Semantic Foundation of the Tagged Signal Model

by

Xiaojun Liu

B.Eng. (Tsinghua University) 1994
M.Eng. (Tsinghua University) 1997

A dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy
in

Engineering-Electrical Engineering and Computer Sciences

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:

Professor Edward A. Lee, Chair
Professor Robert K. Brayton
Professor Thomas A. Henzinger
Professor Michael J. Klass

Fall 2005

Semantic Foundation of the Tagged Signal Model

Abstract

Semantic Foundation of the Tagged Signal Model

by

Xiaojun Liu

Doctor of Philosophy in Engineering-Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Edward A. Lee, Chair

The tagged signal model provides a denotational framework to study properties of various models of computation. It is a generalization of the *Signals and Systems* approach to system modeling and specification. Having different models of computation or aspects of them specified in the tagged signal model framework provides the following opportunities. First, one can compare certain properties of the models of computation, such as their notion of synchrony. Such comparisons highlight both the differences and the commonalities among the models of computation. Second, one can define formal relations among signals and process behaviors from different models of computation. These relations have important applications in the specification and design of heterogeneous embedded systems. Third, it facilitates the cross-fertilization of results and proof techniques among models of computation. This opportunity is exploited extensively in this dissertation.

The main goal of this dissertation is to establish a semantic foundation for the tagged signal model. Both order-theoretic and metric-theoretic concepts and approaches are used. The fundamental concepts of the tagged signal model—signals, processes, and networks of processes—are formally defined. From few assumptions on the tag sets of signals, it is shown that the set of all signals with the same partially ordered tag set and the same value set is

a complete partial order. This leads to a direct generalization of Kahn process networks to tagged process networks.

Building on this result, the order-theoretic approach is further applied to study timed process networks, in which all signals share the same totally ordered tag set. The order structure of timed signals provides new characterizations of the common notion of causality and the discreteness of timed signals. Combining the causality and the discreteness conditions is proved to guarantee the non-Zenoness of timed process networks.

The metric structure of tagged signals is studied from the very specific—the Cantor metric and its properties. A generalized ultrametric on tagged signals is proposed, which provides a framework for defining more specialized metrics, such as the extension of the Cantor metric to super-dense time.

The tagged signal model provides not only a framework for studying the denotational semantics of models of computation, but also useful constructs for studying implementations or simulations of tagged processes. This is demonstrated by deriving certain properties of two discrete event simulation strategies from the behavioral specifications of discrete event processes. A formulation of tagged processes as labeled transition systems provides yet another framework for comparing different implementation or simulation strategies for tagged processes. This formulation lays the foundation to future research in polymorphic implementations of tagged processes.

Professor Edward A. Lee
Dissertation Committee Chair

To Yuan, little crawler Ryan, and my parents Zhanshan Liu and Cuilan Liu.

Acknowledgements

I am deeply grateful to my adviser, Professor Edward Lee. His constant support and guidance over the years make the writing of this dissertation possible. His vision of a computational emphasis in electrical engineering education is a prime motivator of my research. I am also very grateful to Professor Robert Brayton, Professor Thomas Henzinger, and Professor Michael Klass for serving on my qualifying exam and dissertation committees, and for providing valuable advice to my research. Professor Henzinger chaired the qualifying exam. Interactions with his research group at Berkeley have been a great help to me, especially with Shaz Qadeer, Sriram Rajamani, Luca de Alfaro, and Benjamin Horowitz.

It is my privilege to work with the past and present members of the Ptolemy group. Many ideas presented in this dissertation become crystallized from discussions with Adam Cataldo, Eleftherios Matsikoudis, Haiyang Zheng, Jörn Janneck, and John Reekie. Jie Liu's research in mixed-signal and hybrid systems and Yuhong Xiong's work on behavioral types are great sources of ideas. Yuhong and Dr. James Rowson at HP Labs provided me with the opportunity to participate in a joint research project. My work has benefited a lot from discussions and collaborations with Elaine Cheong, John Davis II, Bilung Lee, Stephen Neuendorffer, Sonia Sachs, Winthrop Williams, Bicheng Wu, Yang Zhao, Gang Zhou, and Rachel Zhou. I would also like to thank Christopher Brooks and Mary Stewart for their excellent support.

Finally, I want to thank for the constant support and encouragement from my wife Yuan Wang, parents Zhanshan Liu and Cuilan Liu, sister Xiaohong Liu, and my extended family. My son Ryan's laughter gives me no end of inspiration. I am forever indebted to them.

Contents

List of Figures

Chapter 1

Introduction

This dissertation aims to create a semantic foundation for the tagged signal model [44], and to explore the design of computational frameworks that are built on such a foundation. The research is part of the Ptolemy project [17, 46], which studies the modeling, simulation, and design of concurrent, real-time embedded systems.

Many embedded systems are heterogeneous [7, 18, 28, 54]. They may consist of mechanical, hydraulic, optical, and electronic subsystems. An electronic subsystem may have both analog and digital components and embedded application software running on possibly more than one microprocessor. The specification and design of heterogeneous embedded systems call for the use of various models of computation in concert [46, 59]. The tagged signal model is a meta model that serves to

- compare and contrast certain properties of the various models of computation, such as their notion of synchrony;
- relate, or define the interface among, heterogeneously composed multiple models of computation.

The tagged signal model also provides the foundation for a computational view of signals and systems [47].

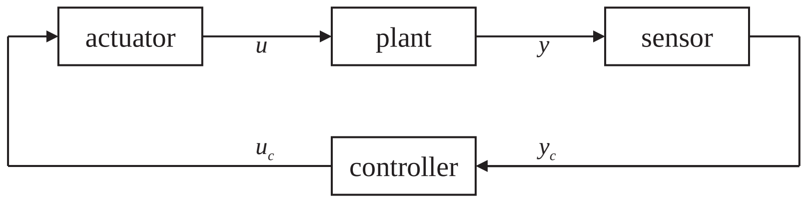

Figure 1.1. Generic block diagram of a feedback control loop.

1.1 Signals and Systems

Signals and Systems is a cornerstone of electrical engineering curricula [11, 64]. Signals carry information among systems that transform or relate signals. Figure 1.1 is a generic block diagram of a feedback control loop. Each block in itself is a system that transforms its input signal to its output signal. For example, the sensor may be a thermometer that converts the signal y, ambient temperature varying over time, to an electric voltage signal y_c. Taken as a whole, the control loop is a system that relates the signals u, y, y_c, and u_c by the simultaneous equations

$$\begin{aligned} y &= \mathrm{plant}(u), \\ y_c &= \mathrm{sensor}(y), \\ u_c &= \mathrm{controller}(y_c), \\ u &= \mathrm{actuator}(u_c), \end{aligned}$$

where plant, sensor, controller, and actuator are functions on signals.

Many physical laws can be modeled as systems like the above. For example, in Newton's Second Law of Motion,

$$F(t) = \mathrm{m}\, a(t),$$

where m is the mass of an object, $F(t)$ is the force applied to the object at time t, and $a(t)$ is the acceleration of the object at time t. As a system, the law relates signals F and a, both of which are functions of time.

An important component of Signals and Systems theory is the mathematical structure on sets of signals. For example, if a set of signals is a *vector space* $\mathcal{V}$ with *basis* $\mathcal{B}$, then any signal in the set is a unique linear combination of the basis signals. Further if $\mathcal{V}$ has a *norm* $\|\cdot\|$, then a notion of approximation can be derived. For any signals $x, y, z \in \mathcal{V}$, y is a better approximation to x than z if and only if

$$\|x - y\| < \|x - z\|.$$

The majority of this dissertation explores the structures on tagged signals that come from order theory [12, 24] and domain theory [2, 32, 72].

1.2 A Computational Perspective

As low cost and high performance microprocessors become widely available, embedded computational systems are now ubiquitous in the living environment. For example, programmable logic controllers are widely adopted in industrial automation. One characteristic of embedded computational systems is their continuous interaction with the physical environment [9]. These systems react to a (conceptually infinite) stream of inputs and generate a (conceptually infinite) stream of outputs.

The need for better understanding and programming paradigms for such systems is recognized in *Structure and Interpretation of Computer Programs* [1], a very influential introductory computer science textbook. Section 3.5 in [1] is on programming with streams, and includes exercises that use streams to simulate RC circuits and to solve differential equations. The computational view is also embraced by Lee and Varaiya in their recent introductory textbook on signals and systems [48].

A theoretical foundation of streams and stream programs is provided by domain theory [2]. Dana Scott pioneered the research in domain theory in search of semantic foundations for programming languages. From his Turing Award lecture [71], the domain of infinite

$$\langle true, false, true, \perp, \dots \rangle$$
$$\vee$$
$$\langle true, false, \perp, \perp, \dots \rangle$$
$$\vee$$
$$\langle true, \perp, \perp, \perp, \dots \rangle$$
$$\vee$$
$$\langle \perp, \perp, \perp, \perp, \dots \rangle$$

Figure 1.2. The information order of the sequences in equation 1.1.

sequences of Boolean values has elements

$$\begin{aligned} &\langle \perp, \perp, \perp, \perp, \dots \rangle, \\ &\langle true, \perp, \perp, \perp, \dots \rangle, \\ &\langle true, false, \perp, \perp, \dots \rangle, \\ &\langle true, false, true, \perp, \dots \rangle. \end{aligned} \tag{1.1}$$

Here the symbol $\perp$ represents undefined. The mathematical structure of the domain is based on a partial order, called the information order, on the sequences. The information order of the sequences in equation 1.1 is illustrated in figure 1.2. The same mathematical concepts and tools are used by Kahn to define the denotational semantics of an elegant model of parallel computation [40].

1.3 The Tagged Signal Model

The tagged signal model [44] provides a framework to formally describe systems of physical processes, computational processes, and their composition. It also provides a meta model to compare and relate the various models of computation that are developed to study these systems.

Specifying a particular model of computation in the tagged signal model framework starts from defining the tag set for signals. As will be formally presented in chapter 2, a

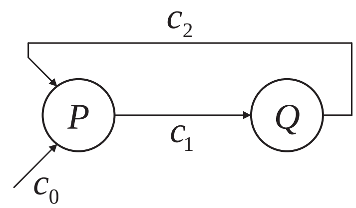

Figure 1.3. A Kahn process network example.

signal is a partial function from its tag set to some set of values. The tag set gives structure to the signal. Answering questions like

What mathematical structure does the tag set have?

Do all signals share the same tag set?

can reveal or formalize many properties of the model of computation.

Consider the Kahn process network (KPN) model of computation [40]. Figure 1.3 illustrates a KPN with two processes P and Q, and three communication channels c_0, c_1, and c_2. Every communication channel connects one producer process to one consumer process. For example, for channel c_1, process P is the producer, and Q is the consumer. For channel c_0, the producer (not shown in the figure) is external to the network, and the consumer is process P. The channels are first-in, first-out (FIFO), and have conceptually infinite capacity.

The communication between the producer and consumer processes of a channel is asynchronous. The producer sends a sequence of data, in units called *tokens*, to the channel. The tokens become available to the consumer after an unpredictable but finite amount of time [40]. To formally specify this asynchrony using the tagged signal model, two signals are associated with every communication channel[1]. As illustrated in figure 1.4, on the producer end, the signal s maps each send action taken by the producer to the token sent. The tag set of signal s, $\{t_k^s \mid k \in \mathbb{N}\}$, is the totally ordered set of send actions. On the consumer end, the signal r maps each receive action taken by the consumer to the token received. The tag set of signal r, $\{t_k^r \mid k \in \mathbb{N}\}$, is the totally ordered set of receive actions. These two tag

[1]This is different from how KPNs are specified in [44]. This alternative aims to make the asynchrony in the communication more explicit.

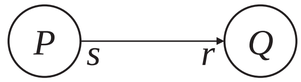

Figure 1.4. The signals associated with a communication channel.

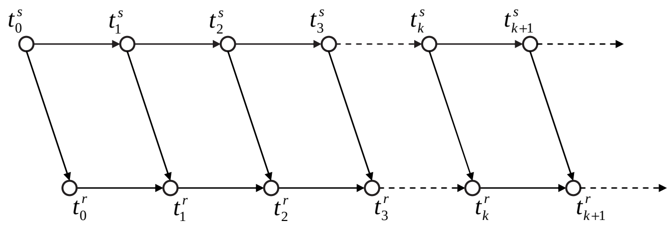

Figure 1.5. The ordering constraints on an asynchronous communication channel.

sets are disjoint. The asynchronous communication over the channel implies the ordering constraints in figure 1.5. Figure 1.6 illustrates the communication ordering constraints in the KPN from figure 1.3.

Every process in a KPN executes a sequential program. Figure 1.7 shows two pseudocode programs. Both programs run in an infinite loop. In each iteration, the process P receives one token each from input signals z and u, and sends their sum to the output signal v. The `receive` action on an input signal will not complete until a token becomes available from the signal, whereas the `send` action on an output signal can always complete without waiting, because the communication channels have infinite capacity.

Every process in a KPN imposes a total order on the tags of its input and output signals. For the processes in figure 1.7, the orders are shown in figure 1.8.

For the KPN in figure 1.3, the complete ordering constraints on the signal tags are the composition of those shown in figures 1.6 and 1.8. The combined constraints have directed loops, such as

$$t_0^z \to t_0^u \to t_0^v \to t_0^x \to t_0^y \to t_0^z.$$

Such a dependency loop implies that the processes P and Q will run into a deadlock, each waiting for a token from the other in order to proceed.

The KPN model of computation imposes few ordering constraints on signals, as illus-

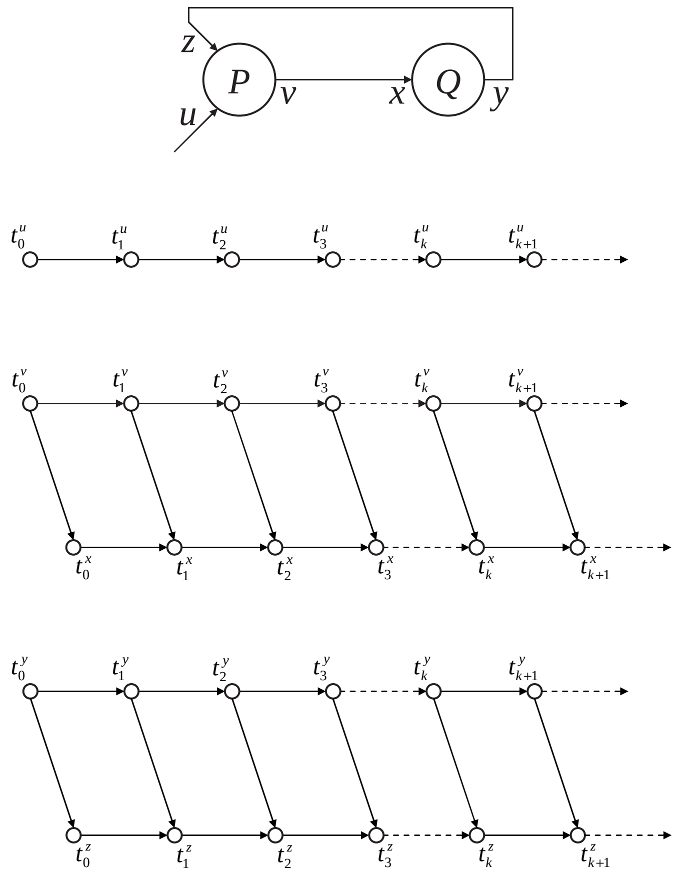

Figure 1.6. Communication ordering constraints in a KPN.

```
Process P(in z, u; out v)
{
  repeat {
    t1 = receive(z)
    t2 = receive(u)
    send(v, t1 + t2)
  }
}
```

```
Process Q(in x; out y)
{
  repeat {
    t = receive(x)
    send(y, t)
  }
}
```

Figure 1.7. Sequential programs of Kahn processes.

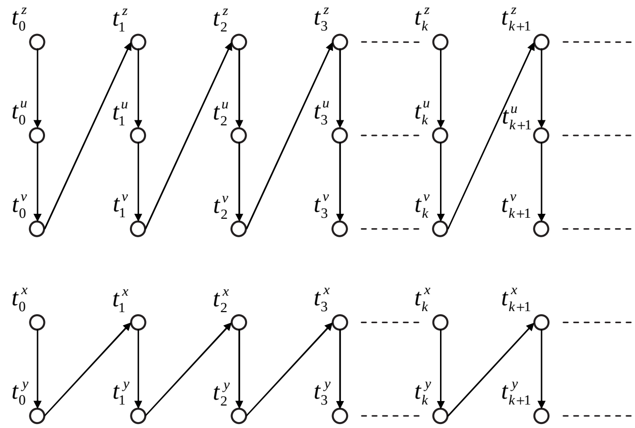

Figure 1.8. The total orders imposed on tags by the Kahn processes from figure 1.7.

```
Process P(in z, u; out v)
{
  fire() {
    t1 = receive(z)
    t2 = receive(u)
    send(v, t1 + t2)
  }
}
```

```
Process Q(in x; out y)
{
  fire() {
    t = receive(x)
    send(y, t)
  }
}
```

Figure 1.9. Processes from figure 1.7 rewritten as SDF processes.

trated by figure 1.6. Properties such as the absence of deadlock cannot be decided without analyzing the behavior or program of the processes. There are many specializations of the KPN model of computation, such as *synchronous dataflow* (SDF) [42], that impose stronger ordering constraints on signals. An SDF process executes a sequence of *firings*. In each firing, the process consumes a fixed number of tokens from its input signals and produces a fixed number of tokens in its output signals. Figure 1.9 shows the processes P and Q from figure 1.7 rewritten as SDF processes. Both processes P and Q consume one token from each input and produce one token in each output per firing. These constraints on token consumption and production imply essentially the same tag ordering constraints as those shown in figure 1.8. For the SDF model of computation, it is possible to check statically whether a network of processes will deadlock.

The motivation of these examples is to show the use of the tagged signal model to elicit the properties of a model of computation—what is the structure of signals, what are the constraints among signals, and so on. These properties determine the mathematical structures on sets of signals. Studying such structures is the theme of this dissertation.

1.4 Overview of the Dissertation

Chapter 2 presents the fundamental concepts of the tagged signal model—signals, processes, and networks of processes. The order structure of signal sets is explored, and is used to characterize the processes that are relations or functions among signal sets. The order structure leads to a generalization of Kahn process networks to tagged process networks.

Chapter 3 explores the implications of assuming that all signals in a network of processes have the same totally ordered tag set. Such tag sets make it possible to formally define the common notion of causality—the output of a process cannot change before its input changes. Conditions that guarantee the causality of a process network are proposed. The second half of chapter 3 studies a further assumption—the discreteness of timed signals—that originates from the need to enumerate the events in a timed process network in computer simulations. To illustrate the definitions and proof techniques, a number of common processes on timed signals are formally defined, and some proofs of their properties are given.

Chapter 4 studies the metric structure of tagged signals. The *Cantor metric* and its properties are reviewed. The metric-theoretic and order-theoretic notions of convergence and finite approximation are compared. Through analyzing an extension of the Cantor metric to the super-dense time, a generalized ultrametric on tagged signals is proposed.

Chapter 5 compares two discrete event simulation strategies. A framework for such comparisons is proposed. Two issues in discrete event simulation—handling dependency loops and advancing simulation time—are formally analyzed. Examples of using the formal results in previous chapters to prove properties of the simulation strategies are presented.

The concluding chapter summarizes the main results and contributions of this dissertation. Some future work directions are discussed, with the hope that this dissertation may serve as a solid foundation.

Chapter 2

Tagged Process Networks

This chapter presents the fundamental concepts in the tagged signal model—signals, processes, and networks of processes—and studies their properties.

2.1 Signals

The definition of a signal and the mathematical structure of signal sets build on the theory of partially ordered sets. Relevant mathematical definitions and results will be included before they are first used.

2.1.1 Partially Ordered Sets and Lattices

Definition 2.1 (Partial Order). Let P be a set. A binary relation $\leq$ on P is a partial order if for all $x, y, z \in P$,

$$\begin{aligned} & x \leq x, && \text{(reflexivity)} \\ & x \leq y \ \text{ and } \ y \leq x \ \text{ imply } \ x = y, && \text{(antisymmetry)} \\ & x \leq y \ \text{ and } \ y \leq z \ \text{ imply } \ x \leq z. && \text{(transitivity)} \end{aligned} \tag{2.1}$$

A set P with a partial order $\leq$ on P is a **partially ordered set** (**poset**).

Notation. To explicitly specify the partial order $\leq$ on a poset P, use $(P, \leq)$.

Definition 2.2. Let S be a subset of a poset P. An element $x \in P$ is a **lower bound** of S if $x \leq s$ for all $s \in S$. x is the **greatest lower bound** of S if x is a lower bound of S and $y \leq x$ for all lower bound y of S. **Upper bound** and **least upper bound** are defined dually.

Notation. $\bigwedge S$ denotes the greatest lower bound of S if it exists, and $\bigvee S$ the least upper bound. $x \wedge y$ and $x \vee y$ are alternative notations of $\bigwedge\{x, y\}$ and $\bigvee\{x, y\}$, respectively.

Definition 2.3 (Lattice). Let $(P, \leq)$ be a non-empty poset.

- If $x \wedge y$ and $x \vee y$ exist for all $x, y \in P$, $(P, \leq)$ is a **lattice**.
- If $\bigwedge S$ and $\bigvee S$ exist for all subset S of P, $(P, \leq)$ is a **complete lattice**.
- If $\bigwedge S$ exists for all non-empty subset S of P, $(P, \leq)$ is a **complete lower semilattice**.

Definition 2.4 (Down-Set). Let P be a poset. A subset S of P is a down-set if for all $x \in S$ and $y \in P$, $y \leq x$ implies $y \in S$.

Lemma 2.5. Let $\mathcal{D}(P)$ be the family of all down-sets of a poset P.

(a) $\mathcal{D}(P)$ is closed under arbitrary union and intersection.

(b) $\mathcal{D}(P)$ with the set inclusion order is a poset $(\mathcal{D}(P), \subseteq)$.

(c) $(\mathcal{D}(P), \subseteq)$ is a complete lattice.

(d) Let $D \subseteq \mathcal{D}(P)$,

$$\bigvee D = \bigcup_{S \in D} S, \quad \bigwedge D = \bigcap_{S \in D} S.$$

2.1.2 Signals

In the tagged signal model, a signal represents the flow of information between physical or computational processes.

Notation. Let X and Y be two sets and $f\colon X \rightharpoonup Y$ a partial function from X to Y.

- For a set $B \subseteq Y$, $f^{-1}(B)$ denotes the *preimage* of B under f,

$$f^{-1}(B) = \{x \in X \mid f(x) \in B\}.$$

- $\mathrm{dom}(f)$ denotes $f^{-1}(Y)$, the subset of X on which the partial function f is defined.

Definition 2.6 (Signal). Let T 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. $\mathcal{S}(T, V)$ denotes the set of all signals with tag set T and value set V.

Definition 2.7 (Event). Let $s \in \mathcal{S}(T, V)$. An element $(t, v) \in T \times V$ is an event of s if $t \in \mathrm{dom}(s)$ and $v = s(t)$.

Notation. Let $e = (t_e, v_e)$ be an event of a signal $s \in \mathcal{S}(T, V)$.

- $\mathrm{tag}(e)$ denote the tag of e, $\mathrm{tag}(e) = t_e$.

- $\mathrm{val}(e)$ denote the value of e, $\mathrm{val}(e) = v_e$.

- $\mathrm{events}(s)$ denote the set of events of s, $\mathrm{events}(s) = \{(t, v) \mid t \in \mathrm{dom}(s) \ \text{ and } \ v = s(t)\}$.

The partial order on the tag set T of a signal $s \in \mathcal{S}(T, V)$ specifies the ordering of events. The event ordering may derive from the timing of events, as in discrete event systems—the tag of an event is its time stamp. Another example is the activation ordering in the actor model [35]. This ordering captures the causal relation between events. Requiring that $\mathrm{dom}(s)$ be a down-set implies that if s has an event e, it has events at all tags $t \leq \mathrm{tag}(e)$. If a signal is defined at tag t, then it is defined at all tags that "come before" t.

Remark 2.8. The signal definition 2.6 is different from that in [44], in which Lee and Sangiovanni-Vincentelli first proposed the tagged signal model. In [44], a signal is a set of events, or equivalently, a subset of $T \times V$. The tag set T is not required to be a poset. When a signal is a *functional signal* or *proper signal* (section II.A of [44]), and T is a poset, it is not required that the subset of T on which the signal is defined is a down-set of T. Any such signal can be matched to a signal by definition 2.6 as follows.

- An arbitrary tag set T can be treated as a poset $(T, =)$, so no generality is lost by requiring T to be a poset in definition 2.6.

- A signal, when defined as a subset of $T \times V$, can have more than one event with the same tag. This is useful in modeling nondeterministic computation. Let $\mathcal{P}(V)$ denote the power set of V. Given $r \subseteq T \times V$, a corresponding $s \in \mathcal{S}((T, =), \mathcal{P}(V))$ can be obtained by letting

$$\begin{aligned} \mathrm{dom}(s) &= \{t \in T \mid \exists v \in V,\ (t, v) \in r\}, \\ s(t) &= \{v \in V \mid (t, v) \in r\},\ \forall t \in \mathrm{dom}(s). \end{aligned} \tag{2.2}$$

  Again no generality is lost by requiring signals to be functional in definition 2.6.

- When the tag set T is a poset, a functional signal in [44] may be defined on a subset of T that is not a down-set. In such cases, the construction given by equation 2.2 is still valid—with the caveat that T assumes the discrete order in the construction.

Example (Tag Sets and Signals). Here are some applications of definition 2.6 to concepts from mathematics, computer science, and electrical engineering.

- *Partial functions.* Let $(X \rightharpoonup Y)$ denote the set of all partial functions from X to Y. X can be treated as a poset $(X, =)$. This partial order is called the *discrete order* on X. With this order, every subset of X is a down-set, so $\mathcal{D}(X)$ is the same as $\mathcal{P}(X)$. Every partial function $f \in (X \rightharpoonup Y)$ is a signal $f \in \mathcal{S}(X, Y)$, so $\mathcal{S}(X, Y)$ is the same as $(X \rightharpoonup Y)$.

- *Streams.* A stream is a finite or infinite sequence of values. The tag set of a stream s is $\{t^s_k \mid k \in \mathbb{N}\}$ with the ordering $t^s_i \leq t^s_j$ for all $i, j \in \mathbb{N}$ such that $i \leq j$. Figure 2.1 illustrates a stream s and its tag set.

  The events of a stream are totally ordered. Figure 2.2 shows the tag set of a signal a consisting of two asynchronous streams r and s. Two tags from different streams are not comparable. The tag set does not have a least element, so the question "What is the first event of a?" has no answer.

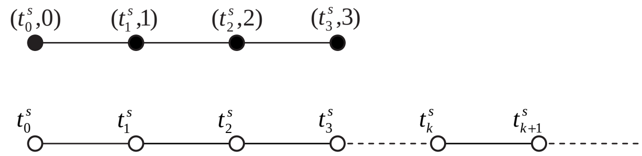

Figure 2.1. A stream s and its tag set. $\mathrm{dom}(s) = \{t_0^s, t_1^s, t_2^s, t_3^s\}$, a down-set of $\{t_k^s \mid k \in \mathbb{N}\}$. This is a *Hasse diagram* with a small variation. Here $t_i^s \leq t_j^s$ if and only if there is a left-to-right path from t_i^s to t_j^s, instead of bottom-up as Hasse diagrams are usually drawn. Circles represent tags, and dots represent events.

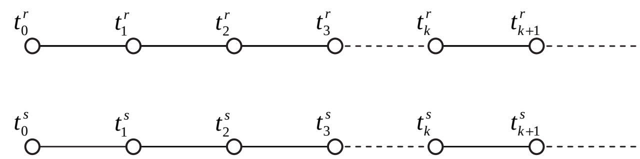

Figure 2.2. The tag set of a signal a consisting of two asynchronous streams r and s.

