A Causality Interface for Deadlock Analysis in Dataflow

Ye Zhou
Edward A. Lee

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2006-51
http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-51.html

May 12, 2006

A Causality Interface for Deadlock Analysis in Dataflow

Ye Zhou and Edward A. Lee*
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
Berkeley, CA 94720, USA
{zhouye, eal}@eecs.berkeley.edu

ABSTRACT

In this paper, we consider a concurrent model of computation called dataflow, where components (actors) communicate via streams of data tokens. Dataflow semantics has been adopted by experimental and production languages used to design embedded systems. The execution of a dataflow actor is enabled by the availability of its input data. One important question is whether a dataflow model will deadlock (i.e., actors cannot execute due to a data dependency loop). Deadlock in many cases can be determined, although it is generally not decidable. We develop a causality interface for dataflow actors based on the general framework we introduced in [1] and show how this causality information can be algebraically composed so that composition of components acquire causality interfaces that are inferred from their components and the interconnections. We illustrate the use of these causality interfaces to statically analyze for deadlock.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques; F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs; D.1.3 [**Programming Techniques**]: Concurrent Programming

General Terms

Design, Reliability, Theory, Verification

Keywords

Actors, Behavioral types, Causality, Dataflow, Deadlock, Interfaces

*This paper describes work that is part of the Ptolemy project, which is supported by the National Science Foundation (NSF award number CCR-00225610), and Chess (the Center for Hybrid and Embedded Software Systems), which receives support from NSF, the State of California Micro Program, and the following companies: Agilent, DGIST, General Motors, Hewlett Packard, Infineon, Microsoft, and Toyota.

1. INTRODUCTION

In this paper, we consider dataflow as a model of computation in the *actor-oriented* [2] sense, where "actors" (components that are in charge of their own actions) communicate by exchanging messages. In dataflow, the messages (signals) are streams of data tokens. Actors execute in response to the availability of input data. Variants of dataflow include Kahn-MacQueen process networks [3], extension to nondeterministic systems [4] and Dennis-style dataflow [5]. The dataflow model of computation has been used in industrial practice, in tools such as SPW from the Alta Group of Cadence, the DSP station from Mentor Graphics, and LabVIEW from National Instruments, as well as in experimental contexts, in frameworks such as Ptolemy developed at Berkeley.

One important question is whether a dataflow network *deadlocks*. Deadlock occurs in feedback loops where actors cannot execute, waiting for input data from each other. Many researchers have tackled this problem using different approaches. For example, Lee and Messerschmitt focus on synchronous dataflow, which is a subset of dataflow models, and present an algorithm to determine deadlock [6]. Buck applies clustering techniques and state traversal analysis to boolean dataflow [7]. Matthews uses a metric-space approach to treat deadlock [8].

In this paper, we give an *interface theory* [9], similar in spirit to resource interfaces [10] and behavioral type systems [11]. Our theory captures *causality* properties of actor-oriented designs. Causality properties reflect in the interface the dependence that particular outputs have on particular inputs. The work here is closest in spirit to the component interfaces of Broy in [12], where causality properties of stream functions are formalized. In this paper, we build a rather specialized theory (of causality only) that is orthogonal to other semantic properties.

Following [9] and common practice in object-oriented design, an actor can have more than one interface. We consider actors with input and output ports, where each input port receives zero or more tokens, and the actor reacts to these tokens by producing tokens on the output ports. One interface of the actor defines the number of ports, gives the ports

names or some other identity, and constrains the data types of the tokens handled by the port [13]. Another interface of the actor defines behavioral properties of the port, such as whether it requires input tokens to be present in order to react [11].

In this paper, we consider a particular behavioral interface that we call a *causality interface*. A preliminary form of causality interfaces is given in [1], where it is applied to discrete-event models [14] and synchronous languages [15] for causality loops. This paper refines the algebra for such interfaces and focuses on dataflow and process networks models.

2. ACTORS AND THEIR COMPOSITION

Dataflow actors communicate with each other via streams. A stream is a potentially infinite sequence of distinct tokens. We define a *prefix order* $\sqsubseteq$ on sequences, where $s_1 \sqsubseteq s_2$ if s_1 is a prefix of s_2. For example, $[x_1, x_2] \sqsubseteq [x_1, x_2, x_3]$. Let $\mathcal{S}$ denote the set of all sequences. $(\mathcal{S}, \sqsubseteq)$ is a complete partial order (CPO). The least element of $\mathcal{S}$ is the empty sequence, denoted $\perp$.

Actors receive and produce signals on *ports*. An *actor* a with N ports is a subset of $\mathcal{S}^N$. A particular $s \in \mathcal{S}^N$ is said to satisfy the actor if $s \in a$. s is called a *behavior* of the actor. Thus an actor is a set of possible behaviors. An actor asserts the constraints on the signals at its ports.

A *connector* c between ports P_c is a particulary simple actor where signals at each port $p \in P_c$ are constrained to be identical. The ports in P_c are said to be *connected*.

A set A of actors and a set C of connectors defines a *composite actor*. The composite actor is defined to be the intersection of all possible behaviors of the actors A and connectors C [16].

In dataflow, ports are either inputs or outputs to an actor but not both. Consider an actor $a \subseteq \mathcal{S}^N$ where $I \subseteq \{1, ..., N\}$ denotes the indices of the input ports, and $O \subseteq \{1, ..., N\}$ denotes the indices of the output ports. $I \cup O = \{1, ..., N\}$ and $I \cap O = \emptyset$. Given a signal tuple $s \in a$, we define $\pi_I(s)$ to be the projection of s on a's input ports, and $\pi_O(s)$ on output ports. The actor is said to be *functional* if

$$\forall\ s, s' \in a, \quad \pi_I(s) = \pi_I(s') \Rightarrow \pi_O(s) = \pi_O(s').$$

Such an actor can be viewed as a function from input signals to output signals. Specifically, given a functional actor a with $|I|$ input ports and $|O|$ output ports, we can define an *actor function* with the form

$$F_a \colon \mathcal{S}^{|I|} \to \mathcal{S}^{|O|}, \tag{1}$$

where $|\cdot|$ denotes the size of a set.

When it creates no confusion, we make no distinction between the actor a (a set of behaviors) and the actor function F_a.

An actor with no input ports (only output ports) is functional if and only if its behavior set is a singleton set. That is, it has only one behavior. An actor with no output ports is always functional.

A composite actor is itself an actor. In addition to the set P of ports contained by the composite actor a, the actor may have a set of Q of external ports, where $Q \bigcap P = \emptyset$ (see figure 1). Input ports in Q may be connected to any input port in P that is not already connected. Output ports in Q may be connected to any single output port in P. If the composite actor has no (external) input ports, it is said to be *closed*. Otherwise it is *open*.

