Deterministic Receptive Processes are Kahn Processes

Stephen A. Edwards* Olivier Tardieu
Columbia University, New York, New York
sedwards@cs.columbia.edu

Abstract

Deterministic asynchronous concurrent formalisms are valuable because determinism greatly simplifies the design and validation of such systems and most concurrent formalisms are nondeterministic.

This paper connects two of the more successful deterministic asynchronous formalisms: Kahn's dataflow networks and Josephs's deterministic receptive processes. The main result: a divergence-free deterministic receptive process is a Kahn process in that it can be modeled by a continuous function from input to output sequences, thus verifying it is compositionally deterministic.

This result provides a bridge between two communities, enabling results from the asynchronous digital hardware community to be used in the context of dataflow computation and vice versa.

1 Introduction

One of the problems with concurrency is that there is so much of it. Many different models of concurrency have been developed across disciplines as diverse as asynchronous VLSI circuits and distributed multiprocessors. This has lead researchers in different disciplines to re-invent hard-won properties and insights.

To help avoid more of this, this paper relates two successful models of concurrent processes that have the very desirable property of determinism: Kahn's processes [10], the model underlying most dataflow computation, and Josephs's receptive processes [6], a popular model for asynchronous VLSI circuits that is closely related to Hoare's Communicating Sequential Processes [4]. We specifically consider Josephs's deterministic receptive processes [7], whose behaviors are constrained to make them deterministic under composition. Kahn's processes also have this property but the constraint is expressed more easily.

Why demand determinism? Certainly, nondeterministic models provide both implementation flexibility and the power to model nondeterministic systems. Simpler verification is the main argument: determinism leads to reproducible simulation behavior and simpler models for formal methods. We believe it is no accident that the two most popular computational models in hardware and software today (i.e., synchronous digital logic and single-threaded imperative programs) are both deterministic. And while it is possible to simulate a nondeterministic system and systematically explore all its behaviors (the simulator of Janin et al. [5], for example, takes snapshots when a nondeterministic choice is made and allows the user to restart the simulation from these points with a different choice), this adds an enormous burden to the already difficult validation task.

At some level, the two concurrency models discussed here are similar: both observe a stream of input events from the environment and produce output events in response. Both are deterministic: a process is compelled to produce the same outputs in response to the same inputs. When composed in parallel, groups of processes remain deterministic.

Details of the models differ greatly. Events travel through unbounded buffers between Kahn processes. Each process may be connected to many buffers, and each buffer has exactly one driver and one receiver. By contrast, a receptive process sees a single, unbuffered stream of input events and produces a single output stream. Since the input stream may come from multiple processes, event order may vary, yet the determinism constraints ensure that the process cannot respond differently under different input arrival orders.

The biggest difference is that receptive processes characterize the input patterns that are assumed never to be observed. Mainly to simplify the mathematics, Josephs models these conditions as "divergences:" by allowing the process to do anything (produce a sequence of outputs and allow any inputs) past that point. This behavior is inherently nondeterministic and has no analog in Kahn's networks, whose processes behave like politicians: unwanted inputs are simply ignored and remain in the input buffer.

The next section presents a motivating example that suggests the main result of the paper—a proof that deterministic receptive processes can be modeled as Kahn processes. The subsequent sections formally define the two process models and present a proof of their equivalence. The paper concludes with a review of related work and a discussion of the implications of the main result.

*Edwards and his group are supported by an NSF CAREER award, a grant from Intel corporation, an award from the SRC, and by New York State's NYSTAR program.

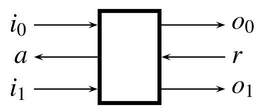

Figure 1: An asynchronous protocol converter.

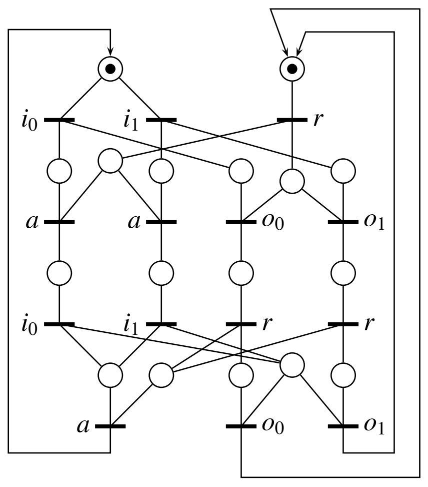

Figure 2: A Petri net describing the protocol converter.

2 An Example

In this section, we discuss the simple process in Figure 1 to illustrate the two concurrent models and the intuition behind the proof of their equivalence in Sections 6 and 7.

The process in Figure 1 is an asynchronous buffer that converts a passive input port (a "push" port) to a passive output port (a "pull" port). Each event models a voltage change on a wire, so the buffer actually looks for pairs of events. The left-side ports repeatedly wait for a rising event on either i_0 or i_1 (signifying 0 and 1 respectively), acknowledge it with a rising event on a, wait for a falling event on the same input channel (i_0 or i_1), and finally acknowledge it with a falling event on a.