- *Discrete time signals.* Consider a signal generated by sampling an audio input with interval $h \in \mathbb{R}$. The tag set for such a discrete time signal is $\{kh \mid k \in \mathbb{N}\}$. This tag set is order-isomorphic to that of a stream, but the values of the tags are relevant when such signals are processed.

2.2 The Prefix Order of Signals

Definition 2.9 (Prefix Order). Let $s_1, s_2 \in \mathcal{S}(T, V)$. s_1 is a prefix of s_2, denoted by $s_1 \preceq s_2$, if and only if

$$\mathrm{dom}(s_1) \subseteq \mathrm{dom}(s_2),$$

$$s_1(t) = s_2(t),\ \forall t \in \mathrm{dom}(s_1).$$

The prefix order on signals is a natural generalization of the prefix order on strings or sequences, and the extension order on partial functions [75].

Example (Prefix Order).

- *Partial functions.* Let $f_1, f_2 \in (X \rightharpoonup Y)$. Considered as signals, $f_1 \preceq f_2$ if and only

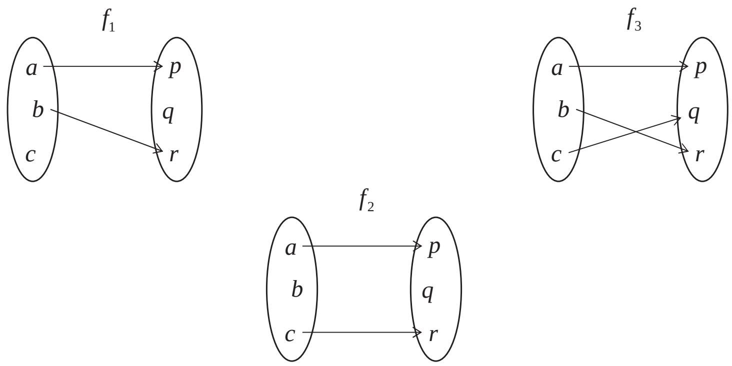

Figure 2.3. The prefix order on partial functions as signals.
$f_1, f_2, f_3 \in (\{a,b,c\} \rightharpoonup \{p,q,r\})$. $f_1 \not\preceq f_2$, $f_1 \preceq f_3$, and $f_2 \not\preceq f_3$.

if f_2 is defined and equal to f_1 everywhere f_1 is defined. The prefix order on partial functions coincides with the extension order. Figure 2.3 illustrates the prefix order on partial functions.

- *Streams*. For two streams s_1 and s_2, $s_1 \preceq s_2$ if s_2 equals s_1 or s_2 can be obtained by appending more values to the sequence of values of s_1. The prefix order on streams is similar to that on strings, which can also be defined as signals. Let Char denote the character set. The set of strings, finite and infinite, is the set of signals $\mathcal{S}(\mathbb{N}, \text{Char})$. (A common notation for such a set is Char**, [24].) Figure 2.4 illustrates the prefix order on signals consisting of two asynchronous streams r and s.

The following lemma characterizes the prefix order in terms of the events of signals, and can be proved easily from definition 2.9.

Lemma 2.10. For any signals $s_1, s_2 \in \mathcal{S}(T, V)$,

$$s_1 \preceq s_2 \iff \text{events}(s_1) \subseteq \text{events}(s_2).$$

2.3 The Order Structure of Signals

The prefix order is a partial order on signals. This section develops the mathematical structure of signal sets as ordered sets.

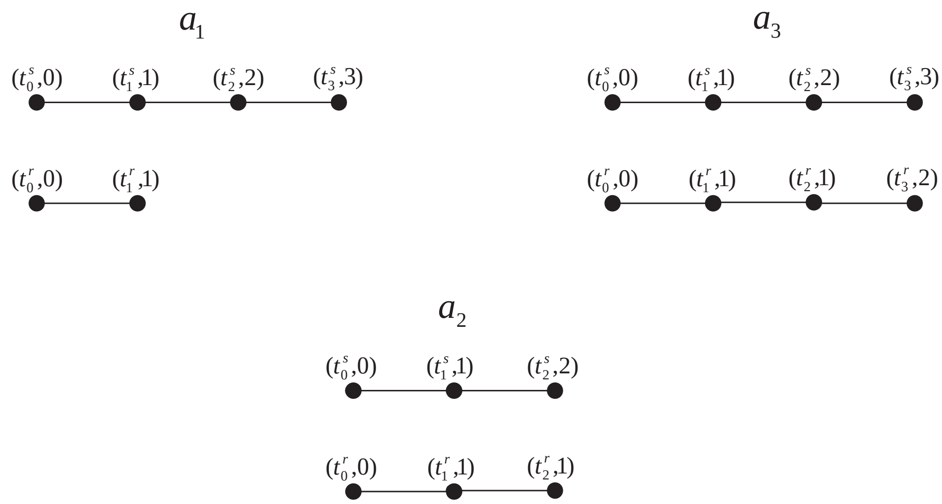

Figure 2.4. The prefix order on signals consisting of two asynchronous streams r and s. $a_1 \not\preceq a_2$, $a_1 \preceq a_3$, and $a_2 \preceq a_3$.

Lemma 2.11 (Poset of Signals). For any poset of tags T and set of values V, the set of signals $\mathcal{S}(T,V)$ with the prefix order $\preceq$ is a poset.

The proof of this lemma is straightforward by verifying that the relation $\preceq$ is reflexive, antisymmetric, and transitive.

Remark 2.12. The poset $\mathcal{S}(T,V)$ has a least element $s_\perp \colon \emptyset \to V$. $s_\perp$ has no events and is called the **empty signal**. If a signal is defined for all tags in T, it is a maximal element of $\mathcal{S}(T,V)$, and is called a **total signal**. Let $\mathcal{S}_\mathrm{t}(T,V)$ denote the set of all total signals in $\mathcal{S}(T,V)$.

Complete posets (CPOs) are an important class of posets used extensively in studying the denotational semantics of programming languages. CPOs are also used in defining the denotational semantics of Kahn process networks in [40]. A generalization of that work will be presented later in this chapter.

Definition 2.13 (Directed Set). Let P be a poset. A subset S of P is directed if for all $x,y \in S$, there exists $z \in S$ such that $x \leq z$ and $y \leq z$, or equivalently z is an upper bound of $\{x,y\}$.

If a set of signals is directed, then for any tag t, all of the signals in the set that are defined at that tag agree on the value. If two signals r and s have an upper bound u, then $\{r, s, u\}$ is a directed set. For all $t \in \text{dom}(r) \cap \text{dom}(s)$, both $r(t)$ and $s(t)$ equal $u(t)$. There is no conflict when both r and s are defined. These observations are formalized in the following lemma.

Lemma 2.14. Let $S \subseteq \mathcal{S}(T, V)$ be a directed subset of signals, and $s \in S$. For all $t \in \text{dom}(s)$ and $r \in S$ such that $t \in \text{dom}(r)$, $r(t)$ equals $s(t)$.

Definition 2.15 (CPO). A poset P is a CPO if P has a least element $\bot$, and every directed subset D of P has a least upper bound.

Lemma 2.16 (CPO of Signals). For any poset of tags T and set of values V, the poset of signals $(\mathcal{S}(T, V), \preceq)$ is a CPO.

Proof. $\mathcal{S}(T, V)$ has the least element $s_\bot$.

Let S be any directed subset of $\mathcal{S}(T, V)$. For all $s \in S$, $\text{dom}(s)$ is a down-set of T. By lemma 2.5, their union

$$D = \bigcup_{s \in S} \text{dom}(s) \tag{2.3}$$

is a down-set of T. Define a signal $r \in \mathcal{S}(T, V)$ such that $\text{dom}(r)$ is D. For each $t \in D$, there exists $s_t \in S$ such that $t \in \text{dom}(s_t)$. Let

$$r(t) = s_t(t).$$

By lemma 2.14, r is well defined.

By its definition, it is clear that r is an upper bound of S. Let u be any upper bound of S,

$$\begin{aligned}
\forall s \in S, s \preceq u &\implies \forall s \in S, \text{dom}(s) \subseteq \text{dom}(u), \\
&\implies \bigcup_{s \in S} \text{dom}(s) \subseteq \text{dom}(u), \\
&\implies \text{dom}(r) \subseteq \text{dom}(u),
\end{aligned}$$

$$\forall t \in \text{dom}(r), \quad \begin{array}{l} r(t) = s_t(t), \\ s_t \preceq u \Rightarrow s_t(t) = u(t), \end{array} \quad \implies r(t) = u(t).$$

r is a prefix of u. r is the least upper bound of S.

Any directed subset S of $\mathcal{S}(T,V)$ has a least upper bound, $\mathcal{S}(T,V)$ is a CPO. $\square$

Lemma 2.17. For any poset of tags T and set of values V, the poset of signals $(\mathcal{S}(T,V),\preceq)$ is a complete lower semilattice.

Proof. By definition, if all non-empty subsets of a poset have a greatest lower bound, then it is a complete lower semilattice.

Let S be any non-empty subset of $\mathcal{S}(T,V)$. Let

$$E = \{t \in \bigcap_{s\in S} \mathrm{dom}(s) \,|\, \forall r, s \in S,\, r(t) = s(t)\}\,,$$

and

$$D = \bigcup_{A\in\mathcal{D}(T) \text{ and } A\subseteq E} A\,. \tag{2.4}$$

D is a subset of E, and by lemma 2.5, D is a down-set of T. Take any signal $r_0 \in S$ and define a signal g such that

$$\mathrm{dom}(g) = D,$$
$$g(t) = r_0(t),\ \forall t \in D.$$

For all $s \in S$, $\mathrm{dom}(g)$ is a subset of $\mathrm{dom}(s)$, and

$$g(t) = r_0(t) = s(t),\ \forall t \in \mathrm{dom}(g),$$

so $g \preceq s$. g is a lower bound of S.

For any lower bound l of S,

$$\forall s \in S,\ l \preceq s \implies \forall s \in S,\ \forall t \in \mathrm{dom}(l),\ s(t) = l(t),$$
$$\implies \mathrm{dom}(l) \subseteq E.$$

By equation 2.4 and $\mathrm{dom}(l) \in \mathcal{D}(T)$, $\mathrm{dom}(l) \subseteq \mathrm{dom}(g)$. For all $t \in \mathrm{dom}(l)$,

$$l(t) = r_0(t) = g(t),$$

so $l \preceq g$. g is the greatest lower bound of S. $\square$

The partial order $\preceq$ can be extended to signal tuples,

$$\forall (s_1, \ldots, s_n), (r_1, \ldots, r_n) \in \mathcal{S}(T_1, V_1) \times \cdots \times \mathcal{S}(T_n, V_n)$$
$$(s_1, \ldots, s_n) \preceq (r_1, \ldots, r_n) \iff s_i \preceq r_i, i = 1, \ldots, n.$$

Lemma 2.18 (Signal Tuples). For any tag sets $T_i, i = 1, \ldots, n$ and value sets $V_i, i = 1, \ldots, n$, the set of signal tuples $\mathcal{S}(T_1, V_1) \times \cdots \times \mathcal{S}(T_n, V_n)$ with the prefix order $\preceq$ is a poset, a CPO, and a complete lower semilattice.

Notation. For a signal tuple $s = (s_1, \ldots, s_n)$ and an index tuple $I = (i_1, \ldots, i_k)$,

$$s|_I = (s_{i_1}, \ldots, s_{i_k}).$$

For example, $(s_1, s_2, s_3)|_{(3,1)} = (s_3, s_1)$. For a set S of signal tuples,

$$S|_I = \{s|_I \mid s \in S\}.$$

2.4 Signal Segments

For two different signals $r, s \in \mathcal{S}(T, V)$ such that $r \preceq s$, s can be obtained by appending their difference to r.

Definition 2.19 (Signal Difference). The difference, $s \setminus r$, of two signals $r, s \in \mathcal{S}(T, V)$ is a partial function from T to V, with

$$\mathrm{dom}(s \setminus r) = \mathrm{dom}(s) \setminus \mathrm{dom}(r)\,,$$
$$(s \setminus r)(t) = s(t),\ \forall t \in \mathrm{dom}(s \setminus r).$$

There is an alternative way to define the difference between signals. It is based on generalizing the concept of an interval on the real line to arbitrary posets.

Definition 2.20 (Interval of a Poset). Let P be a poset and $I \subseteq P$. I is an interval of P if for all $a, b \in I$ and $c \in P$, $a < c$ and $c < b$ imply $c \in I$.

Let $\mathcal{I}(P)$ be the family of all intervals of poset P. Every down-set $D \in \mathcal{D}(P)$ is an interval of P. Down-sets are also called initial segments in the literature [34].

Notation. For any subset A of a poset P, the *down-closure* of A is

$$\downarrow A = \{x \in P \,|\, \exists a \in A,\; x \leq a\}. \tag{2.5}$$

$\downarrow A$ is a down-set of P.

Lemma 2.21. Let P be a poset and $I \in \mathcal{I}(P)$. The set difference $\downarrow I \setminus I$ is a down-set of P.

Proof. Take any $x \in \downarrow I \setminus I$ and $y \in P$ such that $y \leq x$. $x \in \downarrow I$ and $\downarrow I$ is a down-set imply $y \in \downarrow I$. If $y \notin \downarrow I \setminus I$, then $y \in I$. $x \in \downarrow I$ so there exists $z \in I$ such that $x \leq z$. Both y and z are in I, and $y \leq x$, $x \leq z$. But $x \in \downarrow I \setminus I$ implies that $x \notin I$, which contradicts that I is an interval. It must be that $y \in \downarrow I \setminus I$. □

Lemma 2.22. For any poset P,

$$\mathcal{I}(P) = \{E \setminus D \,|\, D, E \in \mathcal{D}(P),\, D \subseteq E\}.$$

That is, every interval of P is expressible as the difference between two down-sets.

Proof. Take any $I \in \mathcal{I}(P)$. $\downarrow I \in \mathcal{D}(P)$, and by lemma 2.21, $\downarrow I \setminus I \in \mathcal{D}(P)$.

$$\forall I \in \mathcal{I}(P), I = \downarrow I \setminus (\downarrow I \setminus I) \implies \mathcal{I}(P) \subseteq \{E \setminus D \,|\, D, E \in \mathcal{D}(P),\, D \subseteq E\}.$$

Take any $D, E \in \mathcal{D}(P)$ such that $D \subseteq E$. For any $a, b \in E \setminus D$ and $c \in P$ such that $a \leq c$ and $c \leq b$, if $c \notin E \setminus D$, then

$$\begin{aligned} c \leq b,\, b \in E,\, E \in \mathcal{D}(P) &\implies c \in E, \\ c \in E,\, c \notin E \setminus D &\implies c \in D, \\ a \leq c,\, c \in D,\, D \in \mathcal{D}(P) &\implies a \in D. \end{aligned}$$

But $a \in E \setminus D$, a contradiction. It must be that $c \in E \setminus D$, so $E \setminus D \in \mathcal{I}(P)$. □

Definition 2.23 (Signal Segment). Let T be a poset of tags and V a set of values. A signal segment g is a partial function from T to V such that $\mathrm{dom}(g) \in \mathcal{I}(T)$.

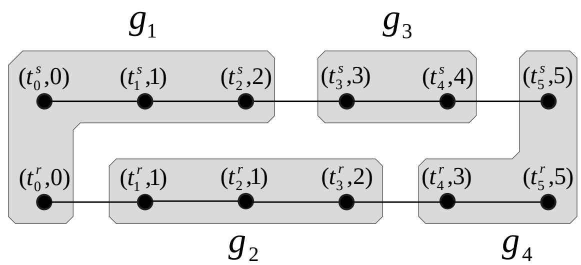

Figure 2.5. Segments of a signal consisting of two asynchronous streams r and s.

Let $\mathcal{G}(T, V)$ denote the set of all signal segments with tag set T and value set V. Every signal $s \in \mathcal{S}(T, V)$ is a signal segment, $\mathcal{S}(T, V) \subseteq \mathcal{G}(T, V)$. An event of a segment is defined the same way as an event of a signal. Figure 2.5 illustrates some segments of a signal consisting of two asynchronous streams.

By the definitions of signal difference and segment, and lemma 2.22, it is easy to establish their equivalence,

$$\mathcal{G}(T, V) = \{s \setminus r \,|\, r, s \in \mathcal{S}(T, V)\}. \tag{2.6}$$

That is, every segment in $\mathcal{G}(T, V)$ can be obtained as the difference between two signals from $\mathcal{S}(T, V)$.

Definition 2.24 (Append). For a segment $g \in \mathcal{G}(T, V)$ and a signal $s \in \mathcal{S}(T, V)$, if $\mathrm{dom}(g) \cap \mathrm{dom}(s) = \emptyset$ and $\mathrm{dom}(g) \cup \mathrm{dom}(s) \in \mathcal{D}(T)$, then a new signal, denoted by $s \ll g$, can be obtained by appending g to s, such that

$$\mathrm{dom}(s \ll g) = \mathrm{dom}(s) \cup \mathrm{dom}(g),$$

$$(s \ll g)(t) = \begin{cases} s(t) & \text{if } t \in \mathrm{dom}(s), \\ g(t) & \text{if } t \in \mathrm{dom}(g). \end{cases}$$

In figure 2.5, segment g_1 is also a signal. $g_1 \ll g_2$ and $(g_1 \ll g_2) \ll g_3$ are signals, but $(g_1 \ll g_2) \ll g_4$ is undefined.

Given a set of signals $\{s_i, i = 0, \dots, n\}$ such that $s_{i-1} \preceq s_i$ for all $i \in \{1, \dots, n\}$, it is easy to verify that

$$s_n = s_0 \ll (s_1 \setminus s_0) \ll (s_2 \setminus s_1) \ll \cdots \ll (s_n \setminus s_{n-1}). \tag{2.7}$$

$\ll$ is left associative when interpreting the above equation.

For a signal $s \in \mathcal{S}(T, V)$, let $\mathcal{F}(s)$ be the set of all segments that can be appended to s (the "futures" of s),

$$\mathcal{F}(s) = \{s' \setminus s \mid s' \in \mathcal{S}(T, V),\ s \preceq s'\}. \tag{2.8}$$

Notation. For a partial function $f\colon A \rightharpoonup B$ and $C \subseteq A$, the **restriction** of f to C, $f\downarrow_C$, is a partial function from C to B such that

$$\mathrm{dom}(f\downarrow_C) = C \cap \mathrm{dom}(f),$$
$$(f\downarrow_C)(c) = f(c),\ \forall c \in \mathrm{dom}(f\downarrow_C).$$

For a set of partial functions $F \subseteq (A \rightharpoonup B)$,

$$F\downarrow_C = \{f\downarrow_C \mid f \in F\}.$$

Lemma 2.25. For any signal $s \in \mathcal{S}(T, V)$,

$$\mathcal{F}(s)\downarrow_{T\setminus\mathrm{dom}(s)} = \mathcal{S}(T \setminus \mathrm{dom}(s), V). \tag{2.9}$$

That is, the futures of a signal, when restricted to the future tags, are the signals having the future tags as the tag set.

Proof. Every segment $g \in \mathcal{F}(s)$ has as domain an interval $I \in \mathcal{I}(T)$ such that $I \cap \mathrm{dom}(s) = \emptyset$ and $I \cup \mathrm{dom}(s) \in \mathcal{D}(T)$. For all $x \in I$ and $y \in T \setminus \mathrm{dom}(s)$ such that $y \leq x$,

$$x \in I \cup \mathrm{dom}(s) \ \text{ and } \ I \cup \mathrm{dom}(s) \in \mathcal{D}(T) \implies y \in I \cup \mathrm{dom}(s),$$
$$y \in I \cup \mathrm{dom}(s) \ \text{ and } \ y \in T \setminus \mathrm{dom}(s) \implies y \in I.$$

I is a down-set of $T \setminus \mathrm{dom}(s)$, so $g\downarrow_{T\setminus\mathrm{dom}(s)} \in \mathcal{S}(T \setminus \mathrm{dom}(s), V)$.

Any signal $r \in \mathcal{S}(T \setminus \mathrm{dom}(s), V)$ has as domain a down-set D of $T \setminus \mathrm{dom}(s)$. Clearly $D \cap \mathrm{dom}(s) = \emptyset$. For any $x \in D \cup \mathrm{dom}(s)$ and $y \in T$ such that $y \leq x$, if $x \in \mathrm{dom}(s)$, a down-set of T, $y \in \mathrm{dom}(s)$. If $x \in D$ but $y \notin \mathrm{dom}(s)$, $D \in \mathcal{D}(T \setminus \mathrm{dom}(s))$ implies $y \in D$. $D \cup \mathrm{dom}(s)$ is a down-set of T, so $r \in \mathcal{F}(s)\downarrow_{T\setminus\mathrm{dom}(s)}$. $\square$

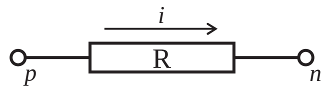

Figure 2.6. An ideal resistor and its electrical signals. R is the resistance. p and n are the voltage signals at its two terminals, and i is the current flow through the resistor.

2.5 Processes

The process definition in the tagged signal model [44] is applicable to both physical and computational processes. Physical laws specify the behavior of physical processes by relating physical quantities measured over time and space. Figure 2.6 shows an ideal resistor with resistance R and its electrical signals. Let $[t_1, t_2]$ be a time interval over which the signals are defined. With $[t_1, t_2]$ as the tag set and $\mathbb{R}$ as the value set[1], p, n, and i are elements of $\mathcal{S}_{\mathrm{t}}([t_1, t_2], \mathbb{R})$. By the Ohm's Law, the signals satisfy the following equation:

$$p(t) - n(t) = \mathrm{R}\, i(t),\ \forall t \in [t_1, t_2]. \tag{2.10}$$

This law specifies a process by the following definition, which derives from both [44] and [10].

Definition 2.26 (Process). A process P is a tuple (n, S, B) where

- n is the arity of the process. It is the number of signals related by the process.
- $S = \mathcal{S}(T_1, V_1) \times \cdots \times \mathcal{S}(T_n, V_n)$ is the signature of the process. T_i is the tag set of the ith signal, and V_i is its value set.
- $B \subseteq \mathcal{S}(T_1, V_1) \times \cdots \times \mathcal{S}(T_n, V_n)$ is the behavior set of the process.

For a process P with behavior set B and arity n, if a tuple of n signals $(s_1, \ldots, s_n)$ is an element of B, then $(s_1, \ldots, s_n)$ is a **behavior** of P. The above definition can be easily adapted to consider only total signals when specifying the signature and behavior set of a process, as illustrated by the following example.

[1] $\mathbb{R}$, the real numbers, and $\mathbb{N}$, the natural numbers including 0, are used as value sets in examples. Properties of value sets, such as data types and physical units, are not considered, as the focus here is on the mathematical structure of signals derived from their tag sets.

Example. For the ideal resistor in figure 2.6, a process R that captures its behavior is $(3, S, B)$ where

- The process relates 3 signals. Its arity is 3. Let the indexes 1, 2, and 3 correspond to p, n, and i respectively.
- Let I denote the interval $[t_1, t_2]$. The signals have the same tag set I and value set $\mathbb{R}$,

$$S = \mathcal{S}_{\mathrm{t}}(I, \mathbb{R}) \times \mathcal{S}_{\mathrm{t}}(I, \mathbb{R}) \times \mathcal{S}_{\mathrm{t}}(I, \mathbb{R}).$$

- The behavior set B contains all signal tuples (p, n, i) that satisfy equation 2.10,

$$B = \{(p, n, i) \in \mathcal{S}_{\mathrm{t}}(I, \mathbb{R}) \times \mathcal{S}_{\mathrm{t}}(I, \mathbb{R}) \times \mathcal{S}_{\mathrm{t}}(I, \mathbb{R}) \,|\, p(t) - n(t) = \mathrm{R}\, i(t),\ \forall t \in I\}.$$

Because the behavior set B is defined by equation 2.10, if any two signals in (p, n, i) are known, the third can be derived from them. This approach to specify process behavior is used in non-causal modeling [30].

Example. Figure 2.7 illustrates a dataflow process [43] that multiplies the corresponding values in the two input streams a and b to produce the output stream m. Let $\star$ denote the multiplication of streams, defined as

$$m = a \star b \iff \begin{cases} t_k^m \in \mathrm{dom}(m) \Leftrightarrow t_k^a \in \mathrm{dom}(a) \quad \text{and} \quad t_k^b \in \mathrm{dom}(b), \\ \qquad\qquad m(t_k^m) = a(t_k^a) b(t_k^b). \end{cases} \tag{2.11}$$

By definition 2.26, this dataflow process M is the tuple $(3, S, B)$ where

- The process has arity 3. The indexes 1, 2, and 3 correspond to a, b, and m respectively. Signal names may be used in place of indexes for better presentation.
- $S = \mathcal{S}(\{t_k^a\}, \mathbb{N}) \times \mathcal{S}(\{t_k^b\}, \mathbb{N}) \times \mathcal{S}(\{t_k^m\}, \mathbb{N})$.
- $B = \{(a, b, m) \,|\, m = a \star b\}$.

The output stream is a function of the input streams, but given two streams a and m, a stream b that satisfies $a \star b = m$ may not exist or may not be unique.

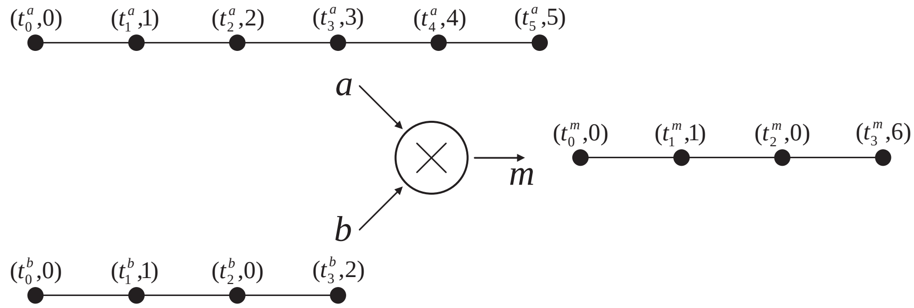

Figure 2.7. A dataflow process that multiplies the corresponding values in the two input streams a and b to produce the output stream m.

Definition 2.27 (Functional Process). A process P, (n, S, B), is functional with respect to a partition $I = (i_1, \ldots, i_k)$ and $O = (o_1, \ldots, o_{n-k})$ of its related signals if for every $r \in S|_I$, there exists exactly one behavior $s \in B$ such that

$$s|_I = r.$$

The process R illustrated in figure 2.6 is functional with respect to the partitions $I = (p, n)$ and $O = (i)$, $I = (i, n)$ and $O = (p)$, and $I = (p, i)$ and $O = (n)$. The process M in figure 2.7 is functional with respect to the partition $I = (a, b)$ and $O = (m)$, but not functional, for example, with respect to $I = (a, m)$ and $O = (b)$.

Notation. For a process P, (n, S, B), and a signal tuple $(s_1, \ldots, s_n)$,

$$P(s_1, \ldots, s_n) \iff (s_1, \ldots, s_n) \in B.$$

If P is functional with respect to index tuples I and O, then for signal tuples $r \in S|_I$ and $s \in S|_O$,

$$s = P(r) \iff \exists p \in B \ \text{ such that } \ r = p|_I, \ s = p|_O\,.$$

Figure 2.8 illustrates the graphical representation of processes.

Figure 2.8. Graphical representation of processes. The process R on the left relates signals p, n, and i, $R(p, n, i)$. The process M on the right is functional with respect to $I = (a, b)$ and $O = (m)$, $m = M(a, b)$.

2.6 Monotonicity, Maximality, and Continuity

For a process $P = (n, S, B)$ that is functional with respect to index tuples I and O, $S|_I$ is the set of its input signal tuples, and $S|_O$ the set of its output signal tuples. By lemma 2.18, both $S|_I$ and $S|_O$, with the prefix order $\preceq$, are posets.

Definition 2.28 (Monotonicity). A functional process P is monotonic if, as a function from $S|_I$ to $S|_O$, it is order-preserving,

$$\forall r, s \in S|_I,\ r \preceq s \implies P(r) \preceq P(s).$$

Recall that a signal $s \in \mathcal{S}(T, V)$ is total if $\mathrm{dom}(s) = T$. A signal tuple is total if all of its components are total.