A visual syntax for a simple three-actor composition is shown in figure 1(a). Here, the actors are rendered as boxes, the ports as triangles, and the connectors as wires between ports. The ports pointing into the boxes are input ports and the ports pointing out of the boxes are output ports. A textual syntax for the same composition might associate a language primitive or a user-defined module with each of the boxes and a variable name with each of the wires.

The composition in figure 1(a) is closed. In figure 1(b), we have added a level of hierarchy by creating an open composite actor a with external ports $\{q_1, q_2, ..., q_6\}$. In figure 1(c), the internal structure of the composite actor is hidden. Using the techniques introduced in this paper, we are able to do that without losing essential causality information of composite actor a.

In fact, any network of actors can be converted to an equivalent hierarchical network, where the composite actor internally has no directed cycles, like that in figure 1(c). A constructive procedure that performs this conversion is easy to develop. Just create one input port and one output port for each signal in the original network. E.g., in figure 1(a), the signal going from $p5$ to $p2$ induces ports $q5$ and $q2$ in figure 1(b) and (c). Then connect the output port providing the signal value ($p5$ in this example) to the new output port ($q5$), and connect the new input port ($q2$) to any input ports that observe the signal ($p2$). This can be done for any network, always resulting in a structure like that in figure 1(c).

It is easy to see that if actors a_1, a_2, and a_3 in figure 1(b) are functional, then the composite actor a in figure 1(c) is functional. Let F_a denote the actor function for actor a. Assuming the component actors are functional, it has the form

$$F_a \colon \mathcal{S}^3 \to \mathcal{S}^3.$$

The feedback connectors in figure 1(c) require the signals at the input ports of a to be the same as the signals at its outputs. Thus the behavior of the feedback composition in figure 1(c) is $s \in \mathcal{S}^3$ that is a fixed point of F_a. That is,

$$F_a(s) = s.$$

A key question, of course, is whether such a fixed point exists (does the composition have a behavior?) and whether it is unique (is the composition determinate?). We define the semantics of the diagram to be the least fixed point (least in the prefix order), if it exists. The least fixed point is assured of existing if F_a is monotonic (order preserving), and a constructive procedure exists for finding that least fixed point if F_a is also (Scott) continuous (in the prefix

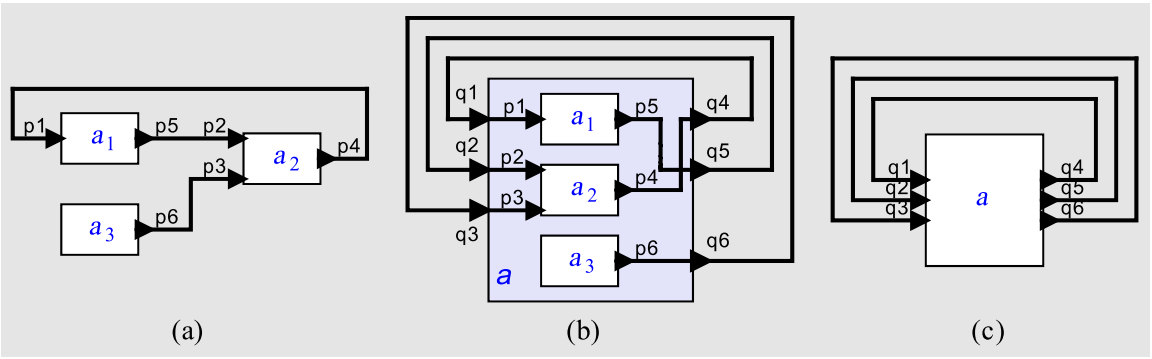

Figure 1: A composition of three actors and its interpretation as a feedback system. $P = \{p1, p2, p3, p4, p5, p6\}$ **is the set of ports contained by the composite actor** a**.** $Q = \{q1, q2, q3, q4, q5, q6\}$ **is the set of external ports of** a**.**

order) [17]. It is easy to show that if a_1, a_2, and a_3 in figure 1(b) have continuous actor functions, then so does a in figure 1(c). Continuity is a property that composes easily.

However, even when a unique fixed point exists and can be found, the result may not be desirable. Suppose for example that in figure 1(c) F_a is the identity function. This function is continuous, so under the prefix order, the least fixed point exists and can be found constructively. In fact, the least fixed point assigns to each port the empty signal. Thus, deadlock manifests itself as a least fixed point of empty or finite sequences. Specifically, we wish to ensure that for a particular dataflow network of actors, if all sources of data are unbounded (external inputs and actors with only output ports can always continue to supply tokens), then all streams in the network are unbounded. A dataflow network that satisfies this requirement is said to be *live*.

Whether such a liveness condition exists may be harder to determine than whether the composition yields a continuous function. In fact, it can be shown that in general this question is undecidable for dataflow models [18]. The causality interfaces we define here provide necessary and sufficient conditions for the deadlock condition. Due to the fundamental undecidability, our necessary and sufficient conditions cannot always be statically checked. But we will show that for many common situations, they are easily checked.

3. CAUSALITY INTERFACES

In this section, we give the definition of causality interfaces, which is refined from [1].

A *causality interface* for an actor a with input ports P_i and outports P_o is a function

$$\delta \colon P_i \times P_o \to D, \tag{2}$$

where D is a partially ordered set with two binary operations $\otimes$ and $\oplus$ that satisfy the axioms given below. The elements of D are called *dependencies*, and $\delta(p_1, p_2)$ denotes the dependency that port p_2 has on p_1.

First, we require that the operators $\oplus$ and $\otimes$ be associative,

$$\forall\, d_1, d_2, d_3 \in D, \quad (d_1 \oplus d_2) \oplus d_3 = d_1 \oplus (d_2 \oplus d_3), \tag{3}$$
$$\forall\, d_1, d_2, d_3 \in D, \quad (d_1 \otimes d_2) \otimes d_3 = d_1 \otimes (d_2 \otimes d_3). \tag{4}$$

Second, we require that $\oplus$ (but not $\otimes$) be commutative,

$$\forall\, d_1, d_2 \in D, \quad d_1 \oplus d_2 = d_2 \oplus d_1, \tag{5}$$

and idempotent,

$$\forall\, d \in D, \quad d \oplus d = d. \tag{6}$$

Third, we require distributivity as follows,

$$\forall\, d_1, d_2, d_3 \in D, \quad d_1 \otimes (d_2 \oplus d_3) = (d_1 \otimes d_2) \oplus (d_1 \otimes d_3), \tag{7}$$
$$\forall\, d_1, d_2, d_3 \in D, \quad (d_2 \oplus d_3) \otimes d_1 = (d_2 \otimes d_1) \oplus (d_3 \otimes d_1). \tag{8}$$