Simultaneously, the ports on the right wait for an event on r, generate an event on either o_0 or o_1 depending on the value received on the left ports, wait for another event on r, and finally generate a matching event on either o_0 or o_1. One possible sequence of events for this process is

$$i_0\ r\ o_0\ a\ i_0\ r\ o_0\ a\ i_1\ r\ o_1\ a\ i_1\ r\ o_1\ a\ i_0\ r\ o_0\ a\ i_0\ r\ o_0\ a.$$

The Petri net in Figure 2 characterizes this process. The places and transitions on the left manage the input port and those on the right manage the output port. The semantics of this capture the notion, for example, that r and i_0 or i_1 may occur independently at the beginning and that once an i_0 is received, both a and o_0 are triggered when r arrives.

A Petri net provides an operational way to characterize the behavior of an asynchronous process. By contrast, Josephs characterizes his processes with an infinite set of valid event sequences. Technically, Josephs processes are triples (I, O, F). For this example, the input alphabet $I = \{i_0, i_1, r\}$, the output alphabet $O = \{o_0, o_1, a\}$, and the set of failures[1] F are all sequences that end with the process blocked waiting for more input plus all those that have an illegal input at a certain point and an arbitrary string of inputs and outputs after. Josephs intends these latter sequences, called divergences, to model undesired input events, implicitly assuming they would never occur in practice.

This process has an infinite number of failures:

$$F = \left\{ \begin{array}{l} \epsilon \text{ (the empty sequence)} \\ i_0 \\ i_0\ r\ a\ o_0 \\ i_0\ r\ o_0\ a \\ r\ i_0\ a\ o_0 \\ r\ i_0\ o_0\ a \\ i_0\ r\ a\ o_0\ i_0 \\ i_0\ r\ a\ o_0\ r \\ i_0\ r\ a\ o_0\ i_0\ r\ a\ o_0 \\ i_0\ r\ a\ o_0\ i_0\ r\ o_0\ a \\ i_0\ r\ a\ o_0\ r\ i_0\ a\ o_0 \\ i_0\ r\ a\ o_0\ r\ i_0\ o_0\ a \\ \qquad\quad \vdots \end{array} \right\}$$

It turns out that this process is both receptive and deterministic by Josephs's definitions (see Section 4), which means composing it in parallel with other deterministic receptive processes preserves its deterministic character. The proof of this is subtle [7] and detailed, but this paper implies a simpler proof, which follows from modeling this process using Kahn's formalism [10], suffices.

We find these relationships among the number of events in each failure:

$$\begin{aligned} o_0 &= \max(r, i_0), \\ o_1 &= \max(r, i_1), \text{ and} \\ a &= \max(r, i_0 + i_1). \end{aligned}$$

Obviously, this model is more abstract than either the Petri net or Josephs's failures, but if the environment behaves as these other models specify, these three equations are enough to constrain the outputs.

These three equations also characterize the behavior of a Kahn process that takes three input streams, one stream for each type of input event, and produces three output streams, one for each output event. In Kahn's formalism, the order in which inputs arrive and outputs are produced on separate channels is irrelevant: a process is simply a function

[1]Failures are sequences that bring the process to a point where it will fail to produce more output without additional input.

that maps input streams to output streams. A Kahn process must behave like a continuous function: applying an infinite sequence to the process must produce the same result as the limit of applying finite prefixes of the same sequence. Providing such a continuous process with more inputs can only produce the same or more output (it is monotonic), and its behavior on infinite inputs may not be somehow unexpected. The addition and maximum operators in the above equations give a continuous function when extended in the obvious way to infinite sequences.

It is no accident that this deterministic receptive process can also be modeled as a Kahn process (i.e., as a continuous mapping from input streams to output streams). In the remainder of this paper, we present both models formally and prove in Section 7 that this can always be done, i.e., that any deterministic receptive process is also a Kahn process.

3 Finite and Infinite Sequences

Throughout this paper, we will be dealing with both finite and infinite sequences of symbols. If A is an alphabet, we write A^* (the Kleene closure) for the set of all finite-length sequences of symbols from the alphabet and A^ω for the set of all finite- and infinite-length sequences.

In this setting, the distinction between unbounded-length sequences (A^*) and truly infinite sequences (A^ω) turns out to be a purely technical point that, unfortunately, requires a fair amount of machinery to prove the unsurprising result that receptive processes are well-behaved in the limit.

4 Josephs's Receptive Processes

Josephs's receptive processes [6] are a variant of Hoare's communicating sequential processes [4] that differ mainly in their treatment of unwelcome inputs. A particular behavior of a process in Josephs's model is a sequence of events: there are no ports, no buffers, and only a slight distinction between input and output events.

Josephs's formalism differs from Hoare's in how it imposes constraints on the environment (equivalently, how it deals with unwelcome inputs). By definition, a receptive process allows any input to arrive at any time, but after an unwelcome input arrives, Josephs models the process as becoming completely nondeterministic and is specifically allowed to babble endlessly, i.e., produce an endless stream of outputs without accepting any further input.

Formally, a receptive process is a triple (I, O, F), where $I \cap O = \emptyset$ are the input and output alphabets, $O \neq \emptyset$, and $F \subseteq (I \cup O)^*$ is the set of failures, i.e., sequences that represent behaviors of the system where it is waiting for additional input or has diverged because of an unwanted input.

