McGill University
School of Computer Science

A Kahn principle for networks of nonmonotonic real-time processes

ACAPS Technical Memo 60

December 6, 1992

Robert Yates

Guang Gao

Advanced Computer Architecture and Program Structures Group

3480 University St • Montreal • Canada • H3A 2A7

Contents

Abstract

We show that the input-output function computed by a network of asynchronous real-time processes is denoted by the unique fixed point of a Scott continuous functional *even though the network or its components may compute a discontinuous function.* This extends a well-known principle of Kahn [Kahn, 1974] to an important class of parallel systems that has resisted the traditional fixed point approach.

We present a fully abstract order-theoretic denotational semantics for networks of asynchronous real-time processes. The time-sensitive nature of the component processes allows them to compute functions which are not Scott continuous, nor even monotonic, on the domain of timed message streams ordered by the usual prefix relation. Because of the discontinuous behavior of the components, the characterization of networks with nonmonotonic processes as fixed points of continuous functionals (the standard approach of denotational semantics, applied to untimed networks of monotonic processes by Kahn in 1974) has been a much-sought but unattained goal. We show that the function computed by any real-time network, even those containing nonmonotonic processes, is identical to the unique fixed point of a continuous network functional whose construction is original.

1 Introduction

Complete partial orders (cpos)[1] and Scott continuous[2] functions as used in the standard approach to denotational semantics [10, 23, 33] embody a notion of finite approximation of infinite limits that is particularly appropriate for digital computation. Moreover, the

[1]A *partial order* $(D, \sqsubseteq)$ is a set D with a binary relation $\sqsubseteq$ that is reflexive, transitive and antisymmetric. Partial order $(D, \sqsubseteq)$ is *complete* if every chain in D has a least upper bound in D. A *chain* is a nonempty subset $X \subseteq D$ such that either $x \sqsubseteq y$ or $y \sqsubseteq x$ for all $x, y \in X$. An element $u \in D$ is an *upper bound* of $X \subseteq D$ if $x \sqsubseteq u$ for all $x \in D$, and u is the *least* upper bound of X (written $u = \bigsqcup X$) if $u \sqsubseteq z$ whenever z is an upper bound of X.

[2]A function $f : D \to E$ from cpo $(D, \sqsubseteq)$ to cpo $(E, \leq)$ is *Scott continuous* if $f(\bigsqcup_{\sqsubseteq} X) = \bigsqcup_{\leq} \{ f(x) \mid x \in X \}$ for any chain X. Then f is also monotonic, i.e., $f(x) \leq f(y)$ whenever $x \sqsubseteq y$.

Knaster-Tarski theorem and variants thereof guarantee the existence of fixed points to serve as solutions of sets of equations resulting from recursive programs or data types. In particular, if $f : D \to D$ is a continuous function on cpo $(D, \sqsubseteq)$ then $\mu f \stackrel{\text{def}}{=} \bigsqcup_{\sqsubseteq} \{ f^n(\perp) \mid n \geq 0 \}$ [3] is guaranteed to be the least fixed point of f. That is, μf is the least element in D such that $\mu f = f(\mu f)$.

Consider the program `f(x) = (cons x (cdr (f x)))` in a lazy variant of a Lisp-like language. The denotation of `f` is the least function satisfying the recursive equation $f(x) = \mathrm{cons}\big(x, \mathrm{cdr}(f(x))\big)$, where functions are ordered pointwise by prefix.[4] To solve the equation, eliminate the recursion to obtain a nonrecursive functional, then take its least fixed point. Thus the meaning of `f` is given by function μF, for functional $F = \lambda f.\, \lambda x.\, \mathrm{cons}\big(x, \mathrm{cdr}(f(x))\big)$. Given any input x, the recursive equation for $f(x)$ will be satisfied by *any* list whose first element is x. This implies that there are many fixed points of F. The *least* fixed point is chosen as the denotation of `f` because it generates no gratuitous information, hence it corresponds to the input-output behavior expected from program `f`. E.g., if $x = a$ then $f(a) = a$ and $f(a) = ab^\infty$ are both valid solutions of equation $f(a) = \mathrm{cons}\big(a, \mathrm{cdr}(f(a))\big)$, but the least fixed point of F yields only the least solution: $[\mu F](a) = a$.

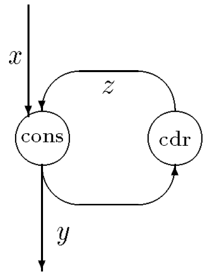
How do we know that functional F really has a fixed point? If F is continuous then the standard fixed point theorem of Scott continuous functions on cpos guarantees that the least fixed point μF exists and is given by the equation $\mu F = \bigsqcup \{ F^n(\perp) \mid n \geq 0 \}$ (where $\perp$ here is the "everywhere empty" function that maps any pair of input streams to the empty stream Λ). But how do we know that F is continuous? The usual justification is that the components of F — cons, cdr and f [5] — are continuous, that application of a function to its arguments is continuous, and that function composition preserves continuity.

Kahn [13] was the first to apply fixed point techniques to parallel programs. Kahn considered networks of determinate processes communicating asynchronously over unbounded

[3]Where $\perp$ is the least or so-called "bottom" element of D, f^n denotes repeated application of f n times, and where $\bigsqcup_{\sqsubseteq} \{ f^n(\perp) \mid n \geq 0 \}$ denotes the least upper bound of chain $\{ f^n(\perp) \mid n \geq 0 \}$.

[4]I.e., function f is less than or equal to function g if $f(x)$ is a prefix of $g(x)$ for all inputs x.

[5]Dummy component f is known to be continuous because the functional F is defined to have the set of all Scott continuous functions of the proper type as its domain.



The network equations for `f`:
x is the input to the network
$y = \mathrm{cons}(x, z)$ is the network output
$z = \mathrm{cdr}(y)$ is an internal arc

Figure 1: A network for program `f(x) = (cons x (cdr (f x)))`.

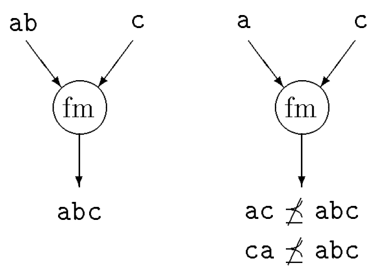
FIFO channels, and observed that the input-output function of any network is denoted by the least fixed point solution of the network's recursion equations. Program `f` could be a network of processes as shown in Figure 1.

The Kahn principle is among the most important and elegant results in the theory of concurrency. However, in its original form it applies only to restricted programming models. In [13], a component process trying to read from one of its input ports must wait until it can read a token before proceeding. This restriction precludes any problematic behavior by ensuring that the component processes compute functions that are continuous on the cpo of streams ordered by prefix.[6]

Some important stream operators are not so well behaved. Consider the *fair merge* operator which takes two possibly infinite streams of tokens and produces an interleaving of them, preserving the relative ordering of tokens from each individual input stream within the result. Clearly this requires the ability to check an input for the presence of data and continue on to do other things if none is present[7] rather than blocking as in Kahn's formulation. Fair merge mimics race conditions in concurrent systems and has been intensively studied [3, 4, 5, 6, 8, 9, 14, 16, 15, 22, 25, 27, 28, 29, 30]. Keller [15] was the first to remark that fair merge is not monotonic in the cpo of streams ordered by prefix (Fig. 2).

[6]More precisely, the domain is the set of stream tuples ordered coordinatewise by prefix, i.e., $(x_1, \ldots, x_n)$ is less than or equal to $(y_1, \ldots, y_n)$ if and only if x_i is a prefix of y_i for all $i \in 1..n$.

[7]This intuition is proved valid by Panangaden and Stark in [28], which also showed that there is a hierarchy of non-determinate merges ranked by fairness.



Input `a` is a prefix of `ab` and `c` is a prefix of `c`, but neither of the possible outputs `ac` nor `ca` is a prefix of `abc`.

Figure 2: Fair merge is not monotonic.

Input `a@`1 (meaning token `a` arrives at time 1) is a prefix of `a@`1_`b@`2 and `c@`3 is a prefix of `c@`3, but output `a@`$(1+\Delta)$_`c@`$(3+\Delta)$ is not a prefix of `a@`$(1+\Delta)$_`b@`$(2+\Delta)$_`c@`$(3+\Delta)$ (Δ is the time needed to transfer a token from input to output). Token sequence `c@`$(3+\Delta)$_`a@`$(1+\Delta)$ is not a meaningful output because the times are not ascending.

Figure 3: Timed merge is not monotonic.

We shall not deal here with non-determinate fair merge, but add a time tag to every token and let *timed merge* interleave them by their arrival order. The primary reason for this is a practical one: real-time systems are extremely important in industry and we would like to contribute to the understanding of them.

We take a real-time network to be a fixed finite set of asynchronous determinate processes $A_1, \ldots, A_k$ that communicate by sending time-stamped tokens over unbounded FIFO channels. For each process or "node" A_a, $\vec{A}_a : \mathcal{S}^{i_a} \to \mathcal{S}^{o_a}$ denotes the function computed by A_a, where $i_a = \#(A_a^{in})$ is the number of input ports (channels) and $o_a = \#(A_a^{out})$ is the number of output ports of A_a. The set of all node output ports is denoted by A^{out}. Token values are chosen from a finite alphabet, and the sequence of tokens that appears on a channel forms a *stream*, with the time stamps strictly increasing and separated by a minimum temporal interval $\Delta > 0$. In this article we assume a single common alphabet, and denote the set of timed streams over this alphabet by $\mathcal{S}$. The time stamps are real values and represent the time of appearance of a token as measured by a conceptual global clock. Each real-time processes executes a sequence of actions determined by its current state and the presence or absence of input tokens. A process may check one or more of its input channels for tokens and proceed to other things whether tokens are present or not. Every action consumes some determinate amount of time, greater than or equal to Δ. Communication between processes is instantaneous; delays can be modelled by interposing delay processes, with unpredictable delays modelled by providing the processes with extra inputs to control the amount of delay added between tokens. A network with i input and s output channels computes a function from i-tuples of streams to s-tuples of streams. Many real systems, from automated bank tellers to automated factories, can be described by such networks.