In addition, we require an additive and a multiplicative identity, called $\mathbf{0}$ and $\mathbf{1}$, respectively that satisfy:

$$\exists\, \mathbf{0} \in D \text{ such that} \quad \forall\, d \in D, \quad d \oplus \mathbf{0} = d$$
$$\exists\, \mathbf{1} \in D \text{ such that} \quad \forall\, d \in D, \quad d \otimes \mathbf{1} = \mathbf{1} \otimes d = d$$
$$\forall\, d \in D, \quad d \otimes \mathbf{0} = \mathbf{0}.$$

The ordering relation $\leq$ on the set D is a partial order, meaning, as usual,

$$\forall\, d \in D, \quad d \leq d$$
$$\forall\, d_1, d_2 \in D, \quad d_1 \leq d_2 \text{ and } d_2 \leq d_1 \;\Rightarrow\; d_1 = d_2$$
$$\forall\, d_1, d_2, d_3 \in D, \quad d_1 \leq d_2 \text{ and } d_2 \leq d_3 \;\Rightarrow\; d_1 \leq d_3.$$

Unless otherwise stated, we use $d_1 < d_2$ to mean $d_1 \leq d_2$ and $d_1 \neq d_2$.

Finally, a key axiom of D relates the operators and the order as follows.

$$\forall\, d_1, d_2 \in D, \quad d_1 \oplus d_2 \leq d_1. \tag{9}$$

Connectors (which are also actors) will always have causality interface $\mathbf{1}$, and lack of dependency between ports will be modeled with causality interface $\mathbf{0}$.

The boolean dependency set and the weighted dependency set we introduced in [1] are two examples that satisfy the above axioms.

4. CAUSALITY INTERFACES FOR DATAFLOW MODELS

We define the dependency set D for dataflow models to be a set of functions:

$$D = (\mathbb{N}_\infty \to \mathbb{N}_\infty),$$

where $(X \to Y)$ denotes the set of total functions with domain X and range contained by Y. $\mathbb{N}_\infty = \mathbb{N} \cup \{\infty\}$, where $\mathbb{N} = \{0, 1, 2, \cdots\}$ is the natural numbers. With appropriate choices for an order and $\oplus$ and $\otimes$ operators, this set forms a dependency set.

We define the order relation such that for any $d_1, d_2 \in D$, $d_1 \leq d_2$ if $\forall\ n \in \mathbb{N}_\infty, d_1(n) \leq d_2(n)$. For two different d_1 and d_2, $d_1 < d_2$ means $\forall\ n \in \mathbb{N}_\infty$, $d_1(n) < d_2(n)$ or $d_1(n) = d_2(n) = \infty$.

The $\oplus$ operation computes the greatest lower bound of two elements in D. I.e., $\forall\ d_1, d_2 \in D$, the function $(d_1 \oplus d_2)\colon \mathbb{N}_\infty \to \mathbb{N}_\infty$ is defined by

$$\forall\ n \in \mathbb{N}_\infty, \quad (d_1 \oplus d_2)(n) = \min\{d_1(n), d_2(n)\}.$$

The $\otimes$ operator is function composition. I.e., $\forall\ d_1, d_2 \in D$, the function $(d_1 \otimes d_2)\colon \mathbb{N}_\infty \to \mathbb{N}_\infty$ is defined by

$$d_1 \otimes d_2 = d_2 \circ d_1$$

or

$$\forall\ n \in \mathbb{N}_\infty, \quad (d_1 \otimes d_2)(n) = d_2(d_1(n)).$$

The additive identity $\mathbf{0}$ is the *infinity function*, $d_\infty\colon \mathbb{N}_\infty \to \mathbb{N}_\infty$, given by

$$\forall\ n \in \mathbb{N}_\infty, \quad d_\infty(n) = \infty.$$

The multiplicative identity $\mathbf{1}$ is the *identity function*, $d_I\colon \mathbb{N}_\infty \to \mathbb{N}_\infty$, given by

$$\forall\ n \in \mathbb{N}_\infty, \quad d_I(n) = n.$$

With these definitions, the dependency set satisfies all of the axioms described in Section 3.

Recall that in dataflow models, tokens at input ports trigger tokens at output ports. For input port p and output port p' of an actor a, $\delta_a(p, p') = d$ is interpreted to mean that given n tokens at port p, there will be $d(n)$ tokens at port p'. That is, given an input stream of length n, the output stream has length $(\delta_a(p, p'))(n)$. Note that, in general, $\delta_a(p, p')$ may depend on the input tokens themselves. This fact is the source of expressiveness that leads to undecidability of liveness. However, as we will show, many situations prove decidable.

A *source actor* has no input ports, so we define the causality interface of a source actor to be a function that maps a fictional *absent input port* and an output port p_o of the actor to the infinity function. I.e.,

$$\delta(\varepsilon, p_o) = d_\infty.$$

This assumes, of course, that the source actor is always able to produce tokens.

A *sink actor* is one with no output ports. Similarly, we define the causality interface of a sink actor to be a function that maps an input port p_i of the actor and a fictional *absent output port* to the *zero function*. I.e.,

$$\delta(p_i, \varepsilon) = d_0,$$

where $d_0(n) = 0, \forall\ n \in \mathbb{N}_\infty$.

The causality interface for a connector is simply the multiplicative identity $\mathbf{1} = d_I$.

For a dataflow network to be live, we require that all causality interfaces of actors be unbounded, unless the actor is a sink actor. Intuitively, an actor with a bounded causality interface cannot produce any tokens beyond the bound, causing starvation of input tokens of any downstream actors.

A (functional) actor a with input ports P_i is said to be *monotonic* (or order preserving) if

$$\forall\ s_1, s_2 \in \mathcal{S}^{|P_i|}, \quad s_1 \sqsubseteq s_2 \ \Rightarrow\ F_a(s_1) \sqsubseteq F_a(s_2),$$

where F_a is the actor function of a.

Intuitively, monotonicity says that if the input signal is extended with additional tokens appended to the end, the output can only be changed by extending it with additional tokens. I.e., giving additional inputs can only result in additional outputs. Thus we have the following property:

PROPERTY 1. *Let p be an input port and p' be an output port of a monotonic actor a. Then $\delta_a(p, p')$ is non-decreasing.*

For the purpose of this paper, we assume all actors are (Scott) continuous, a stronger property than monotonicity. A *chain* in a CPO is a totally ordered subset of the CPO. In a CPO, every chain C has a least upper bound, written $\bigvee C$ (this is what makes the CPO "complete"). An actor a is said to be (Scott) *continuous* if for all chains $C \subseteq \mathcal{S}^{|P_i|}$, the *least upper bound* $\bigvee F_a(C)$ exists and

$$F_a(\bigvee C) = \bigvee F_a(C).$$

Here it is understood that $F_a(C) = \{F_a(s) \mid s \in C\}$.