To be a receptive process, failures must satisfy

$$s \in F\uparrow \Rightarrow st \in F\uparrow \tag{1}$$

$$F\uparrow \subseteq F \tag{2}$$

$$\epsilon \in \hat{F} \tag{3}$$

$$st \in \hat{F} \Rightarrow s \in \hat{F} \tag{4}$$

$$s \in \hat{F} \wedge t \in I^* \Rightarrow st \in \hat{F} \tag{5}$$

where $\uparrow$ and $\hat{}$ are the divergence and trace operators

$$F\uparrow = \{s \mid \{t \in O^* \mid st \in F\} \text{ is infinite}\} \tag{6}$$

$$\hat{F} = \{s \mid \exists t \in O^* \,.\, st \in F\}. \tag{7}$$

A divergence is a sequence after which the process can babble endlessly until the next failure; a trace is a sequence after which at least one output sequence leads to a failure.

These subtle definitions have intuitive consequences:

Lemma 1 (Josephs) *Any sequence may follow a divergence, i.e., $s \in F\uparrow$ iff $\forall t \,.\, st \in F$, and at least one may follow a trace, i.e., $s \in \hat{F}$ iff $\exists t \,.\, st \in F$.*

Corollary 1 *All failures are traces, i.e., $F \subseteq \hat{F}$.*

(1) requires that once a process has diverged, it will remain so. Furthermore,

Lemma 2 (Josephs) *Only unwelcome inputs can cause a divergence, i.e., $st \in F\uparrow \wedge t \in O^* \Rightarrow s \in F\uparrow$*

(2) requires all divergent sequences to themselves be failures, effectively meaning that a divergence may start doing anything immediately.

(3), which requires the empty sequence to be a trace, effectively says the process must start, so the first failure comes after a possibly empty sequence of outputs.

(4) requires the set of traces to be prefixed-closed, which just means that if a trace is allowed, every step necessary to get there is, too.

Finally, (5) makes processes receptive: adding any environmental inputs to the end of a trace always leads to another trace, although possibly a divergent one.

Lemma 3 *Given a trace $s \in F$ followed by an input burst $t \in I^*$, there exists some output burst $u \in O^*$ that will then bring the system to another failure state.*

Proof Since $s \in F$, from Corollary 1 $s \in \hat{F}$ and from (5), it follows that $st \in \hat{F}$ and hence from (7), $\exists u \in O^*$ such that $stu \in F$. □

The set of quiescent traces $\overline{F}$ are those failure traces that are not divergent, i.e., $\overline{F} = F \setminus F\uparrow$.

Lemma 4 *A quiescent trace cannot diverge after an output, i.e., $o \in O \wedge s \in \overline{F} \Rightarrow so \notin F\uparrow$.*

Proof By definition $s \in F$ and $s \notin F\uparrow$. From Lemma 2, it follows that $so \notin F\uparrow$. □

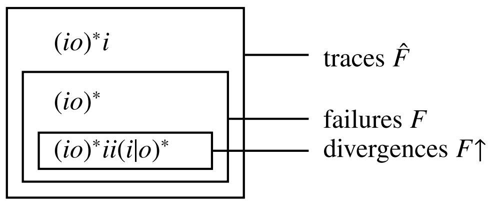

Figure 3: Failures, traces, and divergences expressed with regular expressions for a buffer modeled as a receptive process. The buffer has a single input i and a single output o, alternates between input and output events in normal operation, and diverges after an input glitch (two consecutive i events). The relationships among sets shown here hold for all receptive processes.

Figure 3 shows the failures, traces, and divergences of a receptive process for a simple buffer. Input event i causes output event o, and the next i is meant to come after o. Each event models a transition on a wire. Divergences are well-behaved until two input events occur in a row (i.e., a glitch shorter than the buffer's reaction time), then any sequence of inputs and outputs is allowed. Failures are exactly the divergences plus cases where the system is waiting for the next input. Finally, the traces characterizes all possible behaviors of the process: well-behaved then waiting for input, waiting for output, or having two consecutive inputs appear and then diverging. The one behavior prohibited in the trace set is the appearance of two consecutive outputs before two consecutive inputs have ever appeared. The relationships among the sets in Figure 3 holds for any receptive process.

4.1 Deterministic Receptive Processes

Receptive processes, as defined above, may be nondeterministic (Figure 4 shows an example). To address this, Josephs [7] gives four additional constraints that render receptive processes deterministic:

$$\begin{aligned}
&(\forall v, w \,.\, x = vw \Rightarrow svi \notin F\uparrow) \;\wedge \\
&\qquad i \in I \;\wedge\; sxiu \in F \quad\Rightarrow\quad sixu \in F && (8)\\
&o \in O \;\wedge\; t \in (I \cup (O \setminus \{o\}))^* \;\wedge \\
&\qquad so \in \hat{F} \;\wedge\; st \in F \setminus F\uparrow \quad\Rightarrow\quad \text{false} && (9)\\
&o \in O \;\wedge\; t \in (I \cup (O \setminus \{o\}))^* \;\wedge \\
&\qquad so \in \hat{F} \;\wedge\; st \in \hat{F} \quad\Rightarrow\quad sto \in \hat{F} && (10)\\
&o \in O \;\wedge\; t \in (I \cup (O \setminus \{o\}))^* \;\wedge \\
&\qquad so \in \hat{F} \;\wedge\; stou \in F \setminus F\uparrow \quad\Rightarrow\quad sotu \in F && (11)
\end{aligned}$$

(8) says a process can accept an input earlier without changing its behavior provided the input is legal (does not lead to divergence) at every intervening point.