Definition 2.29 (Maximality). A functional process P is maximal if

$$\forall s \in S|_I,\ P(s) = \bigwedge\{P(r) \,|\, r \in S|_I,\ r \text{ is total, and } s \preceq r\}. \tag{2.12}$$

By lemma 2.18, $S|_O$ is a complete lower semilattice, so the right-hand-side of equation 2.12 is well defined. The behavior of a maximal process is determined by its mapping on total input signals. Such a process maps each input signal to the largest, in the prefix order, output signal that will not be "refuted" by any future input.

Lemma 2.30. Every maximal process is monotonic.

Proof. Let P be a maximal process. For all $s, s' \in S|_I$, let

$$R = \{r \in S|_I \,|\, r \text{ is total and } s \preceq r\},$$

$$R' = \{r' \in S|_I \,|\, r' \text{ is total and } s' \preceq r'\}.$$

$$\begin{aligned} s \preceq s' &\implies R \supseteq R', \\ &\implies \{P(r) \,|\, r \in R\} \supseteq \{P(r') \,|\, r' \in R'\}, \\ &\implies \bigwedge\{P(r) \,|\, r \in R\} \preceq \bigwedge\{P(r') \,|\, r' \in R'\}, \\ &\implies P(s) \preceq P(s'). \end{aligned}$$

P is monotonic. $\square$

For a functional process P and a subset A of $S|_I$, let

$$P(A) = \{P(s) \,|\, s \in A\}.$$

Definition 2.31 (Scott Continuity). A functional process P is (Scott) continuous if for any directed set $D \subseteq S|_I$, $P(D)$, a subset of $S|_O$, is a directed set, and

$$P(\bigvee D) = \bigvee P(D).$$

Lemma 2.32. Every continuous process is monotonic.

Proof. For any two signals $r, s \in S|_I$ such that $r \preceq s$, the set $\{r, s\}$ is a directed set. P is continuous,

$$\bigvee\{P(r), P(s)\} = P(\bigvee\{r, s\}) = P(s),$$

so $P(r) \preceq P(s)$. P is monotonic. $\square$

2.7 Networks of Processes

Complex relations or functions on signals can be defined by creating networks of processes. Figure 2.9 shows a RC low pass filter circuit. The network of processes in figure 2.10 is a specification of the circuit using the tagged signal model. The processes are:

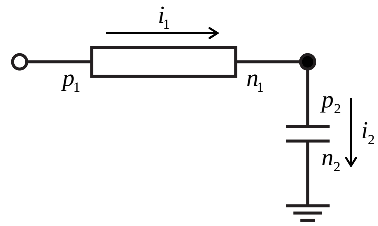

Figure 2.9. A RC low pass filter circuit.

- $R = (3, S_R, B_R)$. R relates signals p_1 and n_1, the voltages at the two terminals of the resistor, and i_1, the current flow through the resistor,

$$(p_1, n_1, i_1) \in B_R \iff p_1 - n_1 = \mathrm{R}\, i_1,$$

  where R is the resistance.

- $C = (3, S_C, B_C)$. C relates signals p_2 and n_2, the voltages at the two terminals of the capacitor, and i_2, the current flow through the capacitor,

$$(p_2, n_2, i_2) \in B_C \iff i_2 = \mathrm{C}\, \frac{d}{dt}(p_2 - n_2)\,,$$

  where C is the capacitance.

- $N = (4, S_N, B_N)$. This process corresponds to the circuit node that connects the resistor and the capacitor. It relates the signals n_1, p_2, i_1, and i_2,

$$(n_1, p_2, i_1, i_2) \in B_N \iff n_1 = p_2 \text{ and } i_1 - i_2 = 0.$$

- $G = (2, S_G, B_G)$. This process corresponds to the ground node in the circuit. It relates the signals n_2 and i_2,

$$(n_2, i_2) \in B_G \iff n_2 = 0.$$

Definition 2.33 (Network of Processes). A network of processes with n signals and m processes is a tuple

$$(n, S, \{P_k, k = 1, \ldots, m\}, \{I_k, k = 1, \ldots, m\}),$$

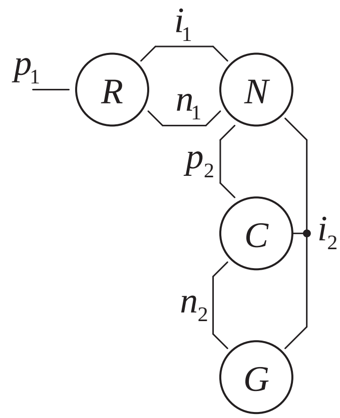

Figure 2.10. The RC circuit from figure 2.9 as a network of processes.

where

$$S = \mathcal{S}(T_1, V_1) \times \cdots \times \mathcal{S}(T_n, V_n)$$

is the signature of the network, and for each $k \in \{1, \ldots, m\}$, $S|_{I_k}$ equals S_{P_k}, the signature of P_k.

For the network in figure 2.10, if the signals are ordered as $(p_1, n_1, i_1, p_2, n_2, i_2)$, then the network is

$$(6, S, \{R, C, N, G\}, \{(1, 2, 3), (4, 5, 6), (2, 4, 3, 6), (5, 6)\}) .$$

Each I_k is called an **incidence tuple**.

By definition 2.33, a network of processes is trivially a process $N = (n, S, B)$ where

$$s \in B \iff \forall k = 1, \ldots, m,\ s|_{I_k} \in B_{P_k} .$$

Although N satisfies the definition of a process, whether its set of behaviors meets the goal of creating the network depends on the properties of the processes in the network and the network structure. The composition of processes in general will not be discussed further. The focus will be on the composition of functional processes. From here on, all processes are assumed to be functional unless explicitly stated otherwise.

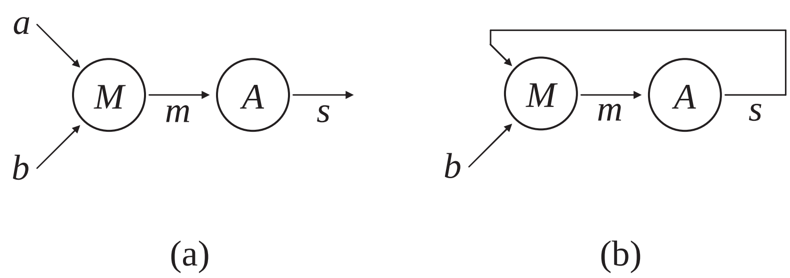

Figure 2.11. Two networks of functional processes.

Figure 2.11 illustrates networks of functional processes. The network in figure 2.11(a) has 4 signals that satisfy the equations

$$m = M(a, b),$$
$$s = A(m).$$

Take signals a and b, which are not the output of any process in the network, as input to the network. The network is a functional process N such that

$$N(a, b) = (m, s) \text{ where } m = M(a, b),\ \ s = A(M(a, b)).$$

The network in figure 2.11(b) has 3 signals that satisfy the equations

$$\begin{aligned} m &= M(s, b), \\ s &= A(m). \end{aligned} \tag{2.13}$$

Take b as the input of the network. The solution to the above equations may have the following properties.

- For some $b \in \mathcal{S}(T_b, V_b)$, the equations have no solution. The network is not a functional process.
- For all $b \in \mathcal{S}(T_b, V_b)$, the equations have a unique solution. The network is a functional process that maps each b to the corresponding solution.
- For all $b \in \mathcal{S}(T_b, V_b)$, the equations have a solution, and for some or all b, more than one solution. Declaring that the network is not a functional process is one alternative.

The other alternative is to develop criteria to choose a solution when there is more than one. The network is a functional process that maps each b to the chosen solution. The latter alternative is employed in the next section to develop tagged process networks.

Definition 2.34 (Network of Functional Processes). A network of functional processes is a network of processes in which all processes are functional, and no signal is the output of more than one process.

2.8 Tagged Process Networks

Kahn process networks [40] are an elegant model of parallel computation. Each signal in a KPN is a stream as illustrated in figure 2.1. Each process is a continuous function (definition 2.31) from its input signals to output signals. Take the network in figure 2.11(b) as an example KPN. For any input b, the signals m and s satisfy the equations 2.13, which can be rewritten as

$$(m, s) = F_b(m, s), \tag{2.14}$$

where

$$F_b \colon \mathcal{S}(T_m, V_m) \times \mathcal{S}(T_s, V_s) \to \mathcal{S}(T_m, V_m) \times \mathcal{S}(T_s, V_s),$$
$$F_b(m, s) = (M(s, b), A(m)).$$

A solution to equation 2.14 is called a *fixed point* of F_b. Because M and A are continuous, F_b is continuous. By Theorem 2.1.19 in [2], F_b has a least fixed point, so equation 2.14 always has at least one solution. Furthermore when it has more than one solution, there exists a solution that is a prefix of all others. Let $\mathrm{fix}(F_b)$ be the least fixed point of F_b. For every input b, $\mathrm{fix}(F_b)$ is chosen as the output of the network. The KPN is a functional process N such that

$$N(b) = \mathrm{fix}(F_b).$$

This approach can be naturally generalized to tagged process networks (TPNs).

Definition 2.35 (Tagged Process Network). A tagged process network is a network of functional processes in which all processes are continuous functions from their input signals to output signals.

The following lemma will be used in establishing the properties of TPNs.

Lemma 2.36. Let D, E be CPOs, and $f\colon D \times E \to E$ a continuous function.

(a) For all $x \in D$, $f_x\colon y \mapsto f(x, y)$ is a continuous function from E to E.

(b) The function $g\colon x \mapsto \text{fix}(f_x)$ from D to E is a continuous function.

Proof.

Part (a). See Lemma 8.10 in [77].

Part (b). Let $[E \to E]$ be the set of all continuous functions from E to E. With the partial order

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

$[E \to E]$ is a CPO. For any directed set $A \subseteq D$, $\{f_x \mid x \in A\}$ is a directed set, and

$$\bigvee\{f_x \mid x \in A\} = f_{\bigvee A}\,,$$

so the function $h\colon x \mapsto f_x$ from D to $[E \to E]$ is continuous. By Theorem 2.1.19 in [2], the function $\text{fix}\colon [E \to E] \to E$ is continuous. The composition of two continuous functions is continuous, $g = \text{fix} \circ h$ is continuous. □

For a TPN N with signature S and processes $\{P_k, k = 1, \ldots, m\}$, let I_N be the index tuple of signals that are not the output of any process, and O_N the index tuple of the other signals. Define a function $F_N\colon S|_{I_N} \times S|_{O_N} \to S|_{O_N}$ as follows.

Let function $g\colon S|_{I_N} \times S|_{O_N} \to S$ be defined by

$$g(a, b) = s \text{ where } s|_{I_N} = a,\ s|_{O_N} = b\,.$$

Let function $h\colon S \to S$ be defined by

$$\begin{aligned} h(s) = r \text{ where} \\ r|_{I_N} &= s|_{I_N}\,, \\ r|_{O_k} &= P_k(s|_{I_k}),\ k = 1, \dots, m. \end{aligned}$$

I_k is the incidence tuple of the input signals of process P_k, and O_k the incidence tuple of its output signals.

The function F_N is

$$F_N(a, b) = h(g(a, b))|_{O_N}\,.$$

Because all processes in the network are continuous functions, F_N is a continuous function. Take the network from figure 2.11(b) as an example,

$$\begin{aligned} I_N &= \{b\}, \\ O_N &= \{m, s\}, \\ g(b, (m, s)) &= (b, m, s), \\ h(b, m, s) &= (b, M(s, b), A(m)), \\ F_N(b, (m, s)) &= (M(s, b), A(m)). \end{aligned}$$

Theorem 2.37 (Tagged Process Network). A TPN N is a functional process with respect to the partition I_N and O_N of its signals,

$$\forall a \in S|_{I_N},\ N(a) = \text{fix}(F_N(a, \cdot)), \tag{2.15}$$

where $F_N(a, \cdot)\colon b \mapsto F_N(a, b)$. N is a continuous function from $S|_{I_N}$ to $S|_{O_N}$.

The proof is straightforward using lemma 2.36. This general theorem is applicable to any model of computation that can be defined as a tagged signal model.

Chapter 3

Discrete Event Process Networks

This chapter focuses on a subclass of tagged process networks (TPNs), in which all signals share a common tag set. The tag set is totally ordered, and is a model of global time in a network of processes.

3.1 Timed Signals

Any non-empty interval of real numbers may be used as the tag set of timed signals. The non-negative real numbers $\mathbb{R}_0 = [0, \infty)$ will be used in most examples as a representative. All value sets of timed signals contain a special element, ε, that represents the absence of a normal value. For any normal value set V, let

$$V_\varepsilon = V \cup \{\varepsilon\}.$$

Definition 3.1 (Timed Signal). Let $T \in \mathcal{I}(\mathbb{R})$ be an interval of real numbers, and V a non-empty set of values. A timed signal is a tagged signal with tag set T and value set V_ε.

A timed signal s is **present** at time $t \in \mathrm{dom}(s)$ if $s(t) \neq \varepsilon$, and otherwise **absent** at t.

Example (Timed Signals). Following are some timed signals from $\mathcal{S}(\mathbb{R}_0, \mathbb{N}_\varepsilon)$.

- A constant signal $const_1$ with value 1 at all times,

$$\begin{aligned} \mathrm{dom}(const_1) &= \mathbb{R}_0, \\ const_1(t) &= 1, \ \forall t \in \mathrm{dom}(const_1). \end{aligned} \tag{3.1}$$

  This signal is illustrated in figure 3.1(a). A timed signal s that is present at all times in $\mathrm{dom}(s)$ is a **continuous-time signal**.

- A clock signal $clock_1$ that is present only at times $t \in \mathbb{N}$,

$$\begin{aligned} \mathrm{dom}(clock_1) &= \mathbb{R}_0, \\ clock_1(t) &= \begin{cases} 1 & \text{if } t \in \mathbb{N}, \\ \varepsilon & \text{if } t \notin \mathbb{N}. \end{cases} \end{aligned} \tag{3.2}$$

  This signal is illustrated in figure 3.1(b).

- A signal $zeno$ that is present at an infinite number of times before time 1,

$$\begin{aligned} \mathrm{dom}(zeno) &= \mathbb{R}_0, \\ zeno(t) &= \begin{cases} 1 & \text{if } t \in \{1 - \frac{1}{2^k} \,|\, k \in \mathbb{N}\}, \\ \varepsilon & \text{otherwise}. \end{cases} \end{aligned} \tag{3.3}$$

  This signal is illustrated in figure 3.1(c).

- A signal $dzeno$ (short for *discrete* Zeno, see section 3.5),

$$\begin{aligned} \mathrm{dom}(dzeno) &= [0, 1), \\ dzeno(t) &= \begin{cases} 1 & \text{if } t \in \{1 - \frac{1}{2^k} \,|\, k \in \mathbb{N}\}, \\ \varepsilon & \text{otherwise}. \end{cases} \end{aligned} \tag{3.4}$$

  This signal is illustrated in figure 3.1(d). All previous examples are total signals, but this one is not a total signal.

Notation (Timed Signals). A timed signal $s \in \mathcal{S}(T, V_\varepsilon)$ can be represented by a tuple $(T, \mathrm{dom}(s), E)$ where

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

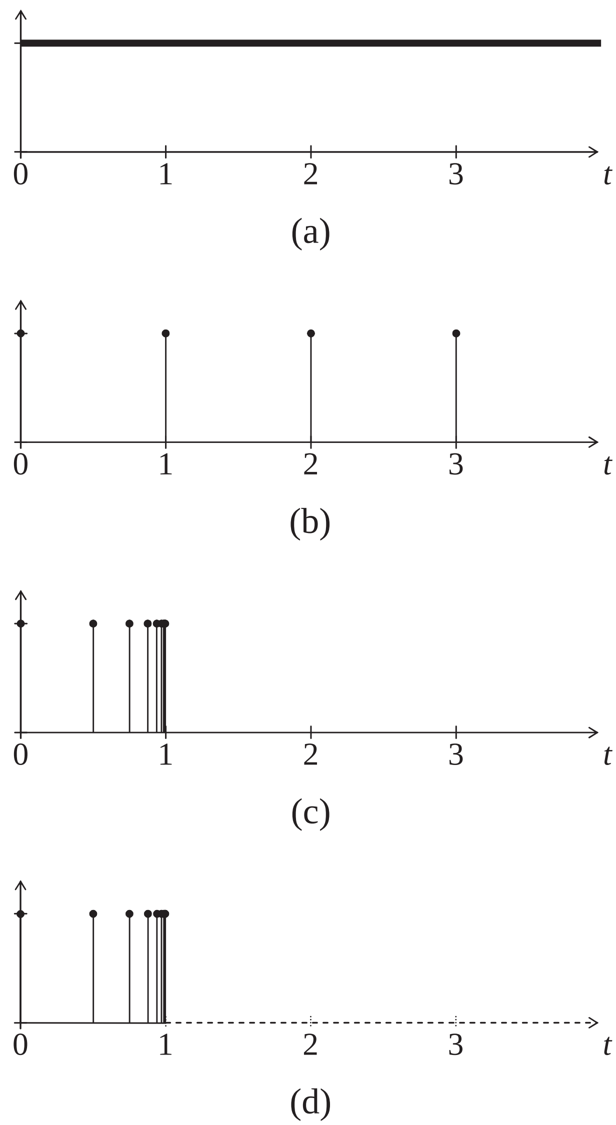

Figure 3.1. Examples of timed signals: (a) $const_1$, (b) $clock_1$, (c) $zeno$, (d) $dzeno$.

An element of E is a **present event** in signal s. If E is a finite set, signal s is called a **finite signal**.

With this notation, the signals in figure 3.1 are

$$
\begin{aligned}
\mathit{const}_1 &= (\mathbb{R}_0, \mathbb{R}_0, \{(t, 1) \,|\, t \in \mathbb{R}_0\}),\\
\mathit{clock}_1 &= (\mathbb{R}_0, \mathbb{R}_0, \{(k, 1) \,|\, k \in \mathbb{N}\}),\\
\mathit{zeno} &= (\mathbb{R}_0, \mathbb{R}_0, \{(1 - \frac{1}{2^k}, 1) \,|\, k \in \mathbb{N}\}),\\
\mathit{dzeno} &= (\mathbb{R}_0, [0, 1), \{(1 - \frac{1}{2^k}, 1) \,|\, k \in \mathbb{N}\}).
\end{aligned}
$$

This notation helps to distinguish between the empty signal $s_\perp$ and the **absent signal** s_ε,

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

The empty signal has no event, whereas the absent signal has "absent event" at all times.

Corollary 3.2 (Timed Signals). Let $T \in \mathcal{I}(\mathbb{R})$ be an interval of real numbers, and V a non-empty set of values. The set of timed signals $\mathcal{S}(T, V_\varepsilon)$ with the prefix order $\preceq$ is both a CPO and a complete lower semilattice.

This corollary is a special case of lemmas 2.16 and 2.17.

Restriction. For a tagged signal $s \in \mathcal{S}(T, V)$ and a down-set $D \in \mathcal{D}(T)$, the restriction of s to D, $s\!\downarrow_D$, is also a tagged signal, such that

$$
\begin{aligned}
\mathrm{dom}(s\!\downarrow_D) &= \mathrm{dom}(s) \cap D,\\
(s\!\downarrow_D)(t) &= s(t),\ \forall t \in \mathrm{dom}(s\!\downarrow_D).
\end{aligned}
\tag{3.5}
$$

For example, $\mathit{dzeno} = \mathit{zeno}\!\downarrow_{[0,1)}$.

For a set of signals $S \subseteq \mathcal{S}(T, V)$, let

$$S\!\downarrow_D = \{s\!\downarrow_D \ | \, s \in S\}.$$

For any tag $t \in T$, let

$$s\!\downarrow_t = s\!\downarrow_{\downarrow\{t\}}.$$

Recall that $\downarrow\{t\}$ is the set of all tags that "come before" t.

Lemma 3.3 (Restriction). The following holds for any tag set T and value set V.

- Given any signal $s \in \mathcal{S}(T, V)$ and down-sets $D_1, D_2 \in \mathcal{D}(T)$ such that $D_1 \subseteq D_2$,

$$s\downarrow_{D_1} \preceq s\downarrow_{D_2} . \tag{3.6}$$

- Given any signals $s_1, s_2 \in \mathcal{S}(T, V)$ such that $s_1 \preceq s_2$, and down-set $D \in \mathcal{D}(T)$,

$$s_1\downarrow_D \preceq s_2\downarrow_D . \tag{3.7}$$

This lemma is very useful in proving the prefix relation between signals.

Totally ordered tag sets have the following properties.

Lemma 3.4. Let T be a totally ordered set. $\mathcal{D}(T)$ is totally ordered by set inclusion.

Proof. For any two down-sets $D_1, D_2 \in \mathcal{D}(T)$, either $D_1 \subseteq D_2$ or $D_2 \subseteq D_1$. Otherwise there exist t_1 such that $t_1 \in D_1,\ t_1 \notin D_2$, and t_2 such that $t_2 \in D_2,\ t_2 \notin D_1$. T is totally ordered, either $t_1 \le t_2$ or $t_2 \le t_1$. If $t_1 \le t_2$,

$$t_1 \le t_2,\ t_2 \in D_2 \implies t_1 \in D_2 .$$

This contradicts $t_1 \notin D_2$. Similarly $t_2 \le t_1$ leads to a contradiction. $\square$

Proposition 3.5. If the tag set T is a totally ordered set, any directed set $D \subseteq \mathcal{S}(T, V)$ is totally ordered.

Proof. For any two signals $s_1, s_2 \in D$, by lemma 3.4, either $\mathrm{dom}(s_1) \subseteq \mathrm{dom}(s_2)$ or $\mathrm{dom}(s_2) \subseteq \mathrm{dom}(s_1)$. D is a directed set, there exists $u \in D$ such that $s_1 \preceq u$ and $s_2 \preceq u$.

$$s_1 \preceq u \implies s_1 = u\downarrow_{\mathrm{dom}(s_1)},$$
$$s_2 \preceq u \implies s_2 = u\downarrow_{\mathrm{dom}(s_2)}.$$

Together with equation 3.6, $\mathrm{dom}(s_1) \subseteq \mathrm{dom}(s_2)$ implies $s_1 \preceq s_2$, and $\mathrm{dom}(s_2) \subseteq \mathrm{dom}(s_1)$ implies $s_2 \preceq s_1$. Any two signals in D are comparable, D is totally ordered. $\square$

3.2 Timed Processes

A **timed process** is a function from timed signals to timed signals. All input and output signals of a timed process have the same tag set. Several representative timed processes are presented in this section, with discussions on their properties, such as continuity and maximality. Some proofs of these properties are included, in order to illustrate how such proofs can be structured.

The Add Process. Suppose that addition $+\colon V \times V \to V$ is defined on a value set V. Let $+_\varepsilon\colon V_\varepsilon \times V_\varepsilon \to V_\varepsilon$ be defined by

$$\begin{array}{c|c|c} +_\varepsilon & b \in V & b = \varepsilon \\ \hline a \in V & a+b & a \\ \hline a = \varepsilon & b & \varepsilon \end{array} \tag{3.8}$$

The *Add* process adds two timed signals $s_1, s_2 \in \mathcal{S}(T, V_\varepsilon)$ by

$$\begin{aligned} \mathit{Add}(s_1, s_2) = s \text{ where} & \\ \mathrm{dom}(s) &= \mathrm{dom}(s_1) \cap \mathrm{dom}(s_2), \\ s(t) &= s_1(t) +_\varepsilon s_2(t). \end{aligned} \tag{3.9}$$

Proposition 3.6. The process $\mathit{Add}\colon \mathcal{S}(T,V_\varepsilon) \times \mathcal{S}(T,V_\varepsilon) \to \mathcal{S}(T,V_\varepsilon)$ defined by equation 3.9 is continuous.

Proof. First, *Add* is monotonic. For any $(s_1, s_2), (r_1, r_2) \in \mathcal{S}(T,V_\varepsilon) \times \mathcal{S}(T,V_\varepsilon)$ such that

$$(s_1, s_2) \preceq (r_1, r_2),$$

let $s = \mathit{Add}(s_1, s_2),\ r = \mathit{Add}(r_1, r_2)$.

$$\begin{aligned} s_1 \preceq r_1,\ \mathrm{dom}(s) \subseteq \mathrm{dom}(s_1) &\implies s_1\!\downarrow_{\mathrm{dom}(s)} = r_1\!\downarrow_{\mathrm{dom}(s)}, \\ s_2 \preceq r_2,\ \mathrm{dom}(s) \subseteq \mathrm{dom}(s_2) &\implies s_2\!\downarrow_{\mathrm{dom}(s)} = r_2\!\downarrow_{\mathrm{dom}(s)}, \\ &\implies s = r\!\downarrow_{\mathrm{dom}(s)}, \\ &\implies s \preceq r. \end{aligned}$$

Because Add is monotonic, for any directed set $D \subseteq \mathcal{S}(T, V_\varepsilon) \times \mathcal{S}(T, V_\varepsilon)$, $Add(D)$ is a directed set, and

$$\bigvee Add(D) \preceq Add(\bigvee D)\,. \tag{3.10}$$

Let $(u_1, u_2) = \bigvee D$, $u = Add(u_1, u_2)$, and $u' = \bigvee Add(D)$. For any $t \in \mathrm{dom}(u)$,

$$\begin{aligned} t \in \mathrm{dom}(u_1) &\implies \exists (p_1, p_2) \in D,\ t \in \mathrm{dom}(p_1), \\ t \in \mathrm{dom}(u_2) &\implies \exists (q_1, q_2) \in D,\ t \in \mathrm{dom}(q_2). \end{aligned}$$

Because D is a directed set, (p_1, p_2) and (q_1, q_2) have an upper bound (w_1, w_2) in D. Let $w = Add(w_1, w_2)$.

$$\begin{aligned} t \in \mathrm{dom}(w_1),\ t \in \mathrm{dom}(w_2) &\implies t \in \mathrm{dom}(w)\,, \\ (w_1, w_2) \preceq (u_1, u_2) &\implies w \preceq u\,, \\ &\implies w\!\downarrow_t = u\!\downarrow_t. \\ w \in Add(D) &\implies w \preceq u', \\ &\implies w\!\downarrow_t = u'\!\downarrow_t. \\ \forall t \in \mathrm{dom}(u),\ u\!\downarrow_t = u'\!\downarrow_t &\implies u \preceq u'. \end{aligned}$$

That is

$$Add(\bigvee D) \preceq \bigvee Add(D)\,.$$

With equation 3.10,

$$Add(\bigvee D) = \bigvee Add(D)\,.$$

Add is continuous. □

Proposition 3.7. If for every $v \in V$, there exist $v_1, v_2 \in V$ such that

$$v_1 + v \neq v,\ v + v_2 \neq v,$$

then Add is maximal.