Since the lengths of the streams in a chain C also form a chain in $\mathbb{N}_\infty$ (a CPO with ordinary order), it is easy to see that the following property holds:

PROPERTY 2. *Let p be an input port and p' be an output port of a (Scott) continuous actor a. Then $\delta_a(p, p')$ is (Scott) continuous.*

Continuity implies monotonicity [17], so it follows that the causality interfaces of a continuous dataflow actor are also non-decreasing.

The following theorem will prove useful in this paper.

THEOREM 1. *If $d : \mathbb{N}_\infty \to \mathbb{N}_\infty$ is a continuous function, then*

1. *d has a least fixed point n_0, given by $\bigwedge\{n \in \mathbb{N}_\infty \mid d(n) \leq n\}$.*
2. *$n_0 = \infty$ if and only if $d_I < d$, where n_0 is the least fixed point of d and $d_I = \mathbf{1}$ is the multiplicative identity.*

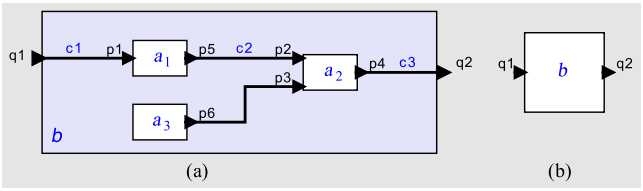

Figure 2: A feedforward composition.

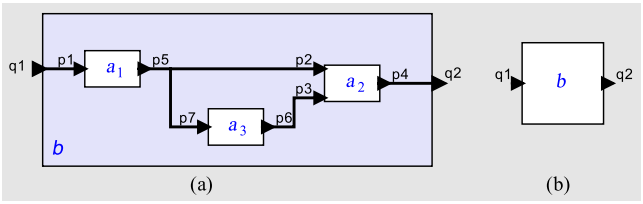

Figure 3: A feedforward composition with parallel paths.

PROOF. Part (1) comes directly from the Knaster-Tarski fixed point theorem [17].

Part (2): If $d_I < d$, then $n_0 = \bigwedge\{n \in \mathbb{N}_\infty \mid d(n) \leq n\} = \infty$.

If $d_I \not< d$, then there is a smallest $n \in \mathbb{N}$ such that $d(n) \leq n$. Then n is the least fixed point of d and n is finite. □

5. COMPOSITION OF CAUSALITY INTERFACES FOR DATAFLOW MODELS

Given a set A of actors, a set C of connectors, and the causality interfaces for the actors and the connectors, we can determine the causality interfaces of the composition and whether the composition is live. To do this, we form a *dependency graph* of ports, and observe that the paths between ports traverse both actors and connectors. We will first discuss feedforward systems and then deal with systems with feedback loops.

5.1 Causality Interfaces for Feedforward Compositions

A feedforward system does not have any cycles in its dependency graph. It is easy to see that a feedforward composition of live actors is always live. To determine the causality interfaces of a composite actor abstracting the feedforward composition, we use the $\otimes$ operator for series composition and the $\oplus$ operator for parallel composition. For example, figure 2 shows a feedforward composition, which is abstracted into a single actor b with external input port $q1$ and output port $q2$. To determine the causality interface of actor b, we need to consider all the paths from $q1$ to $q2$, and $\delta_b(q1, q2)$ is given by

$$\begin{aligned}\delta_b(q1,q2) = \quad & \delta_{c1}(q1,p1) \otimes \delta_{a_1}(p1,p5) \\ & \otimes\delta_{c2}(p5,p2) \otimes \delta_{a_2}(p2,p4) \otimes \delta_{c3}(p4,q2),\end{aligned}$$

where δ_{a_1} and δ_{a_2} are the causality interfaces for actors a_1 and a_2, respectively, and $\delta_{c1}, \delta_{c2}, \delta_{c3}$ are the causality interfaces for connectors $c1, c2, c3$, respectively. Since connectors have causality interface $\mathbf{1}$, the above equation simplifies to

$$\delta_b(q1,q2) = \delta_{a_1}(p1,p5) \otimes \delta_{a_2}(p2,p4).$$

Figure 3 shows a slightly more complicated example, where there are two parallel paths from port $p5$ to port $p4$. We get

$$\begin{aligned}\delta_b(q1,q2) = \delta_{a_1}(p1,p5) \otimes [&\delta_{a_2}(p2,p4) \\ &\oplus(\delta_{a_3}(p7,p6) \otimes \delta_{a_2}(p3,p4))],\end{aligned} \tag{10}$$

where we have omitted the causality interfaces for connectors.

5.2 Causality Interfaces for Feedback Compositions

The dependency graph of a feedback system contains cyclic paths. Given a cyclic path $c = (p_1, p_2, ..., p_n, p_1)$, where p_i's $(1 \leq i \leq n)$ are ports of the composition, we define the *gain* of c to be

$$g_c = \delta(p_1,p_2) \otimes \delta(p_2,p_3) \otimes ... \otimes \delta(p_n,p_1).$$

Note that $c' = (p_i, ..., p_n, p_1, ..., p_i)$ is also a cyclic path, and $g_c \neq g_{c'}$ in general. The ordering of ports of path c' is only a shifted version of that of c. We say that c and c' are two different paths of the same *cycle*.

A *simple cyclic path* is a cyclic path that does not include other cyclic paths. A *simple cycle* is a cycle that does not include other cycles.

We now begin by considering simple cases of feedback systems and build up to the general case. Consider the composition shown in figure 4, where actor a is a feedforward composite actor. From section 5.1, we can determine its causality interface and we know it is live if its component actors are live.

The following three lemmas are useful. The first is an adaptation of lemma 8.10 in [19]:

LEMMA 1. *Consider two CPOs S_1 and S_2, and a continuous function*

$$F_a : S_1 \times S_2 \to S_2.$$

For a given $s_1 \in S_1$, we define the function $F_a(s_1) : S_2 \to S_2$ such that

$$\forall\ s_2 \in S_2, \quad (F_a(s_1))(s_2) = F_a(s_1, s_2).$$

Then for all $s_1 \in S_1$, $F_a(s_1)$ is continuous.

In the context of figure 4(a), this first lemma tells us that if F_a is continuous, then given an input $s_1 \in \mathcal{S}$ at port $p1$, the function $F_a(s_1)$ from port $p2$ to port $p3$ is continuous. Thus, for each s_1, $F_a(s_1)$ has a unique least fixed point, and that fixed point is $\bigvee_{n\in\mathbb{N}}\{(F_a(s_1))^n(\bot)\}$ [17].

