CPO Semantics of Timed Interactive Actor Networks

Xiaojun Liu
Edward A. Lee

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2007-131
http://www.eecs.berkeley.edu/Pubs/TechRpts/2007/EECS-2007-131.html

November 5, 2007

Acknowledgement

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, Bosch, DGIST, Hewlett Packard, Microsoft, National Instruments, and Toyota.

CPO Semantics of Timed Interactive Actor Networks [⋆]

Xiaojun Liu [a],

[a]*xiaojun.liu@sun.com*
Sun Microsystems, Inc.

Edward A. Lee [b]

[b]*eal@eecs.berkeley.edu*
EECS Department, University of California, Berkeley

Abstract

We give a denotational framework for composing interactive components into closed or open systems and show how to adapt classical domain-theoretic approaches to open systems and to timed systems. For timed systems, prior approaches are based on temporal logics, automata theory, or metric-spaces. In this paper, we base the semantics on a CPO with a prefix order, as has been done previously for untimed systems. We show that existence and uniqueness of behaviors are ensured by continuity with respect to this prefix order. Existence and uniqueness of behaviors, however, does not imply that a composition of components yields a useful behavior. The unique behavior could be empty or smaller than expected. We define liveness and show that appropriately defined causality conditions ensure liveness and freedom from Zeno conditions. In our formulation, causality does not require a metric and can embrace a wide variety of models of time.

Key words: semantics, CPOs, posets, interaction, actors, agents, timed systems, process networks, discrete events, actors, dataflow

[⋆] 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, Bosch, DGIST, Hewlett Packard, Microsoft, National Instruments, and Toyota.

1 Introduction

Wegner [63] argues that interaction is more expressive than algorithms. Indeed there is a family of approaches to computing being studied by diverse communities that are distinctly interactive rather than algorithmic. These go under the names of coordination languages, agents, actors, and process networks. They all refactor software into components that co-exist and engage in dialog with one another. Key to the expressiveness of such component interactions is "entanglement" [64], where outputs from a component depend on previous outputs, in dialog with the environment. This is distinct from classical models of computing based on the Turing-Church thesis, which do not model such interaction.

In this paper, we will use the term "actors" for interactive components. [1] In contrast to objects, actors are concurrent, in charge of their own actions. Their environment (which can include other actors) provides them with data, and they react and provide the environment with additional data. Actors engage in dialog with their environment. An immediate consequence is that actor-oriented designs tend to be highly concurrent.

The term "actors" has, of course, been used for models of this type. In the classical actor model of Hewitt and Agha [4,29], components have their own thread of control and interact via message passing. The term "actors" has also been used by the dataflow community [22] to refer to chunks of computation that react to the availability of input data by "firing" and producing output data.

We are using the term "actors" more broadly, inspired by the analogy with the physical world, where actors control their own actions. In fact, the most widespread use of interactive models fitting our notion of actors is not rooted in any of these classical communities, but is rather focused on embedded software (where interaction is intrinsic). For example, the synchronous/reactive languages [11] are "actor-oriented" in our sense. Components react at ticks of a global clock, rather than reacting when other components invoke their methods. In the synchronous language Esterel [13], components exchange data through variables whose values are determined by solving fixed point equations. The Lustre [28] and Signal [12] languages focus more on the flow of data, but are semantically similar. Asynchronous dataflow models based on Kahn process networks [31] are also actor-oriented in our sense, and are used for media intensive embedded signal processing software [21]. The more specialized dataflow models in LabVIEW (from National Instruments) are used in instrumentation systems, configurable hardware design, and embedded software design. Discrete-event (DE) systems are also actor oriented, and are commonly used in hardware design (VHDL and Verilog are DE languages) and

[1] The term "agents" is equally good, but we avoid it because in the mind of many researchers, agents include a notion of mobility, which is orthogonal to interaction and irrelevant to our current discussion.

in modeling and design of networked systems [16,8]. In DE, components interact via timed events, which carry data and a time stamp, and reactions are chronologically ordered by time stamp. Modeling software, such as Simulink (from The MathWorks) and Modelica [26] go further by modeling continuous-time dynamics, where components interact via continuously evolving signals.

Wegner argues that interactive models are less amenable to formalism than algorithmic ones [63]. This is debatable, however. While the formalisms may be more complex (this should be expected), they are no less rigorous. Surrounding the actor-oriented approach are a number of semantic formalisms that complement traditional Turing-Church theories of computation by emphasizing interaction of concurrent components rather than sequential transformation of data. These include stream formalisms [31,15,59] and discrete-event formalisms [66,35]. A few such formalisms are rich enough to embrace a significant variety of actor-oriented models of computation, including interaction categories [2], behavioral types [39,7], interaction semantics [61], and the tagged-signal model [38].

Models of actors that have been previously given include the I/O automata of Lynch and Tuttle [45], which extend the rendezvous semantics of CSP [30] and CCS [50] with notions of input and output. This formalism bases its analysis on interleavings of system state trajectories and uses the automata theoretic concepts of refinement and simulation [51] for analysis.

For timed systems, there is a rich history of formalisms. I/O automata have been extended to Hybrid I/O Automata, which embrace timed models, including continuous-time dynamics [43]. Subsequently, Hybrid I/O Automata were specialized to Timed I/O Automata [33], which describe the passage of time but do not allow interactions of continuous-time dynamics.

An alternative approach builds on temporal logics, also known as tense logics. The pioneering work of Pneuli [54] showed how to use temporal logic for understanding reactive systems. Our actors are (individually) reactive systems, so an actor network is a composition of reactive systems. Temporal logics focus on qualitative aspects of timed behavior such as invariance, precedence, and responsiveness. Many variants are surveyed in [6,24,47], including some that represent quantitative aspects of timing. The TLA-based model of timed systems by Abadi and Lamport [1] includes analysis of Zeno systems, similar in spirit to the study in this paper, though achieved by different methods that are arguably more complicated. In temporal logic models as in I/O automata, a system is described as a sequence of state transitions (variously called “actions”). Various logics provide rules for reasoning about state sequences (linear time logics) or trees of possible state trajectories (branching time logics). Compositions of systems lead to (typically nondeterministic) interleavings of state transitions (interleaving semantics) or sequences of observables (trace semantics). Our approach here is closest to the trace semantics, but deviates completely from the focus on state trajectories.

In [36], one of us argues that concurrent semantics based on interleaved state trajectories introduces spurious nondeterminacy that complicates analysis, limits scalability, and impedes understanding. Fundamentally, the focus on sequences of state transitions is more operational than denotational. In concurrent systems, the "system state" may not be observable, and may not even be well defined. Why base a semantics on such a questionable foundation? In addition, models based on discrete state transitions are not capable of fully embracing the time continuum, admitting for example components whose behavior is given by ordinary differential equations. Our approach in this paper admits such systems, and avoids spurious nondeterminacy.

An alternative approach to the semantics of timed systems that is distinctly denotational builds on metric spaces. In 1988, Reed and Roscoe [57,58] gave a semantic framework for concurrent systems, specifically timed CSP, based on complete metric spaces, saying that they "seem a natural method by which to induce a hierarchy on the various models" and they "appear more appropriate for modelling continuous concepts such as real time." The basic approach is to define a metric (actually, typically an ultrametric) on the set of traces of signals communicated between actors. The actors are then modeled as contraction maps, and the Banach fixed point theorem yields a fixed-point semantics. This approach has been pursued by others for both timed systems [66,35] and more conventional concurrent programs [9,10,20]. In [10], Baier and Majster-Cederbaum compare metric space approaches and CPO-based approaches for concurrent systems using CCS. Our objective here in this paper is to show that CPO-based approaches also work for timed systems, and hence similar tradeoffs can be explored.

In this paper, we seek a similarly denotational semantics, but we are not satisfied with the metric space models. In [42], we collaborated with Matsikoudis to show that the standard metric-space formulation excessively restricts the models of time that can be used. In particular, it cannot handle super-dense time [46,48], used in hardware description languages, hybrid systems modeling, and distributed discrete-event models. Super-dense time is essential to cleanly model simultaneous events without unnecessary nondeterminism, and is related to the interleaving semantics introduced in temporal logic approaches [6]. Moreover, the metric-space approaches do not handle well finite time lines, and time with no origin. Moreover, if we admit continuous-time and mixed signals (essential for hybrid systems modeling) or certain Zeno signals, then causality is no longer equivalent to its formalization in terms of contracting functions. In [42], we give an alternative semantic framework using a generalized ultrametric [55] that overcomes these limitations. The existence and uniqueness of behaviors for such systems comes from the fixed-point theorem of [56], but this theorem gives no constructive method to compute the fixed point. In [17] we go a step further, and for the particular case of super-dense time, we define *petrics*, a generalization of metrics, which we use to generalize the Banach fixed-point theorem to provide a constructive fixed-point theorem. In this paper, we are in part reacting to the ever growing algebraic elaborations of the metric-space approaches, giving an alternative that appears to be simpler. It does not, however, define exactly the same set of models.

One of the issues we address in this paper is the possibility of Zeno behaviors in timed systems, where an infinite number of events occur in a finite time. Most of the prior work either imposes excessive restrictions (lower bounds on time intervals between events [14,58] or simply assumes the problem away [6,44]. Abadi and Lamport [1] confront the problem head on, and establish a linkage between "machine closure" [34] and freedom from Zeno conditions. The potential lack of machine closure, however, proves to be a source of incompleteness; Abadi and Lamport nonetheless give conditions (which they admit are intricate) that are significantly less conservative than the excessive restrictions used by others. Our model is similarly incomplete, and like Abadi and Lamport, we give only sufficient conditions for a system to be non-Zeno.

