Current Trends in the Semantics of Dataflow

Joost N. Kok

RUU-CS-92-22
June 1992

Utrecht University

Department of Computer Science

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

Current Trends in the Semantics of Dataflow

Joost N. Kok

Technical Report RUU-CS-92-22
June 1992

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

ISSN: 0924-3275

Current Trends in the Semantics of Dataflow

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

Abstract

Recently fully abstract models have been proposed for nondeterministic dataflow. The aim of this paper is to present these semantic models for (possibly nondeterministic) dataflow in a uniform way. Besides an operational model we give models based on histories, streams, and traces, and we collect and present the results about full abstraction. Also a method for describing the behavior of networks with equations is given. As a new element we show how the hiaton model (a relational stream model) can be made fully abstract by adding elements to the relation.

1 Introduction

A dataflow net has a finite set of nodes and connections (called channels) between the nodes. The data items that travel over these channels are sometimes called tokens. The channels are directed and behave like (perfect) unbounded first-in/first-out buffers. Channels come in three kinds: internal channels connecting two nodes, input channels and output channels connecting a node with the external world.

We consider only static dataflow networks: the structure of the network is fixed and does not change and each node has a fixed number of incoming and outgoing channels. A semantic model of dataflow nets describes the behavior. In this paper we discuss two new developments in dataflow semantics:

- Fully abstract models [Kok87, Jon89, Rus89, RT89].
- A method for specifying behavior by equations [Mis90].

Our aim is to present these developments in a single framework.

We give four kinds of semantic models for dataflow nets:

1. *Operational Semantics*: The operational model is based on a transition relation and the semantics of a net is given by a set of sequences of transitions. There are three kinds of transitions: input transitions, output transitions and transitions that correspond to computation steps of the nodes.

2. *Trace Semantics*: The trace semantics gives the behavior as a set of sequences (called a trace set) of tuples of channels and tokens.

3. *History Semantics*: The history semantics is a relation over sequences of tokens. Arguments in the relation correspond to channels.

4. *Stream semantics*: The stream semantics is a relation over sequences of tokens plus a special symbol that models silent steps.

The operational semantics is the semantics from which the three other semantics can be derived. For example, the trace semantics can be derived from the operational semantics by picking the labels of input and output transitions.

We define a composition operator on dataflow networks and the trace model is compositional with respect to this operator. A trace set gives a global picture of the behavior of a dataflow net. Therefore it could be of interest to look at more locally distributed semantics. One way to proceed is then to give the semantics as a relation: an argument of the relation corresponds to the contents of a channel. We discuss two relational models: a history model and a stream model. A history semantics assigns to each input and output channel a sequence of tokens. The idea is that such a sequence represents the complete history of the tokens that have passed along this channel during the whole computation. Because we also consider possible nondeterministic behavior of the nodes, the history model is not a function but is given as a relation. The relation for a dataflow net with n input and m output lines has $n + m$ elements: one place for each input and output channel. The history model is in general not compositional with respect to the composition of networks: If we know the history relations of two networks, then we are not always able to construct the history relation for the composition of them. For certain subclasses, the history model is compositional and one such case is given by Kahn in [Kah74].

Next we search for the minimal extension of the history semantics that is compositional. Such a model is called fully abstract (with respect to the history model). We show that the trace model is fully abstract. A consequence is that for the specific case that Kahn considers in [Kah74] the trace and the history semantics are equivalent. In general the trace model distinguishes more dataflow nets than the history model, i.e. is less abstract. (Two dataflow nets are distinguished by the trace model if they have different trace sets.)

Then we ask ourselves the question whether there is a relational model that is fully abstract. A candidate is the stream semantics that gives a relation over streams. A stream is a history to which silent steps (denoted by τ) can be added. Silent steps are used to model an abstract notion of time. They are introduced in [Par83] (where the silent steps are called hiatons). The stream semantics of [Par83] is made fully abstract with respect to the history semantics by adding streams such that a closure condition is satisfied. It becomes then equivalent to the trace semantics. A similar result is shown in [JK89], but there relations over a different domain (vectors of words of tokens) are considered.

Based on [Mis90] we discuss an alternative way to associate sets of traces to dataflow networks: the behavior of a network is specified by sets of equations over traces. One of the problems of this approach is that not all solutions of equations correspond to computations and we have to filter out a subset of the solutions.

The plan of the rest of the paper is as follows. First we give an informal introduction to the dataflow model. Thereafter we give the operational semantics. From that we derive the trace semantics, and we show how to describe trace sets by equations. We introduce the history semantics and we show that the history semantics is in general non-compositional and that the trace semantics is fully abstract with respect to the history semantics. We continue by identifying a subset of networks for which the history semantics is compositional, and we conclude by introducing the stream semantics and show that it is equivalent to the trace semantics.

2 Data Flow model

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

1. *input channels* that transmit data items from the environment to a node,
2. *output channels* that transmit data items from a node to the environment, and
3. *internal channels* that transmit data items from a node to another node in the network.

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

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

Larger networks can be built by *composition* of smaller networks. Given networks $N_1, \ldots, N_k$, where each channel name occurs at most once as an input channel and at most once as an output channel. A composite network is then obtained by connecting input channels to output channels with the same name. The composition operator is a partial operator: for example two networks that both have an input channel with the same name can not be connected according to the definition. (If we want to make such a connecton then we should first do a renaming.)