As shown in Figure 3, timed merge is also nonmonotonic. It is easy to think of other real-time processes which compute nonmonotonic functions, e.g., the process which emits a token every cycle until it receives its first input token. It is this nonmonotonic (hence discontinuous) behavior that has made it difficult to extend Kahn's principle to networks containing merge nodes. In the traditional approach, the existence of the least fixed point of the functional for a set of recursive equations is guaranteed by the Scott continuity of the components. *The functional of a real-time network can have nonmonotonic components, so how can the existence of the least fixed point be guaranteed?*

In the next section we review the traditional network functional for untimed networks of monotonic components. In section 3 we introduce the new network functional for real-time networks with nonmonotonic components. In section 4 we show that the new functional is continuous, and that its least fixed point describes the behavior of the network. Section 5 compares the results to previous work, and conclusions are drawn in the final section, 6.

2 The traditional functional for monotonic networks

In this section we aim to clarify the intuitions behind network functionals. Given a network N with i inputs and s outputs (one for every node output port[8]) composed of a collection of determinate blocking-read processes or "nodes" $A_1, \ldots, A_k$, Kahn's principle states that the function NetFun_N operationally computed by N is described by the least fixed point $\mu\Phi$ of a functional Φ associated with N; i.e., $\mathrm{NetFun}_N = \mu\Phi$, where the functional $\Phi : [\mathcal{S}^i \to \mathcal{S}^s] \to [\mathcal{S}^i \to \mathcal{S}^s]$ associated with network N is defined by the set of k equations

$$\Phi(f)(\sigma) \downharpoonleft A_a^{out} \quad = \quad \vec{A}_a\big((\sigma \otimes f(\sigma)) \downharpoonleft A_a^{in}\big)$$

for all continuous functions $f \in [\mathcal{S}^i \to \mathcal{S}^s]$, input tuples $\sigma \in \mathcal{S}^i$ and nodes $A_a \in A_1, \ldots, A_k$. The domain of functional Φ is the cpo $\big([\mathcal{S}^i \to \mathcal{S}^s], \overset{i\to s}{\preceq}\big)$ of all Scott continuous stream functions of i inputs and s outputs, with the functions ordered pointwise by prefix (i.e., $f \overset{i\to s}{\preceq} g$ if and only if $f(\sigma) \preceq^s g(\sigma)$ for all $\sigma \in \mathcal{S}^s$, where $\preceq^s$ is the coordinatewise prefix order: $(x_1, \ldots, x_n) \preceq^s (y_1, \ldots, y_n)$ if and only if $x_i \preceq y_i$ for all $i \in 1..n$ (each stream x_i is a prefix of the corresponding stream y_i)).

In the left side of the definition of the traditional Φ, expression $\Phi(f)(\sigma)$ denotes the s-tuple representing the value of all the output streams, in some canonical order, produced by function $\Phi(f) : \mathcal{S}^i \to \mathcal{S}^s$ given input $\sigma \in \mathcal{S}^i$; applying $\downharpoonleft A_a^{out}$ then yields an o_a tuple by

[8] In our discussion of the Kahn principle, we (in common with [19]) treat every node output as a network output even though it might only be attached to wires internal to the network.

selecting only those streams representing the values at the output ports of node A_a, again in some canonical order. In the right side, $\sigma\otimes f(\sigma)$ denotes the tuple obtained by joining input $\sigma\in\mathcal{S}^i$ and output $f(\sigma)\in\mathcal{S}^s$ together into a single $i{+}s$-tuple; thus $\sigma\otimes f(\sigma)$ represents the contents of all the source ports of network N computing function f presented with input σ. Then applying $\lfloor A_a^{in}$ to $\sigma\otimes f(\sigma)$ yields an i_a-tuple by selecting only those streams representing the values at the input ports of node A_a. Finally, $\vec{A}_a : \mathcal{S}^{i_a} \rightarrow \mathcal{S}^{o_a}$ is the function computed by node A_a.

The definition of the network functional is related to the *network equations* $H\lfloor A_a^{out} = \vec{A}_a\big(\, H\lfloor A_a^{in} \,\big)$ $(\forall a\in 1..k)$, in the "unknown" variable $H\in\mathcal{S}^{i+s}$.

Because the node processes in the original Kahn principle are required to block on reads until data has arrived, every node function $\vec{A}_a$ is Scott continuous in the coordinatewise prefix order, i.e., from domain $(\mathcal{S}^{i_a}, \preceq^{i_a})$ to codomain $(\mathcal{S}^{o_a}, \preceq^{o_a})$. Moreover, every function used in the definition of the network functional Φ is continuous, so Φ itself is continuous on cpo $\big([\mathcal{S}^i\rightarrow\mathcal{S}^s], \overset{i\to s}{\preceq}\,\big)$ by the fact that composition preserves continuity. Then a standard theorem of semantic domain theory states that the least fixed point of Φ exists and is given by $\mu\Phi \overset{\text{def}}{=} \bigsqcup_{\overset{i\to s}{\preceq}} \big\{\, \Phi^n(\overset{i\to s}{\Lambda}) \;\big|\; n\in\mathbb{N} \,\big\}$ where $\overset{i\to s}{\Lambda}$ is the "everywhere empty" function (defined by $\overset{i\to s}{\Lambda}(\sigma) = \Lambda^s$ for all $\sigma\in\mathcal{S}^i$), the bottom element of Φ's domain $\big([\mathcal{S}^i\rightarrow\mathcal{S}^s], \overset{i\to s}{\preceq}\,\big)$. Since the application operator $\cdot(\sigma) : [\mathcal{S}^i\rightarrow\mathcal{S}^s] \rightarrow \mathcal{S}^s$ is $\overset{i\to s}{\preceq}$–$\preceq^s$-continuous[9] by a standard theorem of semantic domain theory, we obtain $\mu\Phi(\sigma) = \bigsqcup_{\preceq^s} \big\{\, \overset{i\to s}{\Lambda}(\sigma),\ \Phi(\overset{i\to s}{\Lambda})(\sigma),\ \Phi^2(\overset{i\to s}{\Lambda})(\sigma), \ldots \big\}$. In other words, the least fixed point of network functional Φ is a function $\mu\Phi : \mathcal{S}^i \rightarrow \mathcal{S}^s$ whose value at any given input tuple σ is the "limit" (least upper bound) of the increasing sequence $\overset{i\to s}{\Lambda}(\sigma),\ \Phi(\overset{i\to s}{\Lambda})(\sigma),\ \Phi^2(\overset{i\to s}{\Lambda})(\sigma), \ldots$.

To visualize this sequence, picture first the network N with the streams of input tuple σ placed on their individual network input wires, with all other wires holding the empty stream, represented by the $i{+}s$-tuple $\sigma\otimes\Lambda^s$; subtuple Λ^s is equal to $\overset{i\to s}{\Lambda}(\sigma)$, the first element of the sequence. To get the second element in the sequence, the definition of Φ instructs us to apply every node function, selecting the relevant input streams from tuple $\sigma\otimes\Lambda^s$ (the overall network trace generated by the previous application $\overset{i\to s}{\Lambda}(\sigma)$), then collect all the

[9]That is, $\cdot(\sigma)$ is a continuous function from domain $([\mathcal{S}^i\rightarrow\mathcal{S}^s], \overset{i\to s}{\preceq})$ to codomain $(\mathcal{S}^s, \preceq^s)$.

node outputs together into a new s-tuple of streams, call it θ, which is equal to $\Phi(\overset{i\to s}{\Lambda})(\sigma)$. Similarly, to get the third element in the sequence, apply every node function again, this time selecting the relevant input streams from $\sigma \otimes \theta$, the network trace generated by the previous application $\Phi(\overset{i\to s}{\Lambda})(\sigma)$, then collect all the node outputs together into yet another new s-tuple of streams, which is equal to $\Phi(\Phi(\overset{i\to s}{\Lambda}))(\sigma)$. Monotonicity of Φ ensures that $\Phi^n(\overset{i\to s}{\Lambda}) \overset{i\to s}{\preceq} \Phi^{n+1}(\overset{i\to s}{\Lambda})$, from which it follows that $\Phi^n(\overset{i\to s}{\Lambda})(\sigma) \preceq^s \Phi^{n+1}(\overset{i\to s}{\Lambda})(\sigma)$, i.e., for any given network output, the stream at that output in step n is always a prefix of the stream at that output in step $n+1$. Repeat the process until no change is seen on any of the wires. The least fixed point $\mu\Phi$ of network functional Φ is reached when no change is seen for *any* input $\sigma \in \mathcal{S}^i$. Figure 4 depicts an example of this scheme applied to a simple 1-input, 3-output network for the particular input point $\sigma =$ `ac`.