Proof. For any $(s_1, s_2) \in \mathcal{S}(T, V_\varepsilon) \times \mathcal{S}(T, V_\varepsilon)$, let

$$s = Add(s_1, s_2),$$

$$Q = \{(q_1, q_2) \in \mathcal{S}(T, V_\varepsilon) \times \mathcal{S}(T, V_\varepsilon) \,|\, (s_1, s_2) \preceq (q_1, q_2),\ (q_1, q_2) \text{ is total}\},$$

$$q = \bigwedge Add(Q).$$

Because Add is monotonic,

$$\forall (q_1, q_2) \in Q,\ s \preceq Add(q_1, q_2) \implies s \preceq q\,.$$

Let $(r_1, r_2) \in Q$ be the signal such that

$$r_1(t) = \begin{cases} s_1(t) & \text{if } t \in \text{dom}(s_1), \\ \varepsilon & \text{otherwise}, \end{cases}$$

$$r_2(t) = \begin{cases} s_2(t) & \text{if } t \in \text{dom}(s_2), \\ \varepsilon & \text{otherwise}. \end{cases}$$

They are extensions of s_1 and s_2 to total signals by appending the "all absent" future. For any $t_0 \notin \text{dom}(s)$, either $t_0 \notin \text{dom}(s_1)$ or $t_0 \notin \text{dom}(s_2)$. Let v_0 be an arbitrary element of V. If $t_0 \notin \text{dom}(s_1)$, define a total signal r_1' by

$$r_1'(t) = \begin{cases} r_1(t) & t \neq t_0, \\ v_0 & t = t_0,\ r_2(t_0) = \varepsilon, \\ v_1 & t = t_0,\ r_2(t_0) = v,\ v_1 + v \neq v. \end{cases}$$

$Add(r_1, r_2)$ and $Add(r_1', r_2)$ are different at t_0,

$$Add(r_1, r_2)(t_0) = \begin{cases} \varepsilon & r_2(t_0) = \varepsilon, \\ v & r_2(t_0) = v, \end{cases}$$

$$Add(r_1', r_2)(t_0) = \begin{cases} v_0 & r_2(t_0) = \varepsilon, \\ v_1 + v & r_2(t_0) = v. \end{cases}$$

Both (r_1, r_2) and (r_1', r_2) are in the set Q,

$$q \preceq Add(r_1, r_2),\ q \preceq Add(r_1', r_2) \implies t_0 \notin \text{dom}(q)\,.$$

s	$\mathit{Delay}_1(s)$
clock_1	$(\mathbb{R}_0, \mathbb{R}_0, \{(k, 1) \mid k \in \mathbb{N},\ k > 0\})$
dzeno	$(\mathbb{R}_0, [0, 2), \{(2 - \frac{1}{2^k}, 1) \mid k \in \mathbb{N}\})$
$(\mathbb{R}_0, \emptyset, \emptyset)$	$(\mathbb{R}_0, [0, 1), \emptyset)$
$(\mathbb{R}, \emptyset, \emptyset)$	$(\mathbb{R}, \emptyset, \emptyset)$

Figure 3.2. Some behaviors of the Delay_d process, with delay $d = 1$.

Similarly $t_0 \notin \mathrm{dom}(s_2)$ implies $t_0 \notin \mathrm{dom}(q)$. Now that $\mathrm{dom}(q) \subseteq \mathrm{dom}(s)$ and $s \preceq q$, $s = q$. *Add* is maximal. □

The Delay Process. Let d be any positive real number. The Delay_d process shifts every event in its input signal by d into the future.

$$\mathit{Delay}_d \colon \mathcal{S}(T, V_\varepsilon) \to \mathcal{S}(T, V_\varepsilon),$$

$$\mathit{Delay}_d(s) = r \text{ where}$$

$$\mathrm{dom}(r) = \{t \in T \mid t - d \in \mathrm{dom}(s) \text{ or } t - d \notin T\}, \tag{3.11}$$

$$r(t) = \begin{cases} s(t - d) & t - d \in \mathrm{dom}(s), \\ \varepsilon & \text{otherwise.} \end{cases}$$

The Delay_d process is both continuous and maximal. Some examples of its behavior are shown in figure 3.2.

The Merge Process. The *Merge* process 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.

$$\mathit{Merge} \colon \mathcal{S}(T, V_\varepsilon) \times \mathcal{S}(T, V_\varepsilon) \to \mathcal{S}(T, V_\varepsilon),$$

$$\mathit{Merge}(s_1, s_2) = s \text{ where}$$

$$\mathrm{dom}(s) = \mathrm{dom}(s_1) \cap \mathrm{dom}(s_2), \tag{3.12}$$

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

The processes *Add* and $Delay_d$ are both continuous and maximal. This is not the case for the *Merge* process, which is continuous but not maximal. Proving the continuity of *Merge* is similar to that for *Add*. That *Merge* is not maximal is illustrated by the following behavior,

$$\begin{aligned} u_1 &= (\mathbb{R}_0, [0,1], \{(1,1)\}), \\ u_2 &= (\mathbb{R}_0, [0,1), \emptyset), \\ Merge(u_1, u_2) &= u_2. \end{aligned} \tag{3.13}$$

Let *MaxMerge* be the maximal version of the *Merge* process, then

$$MaxMerge(u_1, u_2) = u_1.$$

The definition of the *MaxMerge* process is

$$MaxMerge\colon \mathcal{S}(T, V_\varepsilon) \times \mathcal{S}(T, V_\varepsilon) \to \mathcal{S}(T, V_\varepsilon),$$

$$MaxMerge(s_1, s_2) = s \text{ where}$$

$$\mathrm{dom}(s) = \{t \in \mathrm{dom}(s_1) \mid \forall p \in {\downarrow}\{t\} \backslash \mathrm{dom}(s_2),\ s_1(p) \neq \varepsilon\}, \tag{3.14}$$

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

Intuitively, if the input signal s_1 is continuously present over a time interval right beyond $\mathrm{dom}(s_2)$, then those present events are in the output of *MaxMerge*. To show that *MaxMerge* is not continuous, take u_1 and u_2 as defined in equation 3.13, and let

$$\begin{aligned} r_k &= (\mathbb{R}_0, [0, 1 - \frac{1}{2^k}), \emptyset),\ k \in \mathbb{N}, \\ D &= \{(u_1, r_k),\ k \in \mathbb{N}\}. \end{aligned}$$

D is a directed set, and

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

$$\bigvee MaxMerge(D) \neq MaxMerge(\bigvee D)\,.$$

Figure 3.3 illustrates the timed processes discussed in this section.

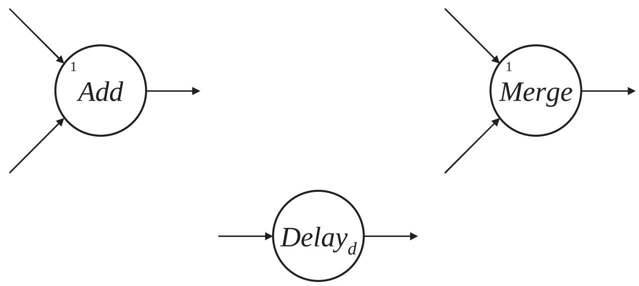

Figure 3.3. Examples of timed processes.

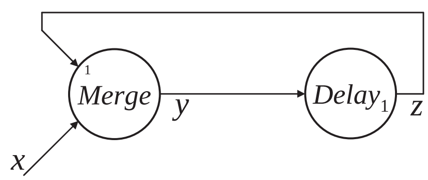

Figure 3.4. A timed process network.

3.3 Timed Process Networks

Definition 3.8 (Timed Process Network). A timed process network is a tagged process network in which all signals are timed signals and have the same tag set $T \in \mathcal{I}(\mathbb{R})$, an interval of real numbers.

Consider the timed process network in figure 3.4. By theorem 2.37, for any input signal x, the output (y, z) of the network is the least fixed point of

$$F\colon (y, z) \mapsto (\mathit{Merge}(z, x), \mathit{Delay}_1(y))\,. \tag{3.15}$$

By Theorem 2.1.19 in [2],

$$\operatorname{fix}(F) = \bigvee_{n \in \mathbb{N}} F^n(s_\perp, s_\perp)\,, \tag{3.16}$$

where F^0 is the identity function, and $F^{n+1} = F \circ F^n$.

Let x be the *zeno* signal from figure 3.1(c). The first 4 values in the sequence

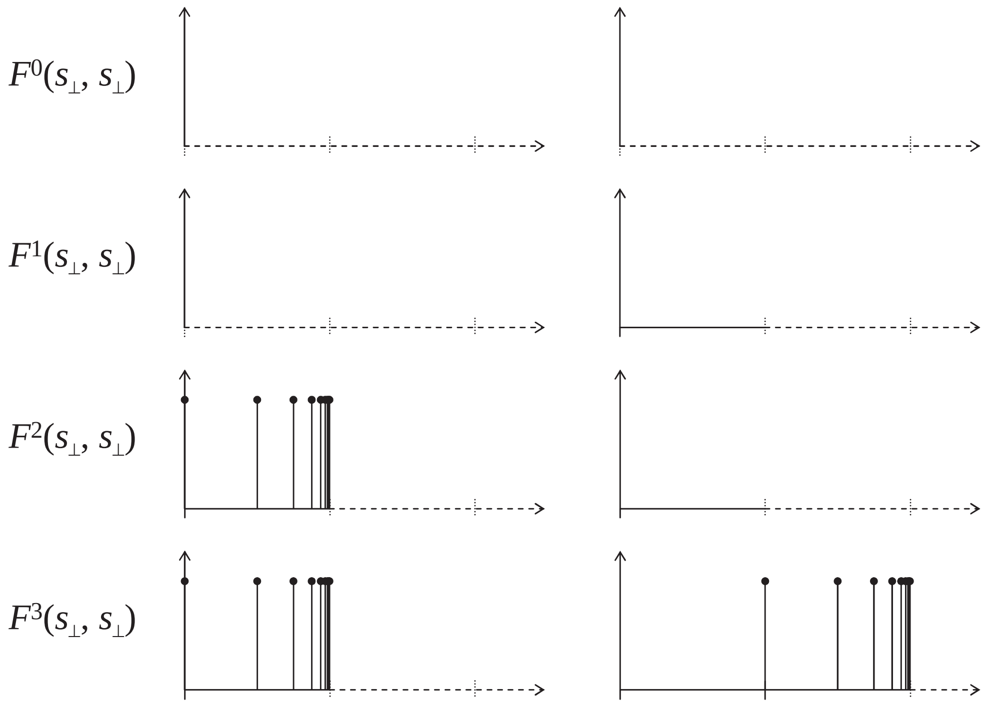

Figure 3.5. Steps in computing the least fixed point of F in equation 3.15.

$\{F^n(s_\perp, s_\perp),\ n \in \mathbb{N}\}$ are illustrated in figure 3.5. By induction on n,

$$\begin{aligned} F^{2n}(s_\perp, s_\perp) = ((&\mathbb{R}_0, [0, n), \{(l - \frac{1}{2^k}, 1) \,|\, l = 1, \ldots, n,\ k \in \mathbb{N}\}), \\ &(\mathbb{R}_0, [0, n), \{(l - \frac{1}{2^k}, 1) \,|\, l = 2, \ldots, n,\ k \in \mathbb{N}\}))\,. \end{aligned} \tag{3.17}$$

Combine equations 3.16 and 3.17,

$$\begin{aligned} \text{fix}(F) = ((&\mathbb{R}_0, \mathbb{R}_0, \{(l - \frac{1}{2^k}, 1) \,|\, l \in \mathbb{N},\ l > 0,\ k \in \mathbb{N}\}), \\ &(\mathbb{R}_0, \mathbb{R}_0, \{(l - \frac{1}{2^k}, 1) \,|\, l \in \mathbb{N},\ l > 1,\ k \in \mathbb{N}\})), \end{aligned} \tag{3.18}$$

which is illustrated in figure 3.6.

Equation 3.16 provides an iterative scheme to compute the least fixed point of F. By this scheme and equation 3.17, for any $t > 0$, $\text{fix}(F) \downarrow_{[0,t]}$ can be determined after a finite number of iterations (applications of F). It is important to note that in this example, the input signal *zeno* is present at an infinite number of times in the finite time interval $[0, 1]$. Equation 3.16 makes it possible to compute the behavior of the network beyond the "Zeno point" in time.

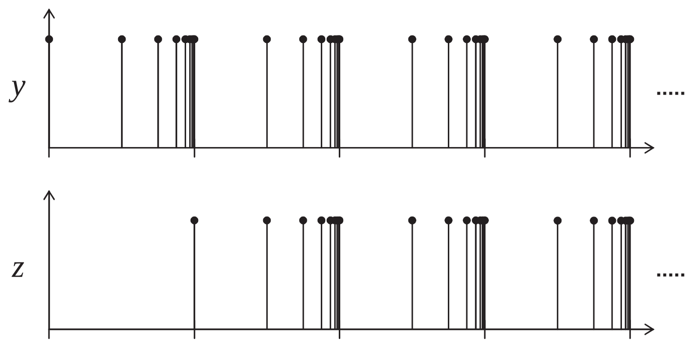

Figure 3.6. The least fixed point of F in equation 3.15.

If the tag set of the signals is $\mathbb{R}_0$, then the network in figure 3.4 has the following property,

$$x \text{ is a total signal} \implies y \text{ and } z \text{ are total signals.}$$

To show this, let

$$D_i = \mathrm{dom}(i),\ \ i = x, y, z.$$

Using abstract interpretation [23], the processes become relations on the domains of the signals,

$$\begin{aligned} D_y &= D_z \cap D_x, \\ D_z &= [0,1) \cup \{t+1 \,|\, t \in D_y\}. \end{aligned} \tag{3.19}$$

$$\begin{aligned} x \text{ is a total signal} &\implies D_x = \mathbb{R}_0, \\ &\implies D_y = D_z, \\ &\implies D_z = [0,1) \cup \{t+1 \,|\, t \in D_z\}. \end{aligned}$$

The only subset of $\mathbb{R}_0$ that satisfies the last equation is $\mathbb{R}_0$, so both y and z are total signals.

Not all timed process networks have this property. The network in figure 3.7 is obtained from that in figure 3.4 by replacing the $Delay_1$ process with the $LookAhead_1$ process, defined

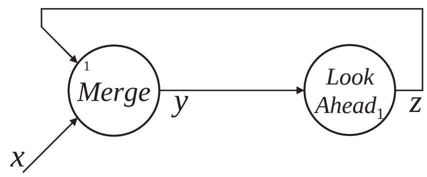

Figure 3.7. A timed process network with a $\mathit{LookAhead}_1$ process.

by the following equation.

$$\begin{aligned} &\mathit{LookAhead}_a \colon \mathcal{S}(T, V_\varepsilon) \to \mathcal{S}(T, V_\varepsilon), \\ &\mathit{LookAhead}_a(s) = r \ \text{ where} \\ &\qquad \mathrm{dom}(r) = \{t \in T \mid t + a \in \mathrm{dom}(s)\}, \\ &\qquad r(t) = s(t + a), \end{aligned} \tag{3.20}$$

for all $a > 0$. $\mathit{LookAhead}_a$ is continuous. For any input signal x, $(s_\perp, s_\perp)$ is the least fixed point of

$$F \colon (y, z) \mapsto (\mathit{Merge}(z, x), \mathit{LookAhead}_1(y)) \,. \tag{3.21}$$

The output of the network is nowhere defined. The next section presents conditions on the processes and network structure such that if all input signals are defined at least on $\downarrow\{t\}$ for some $t \in T$, then all output signals are defined at least on $\downarrow\{t\}$.

3.4 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.

Definition 3.9 (Causality). A timed process P with n input signals and m output signals is causal if it is monotonic, and for all signal tuples $(s_1, \ldots, s_n)$ and $(r_1, \ldots, r_m)$,

$$(r_1, \ldots, r_m) = P(s_1, \ldots, s_n) \implies \bigcap_{i=1}^{n} \mathrm{dom}(s_i) \subseteq \bigcap_{j=1}^{m} \mathrm{dom}(r_j) \,. \tag{3.22}$$

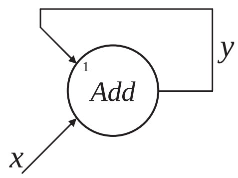

Figure 3.8. A timed process network that is not causal.

A timed process P is causal guarantees that for any two signal tuples $(s_1, \ldots, s_n)$ and $(s'_1, \ldots, s'_n)$, and time t such that

$$t \in \bigcap_{s \in S} \mathrm{dom}(s), \text{ where } S = \{s_1, \ldots, s_n, s'_1, \ldots, s'_n\},$$

$$(s_1, \ldots, s_n)\downarrow_t = (s'_1, \ldots, s'_n)\downarrow_t \implies P(s_1, \ldots, s_n)\downarrow_t = P(s'_1, \ldots, s'_n)\downarrow_t .$$

Among the timed process examples discussed so far, *Add*, $Delay_d$, *Merge*, and *MaxMerge* are causal, whereas $LookAhead_a$ is not causal.

Neither causality nor continuity implies the other. The *MaxMerge* process is causal but not continuous. The $LookAhead_a$ process is continuous but not causal.

A **dependency loop** of length k in a network of processes is a list of k signals $s_i, i = 1, \ldots, k$ and a list of k processes $P_i, i = 1, \ldots, k$ such that s_i is an input signal of $P_i, i = 1, \ldots, k$, s_{i+1} an output signal of $P_i, i = 1, \ldots, k-1$, and s_1 an output signal of P_k. For example, the timed process network in figure 3.4 has a dependency loop of length 2—the list of signals is z, y and the list of processes is *Merge*, $Delay_1$.

If all processes in a timed process network are causal, and there is no dependency loop in the network, such as the network in figure 2.11(a), then the network is a causal process. This may not be the case when there is a dependency loop in the network, as illustrated by the network in figure 3.8. For any input signal x, the output of the network y is the empty signal $s_\perp$.

For any network of causal processes, a sufficient condition that the network defines a causal process is that in every dependency loop, there exists a process that introduces a fixed delay, such as the $Delay_1$ process in figure 3.4. Such processes are special cases of strictly causal processes.

Definition 3.10 (Strict Causality). A timed process P with n input signals and m output signals is strictly causal if it is monotonic, and for all signal tuples $(s_1, \ldots, s_n)$ and $(r_1, \ldots, r_m)$,

$$(r_1, \ldots, r_m) = P(s_1, \ldots, s_n) \implies \begin{array}{c} \bigcap_{i=1}^{n} \mathrm{dom}(s_i) \subset \bigcap_{j=1}^{m} \mathrm{dom}(r_j), \\ \text{or} \\ r_j, j = 1, \ldots, m \text{ are total signals.} \end{array} \tag{3.23}$$

Example. Several strictly causal processes with one input and one output are presented to illustrate the definition. To determine whether such a process P is strictly causal, a good first test is to check whether $P(s_\perp)$ is the empty signal. If P is strictly causal, then $P(s_\perp) \neq s_\perp$. P must "come up with something from nothing."

- If the tag set of the signals is $\mathbb{R}_0$, the $Delay_d$ process is strictly causal. The same holds for any tag set $T \in \mathcal{I}(\mathbb{R})$ that is not a down-set of $\mathbb{R}$.

  If the tag set T is a down-set of $\mathbb{R}$, such as $(-\infty, 0]$ or $\mathbb{R}$, then the $Delay_d$ process is not strictly causal. For such tag sets,

  $$Delay_d(s_\perp) = s_\perp\,.$$

- Let $\mathbb{R}_0$ be the tag set. Define a function $f\colon \mathbb{R}_0 \to \mathbb{R}_0$ by

  $$f(t) = t + e^{-t}\,.$$

  f is injective. This example process E delays the input at t by e^{-t},

  $$\begin{aligned} E(s) = r \text{ where} \\ \mathrm{dom}(r) &= [0,1) \cup \{f(t) \,|\, t \in \mathrm{dom}(s)\}, \\ r(t) &= \begin{cases} s(f^{-1}(t)) & \text{if } t \geq 1, \\ \varepsilon & \text{if } t \in [0,1). \end{cases} \end{aligned}$$

E is strictly causal. The delay introduced by E does not have a positive lower bound over $\mathbb{R}_0$, although there is one over any finite sub-interval of $\mathbb{R}_0$. E is also continuous and maximal.

- This example process F delays the input at t by $2 - 2(t - \lfloor t \rfloor)$,

$$
\begin{aligned}
&F(s) = r \text{ where} \\
&\operatorname{dom}(r) = \begin{cases} [0,1] & \text{if } \operatorname{dom}(s) = \emptyset, \\ [0, \lceil t \rceil] & \text{if } \operatorname{dom}(s) = [0,t), \\ [0, \lfloor t+1 \rfloor] & \text{if } \operatorname{dom}(s) = [0,t], \\ \mathbb{R}_0 & \text{if } \operatorname{dom}(s) = \mathbb{R}_0, \end{cases} \\
&r(t) = \begin{cases} s(2\lfloor t \rfloor - t) & \text{if } t > 1 \text{ and } t \notin \mathbb{N}, \\ s(t-2) & \text{if } t > 1 \text{ and } t \in \mathbb{N}, \\ \varepsilon & \text{if } t \in [0,1]. \end{cases}
\end{aligned} \tag{3.24}
$$

For an input event $e = (t, v)$, F produces the corresponding output event

$$e' = (2\lfloor t \rfloor + 2 - t, v)\,.$$

The time translation performed by F, that is the mapping

$$t \mapsto 2\lfloor t \rfloor + 2 - t\,,$$

is shown in figure 3.9. F is strictly causal and continuous. For any $\delta \in (0,1)$, F delays the input at $1 - \delta$ by 2δ. The delay introduced by F does not have a positive lower bound over the finite interval $[0,1]$.

Theorem 3.11 (Causal Timed Process Network). If all processes in a timed process network are causal, and in every dependency loop in the network there is at least one strictly causal process, then the network is a causal process.

Proof. Let N be such a network with n signals and m processes $P_1, \ldots, P_m$. Without loss of generality, assume that the first k signals are the input of the network. All timed

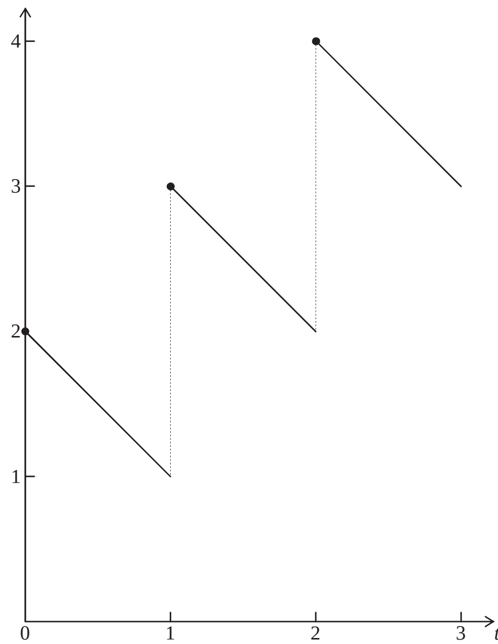

Figure 3.9. The time translation performed by process F in equation 3.24.

process networks are tagged process networks, so by theorem 2.37, N is continuous and thus monotonic.

Suppose that N is not causal. There exist input signals $s_1, \ldots, s_k$ and the corresponding output signals $s_{k+1}, \ldots, s_n$ such that

$$\bigcap_{i=1}^{k} \mathrm{dom}(s_i) \nsubseteq \bigcap_{j=k+1}^{n} \mathrm{dom}(s_j)\,.$$

Let $D = \bigcap_{i=1}^{k} \mathrm{dom}(s_i)$. Recall lemma 3.4, the down-sets of a totally ordered set are totally ordered by set inclusion. Take $j_0 \in \{k+1, \ldots, n\}$ such that $\mathrm{dom}(s_{j_0})$ is the smallest set among $\mathrm{dom}(s_j), j = k+1, \ldots, n$. $\mathrm{dom}(s_{j_0})$ is a strict subset of D. Let P_{l_0} be the process that has s_{j_0} as output, and s_{j_1} be an input of P_{l_0} with the smallest domain among all inputs of P_{l_0}. Because P_{l_0} is causal, $\mathrm{dom}(s_{j_1}) \subseteq \mathrm{dom}(s_{j_0})$. By the picking of j_0, $\mathrm{dom}(s_{j_0}) \subseteq \mathrm{dom}(s_{j_1})$, so $\mathrm{dom}(s_{j_1}) = \mathrm{dom}(s_{j_0})$. s_{j_1} cannot be an input of the network. Let P_{l_1} be the process that has s_{j_1} as output, and continue tracing back to an input s_{j_2} of P_{l_1}, and so on,

$$s_{j_0} \xleftarrow{P_{l_0}} s_{j_1} \xleftarrow{P_{l_1}} s_{j_2} \xleftarrow{P_{l_2}} \cdots\,.$$

There are n signals in the network, so there exist $p, q \in \{0, \ldots, n\}$ such that $p < q$ and

$j_p = j_q$. The signals $s_{j_r}, r = p, \ldots, q-1$ and processes $P_{l_r}, r = p, \ldots, q-1$ form a dependency loop. There is a strictly causal process among these processes. The smallest domain of its input signals equals the domain of an output. This contradicts the strict causality of the process. $\square$

3.5 Discrete Event Signals

An important subclass of timed systems are discrete event (DE) systems [20, 29]. The study of DE process networks parallels that of general timed process networks, starting from the definition of DE signals and their properties.

Definition 3.12 (Discrete Event Signal). A timed signal $s \in \mathcal{S}(T, V_\varepsilon)$ is a discrete event signal if there exists a directed set $D \subseteq \mathcal{S}(T, V_\varepsilon)$ of finite timed signals such that

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

Let $\mathcal{S}_\mathrm{d}(T, V_\varepsilon)$ denote the set of all DE signals with tag set $T \in \mathcal{I}(\mathbb{R})$ and value set V_ε. Among the signals in figure 3.1, $clock_1$ and $dzeno$ are DE signals, but not $const_1$ and $zeno$. 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 three lemmas.

Lemma 3.13. A timed signal s is a DE signal if and only if for all $t \in \mathrm{dom}(s)$, $s\downarrow_t$ 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 $t \in \mathrm{dom}(r)$.

$$r \preceq s \implies s\downarrow_t = r\downarrow_t\,.$$

r is a finite signal implies $r\downarrow_t$ is a finite signal, so is $s\downarrow_t$.

For any timed signal s, let

$$D_s = \{s\downarrow_t \mid t \in \mathrm{dom}(s)\} \cup \{s_\perp\}\,.$$

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

The above equivalent definition of DE signals will be used most often in proving properties of DE signals and processes.

Lemma 3.14. A timed signal $s \in \mathcal{S}(T, V_\varepsilon)$ is a DE signal if and only if

$$s^{-1}(V) \neq \emptyset \implies \min(s^{-1}(V)) \text{ exists,} \tag{3.25}$$

and for all $t \in \mathrm{dom}(s)$, there exists $\delta > 0$ such that

$$s^{-1}(V) \cap (t - \delta, t + \delta) \tag{3.26}$$

is a finite set[1].

Proof. Let s be a DE signal. If $s^{-1}(V)$ is not empty, take any $t_0 \in s^{-1}(V)$. $s\downarrow_{t_0}$ is a finite signal, so $s^{-1}(V) \cap \downarrow\{t_0\}$ is a finite set. Let $t_{\min}$ be the minimum element of this set, then

$$\min(s^{-1}(V)) = t_{\min}\,.$$

For all $t \in \mathrm{dom}(s)$, if $\max(\mathrm{dom}(s))$ exists and $t = \max(\mathrm{dom}(s))$, then s equals $s\downarrow_t$ and is a finite signal. $s^{-1}(V) \cap (t - \delta, t + \delta)$ is a finite set for any $\delta > 0$. If $\max(\mathrm{dom}(s))$ does not exist or $t < \max(\mathrm{dom}(s))$, take any $t' \in \mathrm{dom}(s)$ such that $t < t'$. Let $\delta = t' - t$. $s\downarrow_{t'}$ is a finite signal, so $s^{-1}(V) \cap (t - \delta, t + \delta)$ is a finite set.

For the "if" part of the proof, if $s^{-1}(V)$ is the empty set, then s is a DE signal. If $s^{-1}(V)$ is not empty, let

$$t_{\min} = \min(s^{-1}(V))\,.$$

For all $t \in \mathrm{dom}(s)$, if $t \leq t_{\min}$, then $s^{-1}(V) \cap \downarrow\{t\}$ is either empty or contains only $t_{\min}$, $s\downarrow_t$ is a finite signal. If $t > t_{\min}$, suppose that $s\downarrow_t$ is not a finite signal, then $s^{-1}(V) \cap [t_{\min}, t]$ is an infinite set. $[t_{\min}, t]$ is a compact subset of $\mathbb{R}$, there exists $t^* \in [t_{\min}, t]$ that is a cluster point of $s^{-1}(V) \cap [t_{\min}, t]$. For any $\delta > 0$, $s^{-1}(V) \cap (t^* - \delta, t^* + \delta)$ is an infinite set, a contradiction. For all $t \in \mathrm{dom}(s)$, $s\downarrow_t$ is a finite signal, so s is a DE signal. $\square$

[1] This definition is due to Eleftherios D. Matsikoudis.