We can view the resulting network 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.

Next we give a formal definition of the composition operator:

Definition 2.1 *The dataflow networks $N_1, \ldots, N_k$ are called compatible if each channel name occurs at most once as an input channel and at most once as an output channel in $N_1, \ldots, N_k$. Given compatible networks $N_1, \ldots, N_k$, we define their composition*

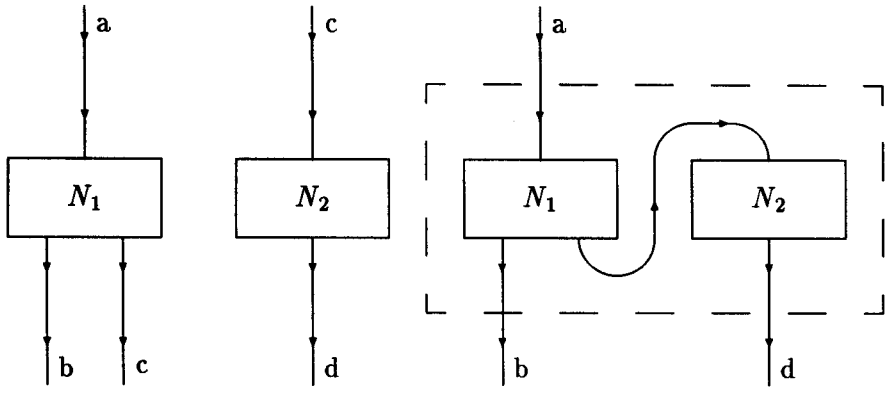

Figure 1: Two networks, N_1 and N_2 (left), and their composition (right).

$N_1 \| \ldots \| N_k$ as the network whose set of nodes is the union of the sets of nodes of $N_1, \ldots, N_k$, and whose set of channels is the union of the channels of $N_1, \ldots, N_k$. The input channels of $N_1 \| \ldots \| N_k$ are those input channels among $N_1, \ldots, N_k$ that to not also occur as output channels, and analogously for the output channels of $N_1 \| \ldots \| N_k$.

Note that our construction does not allow for direct feedback (a network has disjoint input and output channels). However, a direct feedback between two channels can be emulated by composing the network with a buffer that connects the two channels. One of the advantages of not allowing direct feedback is (as we will see later) that the composition operator on trace sets can defined by projections.

3 Operational Semantics

The behavior of dataflow networks can be formalized using labeled transition systems. A labeled transition system has a state, which can be changed by transitions. Transitions can have labels, which represent the communication of data items over channels. Transition systems are often used for operational descriptions of computing systems, e.g. by Plotkin ([Plo81]). First we define a specification of the behavior of individual nodes and then we use these specifications in the definition of the behavior of networks.

We assume a set V of *tokens*, ranged over by d. Let V^* denote the set of finite sequences of data items in V, ranged over by q. The empty sequence is denoted by ϵ. The concatenation of two sequences $q, q' \in V^*$ is denoted by $q.q'$. We assume a set of *nodes*, ranged over by p and a set of *channels* over which c ranges.

Definition 3.1 The *specification* of a node p is a tuple $\langle I_p, O_p, S_p, s_p^0, R_p, \mathcal{F}_p \rangle$ where

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

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

S_p is a set of *states*,

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

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

$\mathcal{F}_p \subseteq \mathcal{P}(R_p)$ is a finite collection of *fairness sets*. Each fairness set is a subset of the set of firings.

The intuitive meaning of a firing $\langle s, \chi_{in}, s', \chi_{out} \rangle$ in a specification of a node p is: "when the node p is in state s and the contents of each incoming channel c starts with the sequence $\chi_{in}(c)$, then these sequences may be consumed, while the node changes its state to s' and the sequence $\chi_{out}(c')$ is produced on each outgoing channel c'". A fairness set represents a set of firings which may not be neglected indefinitely in a run of a network in which the node occurs. For instance, if $\mathcal{F}_p = \{R_p\}$, i.e., the set of all firings is one fairness set, then the node will continue to fire indefinitely, unless it becomes blocked (e.g. for lack of input).

We can now define the behavior of a network N. The definition is a formalization of the intuition that in a network, each node behaves according to its specification, and each channel behaves like an unbounded first-in/first-out buffer.

Definition 3.2 A *specification* of a dataflow network N with the set $P \subseteq$ *nodes* of nodes and the set $C \subseteq$ *channels* of channels is the tuple

$$\langle I_N, O_N, \Sigma_N, \sigma_N^0, R_N, \mathcal{F}_N \rangle,$$

where

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

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

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

σ_N^0 maps each node p to s_p^0, and each channel c to the empty sequence ϵ.

R_N consists of all transitions, which can be generated according to either of the three alternatives below:

1. Internal transitions of one of the nodes: there is a transition $\sigma \xrightarrow{\tau} \sigma'$ if there is a node p with a firing $\langle s, \chi_{in}, s', \chi_{out} \rangle$ and
   - $\sigma(p) = s$, and $\sigma'(p) = s'$.
   - $\sigma'(p') = \sigma(p')$ for all $p' \neq p$.

- $\sigma(c) = (\chi_{in}(c).\sigma'(c))$ for all $c \in I_p$, and $\sigma'(c) = (\sigma(c).\chi_{out}(c))$ for all $c \in O_p$.
- $\sigma(c) = \sigma'(c)$ for channels not in $I_p \cup O_p$.