The second lemma comes from [20]:

LEMMA 2. *Consider two CPOs S_1 and S_2, and a continuous function $F_a : S_1 \times S_2 \to S_2$. Define a function*

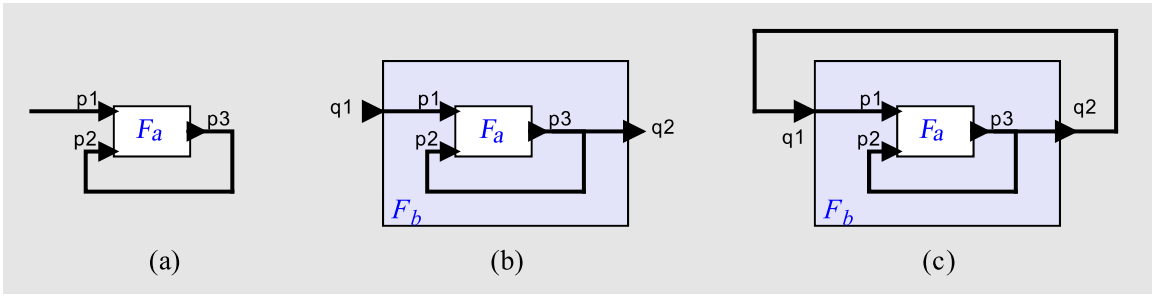

Figure 4: An open composition with feedback loops.

$F_b : S_1 \to S_2$ *such that*

$$\forall\ s_1 \in S_1, \quad F_b(s_1) = \bigvee_{n\in\mathbb{N}} \{(F_a(s_1))^n(\bot_{S_2})\},$$

where $\bot_{S_2}$ *is the least element of* S_2. F_b *is continuous.*

This second lemma tells us that under a least fixed point semantics the composition in figure 4(b) defines a continuous function F_b from port $q1$ to port $q2$.

We now want to find the causality interface for actor b. Let $|s|$ denote the length of the stream $s \in \mathcal{S}$. Given input signal s_1 at port $p1$ and s_2 at $p2$, where $|s_1| = n$ and $|s_2| = m$,

$$|F_a(s_1, s_2)| = \min\{\delta_a(p1, p3)(n), \delta_a(p2, p3)(m)\}.$$

For each $n \in \mathbb{N}_\infty$, we define a function $f_a(n) : \mathbb{N}_\infty \to \mathbb{N}_\infty$ such that

$$\forall\, m \in \mathbb{N}_\infty, \quad (f_a(n))(m) = \min\{\delta_a(p1, p3)(n), \delta_a(p2, p3)(m)\}.$$

$f_a(n)$ is continuous and,

$$\begin{aligned}\delta_b(q1, q2)(n) = |F_b(s_1)| &= |\textstyle\bigvee_{n\in\mathbb{N}}\{(F_a(s_1))^n(\bot)\}| \\ &= \textstyle\bigvee_{n\in\mathbb{N}}\{|(F_a(s_1))^n(\bot)|\} \\ &= \textstyle\bigvee_{n\in\mathbb{N}}\{(f_a(n))^n(0)\}\end{aligned}$$

I.e., $\delta_b(q1, q2)(n)$ is the least fixed point of $f_a(n)$.

The third lemma helps us to find the least fixed point of $f_a(n)$:

Lemma 3. *Consider a continuous function* $\delta : \mathbb{N}_\infty \to \mathbb{N}_\infty$ *and a constant* $K \in \mathbb{N}_\infty$. *We define a function* $g : \mathbb{N}_\infty \to \mathbb{N}_\infty$ *such that*

$$\forall\, m \in \mathbb{N}_\infty, \quad g(m) = \min\{K, \delta(m)\}.$$

Then g *has a least fixed point given by* $m_1 = \min\{K, m_0\}$, *where* m_0 *is the least fixed point of* δ.

Proof. Note that δ and g are continuous and therefore non-decreasing, and $\forall\, m < m_0$, $m < \delta(m)$ (due to Theorem 1). Then

$$\begin{aligned}g(m_1) &= \min\{K, \delta(\min\{K, m_0\})\} \\ &= \min\{K, \delta(K), \delta(m_0)\} \\ &= \min\{K, \delta(K), m_0\}.\end{aligned}$$

1. If $K < m_0$, then $K < \delta(K)$. Therefore, $g(m_1) = K = m_1$.
2. If $m_0 \le K$, then $m_0 = \delta(m_0) \le \delta(K)$. Therefore, $g(m_1) = m_0 = m_1$.

Therefore m_1 is a fixed point of g. Note that $\forall\, m < m_1$, $m < K$ and $m < \delta(m)$. Therefore,

$$m < \min\{K, \delta(m)\} = g(m).$$

Therefore m_1 is the least fixed point of g. □

Corollary 1. *Given the composite actor* b *as shown in figure 4(b),*

1. *The causality interface of* b *is given by*

   $$\forall\ n \in \mathbb{N}_\infty, \quad \delta_b(q1, q2)(n) = \min\{\delta_a(p1, p3)(n), m_0\}$$

   where m_0 *is the least fixed point of* $\delta_a(p2, p3)$.
2. *actor* b *is live if and only if actor* a *is live and* $\mathbf{1} < \delta_a(p2, p3)$, *where* $\mathbf{1} = d_I$ *is the multiplicative identity.*

Proof. Part (1) comes directly by applying $f_a(n)$ to g in Lemma 3.

Part (2): If actor a is live, then $\delta_a(p1, p3)$ is unbounded. If $\mathbf{1} < \delta_a(p2, p3)$, then $m_0 = \infty$. Therefore $\delta_b(q1, q2)$ is unbounded. Thus b is live.

On the other hand, if b is live, then $\delta_a(p1, p3)$ is unbounded and $m_0 = \infty$. This means $\mathbf{1} < \delta_a(p2, p3)$, and actor a is live. □

Given the causality interface for actor b, we now form the nested feedback composition of figure 4(c). We are assured that since b is continuous, this has a unique least fixed point. The composition will be live if and only if $\mathbf{1} < \delta_b(q1, q2)$.

Working towards the structure of figure 1, we add an additional output port to actor a in figure 5. We can easily adapt Lemma 1, 2 and 3 to this situation. Nothing significant changes. We continue to add ports to the actor a, each time creating a nested composite. Since every network can be put into the structure of figure 1(c), we can determine from the causality interface of a, whether a composition is live. Thus we have established the following theorem:

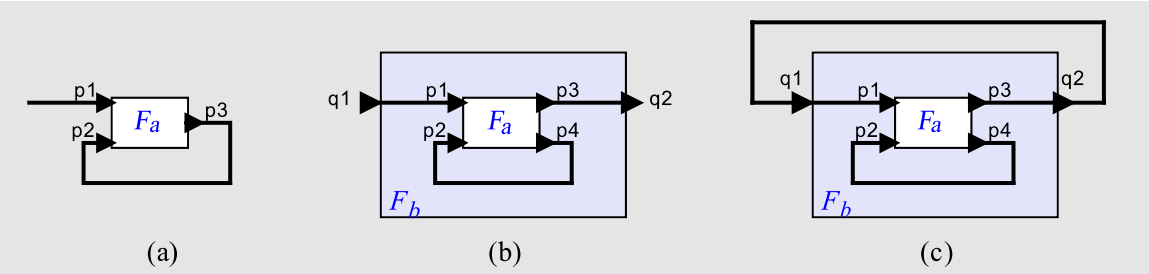

Figure 5: An open system with a feedback connection that has the structure of figure 1.