This paper builds on domain theory [3], developed for the denotational semantics of programming languages [65,60]. But unlike many semantics efforts that focus on system state and transformation of that state (temporal logics and automata theory), we focus on concurrent interactions, and do not even assume that there is a well-defined notion of system state. In particular, we develop a timed version of the fixed-point semantics for process networks as introduced by Kahn [31]. Our version uses the tagged-signal model [38].

In the next section, we explain the structure of programs. In the following section, we review the tagged signal model and define signals, which encompass the communication histories between actors. In section 4, we define compositions of interacting actors and open systems. We show that familiar fixed-point semantics, which are traditionally applied to closed systems, can be extended to open systems. In section 5, we specialize to timed systems, and show that the same fixed point semantics give conditions for existence and uniqueness of behaviors. In contrast to other authors [14,52,66], we do not require causality for existence and uniqueness of behaviors. Causality, however, is useful for liveness, the timed analog of freedom from deadlock. We define strict causality without the use of a metric, and like Naundorf [52], show that strict causality in a feedback loop is sufficient for liveness. This contrasts with other authors [14,66], who require a stronger form of causality called delta causality or time guardedness. Moreover, we extend Naundorf by including open systems, by giving conditions for freedom from Zeno behaviors, and by showing that the fixed point is constructive. We close with a discussion of Zeno conditions in timed systems.

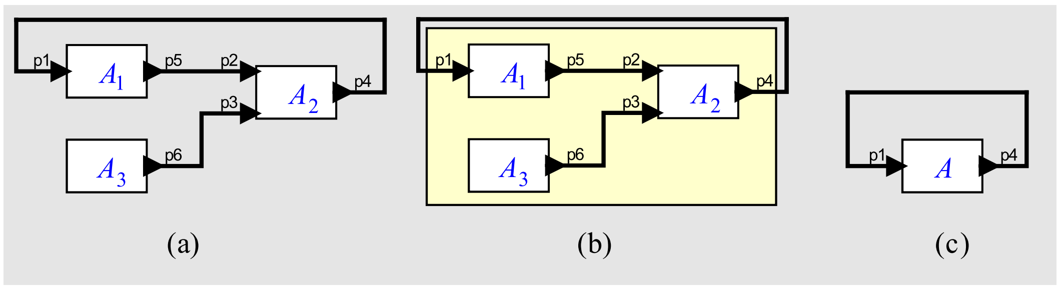

Fig. 1. A composition of three actors and an abstraction.

2 Program Structure

Programs will be given as hierarchical networks of actors like those shown in figure 1. In figure 1(a), a program is given as a network of three actors, A_1, A_2, and A_3. The boxes represent actors, and the triangles on the boxes represent ports. The ports pointing into the boxes are input ports and the ports pointing out of the boxes are output ports. The interconnections (“wires”) between actors represent interaction pathways (”signals”). We take this structure to be static, effectively providing the “source code” for the program. The behavior, of course, may be highly dynamic.

We use a visual syntax here for convenience of exposition, and do not mean to advocate for or against visual syntaxes. A textual syntax for the composition in figure 1(a) 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 synchronous languages Esterel, Lustre, and Signal, for example, have principally textual syntaxes, although recently visual syntaxes for some of them have started to catch on. Ports and connectors are syntactically represented in these languages by variable names. Using the same variable name in two modules implicitly defines ports for those modules and a connection between those ports. Visual syntaxes are more explicit about this architecture. Examples with visual syntaxes include Simulink (from The MathWorks), LabVIEW (from National Instruments), and Ptolemy II [23].

We assume that some actors are “primitive” in the sense that they are not defined in terms of other actors. They might be primitive operations of an actor-oriented language, or software components given in a host language. In the latter case, the actor-oriented language is serving as a coordination language or a composition language.

Actor networks, of course, may be abstracted. In figure 2(b), the three actors are outlined, and then in figure 2(c), aggregated into a single actor. All but two of the ports are hidden. A major objective of this paper is give the semantics of arbitrary aggregations and abstractions like these.

In this paper, we are not assuming a particular syntax, and we consider a family of semantic models rather than a single one. As such, we are not giving a semantics for a particular language, but rather are giving a framework that can be used to develop semantics for a family of languages. Specifically, rather than a particular syntax, we assume only an *abstract syntax*, which asserts that a program is a composition of actors and connectors, that actors are associated with ports, and that a composition is an actor. The family of semantic models that we focus on includes a variety of timed actor-oriented systems.

3 Tagged Signals

The tagged-signal model [38] provides a formal framework for considering and comparing actor-oriented models of computation. It is similar in objectives to the coalgebraic formalism of abstract behavior types in [7], interaction categories [2], and interaction semantics [61]. As with all three of these, the tagged signal model seeks to model a variety of interaction styles between concurrent components.

In the tagged-signal model, each discrete communication between actors is called an **event**. An event is defined to be a pair (t, v), where $t \in T$ is a **tag** and $v \in V$ is a value. A **signal** is a set of events that typically represents the sum total of the communication between two actors along some communication path. The "wires" in figure 1 carry signals. For the systems we are interested in, these sets are very likely infinite. Most applications of the tagged-signal model impose structure on the tag set T and study the consequences of that structure. For example, T might represent causality properties, time, or activation orders.

Later in this paper, we will model time using totally-ordered tag sets. But there is no need to impose that restriction yet. In general, the tag set is partially ordered (a **poset**). A poset $(T, \leq)$ is a set T and a binary relation $\leq$ that is reflexive ($t \leq t$), antisymmetric ($t_1 \leq t_2 \text{ and } t_2 \leq t_1 \Rightarrow t_1 = t_2$), and transitive ($t_1 \leq t_2 \text{ and } t_2 \leq t_3 \Rightarrow t_1 \leq t_3$).

In this paper, we constrain the tagged signal model of [38] in a subtle but important way. Specifically, we assume that a signal is a partial function defined on a down set of T (a similar restriction is made in [52]). Formally,

Definition 1 (Down Set) *Let $(T, \leq)$ be a poset. A subset T' of T is a down set if for all $t' \in T'$ and $t \in T$, $t \leq t'$ implies $t \in T'$.*

Down sets are also called initial segments in the literature [27].

Definition 2 (Signal) *Let $(T, \leq)$ be a poset of tags, and V a non-empty set of values. A signal $s : T \rightharpoonup V$ is a partial function from T to V such that $\mathrm{dom}(s)$ is a down set of T.*

In the above definition, $\mathrm{dom}(s)$ is defined to be the subset of T on which s is defined.

Let $\mathcal{S}$ denote the set of all signals with tag set T and value set V. That is, this is the set of partial functions with domain T and codomain V that are defined on a down set of T. $\mathcal{S}$ is a poset under the **prefix order**, defined next.

Definition 3 (Prefix Order) *For any $s_1, s_2 \in \mathcal{S}$, s_1 is a prefix of s_2, denoted by $s_1 \sqsubseteq s_2$, if and only if* $\mathrm{dom}(s_1) \subseteq \mathrm{dom}(s_2)$, *and* $s_1(t) = s_2(t),\ \forall t \in \mathrm{dom}(s_1)$.

That is, a signal s_1 is a prefix of another signal s_2 if the graph of the function s_1 is a subset of the graph of the function s_2. The prefix order on signals is a natural generalization of the prefix order on strings or sequences, and the extension order on partial functions [62].

A **complete partial order** (CPO) $(P, \leq)$ is a poset where P has least element $\perp_P \in P$, and where every directed subset of P has a least upper bound. A subset $D \subseteq P$ is directed if for all $d_1, d_2 \in D$, $\{d_1, d_2\}$ has an upper bound in D.

A signal set with the prefix order $(\mathcal{S}, \sqsubseteq)$ is a CPO [41]. The least element of $\mathcal{S}$ is $s_\perp \colon \emptyset \to V$, an **empty signal** (it has no events). If a signal is defined for all tags in T, then it is a maximal element of $\mathcal{S}$, and is called a **total signal**.

Note that any pair of signals $\{s_1, s_2\} \subset \mathcal{S}$ has a greatest lower bound $s_1 \wedge s_2 \in \mathcal{S}$. This greatest lower bound is the common prefix, which may be the empty signal if the two signals have nothing in common. In fact, any non-empty subset $S' \subseteq \mathcal{S}$ has a greatest lower bound, which makes $\mathcal{S}$ a **complete semilattice** in addition to a CPO [19].

4 Tagged Systems

Signals, defined in the previous section, represent communication between actors. Actors receive and produce events on **ports**. Thus, a port is associated with a signal, which is a set of events. In this section, we give a declarative definition of actors and show how actors can be composed and abstracted.

4.1 Behaviors

Consider actor A with a finite set of ports $P_A = \{p_1, p_2, ..., p_n\}$. Assume each port sends or receives signals in a signal set $\mathcal{S}_i$ with tag set T_i and value set V_i. Let $\mathcal{S}_A = \mathcal{S}_1 \cup \mathcal{S}_2 \cup ... \cup \mathcal{S}_n$. A **behavior** of A is a function

$$\sigma \colon P_A \to \mathcal{S}_A,$$

with the constraint that $\sigma(p_i) \in \mathcal{S}_i$. A behavior for a set of ports assigns to each port a signal. The set of all behaviors for ports P_A is written $\Sigma_{P_A} \subseteq (P_A \to \mathcal{S}_A)$.