Lemma 3.15. A timed signal $s \in \mathcal{S}(T, V_\varepsilon)$ is a DE signal if and only if $s^{-1}(V)$ is order-isomorphic to a down-set of $\mathbb{N}$, and if $s^{-1}(V)$ is an infinite set, then

$$\operatorname{dom}(s) = \bigcup_{t \in s^{-1}(V)} \downarrow\{t\}\,. \tag{3.27}$$

This definition is used in [45]. If $s^{-1}(V)$ 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 is present at an infinite number of times, then equation 3.27 guarantees that for any $t \in \operatorname{dom}(s)$, s is present at a time later than t.

Remark 3.16. The previous lemmas provide alternative characterizations of DE signals. Definition 3.12 states that DE signals can be approximated by "simple" elements of $\mathcal{S}(T, V_\varepsilon)$, the finite signals. Lemma 3.13 is very useful in proving properties of DE signals. Lemma 3.14 best explains the discreteness of DE signals—for any DE signal s, $s^{-1}(V)$ is a discrete subset of $\operatorname{dom}(s)$. By lemma 3.15, the present events in a DE signal can be treated as a sequence with increasing time tags.

The following lemma summarizes the properties of $\mathcal{S}_\mathrm{d}(T, V_\varepsilon)$.

Lemma 3.17. For any tag set $T \in \mathcal{I}(\mathbb{R})$ and value set V_ε,

(a) $\mathcal{S}_\mathrm{d}(T, V_\varepsilon)$ is a down-set of $\mathcal{S}(T, V_\varepsilon)$.

(b) $\mathcal{S}_\mathrm{d}(T, V_\varepsilon)$ with the prefix order is a CPO.

(c) $\mathcal{S}_\mathrm{d}(T, V_\varepsilon)$ is a complete lower 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 $\mathcal{S}_\mathrm{d}(T, V_\varepsilon)$. As a subset of $\mathcal{S}(T, V_\varepsilon)$, D is also a directed set. By corollary 3.2, there exists $u \in \mathcal{S}(T, V_\varepsilon)$ such that $u = \bigvee D$ in the CPO $\mathcal{S}(T, V_\varepsilon)$. For all $t \in \operatorname{dom}(u)$, there exists $s \in D$ such that $t \in \operatorname{dom}(s)$.

$$s \preceq u,\ t \in \operatorname{dom}(s) \implies u\!\downarrow_t = s\!\downarrow_t .$$

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

Part (c). The proof follows directly from corollary 3.2 and part (a) of this lemma. □

Definition 3.18 (Non-Zeno Signal). A DE signal $s \in \mathcal{S}_{\mathrm{d}}(T, V_\varepsilon)$ is non-Zeno if either s is a finite signal, or s is a total signal, $\mathrm{dom}(s) = T$.

Between the two DE signals in figure 3.1, $clock_1$ is the only non-Zeno signal. $dzeno$ is a Zeno signal—it is present at an infinite number of times in a strict subset of its tag set. 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 3.18. The signal

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

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

Lemma 3.19. For any tag set $T \in \mathcal{I}(\mathbb{R})$ and value set V_ε, the set of all non-Zeno signals $\mathcal{S}_{\mathrm{nz}}(T, V_\varepsilon)$ is a down-set of $\mathcal{S}_{\mathrm{d}}(T, V_\varepsilon)$.

The proof is straightforward as any prefix of a non-Zeno signal is also a non-Zeno signal.

3.6 Discrete Event Processes

A **discrete event process** is a function from DE signals to DE signals. All input and output signals of a DE process have the same tag set. Among the processes discussed in sections 3.2 and 3.3, Add, $Delay_d$, $Merge$, and $LookAhead_a$ are DE processes. $MaxMerge$ is not a DE process, as it has the following behavior,

$$\begin{aligned} s_1 &= (\mathbb{R}_0, [0,1], \{(1,1)\})\,, \\ s_2 &= dzeno, \\ MaxMerge(s_1, s_2) &= (\mathbb{R}_0, [0,1], \{(1 - \frac{1}{2^k}, 1) \mid k \in \mathbb{N}\} \cup \{(1,1)\})\,. \end{aligned}$$

s	s_d
$const_1$	$(\mathbb{R}_0, [0,0], \{(0,1)\})$
$clock_1$	$clock_1$
$zeno$	$dzeno$

Figure 3.10. Examples of DE prefixes. s_d is derived from s by equation 3.28.

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

It is possible to derive a DE process from any timed process. For any timed signal $s \in \mathcal{S}(T, V_\varepsilon)$, let its DE prefix be defined by

$$s_d = \bigvee \{r \in \mathcal{S}_{\mathrm{d}}(T, V_\varepsilon) \,|\, r \preceq s\}\,. \tag{3.28}$$

By lemma 3.17(b), s_d is a DE signal. Some examples of DE prefixes are shown in figure 3.10.

Lemma 3.20. The DE prefix function $\rho\colon \mathcal{S}(T, V_\varepsilon) \to \mathcal{S}_{\mathrm{d}}(T, V_\varepsilon)$, which maps $s \in \mathcal{S}(T, V_\varepsilon)$ to s_d given by equation 3.28, is a continuous function.

Proof. For all $s, s' \in \mathcal{S}(T, V_\varepsilon)$,

$$\begin{aligned} s \preceq s' &\implies \{r \in \mathcal{S}_{\mathrm{d}}(T, V_\varepsilon) \,|\, r \preceq s\} \subseteq \{r \in \mathcal{S}_{\mathrm{d}}(T, V_\varepsilon) \,|\, r \preceq s'\}, \\ &\implies s_d \preceq s'_d\,. \end{aligned}$$

The function ρ is monotonic.

Let $D \subseteq \mathcal{S}(T, V_\varepsilon)$ be a directed set of timed signals, and $u = \bigvee D$. ρ is monotonic implies that $\rho(D)$ is a directed set of DE signals and $\bigvee \rho(D) \preceq \rho(u)$. Continuity requires also $\rho(u) \preceq \bigvee \rho(D)$.

Let $p = \rho(u)$ and $q = \bigvee \rho(D)$. For all $t \in \mathrm{dom}(p)$, $p\downarrow_t$ is a finite signal. $t \in \mathrm{dom}(p)$ implies $t \in \mathrm{dom}(u)$, so there exists $r \in D$ such that $t \in \mathrm{dom}(r)$.

$$r\downarrow_t = u\downarrow_t = p\downarrow_t\,.$$

$r\downarrow_t$ is a finite signal, so

$$p\downarrow_t = r\downarrow_t \preceq \rho(r) \preceq q.$$

For all $t \in \mathrm{dom}(p)$, $p\downarrow_t$ is a prefix of q.

$$p = \bigvee\{p\downarrow_t \mid t \in \mathrm{dom}(p)\},$$

so p is a prefix of q. $\square$

Given any timed process $P\colon \mathcal{S}(T, V_\varepsilon) \to \mathcal{S}(T, V_\varepsilon)$, the derived DE process is

$$\begin{aligned} P_d\colon \mathcal{S}_\mathrm{d}(T, V_\varepsilon) &\to \mathcal{S}_\mathrm{d}(T, V_\varepsilon), \\ P_d(s) &= \rho(P(s))\,. \end{aligned} \tag{3.29}$$

It is straightforward to generalize the above definition to multiple-input, multiple-output processes. Because ρ is continuous, P_d is continuous if P is continuous. Because ρ is not causal, derivation 3.29 does not preserve causality.

Definition 3.21 (Non-Zeno Process). A DE process $P\colon \mathcal{S}_\mathrm{d}(T, V_\varepsilon) \to \mathcal{S}_\mathrm{d}(T, V_\varepsilon)$ is a non-Zeno process if for any non-Zeno signal $s \in \mathcal{S}_\mathrm{d}(T, V_\varepsilon)$, $P(s)$ is a non-Zeno signal.

Such processes are called *simple* processes in [21].

Theorem 3.22. A causal DE process is non-Zeno.

Proof. Let $P\colon \mathcal{S}_\mathrm{d}(T, V_\varepsilon) \to \mathcal{S}_\mathrm{d}(T, V_\varepsilon)$ be a causal DE process. Let $s \in \mathcal{S}_\mathrm{d}(T, V_\varepsilon)$ 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 $s' \in \mathcal{S}_\mathrm{d}(T, V_\varepsilon)$ be a total signal such that

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

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

3.7 Discrete Event Process Networks

In a **discrete event process network**, all signals are DE signals, and all processes are DE processes. The network in figure 3.11 is the same as the timed process network in figure 3.4 but considered as a DE process network. The figure shows the behavior of the network when the input signal x equals *dzeno*.

Because the set $\mathcal{S}_{\mathrm{d}}(T, V_\varepsilon)$ of DE signals with the prefix order is a CPO, if all processes in a DE process network are continuous, the same development of tagged process networks in section 2.8 can be applied directly to DE process networks.

Theorem 3.23 (Discrete Event Process Network). If all processes in a DE process network are continuous, then the network, as a functional process that maps input signals to the least solution of the network equations, is continuous.

Theorem 3.24 (Causal DE Process Network). If all processes in a DE process network are causal and continuous, and in every dependency loop in the network there is at least one strictly causal process, then the network is causal and continuous.

Proof. By theorem 3.23, the network is continuous. Proving that the network is causal is the same as proving theorem 3.11 on the causality of timed process networks. □

Corollary 3.25. A DE process network that satisfies the assumptions of theorem 3.24 is non-Zeno.

This corollary follows directly from theorems 3.24 and 3.22.

What distinguishes the above results from previous work [78, 79] on timed process networks is the observation that "being absent" is significant for timed signals. The conditions on signals and processes are also weaker here—in [79], two present events in a signal must be separated by a minimum time interval, and processes are required to introduce a minimum time delay.

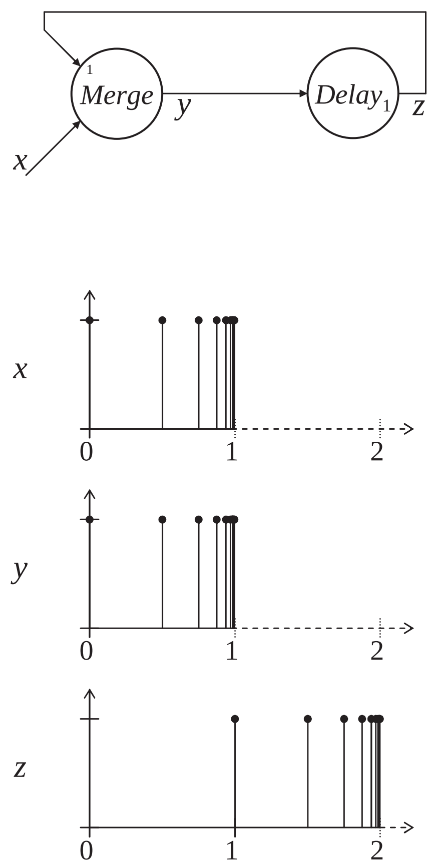

Figure 3.11. A DE process network and its behavior when the input x equals *dzeno*.

3.8 Generalizations and Specializations

3.8.1 Super-Dense Time

So far intervals of $\mathbb{R}$ have been used as the tag sets of timed signals. This choice is due to the familiarity of $\mathbb{R}$ as a model of time. Except lemma 3.14, all definitions and results in the previous sections can be straightforwardly generalized to let any totally ordered set T be the tag set. The super-dense model of time [56] is a prime example of such generalizations.

Definition 3.26 (Super-Dense Time). The super-dense time (SDT) $\mathbb{S}$ is the set $\mathbb{R}_0 \times \mathbb{N}$ with the total order

$$(r_1, n_1) \leq (r_2, n_2) \iff r_1 < r_2, \text{ or } r_1 = r_2 \text{ and } n_1 \leq n_2\,. \tag{3.30}$$

SDT can be similarly defined as $I \times \mathbb{N}$, where $I \in \mathcal{I}(\mathbb{R})$ is any interval of real numbers. SDT has been used in studying the semantics of hybrid systems [41, 50, 55]. A subset of $\mathbb{S}$, $\mathbb{N}\times\mathbb{N}$, is used as the model of time in the hardware description languages CONLAN [65], Verilog [82], and VHDL [62]. $\mathbb{S}$ is in a sense "strictly richer" than $\mathbb{R}_0$ as a model of time, as the following lemma shows.

Lemma 3.27. There is no order-embedding of $\mathbb{S}$ in $\mathbb{R}_0$.

Given an event e in a signal $s \in \mathcal{S}(\mathbb{S}, V_\varepsilon)$ such that $\mathrm{tag}(e)$ equals (r, n), call r the *time* of the event, and n the *step* of the event. With super-dense time, a signal can have multiple events at the same time but different steps. This provides a convenient way to specify processes that can take a sequence of actions at a given time.

Consider the *Merge* process defined in equation 3.12. It has the following behavior,

$$\begin{aligned} s_1 &= (\mathbb{R}_0, \mathbb{R}_0, \{(0,1)\}), \\ s_2 &= (\mathbb{R}_0, \mathbb{R}_0, \{(0,2)\}), \\ \mathit{Merge}(s_1, s_2) &= (\mathbb{R}_0, \mathbb{R}_0, \{(0,1)\}). \end{aligned} \tag{3.31}$$

The input signals s_1 and s_2 are both present at $t = 0$. The event in s_2 is discarded. If both events are to be kept in the output, some alternatives are

n	0	1	2	3	4	5	6
$s_1(0,n)$	ε	v_1	ε	v_2	v_3		
$s_2(0,n)$	ε	u_1	u_2	ε	u_3		
$s(0,n)$	ε	v_1	u_1	u_2	v_2	v_3	u_3

Figure 3.12. A behavior of the *Merge* process with super-dense time, where $s = \mathit{Merge}(s_1, s_2)$.

- Change the output value set from V_ε to $[V]_\varepsilon$, where $[V]$ is the set of lists of values in V. This approach adds structure to signals in their value sets.
- Delay the event from s_2 in the output by some amount of time. But how much? There is no universal answer.
- Use the super-dense time, and delay the event from s_2 by one step in the output.

With super-dense time, the behavior in equation 3.31 becomes

$$\begin{aligned} s_1 &= (\mathbb{S}, \mathbb{S}, \{((0,0),1)\}), \\ s_2 &= (\mathbb{S}, \mathbb{S}, \{((0,0),2)\}), \\ \mathit{Merge}(s_1, s_2) &= (\mathbb{S}, \mathbb{S}, \{((0,0),1), ((0,1),2)\}). \end{aligned} \tag{3.32}$$

A further example is shown in figure 3.12.

Defined as follows, this version of the *Merge* process on SDT signals is causal and continuous, but not maximal.

For any signal $s \in \mathcal{S}(\mathbb{S}, V_\varepsilon)$, $r \in \mathbb{R}_0$, and $k_1, k_2 \in \mathbb{N}$, let

$$p(s, r, k_1, k_2) = |s^{-1}(V) \cap \{(r, n) \mid k_1 \leq n < k_2\}| \,. \tag{3.33}$$

$p(s, r, k_1, k_2)$ is the number of present events in s at time r and between steps k_1 and $k_2 - 1$. Given two signals $s_1, s_2 \in \mathcal{S}(\mathbb{S}, V_\varepsilon)$, let

$$D = \mathrm{dom}(s_1) \cap \mathrm{dom}(s_2) \,.$$

n	0	1	2	3	4	5	6
$s(0,n)$	ε	v_1	v_2	ε	ε	v_3	ε
$s'(d,n)$	v_1	v_2	v_3				

Figure 3.13. A behavior of the $Delay_d$ process with super-dense time, where $s' = Delay_d(s)$.

The *Merge* process only outputs events in s_1, s_2 with tag in D,

$$Merge(s_1, s_2) = Merge(s_1 \downarrow_D, s_2 \downarrow_D)\,.$$

For all $(r, n) \in D$, any input event at (r, n) is placed in the output at time r and step m or $m + 1$, where m is computed by

$$m = \max(\{k + p(s_1, r, k, n) + p(s_2, r, k, n) \,|\, 0 \le k < n\} \cup \{n\})\,.$$

Let $s = Merge(s_1, s_2)$, then

$$\begin{aligned}
m = n,\ s_1(r,n) = s_2(r,n) = \varepsilon &\implies s(r,m) = \varepsilon, \\
s_1(r,n) = u,\ s_2(r,n) = \varepsilon &\implies s(r,m) = u, \\
s_1(r,n) = \varepsilon,\ s_2(r,n) = v &\implies s(r,m) = v, \\
s_1(r,n) = u,\ s_2(r,n) = v &\implies s(r,m) = u,\ s(r,m+1) = v.
\end{aligned}$$

The $Delay_d$ process on SDT signals has the behavior shown in figure 3.13. The equation that defines this $Delay_d$ process is

$$s'(r,n) = \begin{cases} \varepsilon & r < d, \\ s(r-d,k) & p(s, r-d, 0, k+1) = n+1,\ s(r-d,k) \neq \varepsilon, \\ \varepsilon & \{r-d\} \times \mathbb{N} \subseteq \mathrm{dom}(s),\ \forall k \in \mathbb{N},\ p(s, r-d, 0, k+1) \le n. \end{cases} \tag{3.34}$$

This process is strictly causal and maximal, but not continuous.

Figure 3.14 shows a process network with super-dense time. The input signal x is a ramp signal, present at $(k, 0)$ with value k, for all $k \in \mathbb{N}$. The behavior of the network is

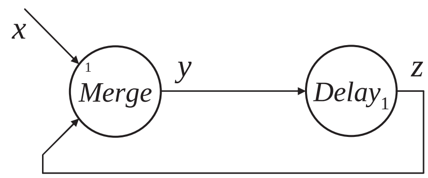

Figure 3.14. A process network with super-dense time.

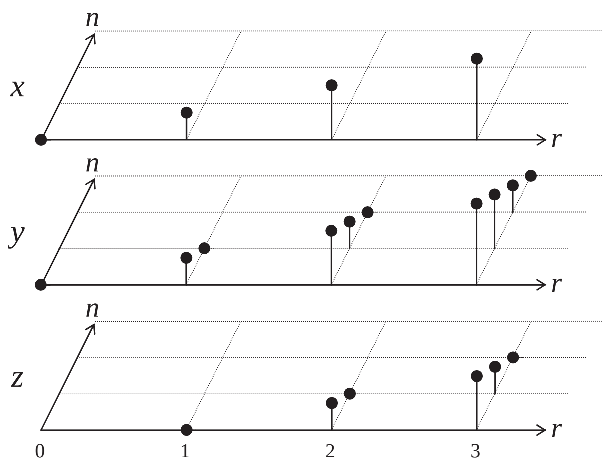

Figure 3.15. A behavior of the process network with super-dense time.

illustrated in figure 3.15. Let E_i be the set of present events in signal i, $i = x, y, z$, then

$$\mathrm{dom}(x) = \mathrm{dom}(y) = \mathrm{dom}(z) = \mathbb{S},$$

$$E_x = \{((k,0),k) \,|\, k \in \mathbb{N}\},$$

$$E_y = \{((k,l),k-l) \,|\, k \in \mathbb{N},\ l \in \mathbb{N},\ 0 \le l \le k\},$$

$$E_z = \{((k,l),k-l-1) \,|\, k \in \mathbb{N},\ l \in \mathbb{N},\ 0 \le l < k\}.$$

3.8.2 Discrete Time Signals, Processes, and Networks

An important subclass of discrete event systems are the *discrete time* (DT) systems that are used extensively in signal processing [63].

A **discrete time signal** with *sampling period* $h > 0$ and *offset* $d \ge 0$ is a DE signal $s \in \mathcal{S}_{\mathrm{d}}(\mathbb{R}_0, V_\varepsilon)$ such that

$$s^{-1}(V) = \mathrm{dom}(s) \cap \{kh + d \,|\, k \in \mathbb{N}\}.$$

All DT signals are non-Zeno.

Lemma 3.28. For all $h > 0$ and $d \geq 0$, the set of all DT signals with sampling period h and offset d, with the prefix order, is both a CPO and a complete lower semilattice.

A **discrete time process** maps input DT signals to output DT signals. The signals may have different sampling periods and offsets. A **discrete time process network** consists of DT signals and DT processes. Theorem 3.24 on DE process networks can be adapted directly to DT process networks.

Theorem 3.29 (Causal DT Process Network). If all processes in a DT process network are causal and continuous, and in every dependency loop in the network there is at least one strictly causal process, then the network is causal and continuous.

Chapter 4

The Metric Structure of Signals

The last two chapters build on domain theory [2], developed for the denotational semantics of programming languages [70, 73, 77]. An alternative approach to denotational semantics is based on metric spaces [3, 6, 25, 67]. This chapter studies various metrics on tagged signals and the application of fixed point theorems to define the behavior of process networks.

4.1 Mathematical Preliminaries

Definition 4.1 (Metric Space). A metric space (X, d) is a set X with a metric function $d\colon X \times X \to \mathbb{R}_0$ such that for all $x, y, z \in X$,

$$\begin{aligned} &d(x, y) = 0 \text{ if and only if } x = y, \\ &d(x, y) = d(y, x), \\ &d(x, z) \leq d(x, y) + d(y, z). \end{aligned} \tag{4.1}$$

If the metric function d also satisfies

$$d(x, z) \leq \max(d(x, y), d(y, z)), \tag{4.2}$$

for all $x, y, z \in X$, (X, d) is an **ultrametric space**.

The value $d(x, y)$ quantifies how close x is an approximation of y, and is called the *distance* between x and y. An element $x \in X$ is the **limit** of a sequence $\{x_k \mid k \in \mathbb{N}\}$ if for all $\epsilon > 0$, there exists $n \in \mathbb{N}$ such that for all $k \geq n$, $d(x_k, x) < \epsilon$. The sequence is said to converge to x, denoted by $x_k \to x$. A sequence $\{x_k \mid k \in \mathbb{N}\}$ is **Cauchy** if for all $\epsilon > 0$, there exists $n \in \mathbb{N}$ such that for all $k, l \geq n$, $d(x_k, x_l) < \epsilon$. A metric space (X, d) is **complete** if every Cauchy sequence converges to some $x \in X$.

Let $B_\delta(x)$ be the set $\{y \in X \mid d(y, x) < \delta\}$. The collection of such sets $\{B_\delta(x) \mid x \in X,\ \delta \in \mathbb{R}_+\}$ is a basis of a *topology* on X. This topology is called the *metric topology* induced by d.

A function $f\colon (X, d) \to (X', d')$ is **continuous** if $x_k \to x$ implies $f(x_k) \to f(x)$. It is **non-expanding** if for all $x, y \in X$,

$$d'(f(x), f(y)) \leq d(x, y). \tag{4.3}$$

f is a **contraction** if there exists $\delta \in (0, 1)$ such that for all $x, y \in X$,

$$d'(f(x), f(y)) \leq \delta\, d(x, y). \tag{4.4}$$

From the theory of metric spaces, the key result used in programming language semantics is the Banach fixed point theorem [33].

Theorem 4.2 (Banach Fixed Point Theorem). Let (X, d) be a complete metric space. If $f\colon (X, d) \to (X, d)$ is a contraction, then

- f has a unique fixed point in X, denoted by $\mathrm{fix}(f)$;
- for all $x \in X$, $f^k(x) \to \mathrm{fix}(f)$.

4.2 Cantor Metric

The Cantor metric can be defined on streams or sequences [19]. The same metric is called the Baire-distance in [26]. The focus here is on the Cantor metric of timed signals [45, 53, 69].

The Cantor-distance function d_{cantor} on timed signals with tag set $T = \mathbb{R}_0$ is defined as

$$
\begin{aligned}
d_{\text{cantor}} \colon \mathcal{S}(T, V_\varepsilon) \times \mathcal{S}(T, V_\varepsilon) \to \mathbb{R}_0, \\
d_{\text{cantor}}(s_1, s_2) = 2^{-\sup\{t \in T \mid s_1\downarrow_t = s_2\downarrow_t\}}.
\end{aligned}
\tag{4.5}
$$

It is understood that $2^{-\sup \emptyset} = 1$, and $2^{-\sup \mathbb{R}_0} = 0$. Using the timed signals in figure 3.1, some examples are

$$
\begin{aligned}
d_{\text{cantor}}(\mathit{const}_1, \mathit{clock}_1) &= 2^{-\sup\{0\}} = 1\,, \\
d_{\text{cantor}}(\mathit{clock}_1, \mathit{zeno}) &= 2^{-\sup[0, \frac{1}{2})} = \frac{1}{\sqrt{2}}\,, \\
d_{\text{cantor}}(\mathit{zeno}, \mathit{dzeno}) &= 2^{-\sup[0,1)} = \frac{1}{2}\,.
\end{aligned}
$$

Lemma 4.3 (Cantor Metric). $(\mathcal{S}(\mathbb{R}_0, V_\varepsilon), d_{\text{cantor}})$ is a complete (ultra)metric space.

Let $\mathbf{M}_{\text{c}}$ denote the metric space $(\mathcal{S}(\mathbb{R}_0, V_\varepsilon), d_{\text{cantor}})$. As an example of a converging sequence in $\mathbf{M}_{\text{c}}$, let

$$
\begin{aligned}
& s_k = \mathit{Delay}_1^k(\mathit{clock}_1),\ k \in \mathbb{N}, \\
& s_k \to s_\varepsilon\,.
\end{aligned}
\tag{4.6}
$$

It is straightforward to extend the Cantor metric to tuples of signals. Let

$$
d^n_{\text{cantor}}((r_1, \ldots, r_n), (s_1, \ldots, s_n)) = \max\{d_{\text{cantor}}(r_i, s_i) \mid 1 \le i \le n\}\,. \tag{4.7}
$$

The set of all n-tuples of signals with d^n_{cantor} is a complete (ultra)metric space, and

$$
(r_{1k}, \ldots, r_{nk}) \to (s_1, \ldots, s_n) \iff r_{ik} \to s_i,\ 1 \le i \le n. \tag{4.8}
$$

4.2.1 Convergence in $(\mathcal{S}(\mathbb{R}_0, V_\varepsilon), \preceq)$ and $(\mathcal{S}(\mathbb{R}_0, V_\varepsilon), d_{\text{cantor}})$

Let $S = \{s_k \mid k \in \mathbb{N}\}$ be a sequence of signals from $\mathcal{S}(\mathbb{R}_0, V_\varepsilon)$ such that $s_k \preceq s_{k+1}$ for all $k \in \mathbb{N}$. Such a sequence is called monotonic. S is a directed set of the CPO $(\mathcal{S}(\mathbb{R}_0, V_\varepsilon), \preceq)$, so there exists $s \in \mathcal{S}(\mathbb{R}_0, V_\varepsilon)$ such that $s = \bigvee S$. For all $k \in \mathbb{N}$,

$$
s_k \preceq s_{k+1} \preceq s \implies d_{\text{cantor}}(s_{k+1}, s) \le d_{\text{cantor}}(s_k, s), \tag{4.9}
$$

but the sequence may not converge to s in $\mathbf{M}_{\mathrm{c}}$. For example, let

$$t_k = 1 - \frac{1}{2^k}\,,$$

$$s_k = dzeno\downarrow_{t_k},$$

$$dzeno = \bigvee\{s_k \mid k \in \mathbb{N}\}.$$

But for all $k \in \mathbb{N}$,

$$d_{\text{cantor}}(s_k, dzeno) > \frac{1}{2}\,,$$

so the sequence $\{s_k \mid k \in \mathbb{N}\}$ does not converge to $dzeno$ in $\mathbf{M}_{\mathrm{c}}$.

Lemma 4.4. Let $S = \{s_k \mid k \in \mathbb{N}\}$ be a monotonic sequence from $\mathcal{S}(\mathbb{R}_0, V_\varepsilon)$ and $s = \bigvee S$. If $\mathrm{dom}(s) = \mathbb{R}_0$, then $s_k \to s$ in $\mathbf{M}_{\mathrm{c}}$.