In other words,

- σ must satisfy $\sigma(p) = s$ and for each incoming channel c to p, the sequence $\chi_{in}(c)$ must be a prefix of $\sigma(c)$.
- σ' differs from σ exactly in that $\sigma(p) = s'$, that for each $c \in I_p$ the initial sequence $\chi_{in}(c)$ has been removed from the state $\sigma(c)$ to get $\sigma'(c)$ and that for each $c \in O_p$ the sequence $\chi_{out}(c)$ has been added to the state $\sigma(c)$ to get $\sigma'(c)$.

2. Input transitions of the form $\sigma \xrightarrow{\langle c,d \rangle} \sigma'$ for channels $c \in I_N$ and $d \in V$, where $\sigma' = \sigma[c \mapsto (\sigma(c).d)]$, i.e., σ' is the same state as σ except that d has been added at the end of c.
3. Output transitions of the form $\sigma \xrightarrow{\langle c,d \rangle} \sigma'$ for output channels $c \in O_N$ and $d \in V$, where $\sigma = \sigma'[c \mapsto (d.\sigma'(c))]$ i.e., σ' is the same state as σ except that d has been removed from the front of c.

$\mathcal{F}_N$ is a finite collection of fairness sets, each of which is of one of the following types:

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

We can now define the *operational semantics* as the set of computations of a network. We say that a transition $\sigma_1 \xrightarrow{e} \sigma_2$ is *enabled* in a state σ if $\sigma = \sigma_1$.

Definition 3.3 A *computation* Γ of a dataflow network N is a finite or infinite sequence

$$\sigma_0 \xrightarrow{e_1} \sigma_1 \xrightarrow{e_2} \cdots \xrightarrow{e_n} \sigma_n \xrightarrow{e_{n+1}} \cdots$$

of transitions, which satisfies the following conditions:

1. σ_0 is the initial state of N,
2. for each $n > 0$, the triple $\sigma_{n-1} \xrightarrow{e_n} \sigma_n$ is a transition of the network,
3. for each fairness set $F \in \mathcal{F}_N$, if there is a σ_n in the computation such that a transition in F is enabled in all states σ_m of the computation with $m \geq n$, then the computation must contain a transition $\sigma_m \xrightarrow{e_{m+1}} \sigma_{m+1} \in F$ with $m \geq n$. Note that for finite computations this condition is equivalent to the condition that no transition from F is enabled in the last state.

4 Trace semantics

Next we give the trace semantics of a dataflow network using the transition system of the previous section. The *trace* of a computation Γ is the sequence of input and output events performed during Γ. A trace can be either finite or infinite. We say that t is a trace of a network N if t is the trace of a computation of N. The denotation T_N of the network N in the trace model is the set of all traces of the network N:

$$T_N = \{t \mid \Gamma \text{ is a computation of } N \text{ and t is the trace of } \Gamma\}.$$

A trace t can be seen as the sequence of communication events that is exchanged with the environment over the input and output channels of the network.

Next we assert that the trace model is compositional with respect to the composition operator on networks. Assume that a network N is the composition of networks $N_1, \ldots, N_k$. We construct the trace set T_N of $N = N_1 \parallel \cdots \parallel N_k$ using the trace sets $T_{N_1}, \ldots, T_{N_k}$ as follows. First introduce the notion of projection: if C is a set of channels then $t|_C$ denotes the subtrace of t in which we only take the communication events over C-channels ("the trace t is projected on C"). Using projections we construct the trace set T_N from the trace sets of the networks $N_1, \ldots, N_k$:

$$T_N = \{\ t|_{(I_N \cup O_N)} \mid t|_{(I_{N_i} \cup O_{N_i})} \in T_{N_i} \text{ for } i = 1, \ldots, k\}$$

Recall that dataflow nets N satisfy the condition that $I_N \cap O_N = \emptyset$ and this enables us to give this direct definition of composition operator based on projections. If we allow direct feedback loops then we need a more complicated definition: see for example [KP85].

Next we address the following question: are there special properties of trace sets that are derived from dataflow networks? An example is the buffering condition for a network N which follows from the fact that all channels in dataflow networks are first-in/first-out buffers:

Definition 4.1 Buffering Condition: *A network N satisfies the buffering condition whenever*

- *t is a trace of the network N, and*
- *the sequence of input/output events t' has the properties*
  - *For each channel c, the sequence of communication events on channel c is the same in t and t'*
  - *For all input channels c_{in}, output channels c_{out}, and positive integers i, j, if in t the i-th communication event on c_{in} precedes the j-th communication event on c_{out}, then that same relationship also holds in t'.*

then t' is also a trace of the network N.

Often this buffering condition is seen as a characteristic property of dataflow networks. and several equivalence-, compositionality- and full abstraction proofs depend on this condition (cf. [Rus89, JK89, RT89]).

5 Trace Equations

In [Mis90] the behavior of networks is given by sets of equations which are called *descriptions.* These sets of equations are an alternative for the operational model.