The prefix order can be generalized to behaviors. Given two behaviors $\sigma_1, \sigma_2 \in \Sigma_{P_A}$, we say $\sigma_1 \sqsubseteq \sigma_2$ if and only if for all $p \in P_A$, $\sigma_1(p) \sqsubseteq \sigma_2(p)$. It is then easy to see that $(\Sigma_{P_A}, \sqsubseteq)$ is a CPO and a complete semilattice.

Other definitions also generalize naturally. For instance, a behavior $\sigma\colon P_A \to \mathcal{S}_A$ is **total** if for all $p \in P_A$, $\sigma(p)$ is total.

4.2 Actors as Sets of Behaviors

An **actor** A with ports P_A is a set of behaviors $A \subseteq \Sigma_{P_A}$. That is, an actor can be viewed as constraints on the signals at its ports. A signal $s \in \mathcal{S}_i$ at port $p_i \in P_A$ is said to **satisfy** an actor A if there is a behavior $\sigma \in A$ such that $s = \sigma(p_i)$.

A **connector** C between ports in set P_C is also a set of behaviors $C \subseteq \Sigma_{P_C}$, but with the constraint that for each behavior $\sigma \in C$, there is a signal $s \in \mathcal{S}_C$ such that

$$\forall\, p \in P_C, \quad \sigma(p) = s.$$

That is, a connector asserts that the signals at a set of ports are identical. A connector is an actor as well. In figure 1, actors are shown as rectangles, ports as triangles, and connectors as lines. In this syntax, actors can share ports with connectors.

Notice that because all signals in a behavior of a connector must be identical, there is a type check that must be performed on actor composition. Moreover, whereas a classical type system would focus only on the value sets V, our type check has to also check the tag sets T. This means that actors communicating through connectors must have compatible semantics on their ports. For example, if an actor sends a stream, the receiving actor must accept a stream. If an actor sends timed events, the receiving actor must accept timed events. Composing incompatible actors will result in empty behaviors.

4.3 Composition of Actors

Given two actors, A with ports P_A and B with ports P_B, the **composition behavior** is the intersection, defined as

$$A \wedge B \subseteq ((P_A \cup P_B) \to (\mathcal{S}_A \cup \mathcal{S}_B)),$$

where

$$A \wedge B = \{\sigma \mid \sigma \upharpoonright P_A \in A \text{ and } \sigma \upharpoonright P_B \in B\},$$

where $\sigma \restriction P$ denotes the restriction of σ to the subset P of ports. Note that this is not the intersection of the graphs of the functions. It is larger.

This composition extends easily to arbitrary sets of actors. A composition A of actors $A_1, \cdots, A_n$ is given by

$$A = A_1 \wedge \cdots \wedge A_n.$$

Such a composition is itself an actor, and is called a *composite actor* when we wish to emphasize that it is composed of other actors. We will consider only compositions of a finite number of actors.

Given an actor A with ports P and a subset $Q \subset P$, an *abstraction* A_Q exposes only those ports in Q. That is,

$$A_Q = \{\sigma \restriction Q \mid \sigma \in A\}.$$

An abstraction of an actor is also an actor.

Notice that our formalism does not require that the ports of distinct actors be disjoint. In fact, an actor will normally share ports with one or more connectors, and also possibly with abstractions. A syntax like that in figure 1 imposes specific constraints. An actor A shares ports only with connectors and (possibly) with abstractions of composite actors that include A. Otherwise, ports of actors are disjoint. Although our semantics does not require such constraints, they are useful syntactic devices in actor-oriented languages.

In many actor-oriented formalisms, ports are either inputs or outputs to an actor but not both. Lynch and Tuttle [45] show that the distinction is important, in that models that do not distinguish inputs from outputs, such as CSP [30] and CCS [50], do not capture the notion of *control*, where one component initiates and controls an event while another reacts to it.

Consider an actor A with ports $P_A = P_i \cup P_o$, where P_i are the input ports, P_o are the output ports, and $P_i \cap P_o = \emptyset$. The actor is said to be **functional** if

$$\forall\, \sigma_1, \sigma_2 \in A, \quad (\sigma_1 \restriction P_i = \sigma_2 \restriction P_i) \Rightarrow (\sigma_1 \restriction P_o = \sigma_2 \restriction P_o).$$

Such an actor can be viewed as a function from input signals to output signals. Specifically, given a functional actor A with input ports P_i and output ports P_o, we can define an **actor function**

$$F_A \colon (P_i \to \mathcal{S}_i) \rightharpoonup (P_o \to \mathcal{S}_o). \tag{1}$$

When it creates no confusion, we make no distinction between the actor (a set of behaviors) and the actor function. If the actor function is total, the actor is said to be **receptive**.

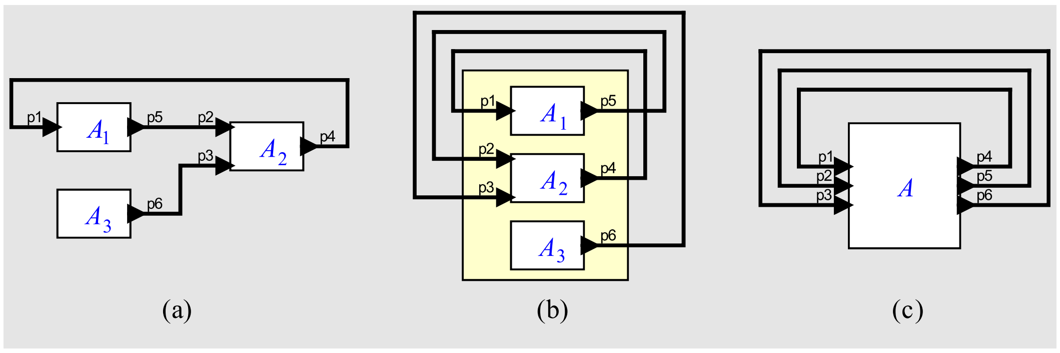

Fig. 2. A composition of three actors and its interpretation as a feedback system.

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 (only input ports) is always functional.

A useful syntactic constraint (which again is not required in the semantics) constrains connectors to have at most one output port. If the output ports of two functional actors are shared by a connector, then unless the output signals of the two actors are identical, the behaviors of the connector and the actors will be empty. Preventing this error through syntactic constraints is advisable when defining a language.

The input ports of a composite actor are the input ports of those actors in the composite that are not shared by connectors with an output port. For example, in figure 1(b), the composite actor represented by the large rectangle has input port p_1. This composite is abstracted to actor A in figure 1(c), where A has only one output port, p_4.

If a composite actor has no input ports, it is said to be **closed**. A composition is **determinate** if it is functional. A key question in many actor-oriented formalisms is, given a set of total functional actors and connectors, is the composition functional and total? This translates into the question of existence and uniqueness of behaviors of compositions. It determines whether a composition is determinate and whether it is receptive. Note that determinacy here is relative to the tag system. Anything not expressed in the tag system is irrelevant. For example, if the tag system is not totally ordered (and hence does not directly express time), then the fact that there are multiple linearizations of partially ordered events does not, by itself, introduce nondeterminacy. This contrasts with formalisms based on system state, which must express progress as progressions from one state to the next, and multiple interleavings are a source of nondeterminacy even if they are semantically irrelevant.

4.4 Fixed Point Semantics

The composition in figure 2(a) can be redrawn as shown in figure 2(b). Figure 2(c) composes the actors A_1, A_2, and A_3 to form actor A, which it then composes with the three connectors. It is easy to see that any block diagram of this type can be redrawn in this way and abstracted to a single actor with the same number of input and output ports, with each output port connected back to a corresponding input port.

It is also easy to see that if actors A_1, A_2, and A_3 in figure 2(b) are functional and receptive, then the composite actor A in figure 2(c) is functional and receptive. Let F_A denote the actor function for actor A. Assuming the component actors are functional and receptive, it has the form

$$F_A\colon (P_i \to \mathcal{S}_i) \to (P_o \to \mathcal{S}_o).$$

The feedback connections in figure 2(c) compose to form an actor with function

$$C\colon (P_o \to \mathcal{S}_o) \to (P_i \to \mathcal{S}_i)$$

that requires the signals at ports P_i to be the same as the signals at ports P_o. The **feedback system function** is thus a composition of the actor function and the feedback connections,

$$(C \circ F_A)\colon (P_i \to \mathcal{S}_i) \to (P_i \to \mathcal{S}_i). \tag{2}$$

Then the behavior of the feedback composition in figure 2(c) is $\sigma \in (P_i \to \mathcal{S}_i)$ that is a fixed point of $C \circ F_A$. That is,

$$(C \circ F_A)(\sigma) = \sigma.$$

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?). This question has been addressed for dataflow process networks using fixed-point theorems on CPOs [31]. For discrete-event models, prior work has defined semantics somewhat differently, by defining a metric space on the set $\mathcal{S}$ of signals [66,35,20], and making causality requirements on the components. We show here that the causality requirements are unnecessary for existence and uniqueness.