Proof. For any $\epsilon > 0$, let

$$t = \max(1 + \log_2 \frac{1}{\epsilon}\,,\ 0).$$

$s = \bigvee S$ implies

$$\mathrm{dom}(s) = \bigcup_{k\in\mathbb{N}} \mathrm{dom}(s_k)\,.$$

$t \in \mathrm{dom}(s)$, so there exists $n \in \mathbb{N}$ such that $t \in \mathrm{dom}(s_n)$. For all $k \geq n$,

$$t \in \mathrm{dom}(s_k),$$

$$s_k\downarrow_t = s\downarrow_t\,,$$

$$d_{\text{cantor}}(s_k, s) \leq 2^{-t} < \epsilon\,.$$

$s_k \to s$ in $\mathbf{M}_{\mathrm{c}}$. □

A converging sequence in $\mathbf{M}_{\mathrm{c}}$ may not be monotonic, such as the sequence defined by equation 4.6. If a sequence is both converging and monotonic, then the limit equals the least upper bound.

Lemma 4.5. If the sequence $S = \{s_k \mid k \in \mathbb{N}\}$ is both converging and monotonic, then $s_k \to \bigvee S$.

Proof. Let $s = \bigvee S$. If $\mathrm{dom}(s) = \mathbb{R}_0$, use lemma 4.4. Otherwise take any $t \in \mathbb{R}_0 \setminus \mathrm{dom}(s)$, and let $\epsilon = 2^{-t}$. S is converging, so it is a Cauchy sequence. There exists $n \in \mathbb{N}$ such that for all $k, l \geq n$,

$$\begin{aligned} d_{\mathrm{cantor}}(s_k, s_l) < \epsilon &\implies s_k{\downarrow_t} = s_l{\downarrow_t}\,, \\ \mathrm{dom}(s) \subseteq [0,t],\ s_k \preceq s,\ s_l \preceq s &\implies \mathrm{dom}(s_k) \subseteq [0,t],\ \mathrm{dom}(s_l) \subseteq [0,t], \\ &\implies s_k = s_l\,. \end{aligned}$$

The sequence S becomes constant after s_n. s_n is the limit and is equal to s. □

Every signal $p \in \mathcal{S}(\mathbb{R}_0, V_\varepsilon)$ that is not total is an *isolated point* in $\mathbf{M}_\mathrm{c}$, or equivalently, $\{p\}$ is an open set in $\mathbf{M}_\mathrm{c}$. Take any $t \in \mathbb{R}_0 \setminus \mathrm{dom}(p)$, and let $\epsilon = 2^{-t}$. The set

$$\{s \in \mathbf{M}_\mathrm{c} \,|\, d_{\mathrm{cantor}}(s, p) < \epsilon\}$$

is an open set in $\mathbf{M}_\mathrm{c}$, and is equal to the singleton set $\{p\}$.

The following lemma shows that removing isolated points from a complete metric space does not affect its completeness.

Lemma 4.6. Let M be a complete metric space, and $I \subset M$ the set of isolated points in M. $M \setminus I$ is a complete metric space.

Proof. For all $x \in I$, $\{x\}$ is an open set in M, so

$$I = \bigcup_{x \in I} \{x\}$$

is an open set in M. $M \setminus I$ is a closed subset of the complete metric space M, so it is also a complete metric space. □

Corollary 4.7. The set of all total timed signals $\mathcal{S}_\mathrm{t}(\mathbb{R}_0, V_\varepsilon)$ with the Cantor metric is a complete (ultra)metric space.

4.2.2 Approximation by Finite Signals

In section 3.5, the "limits" of finite timed signals in $(\mathcal{S}(T,V),\preceq)$, where T is an interval of $\mathbb{R}$, are defined as DE signals. The set of DE signals is a sub-CPO of $(\mathcal{S}(T,V),\preceq)$. What follows are parallel results in the metric space $\mathbf{M}_{\mathrm{c}}$.

Lemma 4.8. For any $s \in \mathcal{S}(\mathbb{R}_0, V_\varepsilon)$, if there exists a sequence of finite signals $\{s_k \mid k \in \mathbb{N}\}$ from $\mathcal{S}(\mathbb{R}_0, V_\varepsilon)$ such that $s_k \to s$, then s is a non-Zeno signal.

Proof. If s is not a total signal, $s_k \to s$ implies that there exists $n \in \mathbb{N}$ such that for all $k \geq n$, $s_k = s$. s is a finite signal, so it is non-Zeno.

If s is a total signal, for any $t > 0$, there exists $m \in \mathbb{N}$ such that

$$d_{\mathrm{cantor}}(s_m, s) < 2^{-t}.$$

This implies $s\downarrow_t = s_k\downarrow_t$, so $s\downarrow_t$ is a finite signal. By lemma 3.13, s is a total DE signal, so it is non-Zeno. $\square$

Lemma 4.9. The set of all non-Zeno signals $\mathcal{S}_{\mathrm{nz}}(\mathbb{R}_0, V_\varepsilon)$ with the Cantor metric is a complete (ultra)metric space.

Proof. Let $\{s_k \mid k \in \mathbb{N}\}$ be a Cauchy sequence from $\mathcal{S}_{\mathrm{nz}}(\mathbb{R}_0, V_\varepsilon)$. By lemma 4.3, the sequence converges to a signal $s \in \mathcal{S}(\mathbb{R}_0, V_\varepsilon)$. Proving s is non-Zeno is similar to proving lemma 4.8. $\square$

Corollary 4.10. The set of all total DE signals, denoted by $\mathcal{S}_{\mathrm{td}}(\mathbb{R}_0, V_\varepsilon)$, with the Cantor metric is a complete (ultra)metric space.

4.3 Causality

The Cantor metric on timed signals quantifies the extent to which two signals are equal. If a timed process $P\colon \mathcal{S}(\mathbb{R}_0, V_\varepsilon) \to \mathcal{S}(\mathbb{R}_0, V'_\varepsilon)$ is causal by definition 3.9, then for all signals

$s_1, s_2 \in \mathcal{S}(\mathbb{R}_0, V_\varepsilon)$ and $t \in \mathbb{R}_0$,

$$s_1 \downarrow_t = s_2 \downarrow_t \implies P(s_1) \downarrow_t = P(s_2) \downarrow_t . \tag{4.10}$$

This leads directly to the following lemma.

Lemma 4.11. If a timed process $P\colon \mathcal{S}(\mathbb{R}_0, V_\varepsilon) \to \mathcal{S}(\mathbb{R}_0, V'_\varepsilon)$ is causal, then for all signals $s_1, s_2 \in \mathcal{S}(\mathbb{R}_0, V_\varepsilon)$,

$$d_{\text{cantor}}(P(s_1), P(s_2)) \leq d_{\text{cantor}}(s_1, s_2).$$

P is non-expanding.

The converse of the above lemma is used in [45, 69] to define the causality of DE processes. Because $\mathcal{S}(\mathbb{R}_0, V_\varepsilon)$ includes non-total and continuous time signals, the converse is not true for general timed signals and timed processes, as illustrated by the following example. Define signals $u_1, u_2 \in \mathcal{S}(\mathbb{R}_0, \mathbb{R}_\varepsilon)$ as

$$\begin{aligned} u_1(t) &= 0, \ \forall t \in \mathbb{R}_0, \\ u_2(t) &= \begin{cases} 0 & \text{if } t \in [0,1], \\ 1 & \text{if } t > 1, \end{cases} \end{aligned} \tag{4.11}$$

and a process $R\colon \mathcal{S}(\mathbb{R}_0, \mathbb{R}_\varepsilon) \to \mathcal{S}(\mathbb{R}_0, \mathbb{R}_\varepsilon)$ as

$$R(s)(t) = \begin{cases} \lim_{r \to t^+} s(r) & \text{if the \textit{right limit} exists,} \\ 0 & \text{otherwise.} \end{cases} \tag{4.12}$$

The process R is non-expanding. $R(u_1) = u_1$, and

$$R(u_2)(t) = \begin{cases} 0 & \text{if } t \in [0,1), \\ 1 & \text{if } t \geq 1. \end{cases} \tag{4.13}$$

The signals u_1 and u_2 are equal over $[0,1]$, but $R(u_1)$ and $R(u_2)$ are equal only over $[0,1)$. R is not causal.

If only total DE signals are considered, the converse of lemma 4.11 is true and can be used to define the causality of DE processes [45, 69]. For two signals $s_1, s_2 \in \mathcal{S}_{\text{td}}(\mathbb{R}_0, V_\varepsilon)$

such that $s1 \neq s2$ and $d_{\text{cantor}}(s_1, s_2) = 2^{-r}$, by lemma 3.14, it can be shown that s_1 and s_2 are equal over $[0, r)$ and $s_1(r) \neq s_2(r)$. If a process P is non-expanding, then $d_{\text{cantor}}(P(s_1), P(s_2)) \leq 2^{-r}$, and $P(s_1)$ and $P(s_2)$ are also equal over $[0, r)$.

Lemma 4.12. If a process $P\colon \mathcal{S}_{\text{td}}(\mathbb{R}_0, V_\varepsilon) \to \mathcal{S}_{\text{td}}(\mathbb{R}_0, V'_\varepsilon)$ is non-expanding, then it is causal—for all signals $s_1, s_2 \in \mathcal{S}_{\text{td}}(\mathbb{R}_0, V_\varepsilon)$ and $t \in \mathbb{R}_0$,

$$s_1 \downarrow_t = s_2 \downarrow_t \implies P(s_1) \downarrow_t = P(s_2) \downarrow_t .$$

If a process $P\colon \mathcal{S}_{\text{t}}(\mathbb{R}_0, V_\varepsilon) \to \mathcal{S}_{\text{t}}(\mathbb{R}_0, V'_\varepsilon)$ is a contraction by a factor $\delta = 2^{-\tau}$, given two signals $s_1, s_2 \in \mathcal{S}_{\text{t}}(\mathbb{R}_0, V_\varepsilon)$ such that $d_{\text{cantor}}(s_1, s_2) = 2^{-r}$,

$$d_{\text{cantor}}(P(s_1), P(s_2)) \leq 2^{-(r+\tau)}.$$

That is, s_1 and s_2 are equal over $[0, r)$ implies $P(s_1)$ and $P(s_2)$ are equal over $[0, r + \tau)$. P reacts to its input at r with a minimum delay τ.

To illustrate how the Banach fixed point theorem is used to define the behavior of a network of processes, consider the network in figure 4.1. For any input signal x, the signals y and z satisfy the equations

$$y = \mathit{Add}(z, x),$$
$$z = \mathit{Delay}_1(y).$$

Eliminate z from the equations,

$$y = \mathit{Add}(\mathit{Delay}_1(y), x).$$

For any given signal x, the function

$$F_x(y) = \mathit{Add}(\mathit{Delay}_1(y), x) \tag{4.14}$$

is a contraction by the factor $\frac{1}{2}$. Take $\mathbb{N}_\varepsilon$ as the value set of signals x, y, and z. By the Banach fixed point theorem, for all $x \in \mathcal{S}(\mathbb{R}_0, \mathbb{N}_\varepsilon)$, F_x has a unique fixed point $\text{fix}(F_x)$ in $\mathcal{S}(\mathbb{R}_0, \mathbb{N}_\varepsilon)$, which, with $z = \mathit{Delay}_1(\text{fix}(F_x))$, is the behavior of the network given input x.

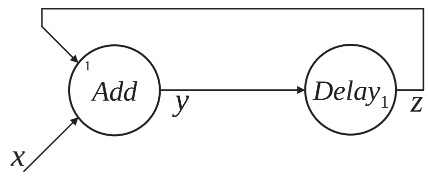

Figure 4.1. A timed process network example.

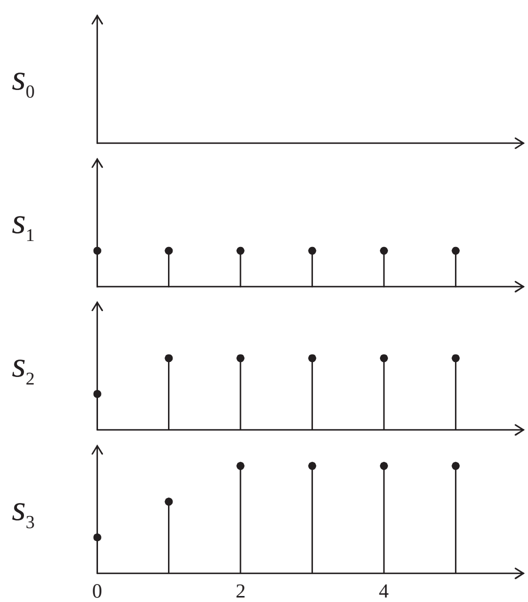

Figure 4.2. Elements from a converging sequence of signals.

The Banach fixed point theorem also states that for all $s \in \mathcal{S}(\mathbb{R}_0, \mathbb{N}_\varepsilon)$,

$$F_x^k(s) \to \text{fix}(F_x).$$

For example, let $x = \mathit{clock}_1$, $s_0 = s_\varepsilon$, and $s_k = F_x^k(s_0)$. It can be shown by induction that

$$d_{\text{cantor}}(s_k, \text{fix}(F_x)) \leq 2^{-k},$$

so s_k equals $\text{fix}(F_x)$ over $[0, k)$. The first four elements from the sequence $\{s_k \mid k \in \mathbb{N}\}$ are shown in figure 4.2, and

$$\text{fix}(F_x) = (\mathbb{R}_0, \mathbb{R}_0, \{(k, k{+}1) \mid k \in \mathbb{N}\}).$$

4.4 Cantor Metric on Alternative Tag Sets

For timed signals, most properties of their order structure studied in chapter 3 do not depend on the choice of totally ordered set as the tag set. This is not the case for the Cantor metric.

Consider the case when the tag set is a finite interval of $\mathbb{R}$, for example $[0, 1)$. By the definition of Cantor metric in equation 4.5, for all signals $s_1, s_2 \in \mathcal{S}([0, 1), V_\varepsilon)$ such that $s_1 \neq s_2$,

$$d_{\text{cantor}}(s_1, s_2) > \frac{1}{2}\,.$$

The metric topology induced by d_{cantor} on $\mathcal{S}([0, 1), V_\varepsilon)$ is the *discrete topology*. The Cantor metric does not provide any useful structure on $\mathcal{S}([0, 1), V_\varepsilon)$.

Another interesting case is to take $\mathbb{R}$ as the tag set. The Cantor-distance between two signals in $\mathcal{S}(\mathbb{R}, V_\varepsilon)$ may be infinity, for example, let

$$\begin{aligned} s_{\text{clk}} &= (\mathbb{R}, \mathbb{R}, \{(k, 1) \mid k \in \mathbb{Z}\}), \\ s_{\text{alt}} &= (\mathbb{R}, \mathbb{R}, \{(2k, 1), (2k{+}1, 0) \mid k \in \mathbb{Z}\}). \end{aligned} \tag{4.15}$$

The signals are illustrated in figure 4.3. The set

$$\{t \in \mathbb{R} \mid s_{\text{clk}}\downarrow_t = s_{\text{alt}}\downarrow_t\}$$

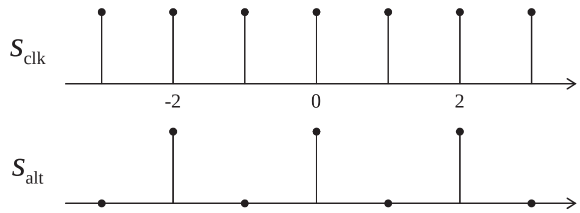

Figure 4.3. Timed signals with tag set $\mathbb{R}$.

is empty. With $\mathbb{R}$ as the tag set, it is understood that $2^{-\sup\emptyset} = \infty$.

$(\mathcal{S}(\mathbb{R}, V_\varepsilon), d_{\text{cantor}})$ is an *extended metric space* [4]. Let relation R on $\mathcal{S}(\mathbb{R}, V_\varepsilon)$ be defined by

$$(s_1, s_2) \in R \iff d_{\text{cantor}}(s_1, s_2) < \infty. \tag{4.16}$$

It is straightforward to show that R is an *equivalence relation*. For all signal $s \in \mathcal{S}(\mathbb{R}, V_\varepsilon)$, let

$$E_s = \{s' \in \mathcal{S}(\mathbb{R}, V_\varepsilon) \mid d_{\text{cantor}}(s', s) < \infty\}$$

be the equivalence class containing s. (E_s, d_{cantor}) is a complete (ultra)metric space.

If a process $P \colon \mathcal{S}(\mathbb{R}, V_\varepsilon) \to \mathcal{S}(\mathbb{R}, V'_\varepsilon)$ is causal, then for all $s \in \mathcal{S}(\mathbb{R}, V_\varepsilon)$,

$$P(E_s) \subseteq E_{P(s)}\,. \tag{4.17}$$

If P is a contraction such that $P(E_s) \subseteq E_s$, then the Banach fixed point theorem is applicable and P has a unique fixed point in E_s. For example,

- the unique fixed point of Delay_1 in E_{s_ε} is s_ε, and in $E_{s_{\text{clk}}}$ is s_{clk},
- $\mathit{Delay}_1(E_{s_{\text{alt}}}) \nsubseteq E_{s_{\text{alt}}}$, so the Banach fixed point theorem is not applicable,
- $\mathit{Delay}_2(E_{s_{\text{alt}}}) \subseteq E_{s_{\text{alt}}}$, and the unique fixed point of Delay_2 in $E_{s_{\text{alt}}}$ is s_{alt}.

An equivalent definition of relation R in equation 4.16 is

$$(s_1, s_2) \in R \iff s_1 \wedge s_2 \neq s_\perp\,, \tag{4.18}$$

that is s_1 and s_2 have a non-empty common prefix—going back in time, s_1 and s_2 are eventually the same. To borrow an analogy from cosmology, the equivalence classes of R

partition $\mathcal{S}(\mathbb{R}, V_\varepsilon)$ into parallel universes, and all signals in an equivalence class originate from the same "Big Bang."

4.5 Generalized Ultrametrics on Signals

For all signals $s_1, s_2 \in \mathcal{S}(\mathbb{R}_0, V_\varepsilon)$, the Cantor metric in essence maps $\mathrm{dom}(s_1 \wedge s_2)$, a down-set of $\mathbb{R}_0$, to an element of $\mathbb{R}_0$, such that

$$\mathrm{dom}(s_1 \wedge s_2) \supseteq \mathrm{dom}(s'_1 \wedge s'_2) \implies d_{\mathrm{cantor}}(s_1, s_2) \leq d_{\mathrm{cantor}}(s'_1, s'_2), \tag{4.19}$$

for all $s'_1, s'_2 \in \mathcal{S}(\mathbb{R}_0, V_\varepsilon)$. The converse is not true, which is the reason why a non-expanding process is not necessarily causal. Because there is no order-embedding of the totally ordered set $(\mathcal{D}(\mathbb{R}_0), \supseteq)$ in $\mathbb{R}_0$, it is impossible to define a metric d on $\mathcal{S}(\mathbb{R}_0, V_\varepsilon)$ such that

$$\mathrm{dom}(s_1 \wedge s_2) \supseteq \mathrm{dom}(s'_1 \wedge s'_2) \iff d(s_1, s_2) \leq d(s'_1, s'_2). \tag{4.20}$$

A generalized ultrametric [67] may satisfy this equivalence.

Definition 4.13 (Generalized Ultrametric). Let X be a set and Γ be a poset with a minimum element 0. A function $d \colon X \times X \to \Gamma$ is a generalized ultrametric if for all $x, y, z \in X$ and $\gamma \in \Gamma$,

$$\begin{aligned} &d(x, y) = 0 \text{ if and only if } x = y, \\ &d(x, y) = d(y, x), \\ &d(x, y) \leq \gamma \text{ and } d(y, z) \leq \gamma \text{ imply } d(x, z) \leq \gamma. \end{aligned} \tag{4.21}$$

A generalized ultrametric on $\mathcal{S}(\mathbb{S}, V_\varepsilon)$ is developed in [21]. $\mathbb{S}$ is the super-dense time from definition 3.26. The down-sets of $\mathbb{S}$ are, for all $r \in \mathbb{R}_0$ and $n \in \mathbb{N}$,

$$\begin{aligned} &[0, r) \times \mathbb{N}, \\ &\downarrow\{(r, n)\}, \\ &[0, r] \times \mathbb{N}, \text{ and} \\ &\mathbb{S}. \end{aligned} \tag{4.22}$$

Note that $[0, 0) \times \mathbb{N}$ is the empty set. Let Γ_{gumsdt} be $\mathbb{R}_0 \times \mathbb{R}_0$ with the lexicographical order. Γ_{gumsdt} is a totally ordered set with $(0, 0)$ as the minimum element. The generalized ultrametric d_{gumsdt} on $\mathcal{S}(\mathbb{S}, V_\varepsilon)$ is defined as

$$\begin{aligned} &d_{\text{gumsdt}} \colon \mathcal{S}(\mathbb{S}, V_\varepsilon) \times \mathcal{S}(\mathbb{S}, V_\varepsilon) \to \Gamma_{\text{gumsdt}}, \\ &d_{\text{gumsdt}}(s_1, s_2) = \begin{cases} (\frac{1}{2^r}, 1) & \text{if } s_1 \neq s_2 \text{ and } \operatorname{dom}(s_1 \wedge s_2) = [0, r) \times \mathbb{N}, \\ (\frac{1}{2^r}, \frac{1}{2^{n+1}}) & \text{if } s_1 \neq s_2 \text{ and } \operatorname{dom}(s_1 \wedge s_2) = \downarrow\{(r, n)\}, \\ (\frac{1}{2^r}, 0) & \text{if } s_1 \neq s_2 \text{ and } \operatorname{dom}(s_1 \wedge s_2) = [0, r] \times \mathbb{N}, \\ (0, 0) & \text{if } s_1 = s_2. \end{cases} \end{aligned} \tag{4.23}$$

Refer to [21] for the application of this generalized ultrametric to the semantics of discrete event systems. Here a generalization of d_{gumsdt} to all tagged signals will be presented.

The function d_{gumsdt} can be decomposed into two functions, one mapping a pair of signals to a down-set of $\mathbb{S}$,

$$\begin{aligned} &d_{\text{ds}} \colon \mathcal{S}(\mathbb{S}, V_\varepsilon) \times \mathcal{S}(\mathbb{S}, V_\varepsilon) \to \mathcal{D}(\mathbb{S}), \\ &d_{\text{ds}}(s_1, s_2) = \begin{cases} \operatorname{dom}(s_1 \wedge s_2) & \text{if } s_1 \neq s_2, \\ \mathbb{S} & \text{if } s_1 = s_2. \end{cases} \end{aligned} \tag{4.24}$$

and the other an order-embedding of $\mathcal{D}(\mathbb{S})$ in Γ_{gumsdt},

$$\begin{aligned} &f_{\text{em}} \colon \mathcal{D}(\mathbb{S}) \to \Gamma_{\text{gumsdt}}, \\ &f_{\text{em}}(D) = \begin{cases} (\frac{1}{2^r}, 1) & \text{if } D = [0, r) \times \mathbb{N}, \\ (\frac{1}{2^r}, \frac{1}{2^{n+1}}) & \text{if } D = \downarrow\{(r, n)\}, \\ (\frac{1}{2^r}, 0) & \text{if } D = [0, r] \times \mathbb{N}, \\ (0, 0) & \text{if } D = \mathbb{S}. \end{cases} \end{aligned} \tag{4.25}$$

d_{gumsdt} is the composition of d_{ds} and f_{em},

$$d_{\text{gumsdt}} = f_{\text{em}} \circ d_{\text{ds}} \,. \tag{4.26}$$

By applying the Cantor metric on both the time and the step axes, the order-embedding f_{em} makes the generalized ultrametric d_{gumsdt} on super-densely timed signals an intuitive extension of the Cantor metric on timed signals defined by equation 4.5. Once the intuition is established, d_{ds} can be taken by itself as a generalized ultrametric on super-densely timed signals. Its definition in equation 4.24 can be generalized to arbitrary tagged signals.

For any tag set T, let the set of generalized ultrametric distances, Γ_T, be the poset

$$\Gamma_T = (\mathcal{D}(T), \supseteq). \tag{4.27}$$

For any two down-sets D, D' of T, D is below D' in Γ_T if and only if $D \supseteq D'$. T is the minimum element of Γ_T and $\emptyset$, the empty set, is the maximum element. By lemma 2.5, the poset $(\mathcal{D}(T), \subseteq)$ is a complete lattice. Γ_T is the same as $(\mathcal{D}(T), \subseteq)$ with the order reversed, so Γ_T is also a complete lattice.

The function d_{ds} in equation 4.24 uses the largest down-set on which two signals are equal as their distance. Its generalization to arbitrary tagged signals is straightforward. For any tag set T and value set V,

$$d_{\mathrm{ds}} \colon \mathcal{S}(T,V) \times \mathcal{S}(T,V) \to \Gamma_T,$$
$$d_{\mathrm{ds}}(s_1, s_2) = \begin{cases} \mathrm{dom}(s_1 \wedge s_2) & \text{if } s_1 \neq s_2, \\ T & \text{if } s_1 = s_2. \end{cases} \tag{4.28}$$

Lemma 4.14 (Generalized Ultrametric on Tagged Signals). For any tag set T and value set V, d_{ds} defined by equation 4.28 is a generalized ultrametric on $\mathcal{S}(T,V)$.

Proof. Note that T is the minimum element of Γ_T. For any $s_1, s_2 \in \mathcal{S}(T,V)$,

$$\begin{aligned} s_1 \neq s_2 &\implies \mathrm{dom}(s_1 \wedge s_2) \subset T, \\ &\implies d_{\mathrm{ds}}(s_1, s_2) \subset T, \\ s_1 = s_2 &\implies d_{\mathrm{ds}}(s_1, s_2) = T, \end{aligned}$$

so $d_{\mathrm{ds}}(s_1, s_2) = T$ if and only if s_1 equals s_2.

$d_{\mathrm{ds}}(s_1, s_2)$ equals $d_{\mathrm{ds}}(s_2, s_1)$ is obvious.

For all $s_1, s_2, s_3 \in \mathcal{S}(T,V)$ and $D \in \mathcal{D}(T)$, given $d_{\mathrm{ds}}(s_1, s_2) \supseteq D$ and $d_{\mathrm{ds}}(s_2, s_3) \supseteq D$, if

$s_1 = s_2$ or $s_2 = s_3$, then trivially $d_{\mathrm{ds}}(s_1, s_3) \supseteq D$. Otherwise,

$$
\begin{aligned}
d_{\mathrm{ds}}(s_1, s_2) \supseteq D, s_1 \neq s_2 &\implies \mathrm{dom}(s_1 \wedge s_2) \supseteq D, \\
&\implies \mathrm{dom}(s_1) \supseteq D, \mathrm{dom}(s_2) \supseteq D, s_1 \!\downarrow_D = s_2 \!\downarrow_D, \\
d_{\mathrm{ds}}(s_2, s_3) \supseteq D, s_2 \neq s_3 &\implies \mathrm{dom}(s_2) \supseteq D, \mathrm{dom}(s_3) \supseteq D, s_2 \!\downarrow_D = s_3 \!\downarrow_D, \\
&\implies \mathrm{dom}(s_1) \supseteq D, \mathrm{dom}(s_3) \supseteq D, s_1 \!\downarrow_D = s_3 \!\downarrow_D, \\
&\implies \mathrm{dom}(s_1 \wedge s_3) \supseteq D, \\
&\implies d_{\mathrm{ds}}(s_1, s_3) \supseteq D.
\end{aligned}
$$

By definition 4.13, d_{ds} is a generalized ultrametric on $\mathcal{S}(T, V)$. $\square$

This lemma shows that for any tag set T and value set V, $(\mathcal{S}(T, V), d_{\mathrm{ds}}, \Gamma_T)$ is a *generalized ultrametric space*. The rest of this section presents the notion of completeness and fixed point theorems on generalized ultrametric spaces [67], and their applications to the tagged signal model.