The solutions of descriptions are traces which give the behavior of the network. Equations are of the form $f := g$ where f and g are functions from traces to some domain. This domain is required to be a complete partial ordering:

Definition 5.1 *A ω-chain in the partial ordering $(X, \sqsubseteq)$ is a sequence $(x_i)_i$ such that $x_i \sqsubseteq x_{i+1}$ for $i = 1, 2, \ldots$.*
For $Y \subseteq X$ we call $x \in X$ an upperbound of Y if

$$\forall y \in Y \;.\; y \sqsubseteq x$$

and is called a least upperbound (lub) if it is an upperbound and

$$\forall x' \in X \;.\; (\forall y \in Y \;.\; y \sqsubseteq x') \Rightarrow x \sqsubseteq x'$$

A ω-complete partially ordered set (cpo) is a triple $(X, \sqsubseteq, \perp)$ with $(X, \sqsubseteq)$ a partial ordering and $\perp \in X$ such that

1. $\forall x \in X \;.\; \perp \sqsubseteq x$
2. *Each ω-chain $(x_i)_i$ in X has a least upperbound $lub_i x_i$ in X*

For example the set of all traces can be turned into a complete partial ordering by choosing the order $\sqsubseteq$ the prefix order $\leq$ on traces: the bottom element is the empty trace.

An example of a function in an equation is the projection on a channel. Connections can be specified by requiring that projections on two channels are the same. The equations are not symmetric: it is not the case that $f := g$ is the same as $g := f$ (this is why we adopt the notation $f := g$ instead of $f = g$).

The lefthand side of an equation represents what is being defined. For example, the equation $f := g$, with $f(t) = t|_{\{b\}}$ and $g(t) = t|_{\{a\}}$, describes an identity node (sometimes also called copy node) which copies the contents of its input channel a to its output channel b.

The problem is that there are in general too many solutions to these equations and only a subset corresponds to actual computations. In order to filter this subset (the *smooth* solutions), we use the complete partial ordering structure. We check which trace solutions correspond to actual computations. For example for the identity node: take solutions traces t such that for every finite prefix t' of t we have that $f(t') \leq g(t')$. For the general restriction we need to make sure that a data item is produced before it is consumed. This can be done by requiring smoothness:

a trace t is a *smooth* solution of $f := g$

whenever

$$f(t) = g(t) \text{ and } \forall t', \langle c, d\rangle \,.\, t' \cdot \langle c, d\rangle \leq t \Rightarrow f(t' \cdot \langle c, d\rangle) \sqsubseteq g(t')$$

In words: if we extend the prefix t' of the trace t by one communication event $\langle c, d\rangle$ then f gives still less (in the cpo ordering) than g.

The nice thing about descriptions is that under certain conditions we can replace left hand sides by right hand sides. (For details we refer to the paper [Mis90].) This replacement can be used to hide connections. Moreover we can take the union of descriptions of networks to get the description for the composition of the networks.

With descriptions more general structures than dataflow networks can be defined. For example up to now we assumed that a channel connects only two nodes. If we use arbitrary descriptions then we can describe channels with several nodes producing and consuming tokens on this channel. This requires an extension of the intuition: a channel should be a multiple buffer (one buffer for each node that consumes tokens from that channel). First assume that there is only one node that produces tokens. Then copies are made of every token that is produced: one copy for each buffer. If there are more producers then they should be consistent, that is produce the same sequence of tokens. Another example is that the buffering condition is not always satisfied for smooth solutions: if we use descriptions for dataflow networks, then we should put identity nodes on all the input and output channels.

We conclude with a small example (more examples can be found in [Mis90]). For this example we assume that tokens are integers and that the name of a channel in an equation denotes the projection from the trace to the history on that channel. Take the following description:

$$\{a := b, d := b, b := 1 \cdot c, c := +1(d)\}$$

This describes a network with four nodes: a node $+1()$ that adds one to each integer that passes this node, a node $1 \cdot c$ that first outputs the integer one and then it copies tokens from its input line to its output line and two copy nodes that send each token its output line. In figure 2 we see the corresponding network.

An example of a trace t is

$$t = \langle b, 1\rangle\langle a, 1\rangle\langle d, 1\rangle\langle c, 2\rangle\langle b, 2\rangle \cdots$$

and there are many more smooth trace solutions for this set of equations.

Here it is allowed to replace lefthand sides by righthand sides of the equations and we then obtain as the set of equations

$$\{a := 1 \cdot (+1(d)), d := 1 \cdot (+1(d))\}$$

Note we have introduced a direct feedback loop and that the channels b, c are hidden. The corresponding network is shown in figure 3.