4.5 Open Systems

Note that the composition in figure 2 is closed (it has no inputs). We can generalize the formulation to allow open compositions like the example in figure 3 (and generalizations, where A is a composite actor and multiple signals are fed back or serve as inputs). In such cases, we partition the input ports of the composition actor

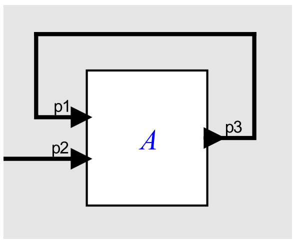

Fig. 3. A composition with feedback and input ports.

into two disjoint sets $P_i = P_i' \cup P_i''$, where P_i' is the set of input ports of actor A that are not connected to any output port of A, and $P_i'' = P_i \setminus P_i'$. Thus, in figure 3, $P_i' = \{p_2\}$ and $P_i'' = \{p_1\}$. Let P_o denote the output ports of A. In figure 3, $P_o = \{p_3\}$. We assume without loss of generality that all output ports are connected back to input ports in P_i''. Then the actor function can be written

$$F_A' \colon (P_i' \to \mathcal{S}_i') \times (P_i'' \to \mathcal{S}_i'') \rightharpoonup (P_o \to \mathcal{S}_o),$$

where $\mathcal{S}_i'$ and $\mathcal{S}_i''$ are the signal sets of ports p_i' and p_i''. As before, we define a connector for the feedback path, which will be a function of the form

$$C \colon (P_o \to \mathcal{S}_o) \to (P_i'' \to \mathcal{S}_i'').$$

The feedback system function is then

$$(C \circ F_A') \colon (P_i' \to \mathcal{S}_i') \times (P_i'' \to \mathcal{S}_i'') \rightharpoonup (P_i'' \to \mathcal{S}_i''). \qquad (3)$$

Given an input behavior $\sigma_i \in (P_i' \to \mathcal{S}_i')$, if the feedback composition of figure 3 has a feedback behavior $\sigma_o \in (P_i'' \to \mathcal{S}_i'')$, then it must be true that

$$(C \circ F_A')(\sigma_i, \sigma_o) = \sigma_o.$$

That is, the behavior on the output ports is a fixed point of a function that is parameterized by the input signal. If this fixed point exists and is unique for all input behaviors, then the **composition function** of figure 3 has the form

$$F \colon (P_i' \to \mathcal{S}_i') \to (P_o \to \mathcal{S}_o). \qquad (4)$$

Because we can model open systems, we no longer need to hide input ports when abstracting systems. Thus, if we assume that nondeterminism (where actors are not functional) is resolved at run-time by external influences, and we model those external influences using input ports, then any source of nondeterminism can be converted into an input port. Thus, the ability to cleanly model open systems significantly reduces the incentive to model nondeterministic systems. If the source of nondeterminism in a system is external events, then a determinate model of an open system is probably better than a nondeterminate model of a closed system. Nonetheless, we conjecture that an adaptation of Plotkin's powerdomain construction would work to provide a generalization to nondeterminate systems [53]. (It needs to be adapted because it is based on transformations of global system state, which is not a well-defined concept in our model.)

We examine next the conditions for existence and uniqueness of the fixed point.

4.6 Existence and Uniqueness of Fixed Points

In this section, we review classical results [19] and apply them to our formulation of actor networks. Let $(D, \sqsubseteq)$ and $(E, \sqsubseteq)$ be CPOs. A function $G\colon D \to E$ is **monotonic** if it is order-preserving,

$$\forall d_1, d_2 \in D,\ \mathrm{d}_1 \sqsubseteq d_2 \implies G(d_1) \sqsubseteq G(d_2).$$

The same function is (Scott) **continuous** if for all directed sets $D' \subseteq D$, $G(D')$ is a directed set and

$$G(\bigvee D') = \bigvee G(D').$$

Here, $G(D')$ is defined in the natural way as $\{G(d) \mid d \in D'\}$, and $\vee X$ denotes the least upper bound of the set X.

It is easy to show that every continuous function is monotonic. A classic fixed point theorem [19] states that if $G\colon D \to D$ for CPO D is continuous, then it has a least fixed point, and that least fixed point is

$$\bigvee\{G^n(\bot_D) \mid n \in \mathbb{N}\}, \tag{5}$$

where $\bot_D$ is the least element of D and $\mathbb{N}$ is the natural numbers.

These results can be immediately applied to closed actor systems like those in figure 2. If each component actor is receptive and continuous, then the system function $C \circ F_A$ of (2) is a continuous function on a CPO. Thus, it has a least fixed point, and that fixed point is given by (5). Following [31], we can define the semantics of the feedback system to be the single unique behavior that is the least fixed point. As we will see, however, this result applies much more broadly than to the process networks of [31].

It is intuitive for actors to be monotonic in the prefix order. Consider an actor A with a single input port, a single output port, and actor function F_A. Consider two possible input signals s_1 and s_2, where $s_1 \sqsubseteq s_2$. That is, s_2 extends (or equals) s_1. If F_A is monotonic, then $F_A(s_1) \sqsubseteq F_A(s_2)$. That is, $F_A(s_2)$ extends (or equals) $F_A(s_1)$. Intuitively, extending the input does not result in changes to "previously produced" outputs (the portion of the output that results from the unextended input). Thus, it is natural for actor functions to be monotonic. Intuitively, if an actor function is also continuous, then this means that the actor does not wait forever before producing output. This behavior is also intuitive and natural. Thus, we conclude that constraining functional actors to be continuous is not onerous.

To handle open systems like those in figure 3, we have a bit more work to do, but again, classic results can be applied almost immediately. As before, let $(D, \sqsubseteq)$ and $(E, \sqsubseteq)$ be CPOs, but now we consider a function of the form

$$G \colon D \times E \to E. \tag{6}$$

For a given $d \in D$, let $G(d) \colon E \to E$ be the function such that

$$\forall\, e \in E, \quad (G(d))(e) = G(d, e).$$

If G is continuous, then for all $d \in D$, $G(d)$ is continuous (lemma 8.10 in [65]). Hence, $G(d)$ has a unique least fixed point, and that fixed point is

$$\bigvee \{(G(d))^n(\bot_E) \mid n \in \mathbb{N}\},$$

where $\bot_E$ is the least element of E.

We recognize immediately that the feedback system function of (3) is a function of form (6). Moreover, if the component actors are receptive and continuous, then the feedback system function will be receptive and continuous, and given an input behavior $\sigma_i \in (P_i' \to \mathcal{S}')$,

$$(C \circ F_A')(\sigma_i) \colon (P_i'' \to \mathcal{S}_i'') \to (P_i'' \to \mathcal{S}_i'')$$

is continuous and hence has a least fixed point. We take that least fixed point to be the semantics of the system. Thus, for any input behavior σ_i, the feedback composition has a unique semantics, and that semantics is a function of the form (4). We now show that that function is receptive and continuous.

Since for any input behavior the system in figure 3 has a unique semantics, the composition function F of (4) is well defined. More interestingly, we can show that if each of the component actors is receptive and continuous, then the composition function F is receptive and continuous. This follows first from the (trivial) observation that F_A', C, and $(C \circ F_A')$ are receptive and continuous, and second from the following theorem.

Theorem 4 *Let $(D, \sqsubseteq)$, $(E, \sqsubseteq)$ be CPOs, and let $G \colon D \times E \to E$ be a continuous function. Define a function $F \colon D \to E$ such that*

$$\forall\, d \in D, \quad F(d) = \bigvee \{(G(d))^n(\bot_E) \mid n \in \mathbb{N}\}.$$

That is, $F(d)$ yields the least fixed point of the function $G(d) \colon E \to E$ (which exists and is unique). F is continuous.

Proof. Let $[E \to E]$ be the set of all continuous functions from E to E. We can define a partial order on this set by $\forall\, p, q \in [E \to E]$,

$$p \sqsubseteq q \iff \forall\, y \in E,\ p(y) \sqsubseteq q(y).$$

With this partial order, $[E \to E]$ is a CPO. For any directed set $D' \subseteq D$, $\{G(d) \mid d \in D'\} \subseteq [E \to E]$ is a directed set, and

$$\bigvee\{G(d) \mid d \in D'\} = G(\bigvee D'),$$

so the function $G\colon D \to [E \to E]$ is continuous. Let fix: $[E \to E] \to E$ denote the function that yields the unique least fixed point of any continuous function in $[E \to E]$. By Theorem 2.1.19 in [3], fix is continuous. Note that

$$F = \mathrm{fix} \circ G.$$

Since this is the composition of two continuous functions, F is continuous. $\square$

5 Timed Interactive Networks

Our framework so far can easily subsume some classical results. For example, if the tag set for all signals is $T = \mathbb{N}$, the natural numbers, then our networks are Kahn process networks [31]. The constraint that signals be defined on a down set of T is natural in this case. However, our framework is more general, and in this paper, we focus on its use for timed interactive networks.

5.1 Models of Time

Our framework admits several models of time. In all cases, the tag set T will be totally ordered. Perhaps the most natural choice, where $T = \mathbb{R}_+$, the non-negative reals, reflects a Newtonian physical view of time. The fact that we include only the non-negative reals implies that our timed interactive networks have a starting point.

