[4] S. Brookes, C. Hoare, and A. Roscoe. A theory of communicating sequential processes. *J. ACM*, 31(3):560–599, 1984.

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

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

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

[8] C. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

[9] B. Jonsson. A model and proof system for asynchronous networks. In *Proc.* 4th *ACM PoDC*, pages 49–58, 1985.

[10] B. Jonsson. *Compositional Verification of Distributed Systems.* PhD thesis, Uppsala University, Sweden, 1987.

[11] B. Jonsson. A fully abstract trace model for dataflow networks. In *Proc.* 16th *ACM PoPL*, pages 155–165, 1989.

[12] B. Jonsson. A hiearchy of compositional models of I/O-automata. In *Proc. MFCS, LNCS 452*, pages 347–354. 1990.

[13] B. Jonsson and J. Kok. Comparing two fully absttract dataflow models. In *Proc. PARLE 89, LNCS 365*, pages 217–234. 1989.

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

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

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

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

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

[19] P. Kosinski. A straight-forward denotational semantics for nondeterminate data flow programs. In *Proc.* 5th *ACM PoPL*, pages 214–219, 1978.

[20] N. Lynch and M. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proc.* 6th *ACM PoDC*, pages 137–151, 1987.

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

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

[23] 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.

[24] E. Olderog and C. Hoare. Specification-oriented semantics for communicating processes. *Acta Informatica*, 23(1):9–66, 1986.

[25] P. Panangaden and V. Shanbhogue. The expressive power of indeterminate dataflow primitives, May 1989. Manuscript.

[26] P. Panangaden, V. Shanbhogue, and E. Stark. Stability and sequentiality in dataflow networks. TR 89-1055, Cornell University, Nov. 1989.

[27] P. Panangaden and E. Stark. Computations, residuals, and the power of indeterminiacy. In *Proc. ICALP '88, LNCS 317*, volume 317 of *Lecture Notes in Computer Science*, pages 439–454. Springer Verlag, 1988.

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

[29] V. Pratt. On the composition of processes. In *Proc.* 9th *ACM PoPL*, pages 213–223, 1982.

[30] V. Pratt. The pomset model of parallel processes: Unifying the temporal and the spatial. In *Proc. Seminar on Concurrency, LNCS 197*, pages 180–196. 1984.

[31] A. Rabinovich and B. Trakhtenbrot. Nets of processes and data flow. In *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, LNCS 354*, pages 574–602. 1989.

[32] G. Reed and A. Roscoe. A timed model for communicating sequential processes. In *Proc. ICALP '86, LNCS 226*, pages 314–323. 1986.

[33] G. Reed and A. Roscoe. Metric spaces as models for real-time concurrency. In *Proc.* 3rd *Workshop on Math. Found. of Progr. Lang. Semantics, LNCS 298.* Springer Verlag, 1988.

[34] J. Russell. Full abstraction for nondeterministic dataflow networks. In *Proc. 30th IEEE FoCS*, 1989.

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

[36] R. J. van Glabbeek and F. W. Vaandrager. Petri net models for algebraic theories of concurency. In *Proc. PARLE, LNCS 259*, pages 224–242. 1987.

[37] J. Zwiers. *Compositionality, Concurrency and Partial Correctness, LNCS 321.* 1989.