What happens if we use the traditional functional on a network containing nodes which compute nonmonotonic functions like timed merge? Figure 5 shows a network similar to that of Figure 4, but where all the tokens carry time tags and all the nodes impart a delay of one time unit; moreover, the blocking-read *alt* node has been replaced by a non-blocking timed merge node *tm*, which passes input tokens to the output in the order of their arrival, regardless of which input stream they belong to (with some rule for settling ties). Observe in Figure 5 that in going from Φ^3 to Φ^4 the network output changes from (`a@2_c@8`, `a@3`, `b@4`) to (`a@2_b@5_c@8`, `a@3`, `b@4`), wherein `a@2_c@8` is not a prefix of `a@2_b@5_c@8`, so set $\{ \overset{i\to s}{\Lambda}(\sigma),\ \Phi(\overset{i\to s}{\Lambda})(\sigma),\ \Phi^2(\overset{i\to s}{\Lambda})(\sigma),\ \ldots \}$ is not a chain in the (coordinatewise) prefix order $\preceq^3$, nor is $\{ \overset{i\to s}{\Lambda},\ \Phi(\overset{i\to s}{\Lambda}),\ \Phi^2(\overset{i\to s}{\Lambda}),\ \ldots \}$ a chain in the pointwise prefix order $\overset{1\to 3}{\preceq}$. Thus it is clear that when the network nodes are nonmonotonic, the traditional network functional Φ may itself be nonmonotonic. If the network functional is not monotonic then neither is it continuous, and the whole underpinning of the traditional Kahn principle collapses.

3 The new functional for real-time networks

How might one redefine the network functional to extend the Kahn principle to networks containing nonmonotonic processes like the timed merge of Figure 5? One thing that must

Alt takes input tokens from its left and right inputs in strict alternation, *hd* takes the first token from its input, and suc converts the value of every token to its successor in the alphabetic order (wrapping around from `z` to `a`).

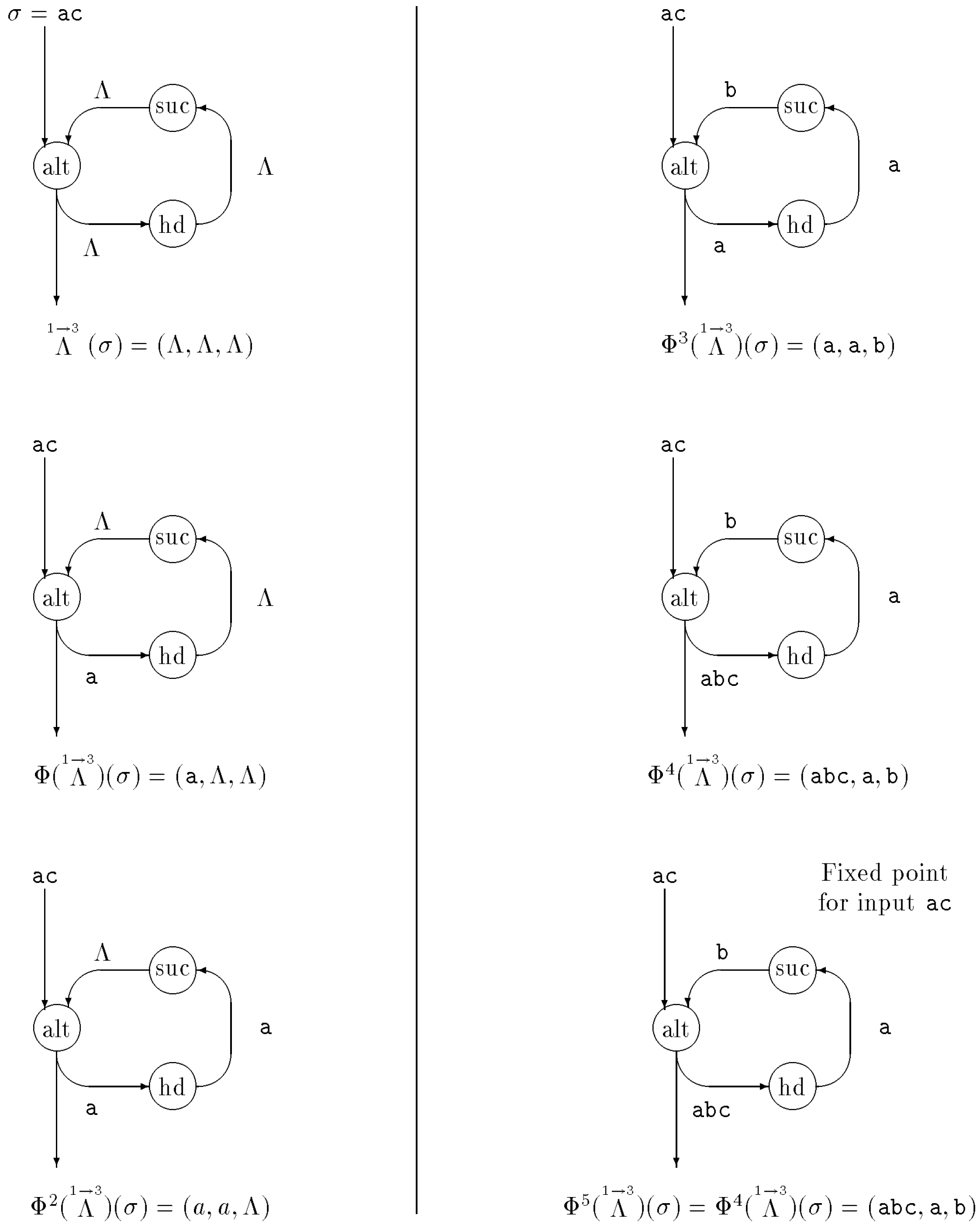


Figure 4: Finding the traditional fixed point $\mu\Phi(\sigma)$.

Tm transmits tokens in order of their time of arrival, *hd* takes the first token from its input, and suc converts the value of every token to its successor in the alphabetic order (wrapping around from `z` to `a`). All three nodes impart a delay of one time unit.

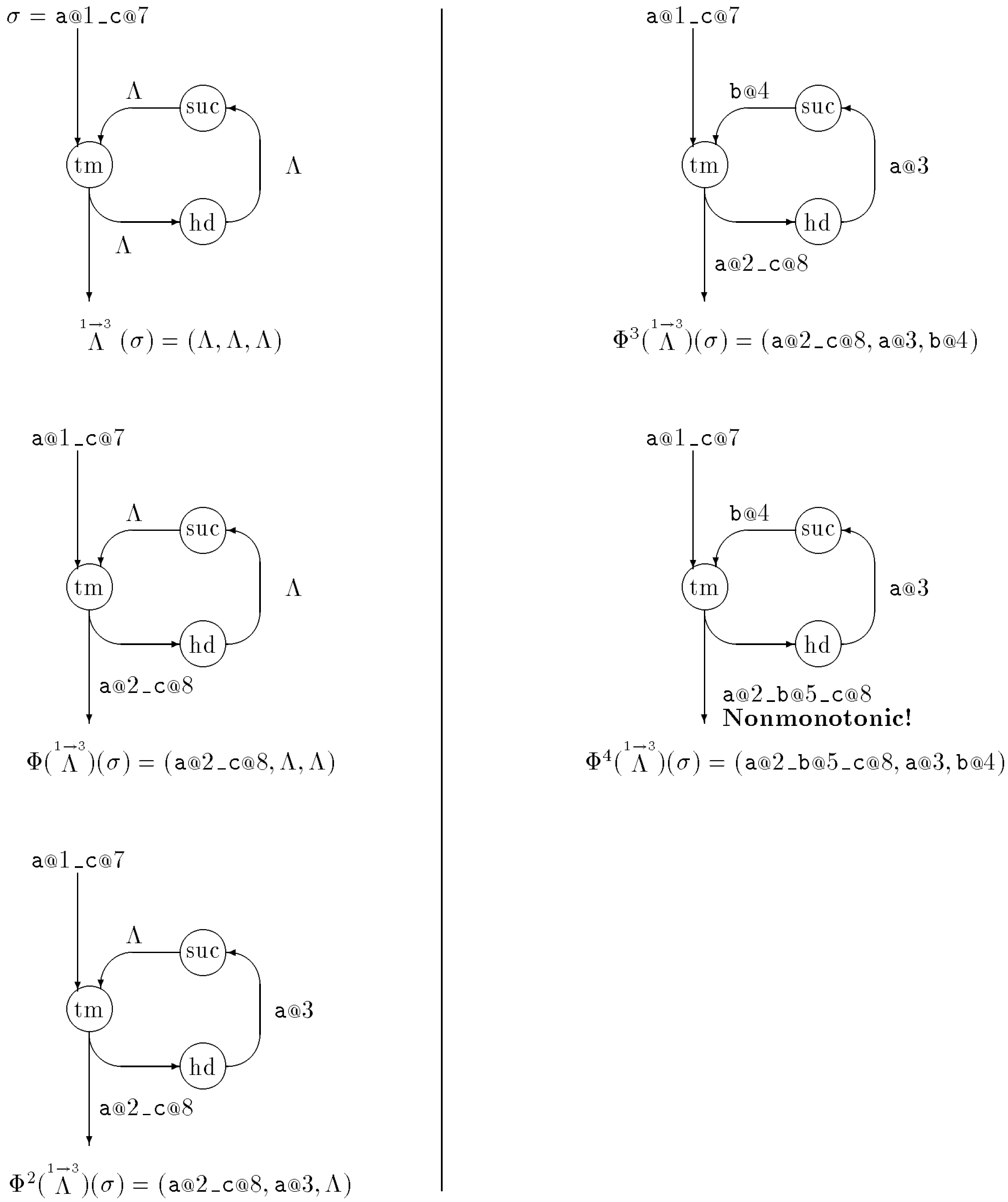


Figure 5: Using the traditional functional on a nonmonotonic network.