A more interesting model of time is super dense time (SDT) [48], where $T = \mathbb{R}_+ \times \mathbb{N}$ with lexical ordering,

$$(r_1, n_1) \leq (r_2, n_2) \iff r_1 < r_2, \text{ or } r_1 = r_2 \text{ and } n_1 \leq n_2\,. \tag{7}$$

This is a total order. SDT can be similarly defined as $T = I \times \mathbb{N}$, where I is any interval of real numbers. SDT has been used in studying the semantics of hybrid systems [32,40,46]. A subset $T = \mathbb{N} \times \mathbb{N}$, is used as the model of time in some hardware description languages (notably VHDL). SDT is in a sense "strictly richer" than $\mathbb{R}_+$ as a model of time, in that one can show that there is no order-embedding of $T = \mathbb{R}_+ \times \mathbb{N}$ in $\mathbb{R}_+$.

We make few constraints on the value sets, but for most models, it is useful to assume that every value set V contains a special element $\varepsilon \in V$ that represents

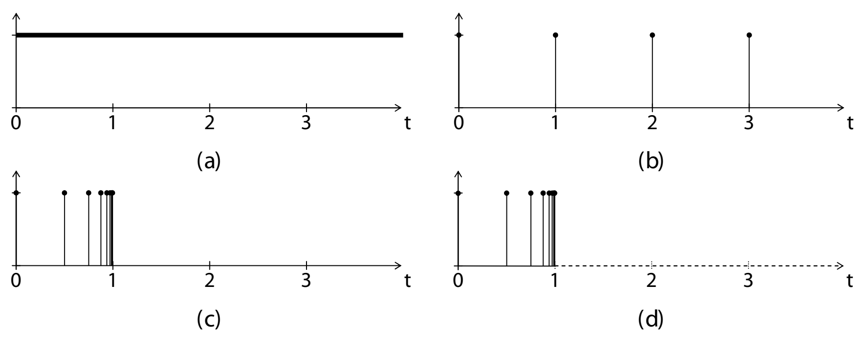

Fig. 4. Examples of timed signals: (a) $const_1$, (b) $clock_1$, (c) $zeno$, (d) $dzeno$.

absence of a value. Without this choice, only signals defined on a connected interval of $\mathbb{R}_+$ including 0 would meet our requirement that signals be defined on a down set. This would unnecessarily constrain us to continuous-time signals.

5.2 Defining Signals

For convenience in giving examples, we will give signals as a tuple, $(\mathrm{dom}(s), E)$ where $\mathrm{dom}(s)$ is the domain of the signal (a down set of T), and E is the set of events that are not absent,

$$E = \{(t, s(t)) \,|\, t \in \mathrm{dom}(s),\ s(t) \neq \varepsilon\}.$$

By implication, all other events with a tag in the domain are absent. If E is a finite set, signal s is called a **finite signal**. For example,

$$\begin{aligned} s_\perp &= (\emptyset, \emptyset), \\ s_\varepsilon &= (T, \emptyset). \end{aligned}$$

The empty signal $s_\perp$ has no events, whereas the absent signal s_ε has absent events (t, ε) for all $t \in T$.

The following examples, with $T = \mathbb{R}_+$ and $V = \{0, 1, \varepsilon\}$, are sketched in figure 4:

$$\begin{aligned} \mathit{const}_1 &= (\mathbb{R}_+, \{(t, 1) \,|\, t \in \mathbb{R}_+\}), \\ \mathit{clock}_1 &= (\mathbb{R}_+, \{(k, 1) \,|\, k \in \mathbb{N}\}), \\ \mathit{zeno} &= (\mathbb{R}_+, \{(1 - 1/2^k, 1) \,|\, k \in \mathbb{N}\}), \\ \mathit{dzeno} &= ([0, 1), \{(1 - 1/2^k, 1) \,|\, k \in \mathbb{N}\}). \end{aligned}$$

The only difference between $zeno$ and $dzeno$ is the tag set.

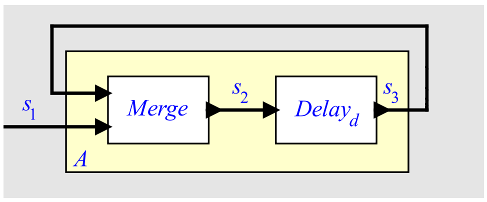

Fig. 5. A composition that can be shown to be live.

5.3 Examples of Actors

We now consider two example actors, $Delay_d$ and $Merge$. Let d be any positive real number. The $Delay_d \colon S \to S$ actor shifts every event in its input signal by d into the future such that if $r = Delay_d(s)$, then

$$\begin{aligned} \mathrm{dom}(r) &= \{t \in T \,|\, t - d \in \mathrm{dom}(s) \text{ or } t - d \notin T\}, \\ r(t) &= \begin{cases} s(t-d) & t - d \in \mathrm{dom}(s), \\ \varepsilon & \text{otherwise.} \end{cases} \end{aligned} \tag{8}$$

The $Merge \colon S^2 \to S$ actor combines the present events in its input signals into its output signal, giving precedence to its first input when both input signals are present at the same time. Specifically, if $s = Merge(s_1, s_2)$, then

$$\begin{aligned} \mathrm{dom}(s) &= \mathrm{dom}(s_1) \cap \mathrm{dom}(s_2), \\ s(t) &= \begin{cases} s_1(t) & s_1(t) \neq \varepsilon, \\ s_2(t) & \text{otherwise.} \end{cases} \end{aligned} \tag{9}$$

It is easy to prove that actors $Delay_d$ and $Merge$ are both continuous [41]. They are also obviously both receptive.

5.4 Live Systems

A closed composition of actors is said to be **live** if all its behaviors are total (if it is determinate, then there is only one behavior). An open composition of actors is live if, given input signals that are total, all behaviors are total. This broadly captures the notions of freedom from deadlock, livelock, and causality loops. By contrast, in [5], a system is live if an infinite number of inputs generates an infinite number of outputs. Our definition is stronger, in that the output must be defined on the entire tag set, and weaker in that there need not be any output at all, and both the output and the input may be continuous-time signals.

Consider the composition shown in figure 5, which has the form of that in figure 3 when the $Merge$ and $Delay_d$ are aggregated. Since both $Merge$ and $Delay_d$ are receptive and continuous, the composite actor A is receptive and continuous, and hence the feedback composition is itself a continuous function, using results from section 4.6 above. We can also show that it is live.

To show that the composition in figure 5 is live, we use abstract interpretation [18], considering the actors only to be relations on the domains of the signals,

$$\begin{aligned}\mathrm{dom}(s_2) &= \mathrm{dom}(s_3) \cap \mathrm{dom}(s_1),\\ \mathrm{dom}(s_3) &= [0, d) \cup \{t + d \,|\, t \in \mathrm{dom}(s_2)\}.\end{aligned}$$

If the input s_1 is total, then $\mathrm{dom}(s_1) = T$ and $\mathrm{dom}(s_2) = \mathrm{dom}(s_3)$. This implies that

$$\mathrm{dom}(s_3) = [0, d) \cup \{t + d \,|\, t \in \mathrm{dom}(s_3)\}.$$

The only subset of $\mathbb{R}_+$ that satisfies the last equation is $\mathbb{R}_+$, so both s_2 and s_3 are total signals.

Not all timed process networks have this property. Suppose we replace the Delay_d actor in figure 5 with an actor $\mathit{LookAhead}_a\colon S \to S$, where a is a positive real number. Given a signal s, the output $r = \mathit{LookAhead}_a(s)$ is defined by

$$\begin{aligned}\mathrm{dom}(r) &= \{t \in T \,|\, t + a \in \mathrm{dom}(s)\},\\ r(t) &= s(t + a),\end{aligned}$$

It is easy to show that $\mathit{LookAhead}_a$ is continuous.

If we replace Delay_d in figure 5 with $\mathit{LookAhead}_a$, the composition still yields a receptive and continuous function from inputs to outputs, because like Delay_d, $\mathit{LookAhead}_a$ is continuous. However, the composition is not live. Given any input s_1, the least fixed point is $s_2 = s_3 = s_\perp$, the empty signal. Thus, the feedback composition gives a function that maps all inputs to the empty signal. This function is certainly receptive and continuous, but it's not very useful. This situation is analogous to deadlock in Kahn process networks.

It is well known that, in general, whether a network of actors is live is undecidable (this is known for Kahn process networks, which are a special case of our framework, so we must assume that in general liveness is undecidable). We have two alternatives. We can specialize the semantics of actors and tag systems to decidable subsets (such as synchronous dataflow [37] and the synchronous/reactive languages [11]), or we can find sufficient conditions for a network to be live, where the sufficient conditions are checkable and not overly restrictive.

The latter approach is commonly used in timed systems such as discrete-event languages [66,35], where a metric space of signals is constructed and contraction maps combined with the Banach fixed point theorem yield live systems. For systems of the types represented by figures 2 and 3, a sufficient condition for a system to be live is that the composite actor A be a contraction map. This corresponds to the more intuitive requirement that every directed loop in a timed actor network include a time delay greater than some $\alpha > 0$. In practice, even though this condition is only sufficient and not necessary, this constraint is not onerous. Designers using discrete-event languages, such as hardware description languages, have no difficulty complying, and no difficulty understanding why the requirement is needed. Indeed, they would consider systems that do not comply but are still live to be pathological. However, in the context of hybrid systems [40], contraction maps are, in fact, overly restrictive.

The metric space approach has been adapted to untimed systems (specifically Kahn process networks) by Matthews [49], who uses a partial metric where the distance of a sequence to itself is greater than zero if the sequence is finite, and is zero only if the sequence is infinite. Matthews develops a generalization of the Banach fixed point theorem to partial metrics and shows that if you have a contraction, then the system is live (he calls the system "complete" rather than "live").