THEOREM 2. *A finite network of continuous dataflow actors is live if and only if for every cyclic path c in the dependency graph, $\mathbf{1} < g_c$, where $\mathbf{1} = d_I$ is the multiplicative identity.*

6. DISCUSSION

Compare the systems in figure 5(b) and (c). In (b), we determine that the composite is live based only on $\delta_a(p2, p4)$. But in (c), we see that there is implicitly another cyclic path $c = (p2, p3, p1, p4, p2)$. (We omit the external ports $q1$ and $q2$ here.) Are we remiss in ignoring that (potential) cyclic path when determining whether the composite in (b) is live? The answer is negative, as we will show in section 9. The cyclic path c does not need to be checked if $c' = (p1, p4, p2, p3, p1)$, which is only a shifted version of c, is checked. c' can be checked using the causality interface of actor b, without knowing explicitly the causality interface of actor a. Thus, our technique achieves a measure of modularity, in that details of a composite system can be hidden; it is only necessary to expose the causality interface of the composite.

A second question that might arise concerns decidability of deadlock. The above theorems give us necessary and sufficient conditions for a dataflow system to be live. However, deadlock is generally undecidable for dataflow models. These statements are not in conflict. Our necessary and sufficient conditions may not be decidable. In particular, the causality interfaces for some actors, e.g., *boolean select* and *boolean switch* [7], are in fact dependent on the data provided to them at the control port. They cannot be statically known by examining the syntactic specification of the dataflow network unless the input stream at the control port can be statically determined. Theorem 2 implies that if for every cyclic path c, $\mathbf{1} < g_c$ is decidable, then deadlock is decidable. More precisely, if we can prove for every c, $\mathbf{1} < g_c$, then the model is live. If we can prove there exists a cyclic path c such that $\mathbf{1} \not< g_c$, then there is at least one (local) deadlock in the model. If we can prove neither of these, then we can draw no conclusion about deadlock.

Certain special cases of the dataflow model of computation make deadlock decidable. For example, in the synchronous dataflow (SDF) model of computation [6], every actor executes as a sequence of firings, where each firing consumes a fixed, specified number of tokens on each input port, and produces a fixed, specified number of tokens on each output port. In addition, an actor may produce a fixed, specified number of tokens on an output port at initialization. Given an SDF actor a with input port p_i and output port p_o, the causality interface function $\delta_a(p_i, p_o)$ is given by

$$\forall\ n \in \mathbb{N}_\infty, \quad (\delta_a(p_i, p_o))(n) = \begin{cases} \lfloor n/N \rfloor \cdot M + I, & \text{if } n < \infty \\ \infty, & \text{if } n = \infty, \end{cases} \tag{11}$$

where N is the number of tokens consumed at p_i in a firing, M is the number of tokens produced at p_o, and I is the number of initial tokens produced at p_o at initialization. Using this, we get the following theorem.

THEOREM 3. *Deadlock is decidable for synchronous dataflow models with a finite number of actors.*

PROOF. Since distributivity holds for the $(\oplus, \otimes)$ algebra on D, it is easy to see that the gain of any cyclic path can be written in the form

$$g = \bigoplus \left(\bigotimes \delta_a(p_i, p_o) \right), \tag{12}$$

where each $\delta_a(p_i, p_o)$ is in the form of (11), and the $\otimes$ and $\oplus$ operators operate on a finite number of δ's.

We first note that for each function δ in the form of (11), the following property holds:

$$\forall\ k, r \in \mathbb{N}, \quad \delta(kN + r) = \delta(r) + kM, \tag{13}$$

which means

$$\delta(kN + r) - (kN + r) = \delta(r) - r + k(M - N).$$

Therefore, $\mathbf{1} < \delta$ if and only if $N \leq M$ and $\forall\ r \in \{1, 2, ..., N-1\}$, $r < \delta(r)$, which can be determined in finite time. Thus $\mathbf{1} < \delta$ is decidable.

Now consider two causality interfaces δ_a and δ_b of some SDF actors, and they satisfy

$$\forall\ k, r \in \mathbb{N}, \qquad \begin{aligned} \delta_a(kN_a + r) &= \delta_a(r) + kM_a \\ \delta_b(kN_b + r) &= \delta_b(r) + kM_b \end{aligned}$$

where we have omitted mention of the ports for notational simplicity. A cascade of δ_a and δ_b would therefore satisfy

$$(\delta_a \otimes \delta_b)(kN_aN_b + r) = (\delta_a \otimes \delta_b)(r) + kM_aM_b,$$

which is also in the form of (13). We can continue to compose any finite number of causality interfaces with the $\otimes$ operator to get an expression of the form $(\otimes\delta)$, where each

δ is a causality interface in the form of (11), and $(\otimes\delta)$ satisfies (13). Thus $\mathbf{1} < (\otimes\delta)$ is decidable.

Now consider the $\oplus$ operation on two functions δ_1 and δ_2 for which we know whether $\mathbf{1} < \delta_1$ and $\mathbf{1} < \delta_2$. Since $\oplus$ computes the greatest lower bound,

$$\mathbf{1} < (\delta_1 \oplus \delta_2) \Leftrightarrow \mathbf{1} < \delta_1 \wedge \mathbf{1} < \delta_2.$$

Thus $\mathbf{1} < (\delta_1 \oplus \delta_2)$ is decidable. This generalizes easily to any expression of the form of (12) over a finite number of actors. □

In [6], it is shown that if a synchronous dataflow model is consistent, then deadlock is decidable. In particular, this is shown by following a scheduling procedure that provably terminates. Our theory applies to both consistent and inconsistent SDF models, and hence is more general. Moreover, it is more straightforward to check whether $\mathbf{1} < g$ than to execute the scheduling procedure described in [6].

7. AN EXAMPLE

Consider the dataflow model in figure 6(a). Assume all the ports produce and consume one token on each firing of the corresponding actor, and that port $p5$ produces $I \in \mathbb{N}$ initial tokens, and all other ports produce zero initial tokens.

First, we notice that there are two cyclic paths starting from $p1$, namely: $c_1 = (p1, p5, p2, p4, p1)$, and $c_2 = (p1, p5, p7, p6, p3, p4, p1)$, where

$$\begin{aligned} g_{c_1} &= \delta_{a_1}(p1, p5) \otimes \delta_{a_2}(p2, p4) \\ g_{c_2} &= \delta_{a_1}(p1, p5) \otimes \delta_{a_3}(p7, p6) \otimes \delta_{a_2}(p3, p4) \end{aligned}$$

and we want to check whether $\mathbf{1} < g_{c_1}$ and $\mathbf{1} < g_{c_2}$. (Below we show that checking c_1 and c_2 is sufficient to conclude liveness. Checks on cyclic paths starting from other ports are unnecessary.)