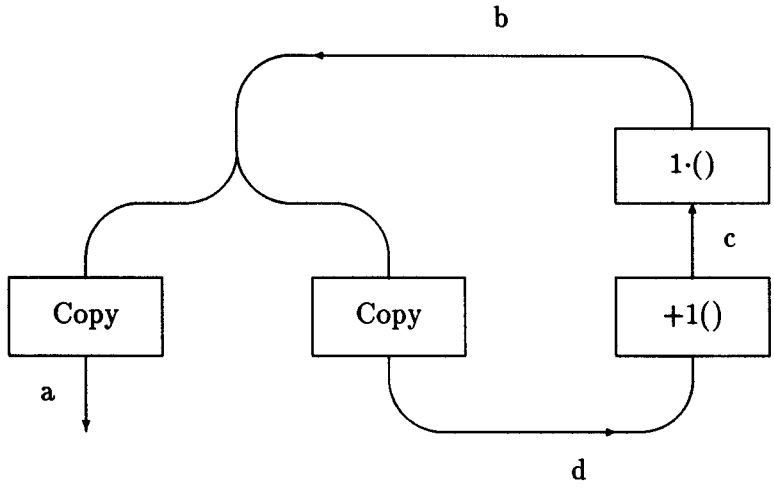

Figure 2: Network corresponding to description

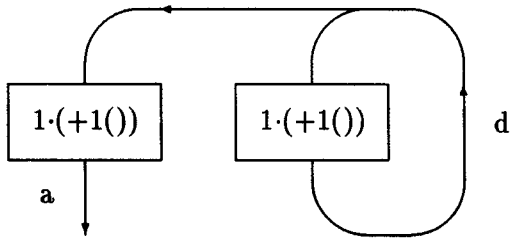

Figure 3: Network after hiding of channels

6 History Semantics

The trace semantics describes the behaviour of a net as a set of communication sequences. In this section we look at a more local model: it describes the behavior of a net by assigning to channels their complete histories (that is, sequences of tokens) that are obtained by projecting traces on channels.

The ***history relation*** H_N of network N is a relation on histories which has an argument for each input and output channel of N that gives the complete sequence of tokens transmitted over that channel in a computation of N. We derive the history relation H_N by using the traces of the network N as follows: (we assume an enumeration $\{c_1, \ldots, c_k\}$ of the input/output channels)

$$H_N = \{(t|_{\{c_1\}}, \ldots, t|_{\{c_k\}}) \mid c_i \in (I_N \cup O_N), i = 1, \ldots k \text{ and } t \text{ is a trace of } N \}.$$

For the full abstraction results in the next section we assume that the history relation is what can be observed from a network (histories are the *observables*). It turns out that

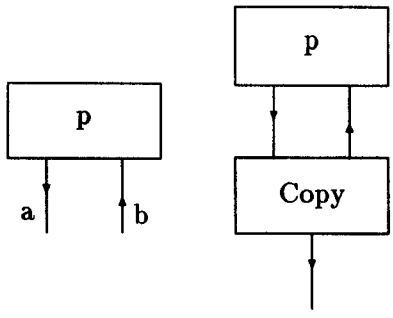

Figure 4: The node p (left) and the node p composed with a *Copy*-node (right)

the fully abstract model is trace model.

Now we turn to the so-called Brock-Ackerman examples that show that the history model is not compositional with respect to the composition operator. In [BA81] it was shown for the first time that a certain class of dataflow networks (with merge nodes) the history model is not compositional. Later more basic examples that do not need merge nodes were given. The one we present here is taken from [Rus89]. Another example can be derived from the proof of the full abstraction of the trace model (which is given in the next section).

Consider the node p of figure 4. It has one input channel and one output channel. Before it starts execution, it decides nondeterministically which of the following two possibilities it is going to perform.

1. either it reads a token from its input channel and outputs a 0 followed by a 1 on its output channel, or
2. it outputs a 0 followed by reading a token from its input channel followed by the output of another 0.

Assume that the incoming channel of p is called b and the outgoing channel a. The history relation is given by ($*$ denotes an arbitrary sequence of tokens of length ≥ 1)

$$\{(*,01),(*,00),(*,01),(\epsilon,\epsilon),(\epsilon,0)\}$$

and the trace set includes

$$\{\langle b,+\rangle\langle a,0\rangle\langle a,1\rangle,\langle a,0\rangle\langle b,+\rangle\langle a,0\rangle\}$$

(where $+$ denotes an arbitrary token).

Next we take a second node p', which behaves the same as p, except for that it can choose a third option: it outputs a 0 followed by reading a token followed by the output of a 1. The history relation of this node p' is the same as for the node p, but there are extra traces, including

$$\langle a,0\rangle\langle b,+\rangle\langle a,1\rangle.$$

But if we compose p and p' with a copy node, then the history relations become different. For the composition with p the possible output histories are the empty history ϵ and 00 and for the context with p' we can have in addition 01. Hence the history semantics is not compositional.

An alternative (as is done in [RT89]) is to consider only the finite prefixes of histories. The same example can be used to show that finite prefixes of histories are also non-compositional. The fully abstract model for finite prefixes of histories are finite prefixes of traces. In [JK91] it is shown that there are no (possibly non-trace based) compositional models between the model based on finite prefixes of traces and the trace model.

7 Full Abstraction

We showed that the trace semantics is compositional by giving the composition operator on trace sets. Now a natural question is

> Is it the minimal extension (that is, does it not make too many distinctions) of the history model that is compositional?

We prove that the trace model is the minimal extension of the history relation that is compositional. First we show that for any two nets N_1 and N_2 with a different trace semantics we can find a context C such that the history semantics of $C[N_1]$ and $C[N_2]$ differ. A context is simply the composition with a network, that is $C[N_1] = N \parallel N_1$ and $C[N_2] = N \parallel N_2$ for some network N. From this result we derive the full abstraction property: the equivalence relation of the trace semantics is the greatest congruence contained in the equivalence relation generated by the history model.