(9) says that once an output can be emitted, it cannot be suppressed. Technically, if o can be emitted at a certain point, it cannot also be the case that another trace that does not include o can follow and lead to a non-divergent failure (i.e., one in which no more outputs will be emitted without additional inputs).

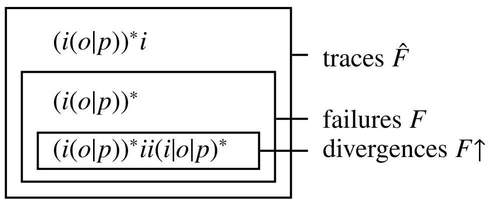

Figure 4: Sets for a nondeterministic receptive process: a buffer that produces either o or p in response to an i.

(10) is broader but weaker: if either an output o or a trace that excludes o may occur, o may still occur after t. Since the process is receptive, a similar property holds for inputs.

Finally, (11) says that if a process can emit o now or safely wait and emit it later (i.e., engage in a trace t that does not include o), emitting it now does not affect what can happen later.

In effect, these four constraints render input arrival and output production orders nearly irrelevant (not quite, though: outputs can not appear before the inputs that produce them). This insight will be crucial in later proofs, which start by discarding input and output event orders.

5 Kahn's Systems

Kahn's networks [10] consist of asynchronously-running concurrent processes that communicate exclusively through point-to-point unbounded first-in, first-out buffers. This is very different than Josephs, whose communication is nearly implicit (i.e., an event generated by one process is seen immediately by another). Kahn describes each process using an Algol-like single-threaded imperative programming language augmented with two constructs: nonblocking write to a particular channel, and a blocking read operation that will wait for an event to arrive on a channel if one is not already in the buffer for this channel.

Restricting reads to be blocking renders the behavior of each process to be a (continuous) function from input sequences to output sequences. By defining the behavior of an entire system as being the least fixed point of these continuous functions, it follows from a theorem usually attributed to Knaster and Tarski [11] that the least fixed point is unique and thus the system has exactly one behavior, i.e., is deterministic.

The functional nature of Kahn's processes is also very different than Josephs's, whose failure sets are more like relations, although we will show that the restriction of determinism implies they behave functionally.

Formally, Kahn starts with a set of events D, whose elements represent messages that can be passed through buffers, and represents the sequence of events flowing through a buffer as members of D^ω, the set of all finite and infinite sequences of events from D.

To relate different behaviors, Kahn uses the usual partial order on elements of D^ω, i.e., for $x, y \in D^\omega$, $x \sqsubseteq y$ when x is a prefix of, or equal to, y. Equivalently, $x \sqsubseteq y$ if y extends x with additional events.

A Kahn process is modeled as a function from a tuple of input sequences to a tuple of output sequences. Any process described in an imperative language with blocking reads and nonblocking writes defines such a continuous function: given (complete) input sequences, consider executing a process on them. The program runs until it hits a read operation, which because it blocks is guaranteed to always return the next event on its input channel. Moreover, further execution of the program can only append events to output channels, making the behavior monotonic. Finally, because there is no way for a process to wait forever before emitting any output, the function is continuous.

Although Kahn expresses processes in an imperative language, Kahn's proof of determinism only requires that each process behaves like a continuous function. In the proof presented here, we, too, only show that Josephs's deterministic receptive processes can be represented with continuous functions, and hence are deterministic using Kahn's argument. In fact, the single-threaded imperative programs Kahn suggests cannot represent many continuous functions on streams. A simple counterexample is the "two wires" process, which simply copies two input streams to two output streams. This cannot be represented with a single process in Kahn's language because it can only wait on a single input at a time, yet it is a continuous function.

6 Modeling Deterministic RPs with Kahn

This section and the next present the main contribution of this paper: a formal connection between Josephs's deterministic receptive processes and Kahn's processes. Specifically, a deterministic receptive process that never diverges (rejects an input) behaves like a Kahn process.

The proof proceeds in three steps. First, we define a way to model a receptive process (i.e., input/output alphabets and a set of failures) as a Kahn-like process where each input and output event travels through its own unique input or output port. Next, we show that the failure set of a deterministic receptive process implies a monotonic and hence functional relationship from finite input sequences to output sequences (i.e., given a particular input sequence, there is exactly one corresponding output sequence). Finally, in the next section, we show that this function is continuous when extended to infinite sequences, completing the proof that the model is a Kahn process.

The way we model deterministic receptive processes as Kahn processes is key. As mentioned earlier, Josephs's restrictions on deterministic processes demand that the process not be sensitive to input arrival order and produce arbitrary output orders. To achieve this, we model a deterministic receptive process as a Kahn-like process in which each input and output event gets a unique port. This discards order and effectively reduces information about each event to a single natural number: the count of the number of times it has arrived or been emitted.