The network is the same as in Fig. 5. *Tm* transmits tokens in order of their time of arrival, *hd* takes the first token from its input, and suc converts the value of every token to its successor in the alphabetic order (wrapping around from `z` to `a`). All three nodes impart a delay of one time unit.

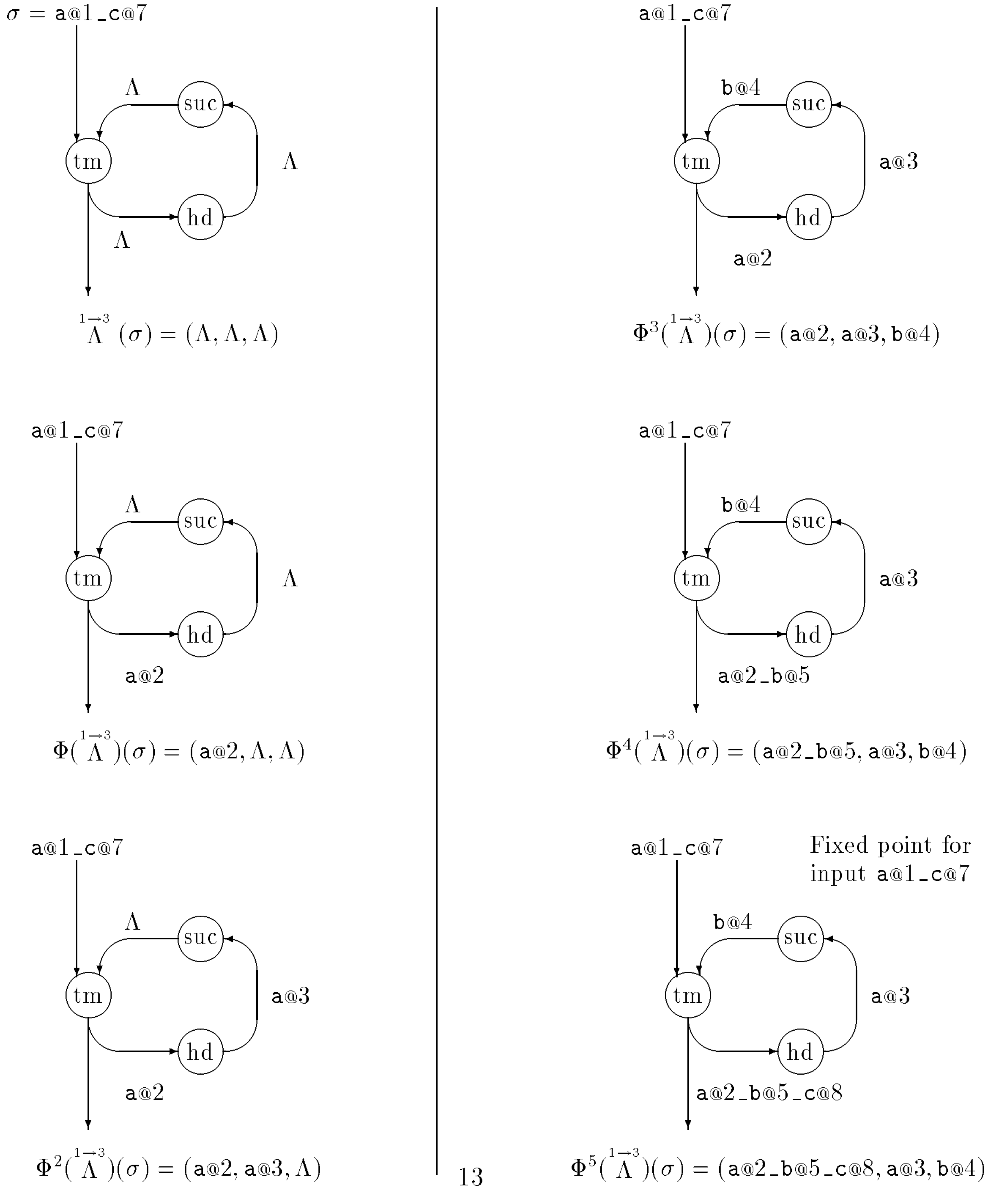


Figure 6: Using the new functional Φ on a nonmonotonic network.

change is the function space — since the networks we want to model do not (in general) compute prefix continuous functions, we cannot expect the new functional to operate on the domain $\left([\mathcal{S}^i \to \mathcal{S}^s], \overset{i\to s}{\preceq}\right)$ of prefix continuous functions. Instead, the new functional will operate on the domain $\left(\langle \mathcal{S}^i \overset{\Delta}{\to} \mathcal{S}^s\rangle, \overset{i\to s}{\preceq}\right)$ of Δ-causal functions, again ordered pointwise by prefix. A function $f : \mathcal{S}^i \to \mathcal{S}^s$ is called Δ-*causal* if there is always a delay of at least $\Delta > 0$ between input "cause" and output "effect," i.e., $f(\sigma)\overset{t}{|} = f(\sigma \overset{t-\Delta}{|})\overset{t}{|}$ for all inputs $\sigma \in \mathcal{S}^i$ and nonnegative times $t \in \mathbb{R}_0$. The notation "$\overset{t}{|}$" denotes the temporal "cutting" function which truncates a stream tuple at time t, i.e., all tokens with time tag greater than t are eliminated.

The principal novelty of the new functional Φ, defined below, is that it cuts off all outputs as soon as the earliest new output token appears. The new functional lets the network "run" at each step only until the earliest new output token appears. Figure 6 shows how the new functional calculates the value of $\mu\Phi\,(\mathtt{a}@1\,\_\,\mathtt{c}@7)$, given the same network as in Figure 5. Observe that the nonmonotonic behavior of Figure 5 is no longer found, due to the fact that the output token c@8 is prevented from appearing before the earlier token b@5. The new network functional Φ is in fact monotonic and continuous on the domain $(\langle \mathcal{S}^i \overset{\Delta}{\to} \mathcal{S}^s\rangle, \overset{i\to s}{\preceq})$ of Δ-causal functions ordered pointwise by prefix, even though the node functions are not prefix continuous.[10] Thus we again have $\overset{i\to s}{\Lambda} \overset{i\to s}{\preceq} \Phi(\overset{i\to s}{\Lambda}) \overset{i\to s}{\preceq} \Phi^2(\overset{i\to s}{\Lambda}) \overset{i\to s}{\preceq} \cdots$ and $\overset{i\to s}{\Lambda}(\sigma) \preceq^s \Phi(\overset{i\to s}{\Lambda})(\sigma) \preceq^s \Phi^2(\overset{i\to s}{\Lambda})(\sigma) \preceq^s \cdots$ just as for the traditional network functional on networks of prefix continuous processes. This can be verified in Figure 6 by noting that the streams on each wire grow prefixwise from step to step.

Definition 1 (The new network functional)
Given any real-time network N, let $\Phi : \langle \mathcal{S}^i \overset{\Delta}{\to} \mathcal{S}^s\rangle \to \langle \mathcal{S}^i \overset{\Delta}{\to} \mathcal{S}^s\rangle$, the network functional *of N, be defined for all $\sigma \in \mathcal{S}^i$ and $p = (a, \grave{p}) \in A^{out}$ by*

$$\Phi(f)(\sigma)[\grave{p}] \;=\; \vec{A}_a\big((\sigma \otimes f_{\mathrm{N}}(\sigma))\,\lfloor A_a^{in}\big)\overset{\natural(\sigma\otimes f_{\mathrm{N}}(\sigma))}{|}[\grave{p}].$$

[10] The proof that Φ is continuous on cpo $\left(\langle \mathcal{S}^i \overset{\Delta}{\to} \mathcal{S}^s\rangle, \overset{i\to s}{\preceq}\right)$ is long and tedious. An outline of the proof is given below; a full proof can be found in [36].

Given any function f from $\langle \mathcal{S}^i \stackrel{\Delta}{\rightarrow} \mathcal{S}^s \rangle$, the above definition says that to calculate the value of the new function $\Phi(f)$ at input σ one performs the following steps:

1. Convert f to f_{N} (this conversion is described in Def. 2).
2. Apply f_{N} to σ and weld input σ to the resulting s-tuple to get an $i+s$-tuple. This forms a new network snapshot $\sigma \otimes f_{\mathrm{N}}(\sigma)$.
3. Project the new snapshot onto the set of ports A_a^{in}, thus picking out the inputs to node A_a. Do this for all the nodes $a \in 1..k$ in the network independently.
4. Apply the node function $\vec{A}_a$ to its inputs.
   Do this for all the nodes $a \in 1..k$ in the network independently.
5. Cut the results at time $\natural(\sigma \otimes f_{\mathrm{N}}(\sigma))$.
   This allows no more than one new token to appear in any output stream. Function $\natural : \mathcal{S}_{\mathrm{N}}^{i+s} \rightarrow \mathbb{R}_0^{\infty}$ is defined by $\natural(H) = \bigsqcup_{\leq} \{\, t \in \mathbb{R}_0 \mid \mathrm{Sat}_{\mathrm{N}}(H,t) \,\}$.
   (See Def. 2 below for the definition of $\mathcal{S}_{\mathrm{N}}^{i+s}$ and $\mathrm{Sat}_{\mathrm{N}}$.)
6. Put each new output stream in its proper place in the new output s-tuple, i.e., the $\grave{p}^{\text{th}}$ output stream of node A_a belongs in position $\dot{p}$, where $p = (a, \grave{p})$. For any node output port $p \in A^{out}$, integer $\dot{p}$ denotes the index of port p within a canonical total ordering of A^{out}.