Definition 7.1 *An equivalence relation R_1 in a set A is said to be contained in an equivalence relation R_2 in a set A if $R_1 \subseteq R_2$.*
An equivalence relation R on dataflow nets is called a congruence if it satisfies

$$\forall N_1, N_2 \;.\; R(N_1, N_2) \Rightarrow \forall C \;.\; R(C[N_1], C[N_2])$$

A semantic function $\mathcal{A}$ (a function with domain the set of dataflow nets) generates an equivalence relation $R_{\mathcal{A}}$: $N_1, N_2 \in R_{\mathcal{A}}$ if and only if $\mathcal{A}(N_1) = \mathcal{A}(N_2)$.

Definition 7.2 *A semantics $\mathcal{A}$ is called fully abstract with respect to a semantics $\mathcal{B}$ if $R_{\mathcal{A}}$ is the greatest congruence contained in $R_{\mathcal{B}}$.*

We have the rather surprising fact that we can use a uniform context that does not depend on the trace sets of the networks (such a context was first given in [Kok87]). Here we take a context similar to the one used in [Rus89]. This context has the property that it uses only deterministic nodes. The rest of this section is devoted to the proof of

Theorem 7.3 *The trace semantics is fully abstract with respect to the history semantics.*

We proceed in the following way. We first prove a lemma that gives a sufficient condition for the full abstractness. In a second lemma we show that this condition holds.

Lemma 7.4 *The trace semantics T is fully abstract with respect to the history semantics H if*

$$\forall N_1, N_2 \;.\; T_{N_1} = T_{N_2} \Leftrightarrow \forall C \;.\; H_{C[N_1]} = H_{C[N_2]}$$

Proof
We show that R_T is the greatest congruence contained in R_H:

1. R_T is a congruence by the compositionality of T.
2. R_T is contained in R_H: if two networks have the same trace set, then the history relations are the same: we can derive the history relation from the trace set.
3. Suppose there is a congruence $R \subseteq R_H$. For any networks N_1, N_2 we have

   $$R(N_1, N_2) \Rightarrow$$

   (R is a congruence)

   $$\forall C \;.\; R(C[N_1], C[N_2]) \Rightarrow$$

   ($R \subseteq R_H$)

   $$\forall C \;.\; R_H(C[N_1], C[N_2]) \Rightarrow$$

   (condition in lemma)

   $$R_T(N_1, N_2)$$

   Hence $R \subseteq R_T$. □

Let C be the context of figure 5. Assume that the context is for networks with n input channels and m output channels. We give a short description of the nodes *Checker* and *Feeder* in this context.

checker The checker is a node with two output channels and $m + 1$ input channels. All the output channels of net N are connected to the input channels of the checker and there is one input channel connected to the feeder. The checker is driven by the feeder in the following way: Along the feeder channel it receives tuples of a channel name and a token. If such a channel name is one of the names of the input channels of the checker, it waits till it receives on that channel the corresponding token. If it receives it, then it puts this tuple on both output channels. If a wrong token arrives, it stops execution. If the channel name is not among the inputs, it just copies it to both outputs.

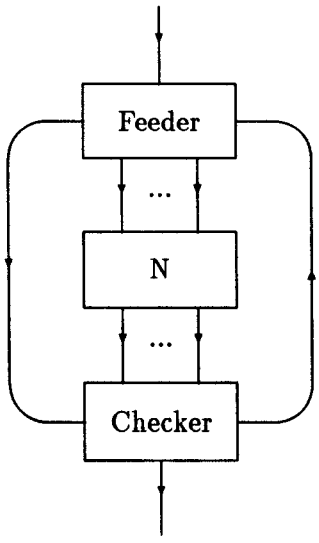

Figure 5: The context used in the full abstraction proof

feeder The feeder has two input channels (one connecting it to the outside world, one to the checker) and $n + 1$ output channels. It is driven by its input channel from the outside world in the following manner. It expects on its input channel tuples of channel names and tokens (i.e. traces). If the channel name is one of its output channels, it sends this token to this channel and moreover it sends it labeled by the channel name on the output channel to the checker. Then the feeder waits till the token comes back from the checker and after this it considers the next input item. If the channel name is not one of its input channels it just sends the tuple to the checker and waits till it returns before continuing.

The crucial lemma in proving the full abstraction is

Lemma 7.5 *For all traces t*

$$t \in T_N \textit{ iff } (t,t) \in H_{C[N]}$$

where C is the context described above.

From this lemma we can obtain the sufficient condition for the full abstraction, as is shown in

Lemma 7.6

$$\forall N_1, N_2 \;.\; T_{N_1} = T_{N_2} \Leftrightarrow \forall C \;.\; H_{C[N_1]} = H_{C[N_2]}$$

Proof:

($\Rightarrow$) Take any two nets N_1, N_2. Assume $T_{N_1} = T_{N_2}$ Take an arbitrary context C. By the compositionality of the trace semantics we derive $T_{C[N_1]} = T_{C[N_2]}$. Because the history semantics can be derived from trace semantics we have $H_{C[N_1]} = H_{C[N_2]}$.

($\Leftarrow$) Take any N_1, N_2 such that for all contexts C we have $H_{C[N_1]} = H_{C[N_2]}$. From the symmetry, it suffices to show

$$T_{N_1} \subseteq T_{N_2}.$$

Take any $t \in T_{N_1}$. We show $t \in T_{N_2}$. Take C as defined before lemma 7.5. By the only-if part of this lemma we have that $(t,t) \in H_{C[N_1]}$. By the assumption we have that $(t,t) \in H_{C[N_2]}$ and hence by the if part of lemma 7.5 also $t \in T_{N_2}$. □

We like to make the following two remarks:

1. In the case that the history model is compositional (for example in the "Kahn" case to be discussed later) it is isomorphic to the trace model. This is a consequence of the full abstraction of the trace model.
2. With the context of the proof used in the full abstraction it is not difficult to construct new Brock-Ackerman examples. Take any two nodes that have the same histories, but different traces: in the context C used in the proof they have different histories.