Formally, we use the projection operator $\downarrow$ to transform a sequence of events into separate streams. It is defined for a sequence s and set A of events as follows: for the empty sequence, $\epsilon \downarrow A = \epsilon$, and

$$as \downarrow A = \begin{cases} a(s \downarrow A) & \text{if } a \in A, \text{ and} \\ s \downarrow A & \text{otherwise.} \end{cases}$$

We also use the obvious singleton and set-valued extensions: for a single event $a \in A$, $s \downarrow a = s \downarrow \{a\}$, and for a set S of sequences of events, we define $S \downarrow A = \{s \downarrow A : s \in S\}$.

Let $\mathcal{I}$ and $\mathcal{O}$ be two functions that map sequences to vectors of inputs and outputs using the projection operator as follows. For a receptive process $P = (I, O, F)$ where $I = \{i_1, \ldots, i_p\}$ and $O = \{o_1, \ldots, o_q\}$, if $f \in F$,

$$\begin{aligned} \mathcal{I}(f) &= (f \downarrow i_1, f \downarrow i_2, \ldots, f \downarrow i_p) \in i_1^* \times \ldots \times i_p^* \text{ and} \\ \mathcal{O}(f) &= (f \downarrow o_1, f \downarrow o_2, \ldots, f \downarrow o_q) \in o_1^* \times \ldots \times o_q^*. \end{aligned} \qquad (12)$$

For example, if $f = i_1 i_2 o_1 o_2 i_1 o_2 o_1 i_2 i_1 o_1 o_2$ then $\mathcal{I}(f) = (i_1 i_1 i_1, i_2 i_2)$ and $\mathcal{O}(f) = (o_1 o_1 o_1, o_2 o_2 o_2)$.

To show that a deterministic receptive process is a Kahn process, we first prove a lemma that shows the relationships in (12) behaves monotonically. As a corollary, it must also be a function. The proof, by contradiction, relies mostly on (8), which can here be interpreted as meaning more inputs can never produce fewer outputs.

Define the $\sqsubseteq$ operator on finite sequences as follows: $s \sqsubseteq t$ if $t = su$ for some possibly empty string u, i.e., if s is a prefix of t. This relation produces a partial order because it is reflexive ($x \sqsubseteq x$), transitive ($x \sqsubseteq y \land y \sqsubseteq z \Rightarrow x \sqsubseteq z$), and antisymmetric ($x \sqsubseteq y \land y \sqsubseteq x \Rightarrow x = y$). We extend it to vectors of sequences in the usual way: $(s_1, \ldots, s_n) \sqsubseteq (t_1, \ldots, t_n)$ if $s_1 \sqsubseteq t_1, \ldots, s_n \sqsubseteq t_n$.

Lemma 5 *The input/output relationship of a deterministic receptive process $P = (I, O, F)$ with no divergence is monotonic, i.e., for $f_1, f_2 \in F$, if $\mathcal{I}(f_1) \sqsubseteq \mathcal{I}(f_2)$ then $\mathcal{O}(f_1) \sqsubseteq \mathcal{O}(f_2)$.*

Proof by contradiction. Assume $f_1, f_2 \in F$ such that $\mathcal{I}(f_1) \sqsubseteq \mathcal{I}(f_2)$, but assume $\mathcal{O}(f_1) \not\sqsubseteq \mathcal{O}(f_2)$. Since the elements of $\mathcal{O}(f_1)$ and $\mathcal{O}(f_2)$ are each sequences of a single output event, this assumption means that some output o appears fewer times in f_2 than it does in f_1.

Since P has no divergences, repeated application of (8) on f_1 and f_2 means there exist failures f'_1 and f'_2 such that all the inputs events of f_1 occur before any outputs in f'_1 and f'_2 begins with the same sequence of input events.

Repeated application of (11) on f'_2 allows us to construct a trace f''_2 that shares a common prefix with f'_1. However, we will fail to construct an f''_2 that has (all of) f'_1 as a prefix since by assumption at least one output event appears less often in f_1 and hence in f'_1 since applying (8) does not affect the number of output events, than it does in f_2 and hence in f'_2 and f''_2 since (8) and (11) do not affect the number of output events.

The existence of f''_2, however, violates (9) since the sequence so begins f'_1 and f''_2 is of the form st, where t is a sequence that does not contain o (there were fewer o events in f_2). The assumption that $\mathcal{O}(f_1) \not\sqsubseteq \mathcal{O}(f_2)$ must therefore be false. □

Corollary 2 *For a deterministic receptive process $P = (I, O, F)$ with no divergences ($F\uparrow = \emptyset$), the input/output relationship is a function, i.e., for $f_1, f_2 \in F$, $\mathcal{I}(f_1) = \mathcal{I}(f_2)$ implies $\mathcal{O}(f_1) = \mathcal{O}(f_2)$.*

7 Extending Receptive Processes to Infinite Sequences

That processes behave monotonically is the most important property in making Kahn networks deterministic, but Kahn's proof requires the processes to be defined on both finite and and infinite sequences of inputs and to behave continuously. We showed in the last section that using the definitions of $\mathcal{I}$ and $\mathcal{O}$ in (12) gives a monotonic function, but it is not defined on infinite sequences. We shall extend this function to infinite sequences and show the resulting function is continuous.