All the components of Φ have already been defined, except for the *trimming functional* $\cdot_{\mathrm{N}}$, which we now introduce.

Definition 2 *Given a TRTM network* $N = (A_1, \ldots, A_k, i, o, s, \mathrm{src})$, *let* $\cdot_{\mathrm{N}} : \langle \mathcal{S}^i \stackrel{\Delta}{\rightarrow} \mathcal{S}^s \rangle \rightarrow \langle \mathcal{S}^i \stackrel{\Delta}{\rightarrow} \mathcal{S}^s \rangle_{\mathrm{N}}$ *be the function defined by* $f_{\mathrm{N}}(\sigma) = \bigsqcup_{\preceq^s} {}_fY_\sigma$ *for all* $f \in \langle \mathcal{S}^i \stackrel{\Delta}{\rightarrow} \mathcal{S}^s \rangle$ *and* $\sigma \in \mathcal{S}^i$, *where*

$$
\begin{aligned}
{}_fY_\sigma &\stackrel{\text{def}}{=} \left\{\, y \mid y \preceq^s f(\sigma) \ \wedge\ \mathrm{Sat}_{\mathrm{N}}\big(\rho \otimes y,\, l\big)\ (\forall \rho \ s.t.\ \rho \stackrel{l-\Delta}{|} = \sigma \stackrel{l-\Delta}{|})\ (\textit{where } l = \mathrm{tlen}(y)) \,\right\} \\
\langle \mathcal{S}^i \stackrel{\Delta}{\rightarrow} \mathcal{S}^s \rangle_{\mathrm{N}} &\stackrel{\text{def}}{=} \left\{\, f \in \langle \mathcal{S}^i \stackrel{\Delta}{\rightarrow} \mathcal{S}^s \rangle \mid \mathrm{Sat}_{\mathrm{N}}\big(\sigma \otimes f(\sigma),\, \mathrm{tlen}(f(\sigma))\big) \textit{ for all } \sigma \in \mathcal{S}^i \,\right\} \\
\mathrm{Sat}_{\mathrm{N}}(H,t) &\stackrel{\text{def}}{\Leftrightarrow} H \stackrel{t}{\downharpoonright} A_a^{out} = \vec{A}_a(H \downharpoonright A_a^{in}) \stackrel{t}{|}\ (\forall a \in 1..k).
\end{aligned}
$$

Codomain $\langle \mathcal{S}^i \overset{\Delta}{\to} \mathcal{S}^s\rangle_{\mathrm{N}}$ of functional $\cdot_{\mathrm{N}}$ is a subspace of domain $\langle \mathcal{S}^i \overset{\Delta}{\to} \mathcal{S}^s\rangle$ wherein $f(\sigma)$ is a temporal prefix of the network solution for input σ, given $f \in \langle \mathcal{S}^i \overset{\Delta}{\to} \mathcal{S}^s\rangle_{\mathrm{N}}$ and $\sigma \in \mathcal{S}^i$. Function $\mathrm{Sat}_{\mathrm{N}} : (\mathcal{S}^{i+s} \times \mathbb{R}^\infty) \to \mathbb{B}$ is a predicate, where $\mathrm{Sat}_{\mathrm{N}}(H, t)$ means that network snapshot H satisfies the network equations up through time t.

The purpose of the trimming functional $\cdot_{\mathrm{N}}$ is to transform an arbitrary Δ-causal function $f \in \langle \mathcal{S}^i \overset{\Delta}{\to} \mathcal{S}^s\rangle$ into a function $f_{\mathrm{N}} \in \langle \mathcal{S}^i \overset{\Delta}{\to} \mathcal{S}^s\rangle_{\mathrm{N}}$ which is prefix-consistent with the network equations and is still Δ-causal. This is done because in taking an arbitrary function f from $\langle \mathcal{S}^i \overset{\Delta}{\to} \mathcal{S}^s\rangle$ there is no guarantee that f will have anything in common with the given network N other than being a Δ-causal function of i inputs and s outputs. The new function f_{N} is guaranteed to satisfy the network equations through its output length at every input point, i.e., $\mathrm{Sat}_{\mathrm{N}}\big(\sigma \otimes f_{\mathrm{N}}(\sigma), \mathrm{tlen}(f_{\mathrm{N}}(\sigma))\big)$ for all $\sigma \in \mathcal{S}^i$. This is accomplished by taking the longest temporal cut of each output such that the desired result is achieved.

The trimming functional is a necessary "preprocessing" step of the network functional because without it Φ may be nonmonotonic when applied to "bad" functions, i.e., functions that are in $\langle \mathcal{S}^i \overset{\Delta}{\to} \mathcal{S}^s\rangle$ but not in $\langle \mathcal{S}^i \overset{\Delta}{\to} \mathcal{S}^s\rangle_{\mathrm{N}}$. As shown in Figure 7, the problem arises because of the presence of nonmonotonic components in real-time networks. The traditional functional for the networks of blocking-read processes treated in Kahn's theory does not need such a trimming step because all the network components compute Scott continuous functions, and so are monotonic.

4 The real-time Kahn principle

We shall show for any real-time network N that the functional Φ is Scott continuous, and that the least fixed point of Φ exactly describes the behavior of N.

The network functional Φ is continuous if and only if $\Phi(\bigsqcup F) = \bigsqcup F'$, where $F \subseteq \langle \mathcal{S}^i \overset{\Delta}{\to} \mathcal{S}^s\rangle$ is an arbitrary $\overset{i\to s}{\preceq}$-chain and $F' \overset{\mathrm{def}}{=} \{\, \Phi(f) \mid f \in F \,\}$. Since $\Phi(\bigsqcup F)$ and $\bigsqcup F'$ are functions in $\langle \mathcal{S}^i \overset{\Delta}{\to} \mathcal{S}^s\rangle$, to prove their equality we need to show

$$\Phi(\bigsqcup F)(\sigma)[\dot{p}] \quad = \quad (\bigsqcup F')(\sigma)[\dot{p}],$$

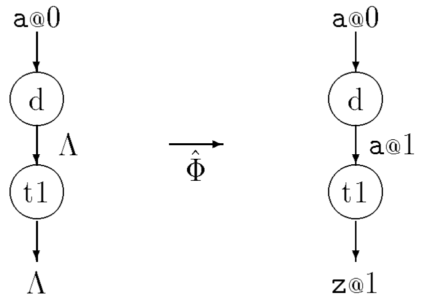
Let $\hat{\Phi}$ be like Φ but without the trimming functional, i.e., $\Phi = \hat{\Phi} \circ \cdot_{\mathrm{N}}$, where

$$\hat{\Phi}(f)(\sigma)[\dot{p}] \quad \stackrel{\text{def}}{=} \quad \vec{A}_a\big((\sigma \otimes f(\sigma)) \lfloor A_a^{in}\big)^{\natural(\sigma \otimes f(\sigma))}\Big| \; [\dot{p}] \qquad \forall\, \sigma \in \mathcal{S}^i,\ p = (a, \dot{p}) \in A^{out}.$$

Let N be the network composed of node d and node *t1* in sequence, where d delays every token by one time unit and *t1* emits a z every cycle until it receives the first input token.

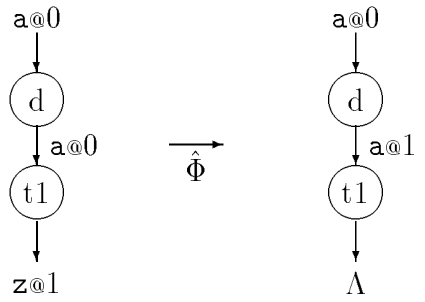
To demonstrate that $\hat{\Phi}$ is nonmonotonic it suffices to find two functions $f, g \in \langle \mathcal{S}^1 \stackrel{\Delta}{\to} \mathcal{S}^2 \rangle$ such that $f \stackrel{1\to 2}{\preceq} g$ but *not* $\hat{\Phi}(f) \stackrel{1\to 2}{\preceq} \hat{\Phi}(g)$. Let $f(x) \stackrel{\text{def}}{=} (\Lambda, \Lambda)$ and $g(x) \stackrel{\text{def}}{=} (\mathtt{z@}1, \mathtt{a@}0)$ for all $x \in \mathcal{S}$. Clearly f and g are both Δ-causal because they are constant, and $f \stackrel{1\to 2}{\preceq} g$ because $f(x) \preceq^2 g(x)$ for all $x \in \mathcal{S}$. Let us see what happens on input $\mathtt{a@}0$.

$f(\mathtt{a@}0) = (\Lambda, \Lambda)$



$\hat{\Phi}(f)(\mathtt{a@}0) = (\mathtt{z@}1, \mathtt{a@}1)$

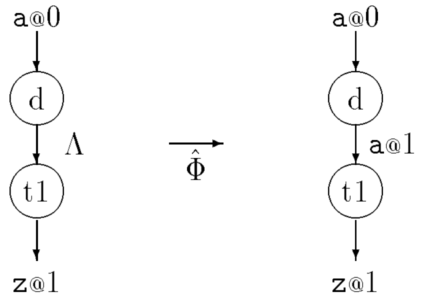
$g(\mathtt{a@}0) = (\mathtt{z@}1, \mathtt{a@}0)$



$\hat{\Phi}(g)(\mathtt{a@}0) = (\Lambda, \mathtt{a@}1)$

Since $\hat{\Phi}(f)(\mathtt{a@}0) = (\mathtt{z@}1, \mathtt{a@}1) \not\preceq^2 (\Lambda, \mathtt{a@}1) = \hat{\Phi}(g)(\mathtt{a@}0)$, $\hat{\Phi}$ is not monotonic. Note that the token $\mathtt{a@}0$ on the output arc of d is inconsistent with d's input $\mathtt{a@}0$. The trimming functional $\cdot_{\mathrm{N}}$ solves the problem by cutting g back to g_{N}, where $g_{\mathrm{N}}(x) = (\mathtt{z@}1, \Lambda)$ for all $x \in \mathcal{S}$.