In [Kah74] it is shown that if the history relations of nodes satisfy certain restrictions, then the history model is compositional and that it can be obtained by a fixed point construction. Let *Hist* denote the set of histories which is a complete partial ordering with the prefix ordering. We need the notion of continuous function:

Definition 7.7 *Let X_1 and X_2 be complete partial orderings. A function $f : X_1 \to X_2$ is called monotonic whenever for all $x_1, x_2 \in X_1$, if $x_1 \sqsubseteq x_2$ then $f(x_1) \sqsubseteq f(x_2)$. A function $f : X_1 \to X_2$ is called continuous whenever it is monotonic and, for each $\omega-$chain $(x_i)_i$ in X_1 we have $f(lub_i x_i) = lub_i f(x_i)$.*

The restriction of [Kah74] is

Restriction 7.8 *Consider only nodes p such that the history relation H_p is a continuous function from input histories to output histories.*

(See [LS89] for a discussion of this restriction in the context of Input/Output automata.)

We assert that history relations are compositional for networks that satisfy the Kahn restriction. We show how to do the composition of history relations and this involves a fixed point construction. We will use the following lemma:

Lemma 7.9 *Let f be a continuous mapping from a complete partial ordering X into itself. Then f has a least fixed point $\mu f = lub_i f^i(\perp)$ satisfying*

1. $f(\mu f) = \mu f$

2. *if* $f(y) = y$ *then* $\mu f \sqsubseteq y$.

For the rest of this section assume a net N in which all the nodes satisfy this restriction. Let $N = N_1 \parallel \cdots \parallel N_k$. For each network N_i, $i = 1, \ldots, k$, the history relation H_{N_i} is a continuous function in

$$(I_{N_i} \to Hist) \to (O_{N_i} \to Hist)$$

(we lift the ordering pointwise to functions). Let C_N be the set of all output channels of $N_1, \ldots, N_k$, that is $C_N = O_{N_1} \cup \cdots \cup O_{N_k}$. We first define a function

$$\Psi : (I_N \to Hist) \times (C_N \to Hist) \to (C_N \to Hist).$$

by

$$\Psi(\phi_1, \phi_2)(c) = H_j(\phi)(c)$$

where j is such that c is an output channel of N_j, that is $c \in O_{N_j}$ (j is unique) and where

$$\phi \; : \; I_{N_j} \to Hist$$

$$\phi = \lambda c. \begin{cases} \phi_1(c) & \text{if } c \in I_N \\ \phi_2(c) & \text{if } c \notin I_N \end{cases}$$

The idea is that we construct new inputs for the history functions of the subnetworks: if a channel is among the input channels of N we use the first argument of Ψ and if not we use the second argument. Then we can derive

$$H_N : (I_N \to Hist) \to (O_N \to Hist)$$

by a fixed point construction (the feedback channels should have the same histories)

$$H_N(\phi_1) = \mu(\lambda \phi_2 . \Psi(\phi_1, \phi_2)).$$

The least fixed point μ can be obtained by iteration from the bottom element of the cpo (the function that assigns to every channel in C_N the empty history).

8 Stream Semantics

In this section we give a relational fully abstract semantics that is a generalization of the hiaton model given by Park in [Par83]. Like the history semantics, it is defined as a relation, but now over streams. A stream is a finite or infinite sequence of which each

element is either a data item in V or a special symbol τ. A history is derived from a stream by forgetting about the τ's.

From the trace semantics T_N we can derive the following stream semantics S_N for a network N.

For a stream s we denote by $s[n]$ the prefix of length n and $hist(s)$ its associated history.

Now $S_N(s_1, \ldots, s_k)$ holds if there is a trace $t \in T_N$ such that:

1. the histories $(hist(s_1), \ldots, hist(s_k))$ are the histories of t, and
2. for each n there is a prefix t' of t such that $(hist(s_1[n]), \ldots, hist(s_k[n]))$ are the corresponding histories of t'.

(The histories of a trace t are the projections $t|_{\{c\}}$ on channels c.)

It is also possible to derive the trace semantics from the stream semantics. Given that the trace semantics is fully abstract, we can conclude that the stream semantics is also fully abstract.

We give (without proof) the closure condition that is satisfied by our stream model and that should be applied to Park's model. It corresponds to the buffering condition on traces: (let $s(i)$ denote the i-th element of the stream s)

Lemma 8.1 *Assume that $S_N(s_1, \ldots, s_k)$ and that there is an integer α such that for each s in $s_1, \ldots, s_k$ either $s(\alpha) = \tau$ or $s(\alpha + 1) = \tau$. If we remove from each s this τ-element and obtain $s'_1, \ldots, s'_k$ then also $S_N(s'_1, \ldots, s'_k)$.*

For example, if $S_N(\tau 0, 1\tau, \tau\tau, 1\tau)$ then also $S_N(0, 1, \tau, 1)$.