We need some additional machinery. First, we extend $\sqsubseteq$ in the obvious way to infinite sequences. This makes $\sqsubseteq$ a complete partial order: any chain C (a totally ordered set: $c_1 \sqsubseteq c_2$ or $c_2 \sqsubset c_1$ for all $c_1, c_2 \in C$) has a (unique) least upper bound $\sqcup C$.

A function is continuous ("well-behaved in the limit") if the limit of its chains is the limit of the function on the chains, i.e., f is continuous iff for any chain C,

$$f(\sqcup C) = \sqcup\{f(c) : c \in C\}. \tag{13}$$

Our solution to extending receptive processes to infinite sequences is somehow obvious but non-trivial to prove: we define the input/output behavior of a process as being exactly the limit as defined by (13). Let $F' \subseteq F$ be a set of failures. If $C = \{\mathcal{I}(f') : f' \in F'\}$ is a chain (i.e., $c_1 \sqsubseteq c_2$ or $c_2 \sqsubseteq c_1$ for all $c_1, c_2 \in C$), then $C' = \{\mathcal{O}(f') : f' \in F'\}$ is also a chain by Lemma 5. We denote $\mathcal{F}$ the set of such sets of failures ($\mathcal{F} \subset \mathcal{P}(F)$), and for each $F' \in \mathcal{F}$, we define

$$\begin{aligned}\bar{\mathcal{I}}(F') &= \sqcup\{\mathcal{I}(f') : f' \in F'\} \text{ and}\\ \bar{\mathcal{O}}(F') &= \sqcup\{\mathcal{O}(f') : f' \in F'\}.\end{aligned} \tag{14}$$

This defines an input/output relation for all finite input sequences—choose a singleton for F'—and all infinite input sequences. For any (possibly infinite) input sequence $s \in i_1^\omega \times \ldots \times i_p^\omega$, there exists $F \in \mathcal{F}$ such that $s = \sqcup\mathcal{I}(F)$. This follows from the process being receptive. For instance, for the input sequence $(i_1i_1i_1, i_2^\infty, i_3i_3)$, consider a set of failures with input sequences $\{(i_1i_1i_1, i_2, i_3i_3), (i_1i_1i_1, i_2i_2, i_3i_3), (i_1i_1i_1, i_2i_2i_2, i_3i_3), \ldots\}$.

To show the relations in (14) define a continuous function from infinite input sequences to infinite output sequences, we first observe that if a finite input sequence is a prefix of the limit of a chain of (potentially infinite) input sequences then it is a prefix of one element of this chain. Intuitively, this means there is only one way to approach a limit.

Lemma 6 *If $X \subset i_1^\omega \times \ldots \times i_p^\omega$ is a chain, $y \in i_1^* \times \ldots \times i_p^*$, and $y \sqsubseteq \sqcup X$ then there exists $x \in X$ such that $y \sqsubseteq x$.*

Proof For a given input i_k, we have $(y \downarrow i_k) \sqsubseteq \sqcup(X \downarrow i_k)$. If $\sqcup(X \downarrow i_k)$ is a finite string s then $s \in (X \downarrow i_k)$. Hence, there exists $x_k \in X$ such that $(x_k \downarrow i_k) = s = \sqcup(X \downarrow i_k)$, so that $(y \downarrow i_k) \sqsubseteq (x_k \downarrow i_k)$. Otherwise, $(X \downarrow i_k)$ contains arbitrarily long and/or infinite sequences, thus there also exists $x_k \in X$ such that $(y \downarrow i_k) \sqsubseteq (x_k \downarrow i_k)$.

By applying this construction to all inputs, we obtain a set $\{x_1, \ldots, x_p\} \subset X$ totally ordered and finite. Let x be its maximum. For every input i_k, $(y \downarrow i_k) \sqsubseteq (x_k \downarrow i_k) \sqsubseteq (x \downarrow i_k)$. As a result, $y \sqsubseteq x$. □

The next lemma and corollary show that the relations in (14) actually represent a monotonic function.

Lemma 7 *The limiting input/output relation of a deterministic receptive process $P = (I, O, F)$ with no divergence is monotonic, i.e., for $F_1, F_2 \in \mathcal{F}$, if $\bar{\mathcal{I}}(F_1) \sqsubseteq \bar{\mathcal{I}}(F_2)$ then $\bar{\mathcal{O}}(F_1) \sqsubseteq \bar{\mathcal{O}}(F_2)$.*

Proof If $f_1 \in F_1$ then $\mathcal{I}(f_1) \sqsubseteq \bar{\mathcal{I}}(F_2)$. By Lemma 6, there exists $f_2 \in F_2$ such that $\mathcal{I}(f_1) \sqsubseteq \mathcal{I}(f_2)$. By Lemma 5, $\mathcal{O}(f_1) \sqsubseteq \mathcal{O}(f_2)$. Thus $\mathcal{O}(f_1) \sqsubseteq \bar{\mathcal{O}}(F_2)$. Finally, $\bar{\mathcal{O}}(F_1) \sqsubseteq \bar{\mathcal{O}}(F_2)$. □

Corollary 3 *The limiting input/output relation of a deterministic process $P = (I, O, F)$ with no divergence is a function, i.e., for $F_1, F_2 \in \mathcal{F}$, if $\bar{\mathcal{I}}(F_1) = \bar{\mathcal{I}}(F_2)$ then $\bar{\mathcal{O}}(F_1) = \bar{\mathcal{O}}(F_2)$.*

Finally we are in a position to prove our main result: that interpreting a divergence-free deterministic receptive process using the constructions in (12) and (14) gives a continuous function and can therefore be interpreted as a Kahn process.