A second way to view this model is to create a hierarchy, as shown in figure 6(b), and there is only one cycle between $q1$ and $q2$. The causality interface of actor b is given in (10), and we want to check whether $\mathbf{1} < \delta_b(q1, q2)$. In fact, we find that $\delta_b(q1, q2) = g_{c_1} \oplus g_{c_2}$. Therefore $\mathbf{1} < \delta_b(q1, q2) \Leftrightarrow \mathbf{1} < g_{c_1} \wedge \mathbf{1} < g_{c_2}$. I.e., both approaches check for the same condition. Our interface theory exposes the necessary causality information for composite actors while other details can be hidden.

Using the second approach we get:

$$\begin{aligned} \delta_b(q1, q2) &= \delta_{a_1}(p1, p5) \otimes [\delta_{a_2}(p2, p4) \\ &\quad \oplus (\delta_{a_3}(p7, p6) \otimes \delta_{a_2}(p3, p4))] \\ &= (d_I + I) \otimes [d_I \oplus (d_I \otimes d_I)] \\ &= d_I + I \end{aligned}$$

If $I = 0$, then $\mathbf{1} = \delta_b(q1, q2)$, and the model deadlocks. If $I > 0$, then $\mathbf{1} < \delta_b(q1, q2)$. The model is live.

This example also shows that our causality interfaces can help in designing a system by properly allocating correct number of initial tokens to prevent deadlock.

8. RELATIONSHIP TO PARTIAL METRICS

Matthews uses a metric-space approach to treat deadlock [8]. He defines a partial metric, which is a distance function:

$$f : \mathcal{S} \times \mathcal{S} \to \mathbb{R}_0,$$

where $\mathcal{S}$ is the set of all sequences and $\mathbb{R}_0$ is the non-negative real numbers. Given two sequences $s_1, s_2 \in \mathcal{S}$,

$$f(s_1, s_2) = 2^{-n},$$

where n is the length of the longest common prefix of s_1 and s_2 (if the two sequences are infinite and identical, $f(s_1, s_2) = 0$). The pair $(\mathcal{S}, f)$ is a complete partial metric space.

We first consider a simple scenario of a continuous dataflow actor a with one input port p_i and one output port p_o and a feedback connection from p_o to p_i. The actor function is F_a and the causality interface is δ_a. According to Theorem 4.1 in [8], this feedback system is deadlock free if F_a is a contraction map in this complete partial metric space, meaning

$$\begin{aligned} &\exists\ c \in \mathbb{R}_0, \quad 0 \le c < 1, \quad \text{such that} \\ &\forall\ s_1, s_2 \in \mathcal{S}, \quad f(F_a(s_1), F_a(s_2)) \le c f(s_1, s_2). \end{aligned}$$

THEOREM 4. *Let a be a continuous dataflow actor with one input port p_i and one output port p_o. The actor function of a is F_a. Then $\mathbf{1} < \delta_a(p_i, p_o) \Leftrightarrow F_a$ is a contraction map in the Matthews partial metric space.*

PROOF. Since there is only one relevant causality interface, we abbreviate $\delta_a(p_i, p_o)$ by δ_a (without showing the dependency on the ports). We begin by showing the forward implication.

Given $s_1, s_2 \in \mathcal{S}$, let s be their longest common prefix, and let $n = |s|$ be its length. Then $|F_a(s)| = \delta_a(n) \ge n + 1$. By monotonicity, $F_a(s)$ is a prefix of $F_a(s_1)$ and $F_a(s_2)$. Therefore,

$$f(F_a(s_1), F_a(s_2)) \le 2^{-\delta_a(n)} \le 2^{-(n+1)} = \frac{1}{2} \cdot f(s_1, s_2),$$

so F_a is a contraction map.

We next show the backward implication. Consider two signals s_1 and $s_2 \in \mathcal{S}$, where $|s_1| = n < \infty$ and s_1 is a strict prefix of s_2. Therefore, we have,

$$\begin{aligned} &f(s_1, s_2) = 2^{-n}, \\ &f(F_a(s_1), F_a(s_2)) = 2^{-\delta_a(n)}. \end{aligned}$$

If F_a is a contraction map, then,

$$2^{-\delta_a(n)} < 2^{-n}$$

Since we can arbitrarily choose s_1 (as long as $|s_1|$ is finite), it follows that $\forall\ n \in \mathbb{N}, n < \delta_a(n) \le \delta_a(\infty)$. This concludes that $\mathbf{1} < \delta_a$. □

In Theorem 5.1 in [8], Matthews gives a sufficient condition for liveness for compositions with more than one feedback loop. We can similarly prove that this sufficient condition is equivalent to the condition in Theorem 2 of this paper. Our Theorem 2 shows that it is also a necessary condition for liveness.

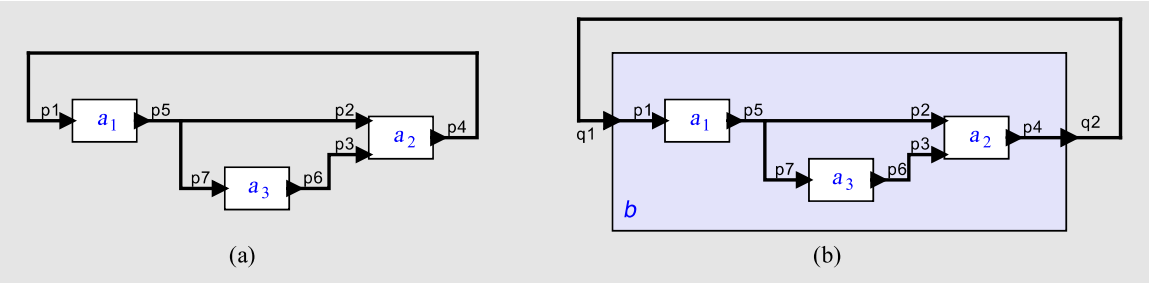

Figure 6: A dataflow model with a feedback loop.

9. COMPUTATION

It is stated in Theorem 2 that a dataflow model is live if and only if for every cyclic path c, $\mathbf{1} < g_c$. We now ask a more practical question. Do we need to verify $\mathbf{1} < g_c$ for every cyclic path c?

Consider a non-simple cyclic path $c = (p_1, ..., p_i, q_1, ..., q_m, p_i, ..., p_n, p_1)$. Therefore $c_1 = (p_1, ..., p_i, p_{i+1}, ..., p_n, p_1)$ and $c_2 = (p_i, q_1, ..., q_m, p_i)$ are two cyclic paths.