Consider a generalized ultrametric space (X, d, Γ). For any $\gamma \in \Gamma \setminus \{0\}$ and $a \in X$, the set

$$B_\gamma(a) = \{x \in X \mid d(x, a) \leq \gamma\} \tag{4.29}$$

is called the **ball** with center a and radius γ. If the generalized ultrametric space is $(\mathcal{S}(T, V), d_{\mathrm{ds}}, \Gamma_T)$ for some tag set T and value set V, then for any signal $s \in \mathcal{S}(T, V)$ and $D \in \mathcal{D}(T)$,

- if $\mathrm{dom}(s) \supseteq D$, that is $\mathrm{dom}(s)$ is below D in the poset Γ_T, $d_{\mathrm{ds}}(s', s) \supseteq D$ holds for all $s' \in \mathcal{S}(T, V)$ such that $\mathrm{dom}(s') \supseteq D$ and $s' \!\downarrow_D = s \!\downarrow_D$, and
- if $\mathrm{dom}(s) \not\supseteq D$, the only solution to the inequality $d_{\mathrm{ds}}(x, s) \supseteq D$ is s itself,

so

$$B_D(s) = \begin{cases} \{s' \in \mathcal{S}(T, V) \mid s \!\downarrow_D \preceq s'\} & \text{if } \mathrm{dom}(s) \supseteq D, \\ \{s\} & \text{if } \mathrm{dom}(s) \not\supseteq D. \end{cases} \tag{4.30}$$

This equation shows that all signals in $\mathcal{S}(T, V)$ that are not total are in a sense isolated.

The balls in a generalized ultrametric space are "super-symmetric," as implied by the following lemma.

Lemma 4.15. Let (X, d, Γ) be a generalized ultrametric space, $x, y \in X$, and $\alpha, \beta \in \Gamma$, such that $0 < \alpha \leq \beta$ and $x \in B_\beta(y)$, then

$$B_\alpha(x) \subseteq B_\beta(y).$$

Proof. For any $z \in X$,

$$\begin{aligned} z \in B_\alpha(x) &\implies d(z, x) \leq \alpha \leq \beta, \\ x \in B_\beta(y) &\implies d(x, y) \leq \beta, \\ d(z, x) \leq \beta \ \text{ and } \ d(x, y) \leq \beta &\implies d(z, y) \leq \beta, \\ &\implies z \in B_\beta(y). \end{aligned}$$

$B_\alpha(x)$ is a subset of $B_\beta(y)$. $\square$

Consider the special case where $\alpha = \beta$. This lemma implies that every point in a ball can be taken as its center.

Definition 4.16 (Spherical Completeness). A generalized ultrametric space (X, d, Γ) is spherically complete if every chain of balls (ordered by inclusion) has a non-empty intersection.

Theorem 4.17. For any tag set T and value set V, the generalized ultrametric space $(\mathcal{S}(T, V), d_{\text{ds}}, \Gamma_T)$ is spherically complete.

Proof. Let J be any totally ordered index set, $\{B_{D_j}(s_j) \mid j \in J, s_j \in \mathcal{S}(T, V), D_j \in \mathcal{D}(T)\}$ be a chain of balls in $(\mathcal{S}(T, V), d_{\text{ds}}, \Gamma_T)$, such that for any $j, k \in J$,

$$j \leq k \implies B_{D_j}(s_j) \supseteq B_{D_k}(s_k).$$

There are two cases to the proof.

- There exists $i \in J$ such that $B_{D_i}(s_i) = \{s_i\}$.

  For all $j \in J$ such that $i \leq j$, $B_{D_j}(s_j) = \{s_i\}$. The chain of balls has the non-empty intersection $\{s_i\}$.

- For all $j \in J$, $B_{D_j}(s_j) \supset \{s_j\}$.

  By equation 4.30, for all $j \in J$,

  $$B_{D_j}(s_j) = \{s \in \mathcal{S}(T,V) \mid s_j \downarrow_{D_j} \preceq s\}. \tag{4.31}$$

  $s_j \downarrow_{D_j}$ is the minimum element of $B_{D_j}(s_j)$ in the prefix order. For all $j, k \in J$ such that $j \leq k$,

  $$B_{D_j}(s_j) \supseteq B_{D_k}(s_k) \implies s_j \downarrow_{D_j} \preceq s_k \downarrow_{D_k}.$$

  The set of signals

  $$\{s_j \downarrow_{D_j} \mid j \in J\}$$

  is a chain in the poset $(\mathcal{S}(T,V), \preceq)$. By lemma 2.16, this chain of signals has a least upper bound, here denoted by s', in $\mathcal{S}(T,V)$. By equation 4.31, $s' \in B_{D_j}(s_j)$ for all $j \in J$. The chain of balls $\{B_{D_j}(s_j) \mid j \in J\}$ has a non-empty intersection that includes s'.

□

Recall that $\mathcal{S}_{\mathrm{t}}(T,V)$ is the set of all total signals with tag set T and value set V.

Corollary 4.18. The generalized ultrametric space $(\mathcal{S}_{\mathrm{t}}(T,V), d_{\mathrm{ds}}, \Gamma_T)$ is spherically complete.

Proof. Based on the proof of theorem 4.17, the only change is at the end of the second case. If the least upper bound s' is not a total signal, then arbitrarily extend it to a total signal. □

Example. Let the tag set T be $\mathbb{R}_0$, the non-negative real numbers, and the value set V be any non-empty set. Consider the generalized ultrametric space $U = (S, d_{\mathrm{ds}}, \Gamma_{\mathbb{R}_0})$ where the signal set S is, respectively,

- $\mathcal{S}(\mathbb{R}_0, V_\varepsilon)$, the timed signals. The corresponding generalized ultrametric space is spherically complete by theorem 4.17.

- $\mathcal{S}_\mathrm{t}(\mathbb{R}_0, V_\varepsilon)$, the total timed signals. The corresponding generalized ultrametric space is spherically complete by corollary 4.18.

- $\mathcal{S}_\mathrm{d}(\mathbb{R}_0, V_\varepsilon)$, the discrete event signals. The corresponding generalized ultrametric space is spherically complete. The proof is the same as that of theorem 4.17 except relying on lemma 3.17 instead of lemma 2.16.

- $\mathcal{S}_\mathrm{td}(\mathbb{R}_0, V_\varepsilon)$, the total discrete event signals. The corresponding generalized ultrametric space is not spherically complete. For illustration, take $\mathbb{Z}$ as the value set. Using the notation for timed signals from section 3.1 in the following equation, for $j = 1, 2, \ldots,$ let

$$\begin{aligned} s_j &= \{\mathbb{R}_0, \mathbb{R}_0, \{(1 - \frac{1}{k}, 1) \,|\, 1 \leq k \leq j\}, \\ D_j &= [0, 1 - \frac{1}{j}]. \end{aligned} \tag{4.32}$$

For all $i, j \in \mathbb{N}$ such that $1 \leq i \leq j$,

$$B_{D_i}(s_i) \supseteq B_{D_j}(s_j).$$

Any signal in the intersection of the chain of balls

$$\{B_{D_j}(s_j) \,|\, j = 1, 2, \ldots\}$$

must have the signal

$$\{\mathbb{R}_0, [0, 1), \{(1 - \frac{1}{k}, 1) \,|\, k = 1, 2, \ldots\}$$

as a prefix, and cannot be a total discrete event signal. The intersection of this chain of balls is empty in the generalized ultrametric space $(\mathcal{S}_\mathrm{td}(\mathbb{R}_0, \mathbb{Z}_\varepsilon), d_\mathrm{ds}, \Gamma_{\mathbb{R}_0})$.

Definition 4.19 (Strict Contraction). Let (X, d, Γ) be a generalized ultrametric space. A function $f \colon X \to X$ is strictly contracting if for all $x, y \in X$ such that $x \neq y$,

$$d(f(x), f(y)) < d(x, y).$$

When the generalized ultrametric space is $(\mathcal{S}(T,V), d_{\text{ds}}, \Gamma_T)$ for some tag set T and value set V, a function $f\colon \mathcal{S}(T,V) \to \mathcal{S}(T,V)$ is strictly contracting if for all signals $r, s \in \mathcal{S}(T,V)$ such that $r \neq s$, either $f(r) = f(s)$, or

$$\text{dom}(f(r) \wedge f(s)) \supset \text{dom}(r \wedge s).$$

Note that if the tag set T is totally ordered, then

$$f \text{ is strictly causal by definition 3.10} \implies f \text{ is strictly contracting,}$$

but not vice versa.

There are several variations of fixed point theorems on generalized ultrametric spaces [67, 68]. The following version is from section 5.2 of [67].

Theorem 4.20. Let (X, d, Γ) be a spherically complete generalized ultrametric space. If a function $f\colon X \to X$ is strictly contracting, then f has a unique fixed point.

Refer to [67] for the proof of this theorem. The proof relies on Zorn's lemma, so it is not constructive. Combining this theorem with theorem 4.17 results in the following theorem.

Theorem 4.21. For any tag set T and value set V, and a function $f\colon \mathcal{S}(T,V) \to \mathcal{S}(T,V)$ such that for all signals $x, y \in \mathcal{S}(T,V)$, $x \neq y$,

$$d_{\text{ds}}(f(x), f(y)) < d_{\text{ds}}(x, y),$$

there is a unique signal $s \in \mathcal{S}(T,V)$ such that $f(s) = s$.

By corollary 4.18, the same claim holds when only total signals are considered.

Theorem 4.22. For any tag set T and value set V, and a function $f\colon \mathcal{S}_{\text{t}}(T,V) \to \mathcal{S}_{\text{t}}(T,V)$ such that for all total signals $x, y \in \mathcal{S}_{\text{t}}(T,V)$, $x \neq y$,

$$d_{\text{ds}}(f(x), f(y)) < d_{\text{ds}}(x, y),$$

there is a unique total signal $s \in \mathcal{S}_{\text{t}}(T,V)$ such that $f(s) = s$.

The above theorems can be further specialized, for example, to consider only discrete event signals. In [60], Naundorf first proved the same result as theorem 4.22. The approach here, by developing a generalized ultrametric on tagged signals, makes it possible to apply more research results in generalized ultrametric spaces, for example from [36], to tagged signals.

Chapter 5

Simulation Strategies for Discrete Event Systems

Processes map potentially infinite input signals to potentially infinite output signals. To implement or simulate process behavior in computers, it is necessary to decompose the mapping into incremental steps, such that each step can be completed by executing a finite number of computer instructions. The following quote comes from Brooks [16].

> Much more often, strategic breakthrough will come from redoing the representation of the data or tables. This is where the heart of a program lies. Show me your flowcharts and conceal your tables, and I shall continue to be mystified. Show me your tables, and I won't usually need your flowcharts; they'll be obvious.

Substitute "data or tables" by signals and "flowcharts" by simulation algorithms, and the result is the key to this chapter.

5.1 Processes as Labeled Transition Systems

A *labeled transition system* (LTS) [61] is a tuple $(\Sigma, L, \delta, \sigma_0)$ where

Σ is a set of states,

L is a set of labels,

δ is a transition relation, $\delta \subseteq \Sigma \times L \times \Sigma$,

σ_0 is the initial state, $\sigma_0 \in \Sigma$.

The following notation is used for a transition $(\sigma, l, \sigma') \in \delta$,

$$\sigma \xrightarrow{l} \sigma'.$$

A *trace* of length n in the LTS consists of a sequence of n states $\sigma_1, \ldots, \sigma_n$ and a sequence of n labels $l_1, \ldots, l_n$ such that

$$\sigma_0 \xrightarrow{l_1} \sigma_1 \xrightarrow{l_2} \sigma_2 \to \cdots \to \sigma_{n-1} \xrightarrow{l_n} \sigma_n.$$

Given a monotonic, single-input, single-output process $P\colon \mathcal{S}(T_1, V_1) \to \mathcal{S}(T_2, V_2)$, construct an LTS L_P as follows.

- The set of states: $\mathcal{S}(T_1, V_1)$.
- The set of labels: $\mathcal{G}(T_1, V_1) \times \mathcal{G}(T_2, V_2)$. Each label is a pair of an input signal segment and an output signal segment.
- For any two signals $s, s' \in \mathcal{S}(T_1, V_1)$, an input signal segment $g \in \mathcal{G}(T_1, V_1)$, and an output signal segment $h \in \mathcal{G}(T_2, V_2)$, $(s, (g, h), s')$ is a transition in L_P if and only if

$$s \preceq s',$$

$$g = s' \setminus s,$$

$$h = P(s') \setminus P(s).$$

- The initial state is the empty signal $s_\perp \in \mathcal{S}(T_1, V_1)$.

It is fairly straightforward to generalize the above construction to multiple-input, multiple-output processes. Each state is a tuple of input signals. Each label is a pair of tuples, one of input signal segments, and the other of output signal segments.

Intuitively, a trace in L_P,

$$s_\perp \xrightarrow{(g_1, h_1)} s_1 \xrightarrow{(g_2, h_2)} s_2 \to \cdots \to s_{n-1} \xrightarrow{(g_n, h_n)} s_n, \tag{5.1}$$

represents a decomposition of the mapping $s_n \mapsto P(s_n)$ into n incremental steps. The input signal s_n is broken into n segments $g_1, \ldots, g_n$, such that

$$s_n = s_\perp \ll g_1 \ll \cdots \ll g_n.$$

The kth incremental step consumes the input signal segment g_k, and produces the output signal segment h_k. The final output $P(s_n)$ is

$$P(s_n) = P(s_\perp) \ll h_1 \ll \cdots \ll h_n.$$

The initial output signal $P(s_\perp)$ will be empty if the process P is *strict* [76].

Consider an SDF process *Scramble* shown at the top of figure 5.1. In each firing, the process consumes two integer tokens from input d, and one boolean token from input c. If the value of the boolean token is true, then the two data tokens are sent to the output in the reverse order as they are read, otherwise they are sent without reversing the order. Figure 5.1 also shows an example of how the input and output signals of *Scramble* are segmented. Because of the token consumption and production constraints on SDF processes, the segmentation of their input and output signals is fixed.

The discrete event model of computation, on the other hand, offers much flexibility in how signals can be segmented and consumed by DE processes. Given an input signal s of a DE process P, different segmentations of s determine different traces in L_P. Various DE simulation strategies can be compared by how they plot the traces in L_P. With this perspective, the rest of the chapter focuses on two such strategies, synchronous DE simulation and asynchronous, process-network-based DE simulation.

Remark 5.1. Recall that the futures of a signal $s \in \mathcal{S}(T, V)$, $\mathcal{F}(s)$, are the segments in $\mathcal{G}(T, V)$ that can be appended to s (page 24). For a monotonic process $P\colon \mathcal{S}(T, V) \to \mathcal{S}(T', V')$, at any state $s \in \mathcal{S}(T, V)$ of the LTS L_P, P can be reduced to a function

$$\begin{aligned} &P_s\colon \mathcal{F}(s) \to \mathcal{F}(P(s)), \\ &P_s(g) = P(s \ll g) \setminus P(s),\ \forall g \in \mathcal{F}(s). \end{aligned} \tag{5.2}$$

All the transitions from s in L_P are of the form

$$s \xrightarrow{(g,\, P_s(g))} s \ll g,\ \forall g \in \mathcal{F}(s).$$

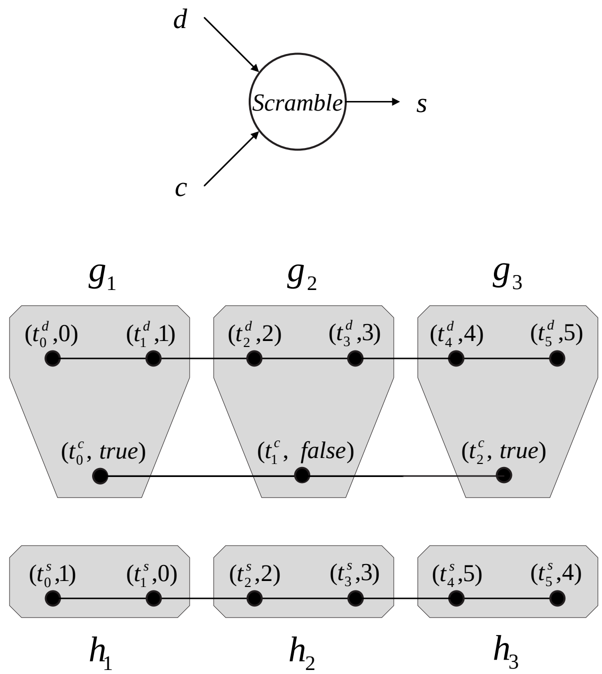

Figure 5.1. The *Scramble* process and segmentation of its input and output signals.

Remark 5.2. The two DE simulation strategies discussed in the rest of this chapter are of the conservative variety. For every process P in a simulation, such strategies only take transitions in L_P that are part of the correct simulation result. For some simulation problems, such as solving ordinary differential equations, intermediate transitions may be required to find fixed point solutions [52, 66]. This is illustrated in figure 5.2 at state s_1. The transitions from s_1 to s_2 and from s_1 to s_3 are intermediate steps in a fixed point iteration. The transitions from s_4 to s_5 and from s_5 to s_6 illustrate speculative simulation. The *Time Warp* [38, 39] simulation strategy is characterized by taking such speculative steps and backtracking when, for example, an input event invalidates the input signal segments h_5 and h_6 in figure 5.2. The speculation can improve simulation performance when backtracking is infrequent.

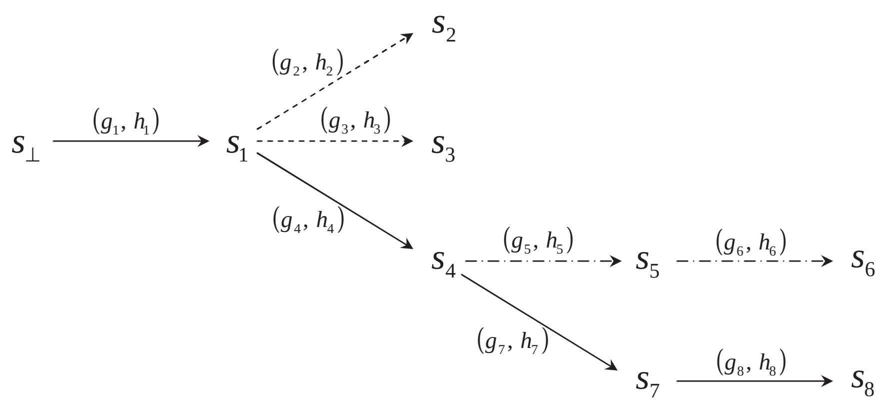

Figure 5.2. Fixed point iteration and backtracking.

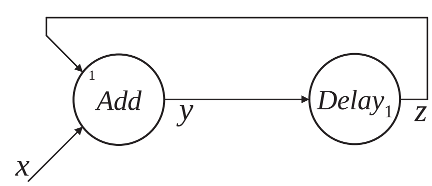

Figure 5.3. A DE process network example.

5.2 Synchronous DE Simulation

Consider the DE process network in figure 5.3. This network is the same as the timed process network in figure 4.1, except that the processes are restricted to DE processes and the signals to DE signals. The *Add* process is a causal, continuous DE process, and the $Delay_1$ process is a strictly causal, continuous DE process. By theorem 3.24, this network, when considered as a function from the input signal x to the tuple of signals (y, z), is a causal and continuous process. If the input signal x is the total signal $clock_1$ from figure 3.1, both output signals y and z are total signals. Figure 5.4 illustrates these signals over the time interval $[0, 4)$.

When the synchronous DE simulation strategy is used to compute the signals y and z, given $clock_1$ as the input, the resulting segmentations of these signals over the same time interval are shown in figure 5.5. This strategy is called synchronous because the segments of all signals are aligned on time, and all processes are simulated in lockstep.

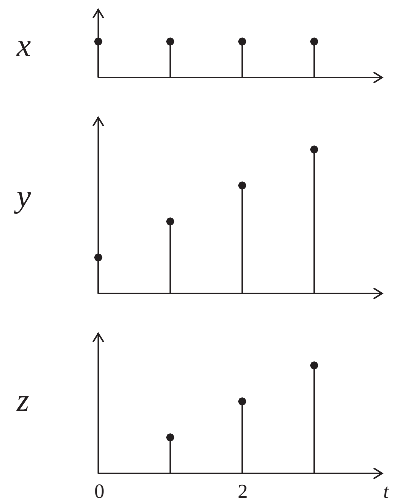

Figure 5.4. A behavior of the DE process network in figure 5.3.

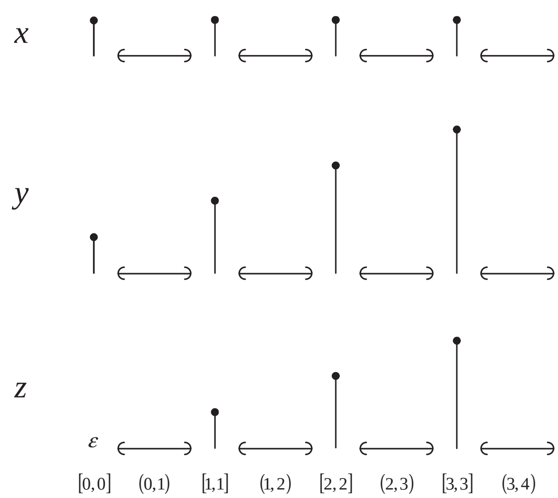

Figure 5.5. Segmentations of the signals in figure 5.3 produced by the synchronous DE simulation strategy. The signal z is absent at time $t = 0$, represented by the symbol ε as the first segment of z.

Given a DE process network that satisfies the conditions of theorem 3.24 and its input signals, the synchronous DE simulation of the network takes two kinds of steps that are interleaved.

- *Solve.* At a time t where at least one signal is present, a solve step computes the values of all signals at time t. Some signals may be absent. The solve steps produce segments over a single point in time.
- *Advance.* Determine the open interval (t, t') over which all signals are absent, but at least one signal is present at t'. Advance the simulation to t'. The advance steps produce segments over an open interval of time.

For an example of the solve step, consider the DE process network in figure 5.3 at time $t = 1$. With the input signal x equal to $clock_1$, the signal values $x(1)$, $y(1)$, and $z(1)$ satisfy the following equations,

$$
\begin{aligned}
&x(1) = 1, \\
&y(1) = z(1) +_{\varepsilon} x(1), \\
&z(1) = y(0).
\end{aligned}
$$

These equations can be solved by simple substitution. Although the network has a dependency loop, there is no circular dependency among the signal values at any time t. This is because the $Delay_1$ process is strictly causal and its output at t does not depend on its input at t, as shown by the following proposition.

Proposition 5.3. Let T be a totally ordered tag set and $P\colon \mathcal{S}(T, V) \to \mathcal{S}(T, V')$ a strictly causal process. For any $t \in T$ and signals $s_1, s_2 \in \mathcal{S}(T, V)$,

$$
s_1(t') = s_2(t'),\ \forall t' < t \implies P(s_1)(t) = P(s_2)(t). \tag{5.3}
$$

Proof. Let $D = \{t' \mid t' < t\}$, and signal $r = s_1\downarrow_D$. Note that if

$$
s_1(t') = s_2(t'),\ \forall t' < t,
$$

then $D \subseteq \text{dom}(s_1)$, and $D \subseteq \text{dom}(s_2)$. By the definition of strict causality on page 51,

$$D = \text{dom}(r) \subset \text{dom}(P(r)).$$

$\text{dom}(P(r))$ is a down-set of T, so $t \in \text{dom}(P(r))$. P is monotonic, so

$$r \preceq s_1 \implies P(r) \preceq P(s_1),$$

and $P(s_1)(t)$ equals $P(r)(t)$. Similarly $P(s_2)(t)$ equals $P(r)(t)$. □

5.2.1 Reactive and Proactive DE Processes

Continue with the above example. After the solve step at time 1, the advance step needs to determine the time $t' > 1$ such that the signals x, y, and z are absent over the time interval $(1, t')$, and at least one of them is present at t'. The input signal x is known to be absent over $(1, 2)$ and present at 2, so $t' \leq 2$. The signal y is now known over the time interval $[0, 1]$. By the definition of the Delay_1 process (page 44), z is known over the time interval $[0, 2]$, and is absent over $(1, 2)$ and present at 2. The *Add* process has the property that at any time, its output signal is present only if at least one of the input signals is present. Taken together these imply $t' = 2$.

Definition 5.4 (Reactive and Proactive DE Processes). Let $I \in \mathcal{I}(\mathbb{R})$ be an interval of real numbers, V and V' two non-empty sets of values, and $P\colon \mathcal{S}_{\text{d}}(I, V_\varepsilon) \to \mathcal{S}_{\text{d}}(I, V'_\varepsilon)$ a DE process. P is reactive if for every $s \in \mathcal{S}_{\text{d}}(I, V_\varepsilon)$,

$$\{t \in I \mid P(s)(t) \neq \varepsilon\} \subseteq \{t \in I \mid s(t) \neq \varepsilon\}. \tag{5.4}$$

P is proactive if it is not reactive.

The above definition can be straightforwardly generalized to multiple-input, multiple-output processes. Examples of reactive DE processes are *Add* and *Merge*. Delay_d and $\mathit{LookAhead}_a$ are proactive DE processes.

In the advance step, a synchronous DE simulation needs to consider only the proactive DE processes in the network to determine the time of the next solve step—the reactive

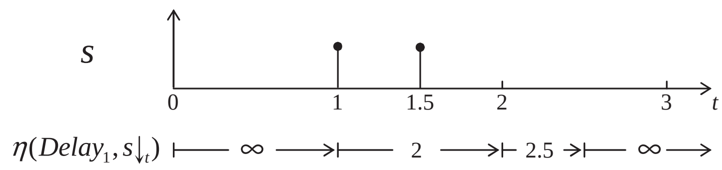

Figure 5.6. Next event time of the $Delay_1$ process with the given input.

processes cannot "spontaneously" produce events. Consider a proactive DE process P, and an input DE signal s that is known up to some time $t_0 \in T$, that is,

$$\mathrm{dom}(s) = \{t \in T \,|\, t \leq t_0\},$$

where T is the tag set of s. Note that s must be a finite signal. The **next event time** $\eta(P, s)$ is defined as follows. Let s' be the total signal that extends s by the "empty future,"

$$s'(t) = \begin{cases} s(t) & \text{if } t \leq t_0, \\ \varepsilon & \text{if } t > t_0. \end{cases} \tag{5.5}$$

s' is a finite DE signal. Let

$$E = \{t \in T \,|\, t > t_0,\ P(s')(t) \neq \varepsilon\}, \tag{5.6}$$

the set of times at which $P(s')$ is present after t_0. Because $P(s')$ is a DE signal, E has a minimum element if it is not empty. The next event time $\eta(P, s)$ is

$$\eta(P, s) = \begin{cases} \min E & \text{if } E \neq \emptyset, \\ \infty & \text{if } E = \emptyset. \end{cases} \tag{5.7}$$

Figure 5.6 illustrates the above definition with the $Delay_1$ process.

Consider a DE process network with m input signals s_l, $l = 1, \ldots, m$. Let P_k, $k = 1, \ldots, n$, be the n proactive processes in the network. Let r_k denote the input signal (tuple) of P_k. After a solve step at time t_0 during a synchronous DE simulation of the process network, the advance step that follows can determine the time t' of the next solve step as the minimum among

$$\eta(P_k, r_k\!\downarrow_{t_0}), k = 1, \ldots, n,$$

the next event time of the proactive processes given their input up to t_0, and the future times at which at least one input signal to the network is present—the elements of the set

$$\bigcup_{1 \leq l \leq m} \{t \mid t > t_0,\ s_l(t) \neq \varepsilon\}.$$

Equation 5.7 defines the next event time $\eta(P, s)$ from the behaviors of process P. Following are some examples of this definition in DE simulation tools and specification formalisms.

- The Synopsys VSS VHDL simulator [51, 74] provides a `cliGetNextEventTime()` function in its C-language interface. The function returns the time of the next simulation event.

- In the DEVS (Discrete Event Specification) formalism [80], atomic components provide a *time advance function*.

- In Ptolemy II [13, 14, 15], synchronous DE simulation is implemented by a set of Java classes in the package `ptolemy.domains.de.kernel`. The major class in this package is called `DEDirector`. An instance of this class, a DE director, controls the simulation of a DE process network. The processes are implemented by *actors*.

  The `DEDirector` class provides the following methods to handle next event time.

  - `fireAt(Time time, Actor actor)`

    An actor that implements a proactive DE process uses this method to inform the DE director of its next event time.

  - `getModelNextIterationTime()`

    A DE system model in Ptolemy II can be structured hierarchically. This method aggregates the next event time of the proactive DE processes in a sub-network by taking their minimum.