Theorem 1 *The limiting input/output function of a deterministic receptive process $P = (I, O, F)$ with no divergence is continuous, i.e., for $\mathcal{F}_1 \subset \mathcal{F}$ and $\mathcal{F}_2 \subset \mathcal{F}$ such that $\{\bar{\mathcal{I}}(F_1) : F_1 \in \mathcal{F}_1\}$ and $\{\bar{\mathcal{I}}(F_2) : F_2 \in \mathcal{F}_2\}$ are totally ordered, if $\sqcup\{\bar{\mathcal{I}}(F_1) : F_1 \in \mathcal{F}_1\} = \sqcup\{\bar{\mathcal{I}}(F_2) : F_2 \in \mathcal{F}_2\}$ then $\sqcup\{\bar{\mathcal{O}}(F_1) : F_1 \in \mathcal{F}_1\} = \sqcup\{\bar{\mathcal{O}}(F_2) : F_2 \in \mathcal{F}_2\}$.*

Proof $\sqcup\{\bar{\mathcal{I}}(F_1) : F_1 \in \mathcal{F}_1\} \in i_1^\omega \times \ldots \times i_p^\omega$, thus there exists $F' \in \mathcal{F}$ such that $\sqcup\{\bar{\mathcal{I}}(F_1) : F_1 \in \mathcal{F}_1\} = \mathcal{I}(F')$.

If $f' \in F'$ then $\mathcal{I}(f') \sqsubseteq \sqcup\{\bar{\mathcal{I}}(F_2) : F_2 \in \mathcal{F}_2\}$. By Lemma 6, there exists $F_2 \in \mathcal{F}_2$ such that $\mathcal{I}(f') \sqsubseteq \bar{\mathcal{I}}(F_2)$. By Lemma 6 again, there exists $f_2 \in F_2$ such that $\mathcal{I}(f') \sqsubseteq \mathcal{I}(f_2)$. By Lemma 5, $\mathcal{O}(f') \sqsubseteq \mathcal{O}(f_2) \sqsubseteq \bar{\mathcal{O}}(F_2) \sqsubseteq \sqcup\{\bar{\mathcal{O}}(F_2) : F_2 \in \mathcal{F}_2\}$, so that $\bar{\mathcal{O}}(F') \sqsubseteq \sqcup\{\bar{\mathcal{O}}(F_2) : F_2 \in \mathcal{F}_2\}$.

If $F_1 \in \mathcal{F}_1$ then $\bar{\mathcal{I}}(F_1) \sqsubseteq \bar{\mathcal{I}}(F')$. By Lemma 7, $\bar{\mathcal{O}}(F_1) \sqsubseteq \bar{\mathcal{O}}(F')$. Hence, $\sqcup\{\bar{\mathcal{O}}(F_1) : F_1 \in \mathcal{F}_1\} \sqsubseteq \bar{\mathcal{O}}(F')$.

To conclude, $\sqcup\{\bar{\mathcal{O}}(F_1) : F_1 \in \mathcal{F}_1\} \sqsubseteq \sqcup\{\bar{\mathcal{O}}(F_2) : F_2 \in \mathcal{F}_2\}$. By symmetry, $\sqcup\{\bar{\mathcal{O}}(F_1) : F_1 \in \mathcal{F}_1\} = \sqcup\{\bar{\mathcal{O}}(F_2) : F_2 \in \mathcal{F}_2\}$. □

In summary, a divergence-free deterministic receptive process behaves like a Kahn process in which each type of input and output event is conveyed through its own separate buffer, an abstraction used to model a receptive process's ability to accept and generate events in any order without producing a fundamental change in behavior. Furthermore, the behavior of the process is "sampled" only at the failures: exactly when the receptive process has emitted all the outputs it is obligated to.

It it worth noting that Josephs's model of divergence does not have a direct analog in Kahn processes since it produces, by definition, a sequence of nondeterministic inputs and outputs. While this is largely a artifact of choosing a particular way to represent the arrival of an unwelcome input (Hoare uses a different approach that adds additional complexity to the mathematics), the issue of unwelcome inputs is handled very differently in the two models.

Kahn's processes in some sense simply ignore unwelcome inputs. If an undesired input is sent on a channel, the receiving process can simply choose never to read from that channel. In dataflow networks, this is somehow natural since the notion of violating a protocol is usually not an issue because the communication behavior of, say, a signal processing application is usually so simple that protocol failures are not usually a concern.