Let $d_1 = \delta(p_1, p_2) \otimes ... \otimes \delta(p_{i-1}, p_i)$, $d_2 = \delta(p_i, p_{i+1}) \otimes ... \otimes \delta(p_n, p_1)$. Then,

$$g_{c_1} = d_1 \otimes d_2$$
$$g_c = d_1 \otimes g_{c_2} \otimes d_2.$$

If $\mathbf{1} < g_{c_1}$ and $\mathbf{1} < g_{c_2}$, then, $\mathbf{1} < g_{c_1} = d_1 \otimes d_2 < g_c$. I.e., checking g_{c_1} and g_{c_2} is sufficient. If c_1 or c_2 are non-simple cyclic paths, we can further decompose them into simple cyclic paths. Thus checking only simple cyclic paths is sufficient.

Now we consider two cyclic paths $c_1 = (p_1, p_2, ..., p_n, p_1)$ and $c_2 = (p_i, ..., p_n, p_1, ...p_i)$ of the same cycle. Let $d_1 = \delta(p_1, p_2) \otimes ... \otimes \delta(p_{i-1}, p_i)$, $d_2 = \delta(p_i, p_{i+1}) \otimes ... \otimes \delta(p_n, p_1)$. d_1 and d_2 are continuous, and,

$$g_{c_1} = d_1 \otimes d_2$$
$$g_{c_2} = d_2 \otimes d_1$$

Since commutativity does not hold for the $\otimes$ operator, $g_{c_1} \neq g_{c_2}$ in general. However, we have the following Lemma:

LEMMA 4. *Let $\delta_1, \delta_2 \in (\mathbb{N}_\infty \to \mathbb{N}_\infty)$ be two continuous functions, and δ_1, δ_2 are unbounded. Then $\mathbf{1} < \delta_1 \otimes \delta_2 \Leftrightarrow \mathbf{1} < \delta_2 \otimes \delta_1$.*

PROOF. If $\mathbf{1} < \delta_1 \otimes \delta_2$, then

$$\forall\, n \in \mathbb{N}, \quad n < \delta_2(\delta_1(n)) \tag{14}$$

Suppose, contrary to this lemma, that $\mathbf{1} \not< \delta_2 \otimes \delta_1$, which implies $\exists\ n_0 \in \mathbb{N}$ s.t. $\delta_1(\delta_2(n_0)) \leq n_0$. Since δ_2 is non-decreasing (due to Property 1),

$$\delta_2(\delta_1(\delta_2(n_0)) \leq \delta_2(n_0) \tag{15}$$

If $\delta_2(n_0) < \infty$, then (15) contradicts (14). If $\delta_2(n_0) = \infty$, then $\delta_1(\infty) \leq n_0$. This contradicts with the fact that δ_1 is unbounded. Therefore $\mathbf{1} < \delta_2 \otimes \delta_1$. □

Thus, it is sufficient to compute the gain of one cyclic path for each simple cycle to check liveness for a continuous dataflow network.

10. CONCLUSION

We have given an interface theory that abstractly represents causality of dataflow actors and that easily composes to get causality interfaces of composite actors. We illustrate the use of such interface information to analyze liveness in dataflow networks. We show that liveness is decidable for synchronous dataflow (whether consistent or not). We also show that the causality analysis only needs to be performed for one cyclic path of each simple cycle.

11. REFERENCES

[1] Lee, E.A., Zheng, H., Zhou, Y.: Causality interfaces and compositional causality analysis. In: Foundations of Interface Technologies (FIT), Satellite to CONCUR, San Francisco, CA (2005)

[2] Lee, E.A.: Model-driven development - from object-oriented design to actor-oriented design. In: Workshop on Software Engineering for Embedded Systems: From Requirements to Implementation (a.k.a. The Monterey Workshop), Chicago (2003)

[3] Kahn, G., MacQueen, D.B.: Coroutines and networks of parallel processes. In Gilchrist, B., ed.: Information Processing, North-Holland Publishing Co. (1977)

[4] de Kock, E.A., Essink, G., Smits, W.J.M., van der Wolf, P., Brunel, J.Y., Kruijtzer, W., Lieverse, P., Vissers, K.A.: YAPI: Application modeling for signal processing systems. In: 37th Design Automation Conference (DAC'00), Los Angeles, CA (2000) 402–405

[5] Dennis, J.B.: First version data flow procedure language. Technical Report MAC TM61, MIT Laboratory for Computer Science (1974)

[6] Lee, E.A., Messerschmitt, D.G.: Synchronous data flow. Proceedings of the IEEE (1987)

[7] Buck, J.T.: Scheduling Dynamic Dataflow Graphs with Bounded Memory Using the Token Flow Model. Ph.d. thesis, University of California, Berkeley (1993)

[8] Matthews, S.G.: An extensional treatment of lazy data flow deadlock. Theoretical Computer Science **151** (1995) 195–205

[9] deAlfaro, L., Henzinger, T.A.: Interface theories for component-based design. In: First International Workshop on Embedded Software (EMSOFT). Volume LNCS 2211., Lake Tahoe, CA, Springer-Verlag (2001) 148–165

[10] Chakrabarti, A., Alfaro, L.d., Henzinger, T.A.: Resource interfaces. In Alur, R., Lee, I., eds.: EMSOFT. Volume LNCS 2855., Philadelphia, PA, Springer (2003) 117–133

[11] Lee, E.A., Xiong, Y.: A behavioral type system and its application in Ptolemy II. Formal Aspects of Computing Journal **16** (2004) 210 – 237

[12] Broy, M.: Advanced component interface specification. In: Proceedings of the International Workshop on Theory and Practice of Parallel Programming (TPPP), London, UK, Springer-Verlag (1994) 369–392

[13] Xiong, Y.: An extensible type system for component-based design. Ph.D. Thesis Technical Memorandum UCB/ERL M02/13, University of California, Berkeley, CA 94720 (2002)

[14] Cassandras, C.G.: Discrete Event Systems, Modeling and Performance Analysis. Irwin (1993)

[15] Benveniste, A., Berry, G.: The synchronous approach to reactive and real-time systems. Proceedings of the IEEE **79** (1991) 1270–1282

[16] Lee, E.A., Sangiovanni-Vincentelli, A.: A framework for comparing models of computation. IEEE Transactions on CAD **17** (1998)

[17] Davey, B.A., Priestly, H.A.: Introduction to Lattices and Order. Cambridge University Press (1990)

[18] Lee, E.A., Parks, T.M.: Dataflow process networks. Proceedings of the IEEE **83** (1995) 773–801

[19] Winskel, G.: The Formal Semantics of Programming Languages. MIT Press, Cambridge, MA, USA (1993)

[20] Liu, X.: Semantic foundation of the tagged signal model. Ph.D. Thesis Technical Memorandum UCB/EECS-2005-31, EECS Department, University of California (2005)