$g_{\mathrm{N}}(\mathtt{a@}0) = (\mathtt{z@}1, \Lambda)$



$\hat{\Phi}(g_{\mathrm{N}})(\mathtt{a@}0) = (\mathtt{z@}1, \mathtt{a@}1)$

Figure 7: Without $\cdot_N$ the network functional is not monotonic.

choosing arbitrary $\sigma \in \mathcal{S}^i$ and $p = (a, \grave{p}) \in A^{out}$.[11] By the definition of Φ we have

$$\Phi(\bigsqcup F)(\sigma)[\dot{p}] \;=\; \vec{A}_a\Big(\big(\sigma \otimes (\bigsqcup F)_{\mathrm{N}}(\sigma)\big) \lfloor A_a^{in}\Big)^{\overset{\natural(\sigma \otimes (\sqcup F)_{\mathrm{N}}(\sigma))}{|}}[\grave{p}].$$

Therefore, to prove Φ continuous we must show

$$\vec{A}_a\Big(\big(\sigma \otimes (\bigsqcup F)_{\mathrm{N}}(\sigma)\big) \lfloor A_a^{in}\Big)^{\overset{\natural(\sigma \otimes (\sqcup F)_{\mathrm{N}}(\sigma))}{|}}[\grave{p}] \;=\; (\bigsqcup F')(\sigma)[\dot{p}]. \tag{1}$$

The definitions of F' and Φ and the continuity of application[12] and indexing give

$$(\bigsqcup F')(\sigma)[\dot{p}] \;=\; \bigsqcup_{\preceq} \Big\{\, \vec{A}_a\big((\sigma \otimes f_{\mathrm{N}}(\sigma)) \lfloor A_a^{in}\big)^{\overset{\natural(\sigma \otimes f_{\mathrm{N}}(\sigma))}{|}}[\grave{p}] \;\Big|\; f \in F \,\Big\}.$$

Thus (1) is equivalent to the continuity of function ${}_{\sigma}\Phi_p : \langle \mathcal{S}^i \overset{\triangle}{\rightarrow} \mathcal{S}^s \rangle \rightarrow \mathcal{S}$ given by

$${}_{\sigma}\Phi_p(f) \;\overset{\text{def}}{=}\; \vec{A}_a\big((\sigma \otimes f_{\mathrm{N}}(\sigma)) \lfloor A_a^{in}\big)^{\overset{\natural(\sigma \otimes f_{\mathrm{N}}(\sigma))}{|}}[\grave{p}],$$

since if ${}_{\sigma}\Phi_p$ is continuous we have ${}_{\sigma}\Phi_p(\bigsqcup F) = \bigsqcup_{\preceq} \big\{\, {}_{\sigma}\Phi_p(f) \;\big|\; f \in F \,\big\}$.

The proof is too long to include here, but in [36] the new network functional Φ is proven

[11] That is, choose an arbitrary node output port p from A^{out}; call the first component a (the node number) and the second component $\grave{p}$ (the local output port number).

[12] In [36] it is proved that the application operator $\cdot(\sigma)$ that maps functions in cpo $\big(\langle \mathcal{S}^i \overset{\triangle}{\rightarrow} \mathcal{S}^s \rangle, \overset{i \rightarrow s}{\preceq}\big)$ to output tuples in cpo $(\mathcal{S}^s, \preceq^s)$ is continuous.

continuous by showing that ${}_\sigma\Phi_p$ is composed of continuous functions.

$$
\begin{array}{lcll}
{}_\sigma\Phi_p & = & \cdot[\grave{p}] \;\circ\; F_4 \;\circ\; F_3 \;\circ\; F_2 \;\circ\; \cdot_{\mathrm{N}} & \\[2em]
{}_\sigma\Phi_p(f) & = & \vec{A}_a\big((\sigma\otimes f_{\mathrm{N}}(\sigma))\lfloor A_a^{in}\big)\Big|^{\natural(\sigma\otimes f_{\mathrm{N}}(\sigma))}[\grave{p}] & \text{the definition of } {}_\sigma\Phi_p \\
{}_\sigma\Phi_p & : & \langle \mathcal{S}^i \stackrel{\triangle}{\to} \mathcal{S}^s\rangle \to \mathcal{S} & \\
 & & \text{ordered by } \stackrel{i\to s}{\preceq} \text{ and } \preceq & \\[2em]
\cdot_{\mathrm{N}} & : & \langle \mathcal{S}^i \stackrel{\triangle}{\to} \mathcal{S}^s\rangle \to \langle \mathcal{S}^i \stackrel{\triangle}{\to} \mathcal{S}^s\rangle_{\mathrm{N}} & \textbf{Step 1: } f_{\mathrm{N}} \text{ is } f \text{ stripped of all} \\
 & & \text{ordered by } \stackrel{i\to s}{\preceq} \text{ and } \stackrel{i\to s}{\preceq} & \text{tokens inconsistent with } N \\[2em]
F_2(g) & = & \sigma\otimes g(\sigma) & \textbf{Step 2: } \sigma\otimes f_{\mathrm{N}}(\sigma) \text{ is the net trace} \\
F_2 & : & \langle \mathcal{S}^i \stackrel{\triangle}{\to} \mathcal{S}^s\rangle_{\mathrm{N}} \to \mathcal{S}_{\mathrm{N}}^{i+s} & \text{produced by } f_{\mathrm{N}} \text{ at } \sigma \\
 & & \text{ordered by } \stackrel{i\to s}{\preceq} \text{ and } \sqsubseteq_{\mathrm{N}}^{i+s} & \\[2em]
F_3(H) & = & \big(H,\, \natural(H)\big) & \textbf{Step 3: } \natural(\sigma\otimes f_{\mathrm{N}}(\sigma)) \text{ is the time} \\
F_3 & : & \mathcal{S}_{\mathrm{N}}^{i+s} \to \mathcal{S}_{\tau}^{i+s} & \text{of the next new token} \\
 & & \text{ordered by } \sqsubseteq_{\mathrm{N}}^{i+s} \text{ and } \sqsubseteq_{\tau}^{i+s} & \\[2em]
F_4(H,t) & = & \vec{A}_a\big(H\lfloor A_a^{in}\big)\Big|^{t} & \textbf{Step 4: } \vec{A}_a(\sigma\otimes f_{\mathrm{N}}(\sigma)\lfloor A_a^{in})\Big|^{\natural(\sigma\otimes f_{\mathrm{N}}(\sigma))} \\
F_4(\perp_\tau) & = & \Lambda^c & \text{run all nodes } A_a \text{ until one} \\
F_4 & : & \mathcal{S}_{\tau}^{i+s} \to \mathcal{S}^c & \text{of them emits a new token} \\
 & & \text{ordered by } \sqsubseteq_{\tau}^{i+s} \text{ and } \sqsubseteq^c & \\
 & & \text{where } c = \#(A_a^{out}) & \\[2em]
\cdot[\grave{p}] & : & \mathcal{S}^c \to \mathcal{S} & \textbf{Step 5: } \vec{A}_a(\sigma\otimes f_{\mathrm{N}}(\sigma)\lfloor A_a^{in})\Big|^{\natural(\sigma\otimes f_{\mathrm{N}}(\sigma))}[\grave{p}] \\
 & & \text{ordered by } \sqsubseteq^c \text{ and } \preceq & \text{is the new stream at node } A_a\text{'s} \\
 & & & p^{\text{th}} \text{ output port.}
\end{array}
$$

Proposition 3 *The functional Φ associated with real-time network N is continuous on cpo* $(\langle \mathcal{S}^i \xrightarrow{\Delta} \mathcal{S}^s \rangle, \overset{i \to s}{\preceq})$.

The next proposition states that Φ transforms every Δ-causal function into one that is guaranteed to be consistent with network N at every input, up through the last output token (but not necessarily beyond). In other words, the result of applying Φ is always a (pointwise) temporal prefix of the full solution of the network equations.

Proposition 4 *Let N be a real-time network, with network functional Φ. Then $\Phi(f) \in \langle \mathcal{S}^i \xrightarrow{\Delta} \mathcal{S}^s \rangle_{\mathrm{N}}$ for all $f \in \langle \mathcal{S}^i \xrightarrow{\Delta} \mathcal{S}^s \rangle$.*

The Kahn principle will follow almost directly from the next proposition, which states that if applying Φ causes no change at a particular snapshot then we have reached the complete solution to the network equations for the given input.

Proposition 5 *Given a real-time network N, let f be any function in $\langle \mathcal{S}^i \xrightarrow{\Delta} \mathcal{S}^s \rangle_{\mathrm{N}}$ and σ be any tuple in $\mathcal{S}^i$. If $f(\sigma) = \Phi(f)(\sigma)$ then* $\mathrm{Sat}_{\mathrm{N}}\big(\sigma \otimes f(\sigma), \infty\big)$.

A real network run certainly satisfies the network equations.

Proposition 6 *For any real-time network N and input $\sigma \in \mathcal{S}^i$, the history $\sigma \otimes \mathrm{NetFun}_N(\sigma)$ obtained by running the network on input σ satisfies the network equations of N; that is,* $\mathrm{Sat}_{\mathrm{N}}\big(\sigma \otimes \mathrm{NetFun}_N(\sigma),\, \infty\big)$.