The metric space approach has also been generalized to handle hybrid systems better. In [42], we give an alternative semantic framework using a generalized ultrametric [55] and the fixed-point theorem of [56]. In [17] we address the particular case of super-dense time and define *petrics*, a generalization of metrics, which we use to generalize the Banach fixed-point theorem to provide a constructive fixed-point theorem.

In this paper, we give a sufficient condition for a system to be live that does not require the machinery of a metric, partial metric, generalized ultrametric, or petric, and yet subsumes these mechanisms as special cases. Our approach is based on a simple and intuitive definition of causality.

5.5 Causality

Causality is the relationship between causes and effects. If a timed process models a physical or computational process, the time of an effect cannot be earlier than the time of the corresponding cause. This intuition is captured by the following definition.

Definition 5 (Causality) *An actor A with input ports P_i and output ports P_o is causal if it is monotonic, and for all behaviors $\sigma \in A$,*

$$\bigcap_{p \in P_i} \mathrm{dom}(\sigma(p)) \subseteq \bigcap_{p \in P_o} \mathrm{dom}(\sigma(p)) \,. \tag{10}$$

An immediate consequence of this definition is that a causal actor is live. Thus, whether a composition of actors is causal will tell us whether it is live.

To understand this definition intuitively, consider the case where the tag set T is totally ordered. Then this definition says that if the inputs to a causal actor are known up to some tag $t \in T$, then the outputs are known at least up to that same tag t.

Also, a consequence of this definition is that if the input signals in one behavior $\sigma \in A$ are the same as the input signals in another behavior $\sigma' \in A$ up to some tag t, then the corresponding output signals will be the same up to the same tag t.

We can make this precise. Let $D(t) = \{\tau \in T \mid \tau \leq t\}$ for some $t \in T$ denote the smallest down set including t. If an actor A is causal, then for any two behaviors $\sigma, \sigma' \in A$ and time t such that

$$t \in \bigcap_{p \in P_i} \mathrm{dom}(\sigma(p)) \cap \mathrm{dom}(\sigma'(p)),$$

$$\begin{aligned} &\forall\, p \in P_i,\ \sigma(p) \upharpoonright D(t) = \sigma'(p) \upharpoonright D(t) \implies \\ &\forall\, p \in P_o,\ \sigma(p) \upharpoonright D(t) = \sigma'(p) \upharpoonright D(t). \end{aligned}$$

This follows immediately from the definition of causality and the fact that the actor is monotonic.

Among the actors discussed so far, Delay_d and Merge are causal, whereas $\mathit{LookAhead}_a$ is not.

Neither causality nor continuity implies the other. The $\mathit{LookAhead}_a$ process is continuous but not causal. A minor variant of the Merge actor that we call $\mathit{MaxMerge}$ is causal but not continuous. The $\mathit{MaxMerge}\colon S^2 \to S$ actor is such that $s = \mathit{MaxMerge}(s_1, s_2)$ is given by

$$\mathrm{dom}(s) = \{t \in \mathrm{dom}(s_1) \mid \forall \tau \in D(t) \setminus \mathrm{dom}(s_2),\ s_1(\tau) \neq \varepsilon\}, \tag{11}$$

$$s(t) = \begin{cases} s_1(t) & s_1(t) \neq \varepsilon, \\ s_2(t) & \text{otherwise.} \end{cases} \tag{12}$$

Intuitively, if the input signal s_1 is continuously present over a time interval beyond $\mathrm{dom}(s_2)$, then those present events are in the output of $\mathit{MaxMerge}$. The "Max" in the name is suggestive that this actor, unlike Merge, produces the maximal output for a given pair of inputs.

Lemma 6 *$\mathit{MaxMerge}$ is not continuous.*

Proof. Assume $T = \mathbb{R}_+$ and consider two signals

$$\begin{aligned} u_1 &= ([0,1], \{(1,1)\}), & (13)\\ u_2 &= ([0,1), \emptyset), & (14)\\ \mathit{MaxMerge}(u_1, u_2) &= u_1. & (15) \end{aligned}$$

Let

$$\begin{aligned} r_k &= ([0, 1 - \frac{1}{2^k}), \emptyset),\ k \in \mathbb{N},\\ D &= \{(u_1, r_k),\ k \in \mathbb{N}\}. \end{aligned}$$

D is a directed set, and

$$\begin{aligned} \mathit{MaxMerge}(u_1, r_k) &= r_k, & \bigvee \mathit{MaxMerge}(D) &= u_2.\\ \bigvee D &= (u_1, u_2), & \mathit{MaxMerge}(\bigvee D) &= u_1. \end{aligned}$$

$$\bigvee \mathit{MaxMerge}(D) \neq \mathit{MaxMerge}(\bigvee D)\,.$$

Hence, the actor is not continuous. □

It is easy to see that any composition of causal actors without directed cycles is itself a causal actor. This is not in general true when there are directed cycles. In this case, we will require that at least one actor in the loop be strictly causal, as defined next.

Definition 7 (Strict Causality) *An actor A with input ports P_i and output ports P_o is strictly causal if it is monotonic, and for all behaviors $\sigma \in A$, either $\sigma(p)$ is total for all $p \in P_o$ or*

$$\bigcap_{p \in P_i} \mathrm{dom}(\sigma(p)) \subset \bigcap_{p \in P_o} \mathrm{dom}(\sigma(p))\,. \qquad (16)$$

Here $\subset$ denotes a strict subset relation. Note that if A is a strictly causal actor with one input and one output, then $A(s_\perp) \neq s_\perp$. A must "come up with something from nothing." This is, of course, why strictly causal actors are useful in directed cycles. Strict causality in our sense serves a similar role to "delta causality" in metric space formulations, but ours does not require a metric.

We might assume that Delay_d is strictly causal, but this is not always the case. If the tag set is $T = \mathbb{R}_+$, then Delay_d is strictly causal for any $d > 0$. The same holds if T is any interval in the reals that is not a down set of $\mathbb{R}$. If T is a down set of $\mathbb{R}$, such as $(-\infty, 0]$ or $\mathbb{R}$ itself, then Delay_d is not strictly causal, as evidenced by the fact that $\mathit{Delay}_d(s_\perp) = s_\perp$.

We finally come to the main result of this section. The following theorem effectively gives us a sufficient condition for networks to be live, since causal actors are live.

Theorem 8 (Causality of Feedback Compositions) *Given a totally ordered tag set and a network of causal, receptive, and continuous actors where in every dependency loop in the network there is at least one strictly causal actor, then the network is a causal, receptive, and continuous actor.*

Proof. (Sketch) We will prove the theorem for networks of the form of figure 3. The composition actor has input port p_2 and output port p_3. Note that since actor A is continuous, the composite actor is receptive and continuous by the results of section 4.6. So we only have to show causality. The generalization to arbitrary networks is notationally more tedious, but conceptually identical, and is given in [41].

We proceed by contradiction. Suppose the composite actor is not causal. Then there exists an input signal s_2 at port p_2 and output signal s_3 at p_3 where $\mathrm{dom}(s_2) \not\subseteq \mathrm{dom}(s_3)$. Since the tag set is totally ordered, the set of down sets of the tag set is totally ordered by set inclusion. Thus, if $\mathrm{dom}(s_2) \not\subseteq \mathrm{dom}(s_3)$, then it must be true that $\mathrm{dom}(s_3) \subset \mathrm{dom}(s_2)$ (a strict subset). The signal s_1 at port p_1 is the same as s_3, so $\mathrm{dom}(s_1) \subset \mathrm{dom}(s_2)$ and $\mathrm{dom}(s_1) \cap \mathrm{dom}(s_2) = \mathrm{dom}(s_1)$. Hence, strict causality requires $\mathrm{dom}(s_1) \subset \mathrm{dom}(s_3)$, but we have $\mathrm{dom}(s_1) = \mathrm{dom}(s_3)$, a contradiction. □

Note that this proof is not constructive. It does not tell us how to find the behavior of the actor network, it just tells us that there is a well-defined behavior, and it implies that if the input is total then the output is total. Since we assume the actors are receptive and continuous, the behavior of the network is the same as obtained constructively by theorem 4. However, although theorem 4 is constructive, behaviors of the system may not be computable in practice. We examine this issue next.

6 Discrete-Event Systems

An important subclass of timed systems are discrete event (DE) systems [16,25,35]. Here, we give a strong definition of such systems, showing that they provide a subset of timed systems that can be computed one event at a time. In particular, appropriately constrained DE systems yield a countable set of events and avoid Zeno conditions, which in practice can be as big an obstacle to practical utility as lack of liveness. We begin with the definition of DE signals and their properties.

Definition 9 (Discrete Event Signal) *A timed signal $s \in \mathcal{S}$ is a discrete event (DE) signal if there exists a directed set $D \subseteq \mathcal{S}$ of finite timed signals such that*

$$s = \bigvee D\,.$$

Let $\mathcal{S}_{\mathrm{d}} \subseteq \mathcal{S}$ denote the set of all DE signals with the same tag and value sets as $\mathcal{S}$. Among the signals in figure 4, $clock_1$ and $dzeno$ are DE signals, but not $const_1$ and $zeno$.[2] Both the empty signal $s_\perp$ and the absent signal s_ε are DE signals.