5.3 Asynchronous DE Simulation

In an asynchronous DE simulation of a DE process network, each DE process is simulated by a corresponding computational process (or thread). The computational processes

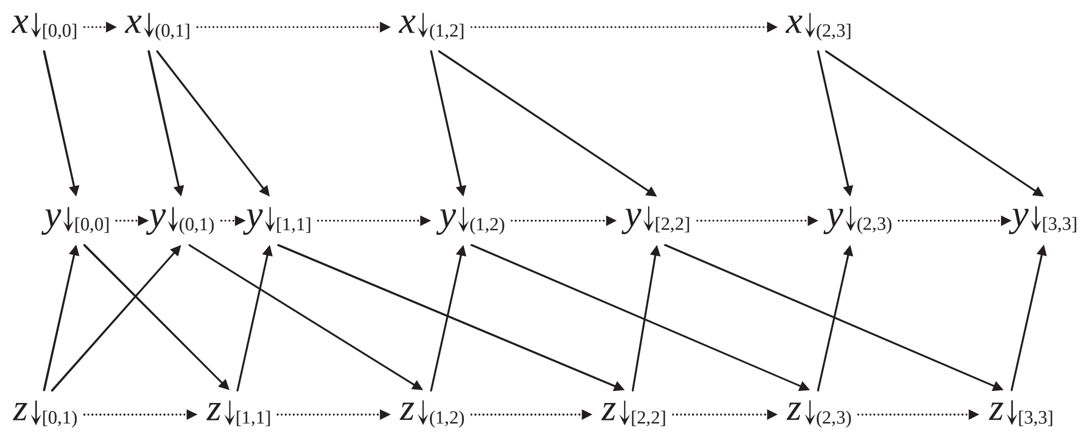

Figure 5.7. The signal segmentations produced by an asynchronous DE simulation. The dotted arrows represent the order in which the segments of one signal are produced and consumed. The solid arrows represent the dependency of an output signal segment on an input signal segment.

execute in parallel, and communicate asynchronously by messages that represent DE signal segments. This simulation strategy was originally proposed to parallelize or distribute large-scale simulation tasks [5, 22, 31].

Consider the DE process network in figure 5.3, with the input signal x equal to the $clock_1$ signal. Figure 5.7 shows an example of the signal segmentations produced by an asynchronous DE simulation of the network. Only the signal segments within the time interval $[0, 3]$ are included in the figure.

The segments of signal x are provided as external stimuli to the simulation. Each segment of x has exactly one present event at the end. The process $Delay_1$ is non-strict,

$$Delay_1(s_\perp) = (\mathbb{R}_0, [0, 1), \emptyset),$$

so the first segment of its output signal z,

$$z\downarrow_{[0,1)} = Delay_1(s_\perp),$$

does not depend on any input signal segment.

Given a DE process P, the program of the computational process that simulates P can be derived from the LTS L_P defined in section 5.1. Such a pseudocode program is shown in figure 5.8. Figure 5.9 shows some initial steps in executing this pseudocode program

Let state $s = s_\perp$.

If P is non-strict, produce the output segment $P(s_\perp)$.

Loop:

> Consume an input segment g.
>
> Determine the transition $s \xrightarrow{(g,h)} s'$ in L_P.
>
> If h is not empty, produce the output segment h.
>
> Let state $s = s'$.

Figure 5.8. Pseudocode program to simulate a DE process P.

input segment g	state s	next state s'	output segment h
	$s_\perp$		$z\downarrow_{[0,1)}$
$y\downarrow_{[0,0]}$	$s_\perp$	$y\downarrow_{[0,0]}$	$z\downarrow_{[1,1]}$
$y\downarrow_{(0,1)}$	$y\downarrow_{[0,0]}$	$y\downarrow_{[0,1)}$	$z\downarrow_{(1,2)}$
$y\downarrow_{[1,1]}$	$y\downarrow_{[0,1)}$	$y\downarrow_{[0,1]}$	$z\downarrow_{[2,2]}$
$y\downarrow_{(1,2)}$	$y\downarrow_{[0,1]}$	$y\downarrow_{[0,2)}$	$z\downarrow_{(2,3)}$

Figure 5.9. Steps in simulating the $Delay_1$ process. The second row corresponds to producing the initial output segment $Delay_1(s_\perp)$.

for the $Delay_1$ process. If the process P has multiple input signals, a segment from one of the input signals is consumed in each iteration of the program. A tuple with all empty segments except the one just consumed is formed to determine the transition in L_P. Figure 5.10 shows some initial steps in the simulation of the *Add* process.

If a DE process network satisfies the assumptions of theorem 3.24, and every process in the network is simulated according to the program in figure 5.8, and every output segment produced is eventually consumed by the receiving process, then theorem 3.24 guarantees that the asynchronous DE simulation of the process network will not deadlock. Because the assumptions of theorem 3.24 are weaker than previous approaches, such as [58], the

input segment g	state s	next state s'	output segment h
$z\downarrow_{[0,1)}$	$(s_\perp, s_\perp)$	$(z\downarrow_{[0,1)}, s_\perp)$	
$x\downarrow_{[0,0]}$	$(z\downarrow_{[0,1)}, s_\perp)$	$(z\downarrow_{[0,1)}, x\downarrow_{[0,0]})$	$y\downarrow_{[0,0]}$
$x\downarrow_{(0,1]}$	$(z\downarrow_{[0,1)}, x\downarrow_{[0,0]})$	$(z\downarrow_{[0,1)}, x\downarrow_{[0,1]})$	$y\downarrow_{(0,1)}$
$z\downarrow_{[1,1]}$	$(z\downarrow_{[0,1)}, x\downarrow_{[0,1]})$	$(z\downarrow_{[0,1]}, x\downarrow_{[0,1]})$	$y\downarrow_{[1,1]}$
$z\downarrow_{(1,2)}$	$(z\downarrow_{[0,1]}, x\downarrow_{[0,1]})$	$(z\downarrow_{[0,2)}, x\downarrow_{[0,1]})$	
$x\downarrow_{(1,2]}$	$(z\downarrow_{[0,2)}, x\downarrow_{[0,1]})$	$(z\downarrow_{[0,2)}, x\downarrow_{[0,2]})$	$y\downarrow_{(1,2)}$
$z\downarrow_{[2,2]}$	$(z\downarrow_{[0,2)}, x\downarrow_{[0,2]})$	$(z\downarrow_{[0,2]}, x\downarrow_{[0,2]})$	$y\downarrow_{[2,2]}$

Figure 5.10. Steps in simulating the *Add* process.

asynchronous DE simulation strategy is made applicable to a larger set of DE process networks.

Chapter 6

Conclusion

This dissertation studies the semantic foundation of the tagged signal model. The approach originates from a simple observation—the derivation of the Kahn process network semantics is valid as long as:

- the set of all signals that can be communicated through a channel between two processes is a complete partial order;
- the processes are Scott continuous functions from their input signals to output signals.

This started the research to establish the mathematical structure of tagged signals, and the rest is summarized in the following section.

6.1 Summary of Results

The fundamental concepts of the tagged signal model—signals, processes, and networks of processes—are formally defined in chapter 2. The order structure of signals is established. The key results are that for any partially ordered set of tags T and any set of values V, the set of signals $\mathcal{S}(T, V)$ is both a complete lower semilattice (lemma 2.17) and a complete partial order (lemma 2.16). The latter result leads to a direct generalization of Kahn process networks to tagged process networks (theorem 2.37). Few assumptions are made on the tags of signals when developing the results in chapter 2. This makes the results applicable to any model of computation specified in the tagged signal model framework.

Chapter 3 focuses on a subclass of tagged process networks in which all signals share the same totally ordered tag set. The common notion of causality is formally defined, and conditions are developed that guarantee the causality of timed process networks (theorem 3.11). The discreteness of timed signals is defined as being approximable by finite timed signals. Several characterizations of discrete event signals are compared and are shown to be equivalent. For any totally ordered tag set T and value set V, the set of discrete event signals $\mathcal{S}_{\mathrm{d}}(T, V_\varepsilon)$ is a sub-CPO of the set of timed signals $\mathcal{S}(T, V_\varepsilon)$ (lemma 3.17). The combination of the causality and the discreteness assumptions is proved to guarantee the non-Zenoness of timed process networks (theorem 3.24).

Chapter 4 explores the metric structure of tagged signals. Properties of the Cantor metric and its extensions to alternative tag sets and super-dense time are analyzed. The relations between the metric-theoretic and order-theoretic notions of convergence and finite approximation are determined. The main contribution of this chapter is the proposed generalized ultrametric on tagged signals (lemma 4.14). The generalized ultrametric provides a foundation for defining more specialized metrics on tagged signals. It also paves the way to apply the many research results in generalized ultrametric spaces to the tagged signal model.

Chapter 5 presents a formulation of tagged processes as labeled transition systems. This formulation provides a framework for comparing different implementation or simulation strategies for tagged processes. Two discrete event simulation strategies are studied using this framework. For synchronous discrete event simulation, the handling of dependency loops and the advancing of simulation time are derived from the behaviors of discrete event processes. For asynchronous discrete event simulation, results from chapter 3 are used to show that the simulation computes the correct network behavior in the limit.

6.2 Future Work

Mathematical structure of tagged signals. Several developments in this dissertation follow a similar trajectory. The first step is to study the properties of a mathematical structure of (sets of) signals; the second step is to use this structure to characterize the processes that are functions on signal sets, such as continuity and causality; and the third step is to determine conditions under which the characterizations are compositional. The main structures studied in this dissertation are complete partial orders and generalized ultrametric spaces. Many more sophisticated structures have been developed in the order theory [24] in mathematics and domain theory [2] in computer science. Future research in this direction may start with answering questions like under what conditions on the tag set T and value set V the set of signals $\mathcal{S}(T, V)$ is a *continuous domain* or an *algebraic domain*? What are the *compact* elements in $\mathcal{S}(T, V)$?

Space-time. In many physical processes, the physical quantities involved are functions of both space and time, such as electric and magnetic fields. The tag set of these field signals is $\mathbb{R}^3 \times \mathbb{R}$, where the first component is the 3-dimensional space and the second component is time. A partial order on this tag set can be defined by

$$(\vec{x}_1, t_1) \leq (\vec{x}_2, t_2) \iff \frac{\|\vec{x}_1 - \vec{x}_2\|}{c} \leq t_2 - t_1, \tag{6.1}$$

where c is the speed of light. By this order, $(\vec{x}_1, t_1)$ is below $(\vec{x}_2, t_2)$ if and only if $(\vec{x}_2, t_2)$ is on or inside the *future light cone* of $(\vec{x}_1, t_1)$. The spatial component of the tag set may be replaced by an abstract set L of locations, for example, to represent the nodes in a communication network. With such tag sets, the tagged signal model can be used to study computational processes over space and time. The specification, simulation, and implementation of sensor network applications [8, 81] may benefit from such studies.

Polymorphic implementation. Given a monotonic tagged process $P\colon \mathcal{S}(T_1, V_1) \to \mathcal{S}(T_2, V_2)$ and an input signal $s \in \mathcal{S}(T_1, V_1)$, the labeled transition system L_P defined in section 5.1 represents the decompositions of the mapping $s \mapsto P(s)$ into incremental steps as traces in L_P from the initial state $s_\perp$ to s. The labels in L_P are pairs of an input signal segment and an output signal segment. A program I that implements the process P must have these segments mapped into data structures that are manipulated by the program. The program I can be specified as an LTS L_I as follows.

- Σ_I is the set of program states. Elements of Σ_I are denoted by p, with subscripts as needed.
- $A \times B$ is the set of labels, where A is the set of values of the input data type, and B is the set of values of the output data type.
- Given two program states $p_1, p_2 \in \Sigma_I$, an input value $a \in A$, and an output value $b \in B$, $(p_1, (a, b), p_2)$ is a transition in L_I if and only if when I is in state p_1 and is invoked with the input value a, it produces the output value b and changes its program state to p_2.
- The initial state is an element $p_0 \in \Sigma_I$.

The following pair of maps establish the relation between L_P and L_I.

$$M_i \colon \mathcal{G}(T_1, V_1) \rightharpoonup A \tag{6.2}$$

maps (a subset of) input signal segments to the input values of program I. Note that M_i is a partial function, so it may place constraints on the segmentation of input signals.

$$M_o \colon \mathcal{S}(T_2, V_2) \times B \rightharpoonup \mathcal{G}(T_2, V_2) \tag{6.3}$$

maps the output values of program I to output signal segments, depending on the past output of the process. For any $s' \in \mathcal{S}(T_2, V_2)$ and $b \in B$, if $M_o(s', b)$ is defined, then $M_o(s', b) \in \mathcal{F}(s')$. It is important to emphasize that the maps M_i and M_o do not depend on the behaviors of the process P or program I, but only on the signals $\mathcal{S}(T_2, V_2)$, the segments $\mathcal{G}(T_1, V_1)$ and $\mathcal{G}(T_2, V_2)$, and the data types A and B.

Given L_P, L_I, and the maps M_i and M_o, the program I **implements** the process P if and only if for every trace

$$s_\perp \xrightarrow{(g_1, h_1)} s_1 \xrightarrow{(g_2, h_2)} s_2 \to \cdots \to s_{n-1} \xrightarrow{(g_n, h_n)} s_n \tag{6.4}$$

in L_P such that $M_i(g_k)$ is defined for $k = 1, \ldots, n$, the following

$$p_0 \xrightarrow{(M_i(g_1), b_1)} p_1 \xrightarrow{(M_i(g_2), b_2)} p_2 \to \cdots \to p_{n-1} \xrightarrow{(M_i(g_n), b_n)} p_n \tag{6.5}$$

is a trace in L_I and

$$M_o(s'_{k-1}, b_k) = h_k,\ k = 1, \ldots, n, \tag{6.6}$$

where

$$s'_k = P(s_\perp) \ll h_1 \ll \cdots \ll h_k,\ k = 1, \ldots, n.$$

It is understood that $s'_0 = P(s_\perp)$. This relation is analogous to the classical simulation relation between labeled transition systems [57].

A program I may implement multiple processes defined in different models of computation as illustrated below.

$$L_P \underset{M_i}{\overset{M_o}{\leftrightarrows}} L_I \underset{M'_i}{\overset{M'_o}{\rightleftarrows}} L_Q\,. \tag{6.7}$$

By defining the maps M_i and M_o, and M'_i and M'_o appropriately, the input and output signal segments associated with processes P and Q are mapped to the same data types manipulated by program I. This approach to reuse is a major research topic of the Ptolemy project [49], and is called *domain polymorphism*. It is the guiding principle in designing the actor library in Ptolemy II. The above formulation is an initial proposal to formalize the design practices. Further research may start from defining the maps M_i and M_o for the various models of computation implemented in Ptolemy II, and studying the properties of the implementation relation in equations 6.5 and 6.6. Programs can be written in the *CAL* actor language [27, 37], which defines its actor model using labeled transition systems and is well integrated into Ptolemy II.

Bibliography

[1] Harold Abelson and Gerald Jay Sussman, with Julie Sussman. *Structure and Interpretation of Computer Programs (2nd ed.)*. The MIT Press, 1996.

[2] Samson Abramsky and Achim Jung. Domain theory. In *Handbook of Logic in Computer Science (vol. 3): Semantic Structures*, pages 1–168. Oxford University Press, Oxford, UK, 1994.

[3] André Arnold and Maurice Nivat. Metric interpretations of infinite trees and semantics of non deterministic recursive programs. *Theoretical Computer Science*, 11:181–205, 1980.

[4] N. Aronszajn and P. Panitchpakdi. Extension of uniformly continuous transformations and hyperconvex metric spaces. *Pacific Journal of Mathematics*, 6(3):405–439, 1956.

[5] Rajive L. Bagrodia, K. Mani Chandy, and Jayadev Misra. A message-based approach to discrete-event simulation. *IEEE Transactions on Software Engineering*, 13(6):654–665, 1987.

[6] C. Baier and M. Majster-Cederbaum. Denotational semantics in the CPO and metric approach. *Theoretical Computer Science*, 135:171–220, 1994.

[7] Felice Balarin, Massimiliano Chiodo, Paolo Giusto, Harry Hsieh, Attila Jurecska, Luciano Lavagno, Claudio Passerone, Alberto Sangiovanni-Vincentelli, Ellen Sentovich, Kei Suzuki, and Bassam Tabbara. *Hardware-Software Co-Design of Embedded Systems : The POLIS Approach*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[8] Philip Baldwin, Sanjeev Kohli, Edward A. Lee, Xiaojun Liu, and Yang Zhao. Modeling of sensor nets in Ptolemy II. In *Information Processing in Sensor Networks (IPSN)*, Berkeley, CA, USA, 2004.

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

[10] Albert Benveniste. Compositional and uniform modeling of hybrid systems. In *Hybrid Systems*, pages 41–51, 1995.

[11] Frederick C. Berry, Philip S. DiPiazza, and Susan L. Sauer. The future of electrical and computer engineering education. *IEEE Transactions on Education*, 46(4):467–476, Nov 2003.

[12] Garrett Birkhoff. *Lattice Theory*. American Mathematical Society, 3rd edition, 1967.

[13] C. Brooks, E. A. Lee, X. Liu, S. Neuendorffer, Y. Zhao, and H. Zheng. Heterogeneous concurrent modeling and design in Java (volume 1: Introduction to Ptolemy II). Technical Report UCB/ERL M05/21, University of California, July 2005.

[14] C. Brooks, E. A. Lee, X. Liu, S. Neuendorffer, Y. Zhao, and H. Zheng. Heterogeneous concurrent modeling and design in Java (volume 2: Ptolemy II software architecture). Technical Report UCB/ERL M05/22, University of California, July 2005.

[15] C. Brooks, E. A. Lee, X. Liu, S. Neuendorffer, Y. Zhao, and H. Zheng. Heterogeneous concurrent modeling and design in Java (volume 3: Ptolemy II domains). Technical Report UCB/ERL M05/23, University of California, July 2005.

[16] Frederick P. Brooks Jr. *The Mythical Man-Month: Essays on Software Engineering.* Addison-Wesley, 1995.

[17] Joseph Buck, Soonhoi Ha, Edward A. Lee, and David G. Messerschmitt. Ptolemy: a framework for simulating and prototyping heterogeneous systems. In *Readings in Hardware/Software Co-Design*, pages 527–543. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[18] Joseph Buck and Radha Vaidyanathan. Heterogeneous modeling and simulation of embedded systems in El Greco. In *CODES '00: Proceedings of the 8th International Workshop on Hardware/Software Codesign*, pages 142–146, New York, NY, USA, 2000. ACM Press.

[19] Cristian S. Calude, Solomon Marcus, and Ludwig Staiger. A topological characterization of random sequences. *Information Processing Letters*, 88(5):245–250, 2003.

[20] Christos G. Cassandras. *Discrete Event Systems: Modeling and Performance Analysis.* Richard D. Irwin Publ., 1993.

[21] Adam Cataldo, Edward A. Lee, Xiaojun Liu, Eleftherios D. Matsikoudis, and Haiyang Zheng. Discrete-event systems: Generalizing metric spaces and fixed point semenatics. Technical Report UCB/ERL M05/12, University of California, April 2005.

[22] K. M. Chandy and J. Misra. Asynchronous distributed simulation via a sequence of parallel computations. *Communications of the ACM*, 24(4):198–206, 1981.

[23] Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL '77: Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, pages 238–252, New York, NY, USA, 1977. ACM Press.

[24] B.A. Davey and H. A. Priestley. *Introduction to Lattices and Order (2nd ed.).* Cambridge University Press, 2002.

[25] Jaco de Bakker and Erik de Vink. *Control Flow Semantics.* MIT Press, Cambridge, MA, USA, 1996.

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

[27] Johan Eker and Jörn W. Janneck. CAL language report: Specification of the CAL actor language. Technical Report UCB/ERL M03/48, EECS Department, University of California, Berkeley, 2003.

[28] Johan Eker, Jörn W. Janneck, Edward A. Lee, Jie Liu, Xiaojun Liu, Josef Ludvig, Stephen Neuendorffer, Sonia Sachs, and Yuhong Xiong. Taming heterogeneity—the Ptolemy approach. *Proceedings of the IEEE, Special Issue on Modeling and Design of Embedded Software*, 91(1):127–144, January 2003.

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

[30] Peter Fritzson and Vadim Engelson. Modelica–a unified object-oriented language for system modelling and simulation. In *ECCOP '98: Proceedings of the 12th European Conference on Object-Oriented Programming*, pages 67–90, London, UK, 1998. Springer-Verlag.

[31] Richard M. Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10):30–53, 1990.

[32] Gerhard Gierz, Karl Heinrich Hofmann, Klaus Keimel, Jimmie D. Lawson, Michael Mislove, and Dana S. Scott. *Continuous Lattices and Domains*, volume 93 of *Encyclopedia of Mathematics and its Applications.* Cambridge University Press, 2003.

[33] Andrzej Granas and James Dugundji. *Fixed Point Theory.* Springer-Verlag, 2003.

[34] Yuri Gurevich. Evolving algebras 1993: Lipari Guide. In Egon Börger, editor, *Specification and Validation Methods*, pages 9–37. Oxford University Press, 1994.

[35] Carl Hewitt and Henry Baker Jr. Actors and continuous functionals. Technical Report 194, MITLCS, December 1977.

[36] Pascal Hitzler. *Generalized Metrics and Topology in Logic Programming Semantics.* PhD thesis, Department of Mathematics, National University of Ireland, University College Cork, Jan 2001.

[37] Jörn W. Janneck. Actors and their composition. *Formal Aspects of Computing*, 15(4):349–369, Dec 2003.

[38] David R. Jefferson. Virtual time. *ACM Transactions on Programming Languages and Systems*, 7(3):404–425, 1985.

[39] David R. Jefferson. Virtual time II: storage management in conservative and optimistic systems. In *PODC '90: Proceedings of the 9th Annual ACM Symposium on Principles of Distributed Computing*, pages 75–89, New York, NY, USA, 1990. ACM Press.

[40] Gilles Kahn. The semantics of a simple language for parallel programming. In J. L. Rosenfeld, editor, *Information Processing*, pages 471–475, Stockholm, Sweden, Aug 1974. North Holland, Amsterdam.

[41] Arjun Kapur. *Interval and Point-Based Approaches to Hybrid System Verification.* PhD thesis, Department of Computer Science, Stanford University, Sep 1997.

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

[43] E. A. Lee and T. M. Parks. Dataflow process networks. *Proceedings of the IEEE*, 83(5):773–801, 1995.

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

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

[46] Edward A. Lee. Overview of the Ptolemy project. Technical Report UCB/ERL M03/25, University of California, Berkeley, July 2003.

[47] Edward A. Lee and Pravin Varaiya. Introducing signals and systems—the Berkeley approach. In *Proceedings of the First Signal Processing Education Workshop (SPE2000)*, Oct 2000.

[48] Edward A. Lee and Pravin Varaiya. *Structure and Interpretation of Signals and Systems*. Addison-Wesley, 2003.

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

[50] Edward A. Lee and Haiyang Zheng. Operational semantics of hybrid systems. In *Hybrid Systems: Computation and Control: 8th International Workshop, HSCC 2005*, volume 3414 of *Lecture Notes in Computer Science*, pages 25–53, 2005.

[51] J. Liu, B. Wu, X. Liu, and E. A. Lee. Interoperation of heterogeneous CAD tools in Ptolemy II. In *Proc. SPIE Vol. 3680, Design, Test, and Microfabrication of MEMS and MOEMS*, pages 249–258, March 1999.

[52] Jie Liu and Edward A. Lee. A component-based approach to modeling and simulating mixed-signal and hybrid systems. *ACM Transactions on Modeling and Computer Simulation*, 12(4):343–368, 2002.

[53] Jie Liu and Edward A. Lee. On the causality of mixed-signal and hybrid models. In *6th International Workshop on Hybrid Systems: Computation and Control (HSCC '03)*, Prague, Czech Republic, 2003.

[54] Xiaojun Liu, Jie Liu, Johan Eker, and Edward A. Lee. Heterogeneous modeling and design of control systems. In Tariq Samad and Gary Balas, editors, *Software-Enabled Control: Information Technology for Dynamical Systems*, pages 105–122. Wiley-IEEE Press, 2003.

[55] Oded Maler, Zohar Manna, and Amir Pnueli. From timed to hybrid systems. In *Proceedings of the Real-Time: Theory in Practice, REX Workshop*, pages 447–484, London, UK, 1992. Springer-Verlag.

[56] Zohar Manna and Amir Pnueli. Verifying hybrid systems. In *Hybrid Systems*, pages 4–35, 1992.

[57] R. Milner. *Communication and Concurrency.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[58] Jayadev Misra. Distributed discrete-event simulation. *ACM Computing Surveys*, 18(1):39–65, 1986.

[59] National Research Council Staff. *Embedded Everywhere: A Research Agenda for Networked Systems of Embedded Computers.* National Academy Press, Washington, DC, USA, 2001.

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

[61] Rocco De Nicola and Frits Vaandrager. Three logics for branching bisimulation. *Journal of the ACM*, 42(2):458–487, 1995.

[62] F. Nicoli. Denotational semantics of a behavioral subset of VHDL. In *DATE '98: Proceedings of the Conference on Design Automation and Test in Europe*, pages 975–976, Washington, DC, USA, 1998. IEEE Computer Society.

[63] Alan V. Oppenheim, Ronald W. Schafer, and John R. Buck. *Discrete-Time Signal Processing (2nd ed.).* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1999.

[64] Alan V. Oppenheim, Alan S. Willsky, and S. Hamid Nawab. *Signals & systems (2nd ed.).* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[65] Robert Piloty and Dominique Borrione. The CONLAN project: Status and future plans. In *DAC '82: Proceedings of the 19th Conference on Design Automation*, pages 202–212, Piscataway, NJ, USA, 1982. IEEE Press.

[66] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C (2nd ed.): The Art of Scientific Computing.* Cambridge University Press, New York, NY, USA, 1992.

[67] Sibylla Priess-Crampe and Paulo Ribenboim. Logic programming and ultrametric spaces. *Rendiconti di Matematica, Serie VII*, 19:155–176, 1999.

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

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

[70] David A. Schmidt. *Denotational Semantics: A Methodology for Language Development.* William C. Brown Publishers, Dubuque, IA, USA, 1986.

[71] Dana S. Scott. Logic and programming languages. *Communications of the ACM*, 20(9):634–641, 1977.

[72] Dana S. Scott. Domains for denotational semantics. In *Proceedings of the 9th Colloquium on Automata, Languages and Programming*, pages 577–613, London, UK, 1982. Springer-Verlag.

[73] Joseph E. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory.* MIT Press, Cambridge, MA, USA, 1977.

[74] Synopsys, Inc. *VHDL System Simulator C-Language Interface.* Oct 1999.

[75] Paul Taylor. *Practical Foundations of Mathematics.* Cambridge University Press, 1999.

[76] C. J. van Rijsbergen, Viggo Stoltenberg-Hansen, Ingrid Lindström, and Edward R. Griffor. *Mathematical Theory of Domains.* Cambridge University Press, New York, NY, USA, 1994.

[77] Glynn Winskel. *The Formal Semantics of Programming Languages: An Introduction.* MIT Press, Cambridge, MA, USA, 1993.

[78] Robert Kim Yates. Networks of real-time processes. In *International Conference on Concurrency Theory*, Lecture Notes in Computer Science, pages 384–397, 1993.

[79] Robert Kim Yates and Guang R. Gao. A Kahn principle for networks of nonmonotonic real-time processes. In *Parallel Architectures and Languages Europe*, pages 209–227, 1993.

[80] Bernard P. Zeigler, Tag Gon Kim, and Herbert Praehofer. *Theory of Modeling and Simulation.* Academic Press, Inc., Orlando, FL, USA, 2000.

[81] Feng Zhao and Leonidas Guibas. *Wireless Sensor Networks: An Information Processing Approach.* Morgan Kaufmann, May 2004.

[82] Huibiao Zhu, Jonathan P. Bowen, and Jifeng He. From operational semantics to denotational semantics for Verilog. In *CHARME '01: Proceedings of the 11th IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, pages 449–466, London, UK, 2001. Springer-Verlag.