The delays imparted by causal functions ensure that any solution to the network equations is unique.

Proposition 7 *Let $N = (A_1, \ldots, A_k, i, o, s, \mathrm{src})$ be a network. For any $\sigma \in \mathcal{S}^i$, $\theta \in \mathcal{S}^s$ and $t \in \mathbb{R}^\infty$ such that* $\mathrm{Sat}_{\mathrm{N}}(\sigma \otimes \theta, t)$, $\mathrm{Sat}_{\mathrm{N}}(\sigma \otimes \eta, t) \Leftrightarrow \theta\overset{t}{|} = \eta\overset{t}{|}$.

At last everything needed to prove the extended Kahn principle is assembled. We have tried to make it clear in the earlier examples and discussion just how repeated application of the network functional actually traces out, for any given network input, the same sequence of snapshots that would be produced by actually running the network. If the explanations have fulfilled their purpose, the validity of the Kahn principle for real-time networks should come as no surprise.

Theorem 8 (Kahn principle for real-time networks)
Let N be a real-time network, with network functional Φ.
Then $\mathrm{NetFun}_N = \mu\Phi$*, the least fixed point of* Φ*.*

Proof. By Prop. 3, functional Φ is Scott continuous on cpo $\big(\langle \mathcal{S}^i \xrightarrow{\triangle} \mathcal{S}^s\rangle, \overset{i\to s}{\preceq}\big)$, so the least fixed point $\mu\Phi = \bigsqcup_{\overset{i\to s}{\preceq}} \{\, \Phi^n(\overset{i\to s}{\Lambda}) \mid n \in \mathbb{N} \,\}$ is well defined (where $\overset{i\to s}{\Lambda}$ is the "everywhere empty" function, the bottom element of cpo $(\langle \mathcal{S}^i \xrightarrow{\triangle} \mathcal{S}^s\rangle, \overset{i\to s}{\preceq})$).

We must show that $\mathrm{NetFun}_N(\sigma) = \mu\Phi(\sigma)$ for all $\sigma \in \mathcal{S}^i$. Proposition 6 warrants that $\sigma \otimes \mathrm{NetFun}_N(\sigma)$ satisfies the network equations, i.e., $\mathrm{Sat}_{\mathrm{N}}\big(\sigma \otimes \mathrm{NetFun}_N(\sigma),\, \infty\big)$. We know by Proposition 7 that any two solutions to the network equations sharing the same network input are identical. Therefore to prove $\mathrm{NetFun}_N(\sigma) = \mu\Phi(\sigma)$ it suffices to show that $\sigma \otimes \mu\Phi(\sigma)$ satisfies the network equations, i.e., $\mathrm{Sat}_{\mathrm{N}}\big(\sigma \otimes \mu\Phi(\sigma),\, \infty\big)$. In fact, we shall prove that $\sigma \otimes f(\sigma)$ satisfies the network equations given *any* fixed point $f = \Phi(f)$.

Let $f = \Phi(f)$ be an arbitrary fixed point of network functional Φ and σ be an arbitrary tuple from $\mathcal{S}^i$. Then we have

$$
\begin{array}{rcll}
\Phi(f) & \in & \langle \mathcal{S}^i \xrightarrow{\triangle} \mathcal{S}^s\rangle_{\mathrm{N}} & \text{by Prop 4} \\
f & \in & \langle \mathcal{S}^i \xrightarrow{\triangle} \mathcal{S}^s\rangle_{\mathrm{N}} & \text{since } f = \Phi(f) \\
f(\sigma) & = & \Phi(f)(\sigma) & \text{since } f = \Phi(f) \\
\multicolumn{3}{r}{\mathrm{Sat}_{\mathrm{N}}\big(\sigma \otimes f(\sigma),\, \infty\big)} & \text{by Prop 5} \\
\multicolumn{3}{r}{\mathrm{Sat}_{\mathrm{N}}\big(\sigma \otimes \mathrm{NetFun}_N(\sigma),\, \infty\big)} & \text{by Prop 6} \\
\mathrm{NetFun}_N(\sigma)\overset{\infty}{|} & = & f(\sigma)\overset{\infty}{|} & \text{by Prop 7} \\
\mathrm{NetFun}_N(\sigma) & = & f(\sigma) & \text{since } x = x\overset{\infty}{|} \text{ for any } x \in \mathcal{S}^n.
\end{array}
$$

Thus $\mathrm{NetFun}_N = f$ for any fixed point $f = \Phi(f)$. In particular, $\mathrm{NetFun}_N = \mu\Phi$.
$\Box$ (End of proof of Theorem 8.)

In other words, the function computed by any real-time network is identical to the least fixed point $\mu\Phi$ of network functional Φ. In fact, $\mu\Phi$ is the *unique* fixed point of Φ.

The equivalence between denotation and operation means that the denotational semantics is fully abstract.[13] However, the identity $\mathrm{NetFun}_N = \mu\Phi$ does not capture the kind of equivalence we really are after, since the output of both NetFun_N and $\mu\Phi$ includes the contents of all the internal arcs of network N. As mentioned in the discussion of NetFun_N, the internal arcs of the network are treated as outputs (following the approach taken by Lynch and Stark in [19]) because this simplifies the proof of the Kahn principle. However, we would in fact prefer two networks to be equivalent (both operationally and denotationally) if given the same input they produce the same streams on their true output ports, regardless of the number or contents of any internal arcs.[14] For instance, a network composed of two nodes in sequence each of which delays its input tokens by t ought to be equivalent to a network with a single node which delays its inputs by $2t$. We would like the internal arcs to be hidden. Let $\mathrm{Hide} : (\mathcal{S}^i \to \mathcal{S}^s) \to (\mathcal{S}^i \to \mathcal{S}^o)$ be the functional defined by

$$\mathrm{Hide}(f)(\sigma) \quad = \quad \big(f(\sigma)\big) \lfloor N^{out}$$

for all $\sigma \in \mathcal{S}^i$. That is, Hide takes a function $f : \mathcal{S}^i \to \mathcal{S}^s$ and produces a function $\mathrm{Hide}(f) : \mathcal{S}^i \to \mathcal{S}^o$ by discarding, for all inputs $\sigma \in \mathcal{S}^i$, all the streams of $f(\sigma)$ that correspond to internal arcs of the network N. By taking the denotation of a network to be $\mathrm{Hide}(\mu\Phi)$ we get a fully abstract semantics in which the internal arcs are unobservable, as desired.

[13] A semantics is *fully abstract* if any two program phrases behave identically whenever their denotations are identical. In the case of real-time networks, our semantics is fully abstract because ($\mathrm{NetFun}_{N_1} = \mathrm{NetFun}_{N_2}$) $\Leftrightarrow$ ($\mu\Phi_{N_1} = \mu\Phi_{N_2}$), where Φ_{N_1} is the functional associated with network N_1, and Φ_{N_2} is the functional associated with network N_2.

[14] By a *true output* we mean any arc that connects the output of a node to the environment, and by an *internal* arc we mean any arc that connects two nodes.

5 Comparison to previous work

The literature on the semantics dataflow networks with timed merge is nonexistent. Work in the theory of timed systems has centered on temporal logic ([11, 18] and others) and algebraic-operational models like CSP ([32] and others) and CCS ([7, 21, 37] and others), with some work being done in automata theory ([2, 12, 20] and others). By contrast, the literature on the semantics of networks with indeterminate operators would easily fill a large book ([1, 3, 4, 5, 6, 14, 16, 15, 17, 24, 29, 26, 27, 28, 25, 30, 31, 34, 35] and many more). We comment only on those works most closely related to timed merge.

Hiaton theories [4, 14, 25, 29] make fair merge monotonic by inserting special tokens into streams to represent the absence of data. Hiaton theories are related to the semantics presented here in so far as hiatons are sometimes construed to signify the passage of an unspecified length of time.

Boussinot [4] was the first to employ hiatons in the semantics of networks with non-monotonic merge operators. Hiatons, used together with oracles, do make merge continuous and solve the causality anomalies of Keller [15] and Brock and Ackerman [5]. However, Boussinot's semantics lacks full abstractness.[15] For example, the "identity" function id_τ on hiatonned streams must prepend a hiaton to the head of its output, with the effect that $\mathrm{id}_\tau(x) = \tau x \neq \tau\tau x = \mathrm{id}_\tau(\mathrm{id}_\tau(x))$. Thus the denotational semantics distinguishes between a network composed of two id_τ nodes in sequence and the network composed of a single id_τ node, whereas operationally the networks are indistinguishable.

Park [29] simplified and improved upon Boussinot's theory. He also presents a lucid discussion of the problematic nature of fair merge and shows why hiatons (or something like them) are needed if fair merge is to be continuous. Like [4], the result is a fixed point semantics lacking full abstractness, and for the same reasons.

[15]To say that a semantics is *fully abstract* means that two program phrases behave identically (are interchangeable in any context) if and only if their denotations are identical. Full abstraction is considered a desirable property, since it means that the semantics has just the right information to be compositional, and no more.

In an unusual use of abstract interpretation justified by category theory, Panangaden [25] injects hiatons into streams to derive the meaning of a network, and then filters them out. In this theory hiatons are generated only when a poll operation detects that no input data is present; this is a marked advantage over the work of Boussinot and Park. The semantics described in [25] might be fully abstract, but this is still an open question. The appeal to abstract interpretation and category theory does not have the same intuitive appeal as order-theoretic continuity, though this may be due to its unfamiliarity.