There are several equivalent definitions of DE signals, as established by the following lemmas.

Lemma 10 *A timed signal s is a DE signal if and only if for all $t \in \mathrm{dom}(s)$, $s \upharpoonright D(t)$ is a finite signal.*

Proof. Let s be a DE signal and D a directed set of finite signals such that $s = \bigvee D$. For all $t \in \mathrm{dom}(s)$, there exists $r \in D$ such that $t \in \mathrm{dom}(r)$.

$$r \sqsubseteq s \implies s \upharpoonright D(t) \ = r \upharpoonright D(t).$$

r is a finite signal implies $r \upharpoonright D(t)$ is a finite signal, so is $s \upharpoonright D(t)$.

For any timed signal s, let

$$D_s = \{s \upharpoonright D(t) \,|\, t \in \mathrm{dom}(s)\} \cup \{s_\perp\}\,.$$

D_s is a directed set and $s = \bigvee D_s$. If for all $t \in \mathrm{dom}(s)$, $s \upharpoonright D(t)$ is finite, then s is a DE signal. □

Lemma 11 *A timed signal $s \in \mathcal{S}$ is a DE signal if and only if $s^{-1}(V \setminus \{\varepsilon\})$ is order-isomorphic to a down set of $\mathbb{N}$, and if $s^{-1}(V \setminus \{\varepsilon\})$ is an infinite set, then*

$$\mathrm{dom}(s) = \bigcup_{t \in s^{-1}(V \setminus \{\varepsilon\})} D(t)\,. \tag{17}$$

This definition is used in [35]. If $s^{-1}(V \setminus \{\varepsilon\})$ is order-isomorphic to a down set of $\mathbb{N}$, then the present events of s can be enumerated in the order of their time. If s

[2] The only difference between *zeno* and *dzeno* is the domain of the signal. For *dzeno*, the domain is $[0, 1)$, the left-closed interval between 0 and 1. Given any $t \in [0, 1)$, there are only a finite number of events before tag t, so by lemma 10, *dzeno* is a DE signal.

is present at an infinite number of times, then equation 17 guarantees that for any $t \in \mathrm{dom}(s)$, s is present at a time later than t.

With these lemmas, we have three equivalent definitions of DE signals. Definition 9 states that DE signals can be approximated by "simple" elements of $\mathcal{S}$, the finite signals. Lemma 10 is very useful in proving properties of DE signals. By lemma 11, the present events in a DE signal can be treated as a sequence with increasing time tags.

The following lemma summarizes the properties of $\mathcal{S}_\mathrm{d}$, the set of DE signals.

Lemma 12 *For any totally ordered tag set T,*

(a) $\mathcal{S}_\mathrm{d}$ is a down set of $\mathcal{S}$.
(b) $\mathcal{S}_\mathrm{d}$ with the prefix order is a CPO.
(c) $\mathcal{S}_\mathrm{d}$ is a complete semilattice.

Proof. Part (a) is straightforward, as any prefix of a DE signal is also a DE signal.

Part (b). Let D be a directed set of DE signals from $\mathcal{S}_\mathrm{d}$. As a subset of $\mathcal{S}$, D is also a directed set. Since $\mathcal{S}$ is a CPO, there exists $u \in \mathcal{S}$ such that $u = \bigvee D$ in the CPO $\mathcal{S}$. For all $t \in \mathrm{dom}(u)$, there exists $s \in D$ such that $t \in \mathrm{dom}(s)$.

$$s \sqsubseteq u,\ t \in \mathrm{dom}(s) \implies u \restriction D(t) = s \restriction D(t).$$

$s \restriction D(t)$ is a finite signal, so is $u \restriction D(t)$. u is a DE signal, so D has a least upper bound in $\mathcal{S}_\mathrm{d}$. $\mathcal{S}_\mathrm{d}$ is a CPO.

Part (c). The proof follows directly from the fact that $\mathcal{S}$ is a complete semilattice and part (a) of this lemma. $\square$

Definition 13 (Non-Zeno Signal) *A DE signal $s \in \mathcal{S}_\mathrm{d}$ is non-Zeno if either s is a finite signal, or s is a total signal, $\mathrm{dom}(s) = T$.*

Of the signals in figure 4, $clock_1$ is the only non-Zeno DE signal. The only other DE signal, $dzeno$, is a Zeno signal—it is neither total nor finite. Intuitively, it is Zeno because it is present at an infinite number of times in a strict subset of its tag set. The significance of this is that if the signal is computed by enumerating its present events ordered by time, then any $t \in T \setminus \mathrm{dom}(dzeno)$ cannot be covered in any finite number of computational steps.

Note the role of the tag set T in definition 13. If we change the tag set to $T = [0, 1)$, then the signal

$$([0,1), \{(1 - \frac{1}{2^k}, 1) \mid k \in \mathbb{N}\})$$

is present at the same set of times as $dzeno$, but it is a non-Zeno signal because its tag set T is $[0, 1)$ and it becomes a total signal.

A key property of non-zeno DE signals is that all approximations defined over a subset of T have a finite number of (non-absent) events. This property is extremely helpful when computing the signals in a composition. It means that a computation can successively approximate signals over downsets of T, iteratively increasing these downsets towards the limit of T, and the computation will never have to represent more than a finite number of events. Discrete-event simulators, for example, execute a composition in precisely this manner, by advancing time and representing signals up to the advancing time. This observation motivates the following definitions.

Definition 14 (Non-Zeno closed compositions) *A closed composition is non-Zeno if it has a finite number of behaviors and every signal in every behavior is a non-Zeno DE signal.*

Definition 15 (Non-Zeno open compositions) *An open composition is non-Zeno if given non-Zeno DE signals for inputs it has a finite number of behaviors and every signal in every behavior is a non-Zeno DE signal.*

6.2 Discrete-Event Actors

Definition 16 (Discrete-Event Actor) *A* ***discrete event actor*** *is a function from DE signals to DE signals.*

All input and output signals of a DE actor have the same tag set. Among the actors discussed above, $Delay_d$, $Merge$, and $LookAhead_a$ are DE actors. $MaxMerge$ is not a DE actor, as it has the following behavior,

$$\begin{aligned} s_1 &= ([0,1], \{(1,1)\})\,, \\ s_2 &= dzeno, \\ MaxMerge(s_1, s_2) &= ([0,1], \{(1 - \frac{1}{2^k}, 1) \mid k \in \mathbb{N}\} \cup \{(1,1)\})\,. \end{aligned}$$

$MaxMerge(s_1, s_2)$ is not a DE signal.

Definition 17 (Non-Zeno Actor) *A DE actor $P\colon \mathcal{S}_\mathrm{d} \to \mathcal{S}_\mathrm{d}$ is a* ***non-Zeno actor*** *if for any non-Zeno signal $s \in \mathcal{S}_\mathrm{d}$, $P(s)$ is a non-Zeno signal.*

Such actors are called *simple processes* in [17].

Theorem 18 *A causal DE actor is non-Zeno.*

Proof. Let $P\colon \mathcal{S}_\text{d} \to \mathcal{S}_\text{d}$ be a causal DE process. Let $s \in \mathcal{S}_\text{d}$ be any non-Zeno signal. If s is a total signal, P is causal implies $P(s)$ is a total DE signal. $P(s)$ is non-Zeno.

If s is not a total signal, then s is finite. Let $s' \in \mathcal{S}_\text{d}$ be a total signal such that

$$s'(t) = \begin{cases} s(t) & \text{if } t \in \operatorname{dom}(s), \\ \varepsilon & \text{otherwise.} \end{cases}$$

s' is a total non-Zeno signal, and $s \sqsubseteq s'$. $P(s')$ is a non-Zeno signal. P is causal, so it is monotonic by definition. $P(s) \sqsubseteq P(s')$, so $P(s)$ is non-Zeno. $\square$

6.3 Composition of Discrete-Event Actors

Combining the previous results, from section 4, we know that if all actors in a network of DE actors are continuous, then the network, as a functional actor that maps input signals to the least solution of the network equations, is continuous. The following theorem is proved essentially identically to theorem 8.

Theorem 19 (Causal DE Process Network) *If all actors in a DE actor network are causal and continuous, and in every dependency loop in the network there is at least one strictly causal actor, then the network is causal and continuous.*

Corollary 20 *A DE actor network that satisfies the assumptions of theorem 19 is non-Zeno.*

This corollary follows directly from theorems 19 and 18. Note that unlike [67,66,35] and most other treatments of DE systems, we do not require that two present events in a signal be separated by a minimum time interval, nor that actors be required to introduce a minimum time delay.

7 Conclusion

We have given a domain-theoretic denotational framework for timed interactive systems. We have shown that classical CPO-based techniques determine existence and uniqueness of (least fixed point) solutions, while causality determines liveness. In particular, strict causality, the definition of which does not require a metric space, ensures live feedback loops, which in turn ensures freedom from Zeno conditions. Our approach contrasts with metric space approaches, where contractions and the Banach fixed point theorem ensure existence and uniqueness of fixed points together with liveness. Separating liveness concerns from existence/uniqueness concerns allows us to admit non-causal components. Moreover, our approach does not require a metric space and consequently embraces easily a wide variety of models of time, including super-dense time.

8 Acknowledgements

Thanks to Adam Cataldo, Eleftherios D. Matsikoudis, and the anonymous reviewers for many helpful suggestions.