Acknowledgements

We like to thank Bengt Jonsson for the stimulating discussions on the topic of this paper. The first parts of the paper are adapted versions of sections originally written by him. We also like to thank Kaisa Sere for all her efforts to improve the presentation. The members of the Amsterdam Concurrency Group headed by Jaco de Bakker are also acknowledged. Moreover we like to acknowledge the anonymous referees for their valuable suggestions.

References

[BA81] J. Brock and W. Ackerman. Scenarios: a model of non-determinate computation. In *Formalization of Programming Concepts*, number 107 in Lecture Notes in Computer Science, pages 252–259. Springer Verlag, 1981.

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

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

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

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

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

[CM81] K. Chandy and J. Misra. Proofs of networks of processes. *IEEE Transactions on Software Engineering*, SE-7(4):417–426, July 1981.

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

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

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

[JK91] B. Jonsson and J. Kok. Towards a complete hierarchy of compositional dataflow models. In *Proceedings Theoretical Aspects of Computer Software*, volume 526 of *Lecture Notes in Computer Science*, pages 204–225. Springer Verlag, 1991.

[Jon85] B. Jonsson. A model and proof system for asynchronous networks. In *Proceedings* 4^{th} *ACM PoDC*, pages 49–58, 1985.

[Jon87] B. Jonsson. *Compositional Verification of Distributed Systems.* PhD thesis, Uppsala University, Sweden, 1987.

[Jon89] B. Jonsson. A fully abstract trace model for dataflow networks. In *Proc.* 16^{th} *ACM PoPL*, pages 155–166, 1989.

[Jon90] B. Jonsson. A hierarchy of compositional models of I/O-automata. In *Proceedings MFCS*, volume 452 of *Lecture Notes in Computer Science*, pages 347–354. Springer Verlag, 1990.

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

[Kok86] J. Kok. Denotational semantics of nets with nondeterminism. In *European Symposium on Programming*, volume 206 of *Lecture Notes in Computer Science*, pages 237–249. Springer Verlag, 1986.

[Kok87] J. Kok. A fully abstract semantics for data flow nets. In *Proceedings PARLE*, volume 259 of *Lecture Notes in Computer Science*, pages 351–368. Springer Verlag, 1987.

[Kos78] P. Kosinski. A straight-forward denotational semantics for nondeterminate dataflow programs. In *Proceedings* 5^{th} *ACM PoPL*, pages 214–219, 1978.

[KP85] R. Keller and P. Panangaden. Semantics of networks containing indeterminate operators. In *Seminar on Concurrency*, volume 197 of *Lecture Notes in Computer Science*, pages 479–496. Springer Verlag, 1985.

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

[LS89] N. Lynch and E. Stark. A proof of the kahn principle for input/output automata. *Information and Computation*, 82(1):81–92, July 1989.

[LT87] N. Lynch and M. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proceedings* 6^{th} *ACM PoDC*, pages 137–151, 1987.

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

[Mis90] J. Misra. Equational reasoning about nondeterministic processes. *Formal Aspects of Computer Science*, 2:167–192, 1990.

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

[OH86] E. Olderog and C. Hoare. Specification-oriented semantics for communicating processes. *Acta Informatica*, 23(1):9–66, 1986.

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

[Plo81] G. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, 1981.

[Pra82] V. Pratt. On the composition of processes. In *Proceedings* 9^{th} *ACM PoPL*, pages 213–223, 1982.

[Pra84] V. Pratt. The pomset model of parallel processes: Unifying the temporal and the spatial. In *Proceedings Seminar on Concurrency*, volume 197 of *Lecture Notes in Computer Science*, pages 180–196. Springer Verlag, 1984.

[PS88] P. Panangaden and E. Stark. Computations, residuals, and the power of indeterminacy. In *Proceedings ICALP*, volume 317 of *Lecture Notes in Computer Science*, pages 439–454. Springer Verlag, 1988.

[PS89] P. Panangaden and V. Shanbhogue. The expressive power of indeterminate dataflow primitives. 1989. Manuscript.

[PSS90] P. Panangaden, V. Shanbhogue, and E. Stark. Stability and sequentiality in dataflow networks. In *Proceedings ICALP*, number 443 in Lecture Notes in Computer Science, pages 181–194. Springer Verlag, 1990.

[RR86] G. Reed and A. Roscoe. A timed model for communicating sequential processes. In *Proceedings ICALP*, volume 226 of *Lecture Notes in Computer Science*, pages 314–323. Springer Verlag, 1986.

[RR88] G. Reed and A. Roscoe. Metric spaces as models for real-time concurrency. In *Proceedings 3rd Workshop on Mathematical Foundations of Programming Language Semantics*, volume 298 of *Lecture Notes in Computer Science*, pages 331–344. Springer Verlag, 1988.

[RT89] 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*, volume 354 of *Lecture Notes in Computer Science*, pages 574–602. Springer Verlag, 1989.

[Rus89] J. Russell. Full abstraction for nondeterministic dataflow networks. In *Proceedings 30th IEEE FoCS*, 1989.

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

[vGV87] R. van Glabbeek and F. Vaandrager. Petri net models for algebraic theories of concurrency. In *Proceedings PARLE*, volume 259 of *Lecture Notes in Computer Science*, pages 224–242. Springer Verlag, 1987.

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