Kearney and Staples [14] make some technical improvements on Park's work by relaxing some of the rules on hiaton generation and by giving a compositional treatment of oracle inputs. However, the result is still not fully abstract.

Park says of hiatons that "[t]his device is displeasing, since it does not seem sufficiently 'abstract' " [29]. We would argue that hiatons are either too abstract or not abstract enough. If time delays are unobservable in the operational semantics then hiatons are not abstract enough, as seen by the lack of full abstraction in theories using them. On the other other hand, if time delays *are* to be observable in the operational model then hiatons are too abstract to carry much useful information. Hiatons could be *made* useful if they were to represent something concrete, like a failed poll as in [25] or a specific time delay. Hiatons are an abstraction of delays. Why not just put real delays into the semantics?

6 Conclusions

We have set forth a denotational semantics of real-time dataflow networks. To that end we described timed streams, their orderings and some operations upon them. A new network functional was defined and shown to be Scott continuous, even when the network components are nonmonotonic. Subsequently it was shown that the least fixed point of the network functional is identical to the function computed by the network. The least fixed point of the functional is in fact the unique fixed point.

The method is successful because iterating the network functional starting from the everywhere-empty function traces out, input point-wise, the same sequence of snapshots that an observer would see traced out by a real network execution. The conclusion to draw is that it really doesn't matter whether the network components are monotonic or not. Since the function space is ordered pointwise, the continuity of the network functional is determined by what it does to chains of functions, not by what the component functions do to chains of input tuples.

One possible extension to this work would be to develop a theory that associates tokens with time intervals, or some other kind of timing constraint, rather than single instants of time. An interval would mean that the event represented by the token must occur at some undetermined instant within the given interval. Such a theory might be of more practical benefit than a totally untimed theory, which at any rate would be just a special case of the interval theory.

The ultimate test of the work presented here is whether it contributes to the understanding of parallel systems. There is much room for improvement in the design, construction and validation of parallel and real-time programs. The most successful methods for reasoning about concurrency have been the operational-algebraic methods typified by CSP and CCS. How might the ideas explored here be used to develop new or modify existing techniques for reasoning about timed and untimed parallel programs?

References

[1] S. Abramsky. A generalized Kahn principle for abstract asynchronous networks. In M. Main, A. Melton, M. Mislove, and D. Schmidt, editors, *Proc. of the Fifth Workshop on Mathematical Foundations of Programming Semantics*. Springer-Verlag, LNCS-442, 1990.

[2] R. Alur and D. Dill. The theory of timed automata. In J.W. deBakker, C. Huizing, W.P deRoever, and G. Rozenberg, editors, *Real-Time: Theory in Practice*, pages 45–73. Springer-Verlag, LNCS 600, 1991. Proceedings of the REX Workshop.

[3] D.B. Benson, P. Panangaden, and J.R. Russell. Defining fair merge as a colimit: Towards a fixed-point theory for indeterminate dataflow. In M. Z. Kwiatkowska, M. W. Shields, and R. M. Thomas, editors, *Semantics of Concurrency, Proceedings of the International BCS-FACS Workshop*. Springer-Verlag, 1990.

[4] F. Boussinot. Proposition de sémantique dénotationelle pour des réseaux de processus avec operateur de mélange équitable. *Theoretical Computer Science*, 18:173–206, 1982.

[5] J. D. Brock and W. B. Ackerman. Scenarios: A model of non-determinate computation. In J. Diaz and I. Ramos, editors, *International Colloquium on Formalization of Programming Concepts*, pages 252–259. Springer-Verlag, LNCS-107, 1981.

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

[7] L. Chen. An interleaving model for real-time systems. In A. Nerode and M. Taitslin, editors, *Logical Foundations of Computer Science: Proc. of the Second Int. Symp., Tver, Russia*, pages 81–92. Springer-Verlag, LNCS 620, 1992.

[8] J. W. deBakker, J. A. Bergstra, J. W. Klop, and J.-J. Ch. Meyer. Linear time and branching time semantics for recursion with merge. *Theoretical Computer Science*, 34:134–156, 1984.

[9] J. B. Dennis. A language design for structured concurrency. In *Design and Implementation of Programming Languages*, pages 231–243. Springer-Verlag, LNCS-54, 1976.

[10] Carl A. Gunter. *Semantics of Programming Languages: Structures and Techniques.* MIT Press, 1992.

[11] E. Harel, O. Lichtenstein, and A. Pnueli. Explicit clock temporal logic. In *Proc. of the Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 402–413, 1990.

[12] T.A. Henzinger, Z. Manna, and A. Pnueli. Timed transition systems. In J.W. deBakker, C. Huizing, W.P deRoever, and G. Rozenberg, editors, *Real-Time: Theory in Practice*, pages 226–251. Springer-Verlag, LNCS 600, 1991. Proceedings of the REX Workshop.

[13] G. Kahn. The semantics of a simple language for parallel processing. In *Information Processing 74*, pages 471–475, 1974.

[14] P. Kearney and J. Staples. An extensional fixed-point semantics for nondeterministic data flow. *Theoretical Computer Science*, 91:129–179, 1991.

[15] R. M. Keller. Denotational models for parallel programs with indeterminate operators. In E. J. Neuhold, editor, *Formal Descriptions of Programming Concepts*, pages 337–365, Amsterdam, 1978. North-Holland.

[16] R. M. Keller and P. Panangaden. Semantics of networks containing indeterminate operators. In *Proc. of the 1984 CMU Seminar on Concurrency*, pages 479–496. Springer-Verlag, LNCS 197, 1985.

[17] Joost N. Kok. Denotational semantics of nets with nondeterminism. In B. Robinet and R. Wilhelm, editors, *European Symposium on Programming*, pages 237–249. Springer-Verlag, LNCS-213, 1986.

[18] R. Koymans. Specifying real-time properties with metric temporal logic. *Real Time Systems*, 2(4):255–291, 1990.

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

[20] M. Merritt, F. Modugno, and M.R. Tuttle. Time-constrained automata. In J.C.M. Baeten and J.F. Groote, editors, *Proceedings of Concur '91*, pages 408–423. Springer-Verlag, LNCS 527, 1991.

[21] F. Moller and C. Tofts. A temporal calculus of communicating systems. In J.C.M. Baeten and J.W. Klop, editors, *Proceedings of Concur '90*, pages 401–415. Springer-Verlag, LNCS 458, 1990.

[22] Y.N. Moschovakis. A model of concurrency with fair merge and full recursion. *Information and Computation*, 93:119–171, 1991.

[23] P.D. Mosses. Denotational semantics. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 575–631. MIT Press/Elsevier, 1990.

[24] D. Murphy. Approaching a real-timed concurrency theory. In M.Z. Kwiatkowska, M.W. Shields, and R.M. Thomas, editors, *Semantics for Concurrency: Proc. of the BCS-FACS Workshop, Leicester, UK*, pages 295–310. Springer-Verlag, July 1990.

[25] P. Panangaden. Abstract interpretation and indeterminacy. In S. D. Brookes, A. W. Roscoe, and G. Winskel, editors, *Proc. of the Seminar on Concurrency*, pages 497–511. Springer-Verlag, LNCS-197, 1985.

[26] P. Panangaden and J. R. Russell. A category-theoretic semantics for unbounded indeterminacy. In *Proc. of the Fifth Conference on Mathematical Foundations of Programming Semantics*, 1989.

[27] P. Panangaden and V. Shanbhogue. On the expressive power of indeterminate dataflow primitives. *Information and Computation*, 98(1):99–131, May 1992.

[28] P. Panangaden and E. W. Stark. Computations, residuals and the power of indeterminacy. In T. Lepisto and A. Salomaa, editors, *Proc. of the Fifteenth International Colloquium on Automata Languages and Programming*, pages 348–363. Springer-Verlag, LNCS-317, 1988.

[29] D. Park. The fairness problem and nondeterministic computing networks. In J.W. de Bakker and J. van Leeuwen, editors, *Foundations of Computer Science IV, Part 2*, pages 133–161. Mathematical Centre, Tract #159, 1983.

[30] V. Pratt. On the composition of processes. In *Conf. Rec. of the Ninth Ann. ACM Symp. on Principles of Programming Languages*, pages 213–223, Jan. 1982.

[31] A. Rabinovich and B.A. Trakhtenbrot. Nets of processes and data flow. In J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, pages 574–602. Springer-Verlag, LNCS-354, 1989.

[32] G.M. Reed and A.W. Roscoe. A timed model for communicating sequential processes. *Theoretical Computer Science*, 58:249–261, 1988.

[33] D. A. Schmidt. *Denotational Semantics: A Methodology for Language Development.* Wm. C. Brown Publishers, 1988.

[34] E. W. Stark. A simple generalization of Kahn's principle to indeterminate dataflow networks. In M. Z. Kwiatkowska, M. W. Shields, and R. M. Thomas, editors, *Semantics of Concurrency, Proceedings of the International BCS-FACS Workshop.* Springer-Verlag, 1990. Available as SUNY Stonybrook TR 89–29.

[35] E.W. Stark. Dataflow networks are fibrations. In D.H. Pitt, P.-L. Curien, S. Abramsky, A.M. Pitts, A. Poigne, and D.E. Rydeheard, editors, *Category Theory and Computer Science*, pages 261–281. Springer-Verlag, LNCS 530, 1991.

[36] R.K. Yates. *Semantics of timed dataflow networks.* PhD thesis, McGill University, 1992.

[37] Wang Yi. CCS + time = an interleaving model for real time systems. In *Proc. of 18th ICALP*, pages 217–228. Springer-Verlag, LNCS 510, 1991.