References

[1] M. Abadi and L. Lamport. An old-fashioned recipe for real time. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(5):1543 – 1571, 1994.

[2] S. Abramsky, S. J. Gay, and R. Nagarajan. Interaction categories and the foundations of typed concurrent programming. In M. Broy, editor, *Deductive Program Design: Proceedings of the 1994 Marktoberdorf Summer School*, NATO ASI Series F. Springer-Verlag, 1995.

[3] S. Abramsky and A. Jung. Domain theory. In *Handbook of logic in computer science (vol. 3): semantic structures*, pages 1–168. Oxford University Press, Oxford, UK, 1995.

[4] G. Agha. Concurrent object-oriented programming. *Communications of the ACM*, 33(9):125–140, 1990.

[5] B. Alpern and F. B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, 1985.

[6] R. Alur and T. Henzinger. Logics and models of real time: A survey. In J. W. De Bakker, C. Huizing, W. P. De Roever, and G. Rozenberg, editors, *REX Workshop*, volume LNCS 600, pages 74–106, Mook, The Netherlands, 1991. Springer.

[7] F. Arbab. Abstract behavior types : A foundation model for components and their composition. *Science of Computer Programming*, 55:3–52, 2005.

[8] J. R. Armstrong and F. G. Gray. *VHDL Design Representation and Synthesis*. Prentice-Hall, second edition, 2000.

[9] A. Arnold and M. Nivat. Metric interpretations of infinite trees and semantics of non deterministic recursive programs. *Fundamenta Informaticae*, 11(2):181–205, 1980.

[10] C. Baier and M. E. Majster-Cederbaum. Denotational semantics in the cpo and metric approach. *Theoretical Computer Science*, 135(2):171–220, 1994.

[11] A. Benveniste and G. Berry. The synchronous approach to reactive and real-time systems. *Proceedings of the IEEE*, 79(9):1270–1282, 1991.

[12] A. Benveniste and P. L. Guernic. Hybrid dynamical systems theory and the signal language. *IEEE Tr. on Automatic Control*, 35(5):525–546, 1990.

[13] G. Berry and G. Gonthier. The esterel synchronous programming language: Design, semantics, implementation. *Science of Computer Programming*, 19(2):87–152, 1992.

[14] M. Broy. Refinement of time. *Theoretical Computer Science*, 253:3–26, 2001.

[15] M. Broy and G. Stefanescu. The algebra of stream processing functions. *Theoretical Computer Science*, 258:99–129, 2001.

[16] C. G. Cassandras. *Discrete Event Systems, Modeling and Performance Analysis*. Irwin, 1993.

[17] A. Cataldo, E. A. Lee, X. Liu, E. Matsikoudis, and H. Zheng. A constructive fixed-point theorem and the feedback semantics of timed systems. In *Workshop on Discrete Event Systems (WODES)*, Ann Arbor, Michigan, 2006.

[18] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Symposium on Principles of Programming Languages (POPL)*, pages 238–252. ACM Press, 1977.

[19] B. A. Davey and H. A. Priestly. *Introduction to Lattices and Order*. Cambridge University Press, 1990.

[20] J. W. de Bakker and E. P. de Vink. Denotational models for programming languages: Applications of Banachs fixed point theorem. *Topology and its Applications*, 85:35–52, 1998.

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

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

[23] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong. Taming heterogeneity—the Ptolemy approach. *Proceedings of the IEEE*, 91(2):127–144, 2003.

[24] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 995–1072. Elsevier Science Publishers (North-Holland), 1990.

[25] G. S. Fishman. *Discrete-Event Simulation: Modeling, Programming, and Analysis*. Springer-Verlag, 2001.

[26] P. Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. Wiley, 2003.

[27] Y. Gurevich. Evolving algebras 1993: Lipari Guide. In E. Börger, editor, *Specification and Validation Methods*, pages 9–37. 1994.

[28] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous data flow programming language LUSTRE. *Proceedings of the IEEE*, 79(9):1305–1319, 1991.

[29] C. Hewitt. Viewing control structures as patterns of passing messages. *Journal of Artifical Intelligence*, 8(3):323363, 1977.

[30] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8), 1978.

[31] G. Kahn. The semantics of a simple language for parallel programming. In *Proc. of the IFIP Congress 74*. North-Holland Publishing Co., 1974.

[32] A. Kapur. *Interval and Point-Based Approaches to Hybrid Systems Verification*. Ph.d., Stanford University, 1997.

[33] D. K. Kaynar, N. Lynch, R. Segala, and F. Vaandrager. *The Theory of Timed I/O Automata*. Synthesis Lectures on Computer Science. Morgan Claypool Publishers, 2006.

[34] L. Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, 1994.

[35] E. A. Lee. Modeling concurrent real-time processes using discrete events. *Annals of Software Engineering*, 7:25–45, 1999.

[36] E. A. Lee. Concurrent semantics without the notions of state or state transitions. In E. Asarin and P. Bouyer, editors, *Proceedings of the International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS)*, volume LNCS 4202, page pp. 1831, Paris, 2006. Springer-Verlag.

[37] E. A. Lee and D. G. Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 1987.

[38] E. A. Lee and A. Sangiovanni-Vincentelli. A framework for comparing models of computation. *IEEE Transactions on Computer-Aided Design of Circuits and Systems*, 17(12):1217–1229, 1998.

[39] E. A. Lee and Y. Xiong. A behavioral type system and its application in Ptolemy II. *Formal Aspects of Computing Journal*, 16(3):210 – 237, 2004.

[40] E. A. Lee and H. Zheng. Operational semantics of hybrid systems. In M. Morari and L. Thiele, editors, *Hybrid Systems: Computation and Control (HSCC)*, volume LNCS 3414, pages pp. 25–53, Zurich, Switzerland, 2005. Springer-Verlag.

[41] X. Liu. Semantic foundation of the tagged signal model. Phd thesis, EECS Department, University of California, December 20 2005.

[42] X. Liu, E. Matsikoudis, and E. A. Lee. Modeling timed concurrent systems. In *CONCUR 2006 - Concurrency Theory*, volume LNCS 4137, Bonn, Germany, 2006. Springer.

[43] N. Lynch, R. Segala, F. Vaandrager, and H. Weinberg. Hybrid I/O automata. In R. Alur, T. Henzinger, and E. Sontag, editors, *Hybrid Systems III*, volume LNCS 1066, pages 496–510. Springer-Verlag, 1996.

[44] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[45] N. A. Lynch and M. R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *ACM Symposium on Principles of Distributed Computing*, pages 137 – 151, Vancouver, British Columbia, Canada, 1987.

[46] O. Maler, Z. Manna, and A. Pnueli. From timed to hybrid systems. In *Real-Time: Theory and Practice, REX Workshop*, pages 447–484. Springer-Verlag, 1992.

[47] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer, Berlin, 1992.

[48] Z. Manna and A. Pnueli. Verifying hybrid systems. *Hybrid Systems*, pages 4–35, 1992.

[49] S. G. Matthews. An extensional treatment of lazy data flow deadlock. *Theoretical Computer Science*, 151(1):195–205, 1995.

[50] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, 1980.

[51] R. Milner. Elements of interaction. *Communications of the ACM*, 36:78–89, 1993.

[52] H. Naundorf. Strictly causal functions have a unique fixed point. *Theoretical Computer Science*, 238(1-2):483–488, 2000.

[53] G. Plotkin. A powerdomain construction. *SIAM Journal on Computing*, 5(3):452–487, 1976.

[54] A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science*, pages 46–57. IEEE Computer Society Press, 1977.

[55] S. Priess-Crampe and P. Ribenboim. Generalized ultrametric spaces I. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 66:55–73, 1996.

[56] S. Priess-Crampe and P. Ribenboim. Fixed point and attractor theorems for ultrametric spaces. *Forum Mathematicum*, 12:53–64, 2000.

[57] G. M. Reed and A. W. Roscoe. Metric spaces as models for real-time concurrency. In *3rd Workshop on Mathematical Foundations of Programming Language Semantics*, pages 331–343, London, UK, 1988.

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

[59] J. J. M. M. Rutten. A coinductive calculus of streams. *Mathematical Structures in Computer Science*, 15(1):93–147, 2005.

[60] J. E. Stoy. *Denotational Semantics*. MIT Press, Cambridge, MA, 1977.

[61] C. L. Talcott. Interaction semantics for components of distributed systems. In *Formal Methods for Open Object-Based Distributed Systems (FMOODS)*, 1996.

[62] P. Taylor. *Practical Foundations of Mathematics*. Cambridge University Press, 1999.

[63] P. Wegner. Interactive foundations of computing. *Theoretical Computer Science*, 192(2):315–351, 1998.

[64] P. Wegner, F. Arbab, D. Goldin, P. McBurney, M. Luck, and D. Roberson. The role of agent interaction in models of computation (panel summary). In *Workshop on Foundations of Interactive Computation*, Edinburgh, 2005.

[65] G. Winskel. *The Formal Semantics of Programming Languages*. MIT Press, Cambridge, MA, USA, 1993.

[66] R. K. Yates. Networks of real-time processes. In E. Best, editor, *Proc. of the 4th Int. Conf. on Concurrency Theory (CONCUR)*, volume LNCS 715. Springer-Verlag, 1993.

[67] R. K. Yates and G. R. Gao. A kahn principle for networks of nonmonotonic real-time processes. In *Parallel Architectures and Languages, Europe*, pages 209–227, 1993.