Josephs and others in the asynchronous VLSI community, by contrast, are often concerned exclusively with protocols and not with data communication per se because of the low level of abstraction necessary at the level of asynchronous digital logic. In fact, Josephs's failures are a mathematically elegant way of expressing both the input/output behavior of a process and the expectations it places on its environment. Kahn focused purely on input/output behavior.

8 Related Work

Udding [17] was one of the first to formalize the notion of delay-insensitivity for asynchronous circuits. Working in Van de Snepscheut's trace theory [22], Udding defined a set of constraints on traces much like those due to Josephs [7]. Although Udding did not provide a proof of compositionality like that of Josephs, the two formalisms are very similar. Josephs and Udding later collaborated on a delay-insensitive algebra for expressing trace sets [9]. Separately, Mallon and Udding [14] developed a tool for converting processes expressed in such an algebra to finite automata.

Concurrently, van Berkel et al. [20, 18, 19, 21] developed the notion of handshake circuits, an asynchronous design style that uses a library of processes that behave roughly like Josephs's deterministic receptive processes. With the exception of nondeterministic components such as arbiters, systems composed of handshake components are deterministic by construction. This work led to the Balsa/Tangram asynchronous circuit synthesis system [1] in commercial use at Philips. Josephs, Nowick, and van Berkel [8] provide a survey of this and other asynchronous design styles.

Janin, Bardsley, and Edwards [5] show how to simulate asynchronous circuits in Balsa. Their simulation runs until it encounters a nondeterministic choice, such as at an arbiter. It then takes a snapshot of the system state, makes a particular choice, and continues. Later, a user can restart the simulation from one of these choice points and proceed with a different choice. While restarting is a fairly unique ability for a simulator, it illustrates the challenges of simulating a nondeterministic formalism.

Others, including Peeters and van Berkel [16] and O'Leary and Brown [15] use the handshake circuit idea in a synchronous setting, illustrating its power as a model of computation beyond asynchronous implementations.

Dechering et al. [2] apply ideas from delay insensitivity theory to a particular model of embedded systems, but do little more than relate the two.

Kahn's 1974 paper [10] became the theoretical underpinning of most dataflow computing models. Relying heavily on Kahn's inherent determinism, Lee's Synchronous Dataflow [12] is a restricted class of Kahn networks whose behavior can be completely predicted at compile-time ideal for signal processing. Lee and Parks [13] present additional models based on Kahn's systems.

9 Conclusions

The main result of this paper is that Josephs's deterministic receptive processes behave like Kahn processes (specifically, divergence-free receptive processes: those that never consider any input patterns to be erroneous). The immediate result is that Kahn's simple proof of compositional determinism also applies to deterministic receptive processes, mirroring Josephs's more complex proof [7].

The equivalence of the two modeling styles—the trace-based style of receptive processes and the partial order and continuous function style of Kahn—is the broader result, enabling a variety of cross-pollination possibilities. Excellent work has been done on both of these models, but until now, few techniques have been transferred. Now, for example, the handshake circuit style developed in the asynchronous logic community [20] could be applied in a Kahn setting. Similarly, it might be possible to apply techniques and restrictions developed for dataflow networks [13] in the asynchronous logic world.

The original motivation for this paper was a desire to impose a deterministic model of computation on the author's SHIM language for hardware/software codesign [3], which currently consists of synchronous hardware interacting with single-threaded software. Each model by itself is deterministic, but their composition is not. The results here suggest that imposing either a Kahn or Josephs-like model on the language to guarantee determinism would be equivalent, and that ideas from both the asynchronous digital hardware world and the dataflow software world could be successfully combined.

Acknowledgements

Steve Nowick introduced me to Josephs's work. Josephs gave me insight into his motivation for divergences.

References

[1] Andrew Bardsley. Balsa: An asynchronous circuit synthesis system. Master's thesis, University of Manchester, England, 1998.

[2] Paul Dechering, Rix Groenboom, Edwin de Jong, and Jan Tijmen Udding. Formalization of a software architecture for embedded systems: a process algebra for SPLICE. In *Proceedings of the 32nd Hawaii International Conference on System Sciences*, Maui, Hawaii, January 1999.

[3] Stephen A. Edwards. SHIM: A language for hardware/software integration. In *Proceedings of SYNCHRON*, Schloss Dagstuhl, Germany, December 2004.

[4] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, Upper Saddle River, New Jersey, 1985.

[5] Lilian Janin, Andrew Bardsley, and Doug A. Edwards. Simulation and analysis of synthesised asynchronous circuits. *International Journal of Simulation Systems, Science & Technology*, 4(3–4):31–43, 2003.

[6] Mark B. Josephs. Receptive process theory. *Acta Informatica*, 29(1):17–31, February 1992.

[7] Mark B. Josephs. An analysis of determinacy using a trace-theoretic model of asynchronous circuits. In *Proceedings of the Ninth International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 121–130, Vancouver, BC, Canada, May 2003.

[8] Mark B. Josephs, Steven M. Nowick, and C. H. (Kees) van Berkel. Modeling and design of asynchronous circuits. *Proceedings of the IEEE*, 87(2):232–242, February 1999.

[9] Mark B. Josephs and Jan Tijmen Udding. An overview of D-I algebra. In *Proceedings of the 26th Hawaii International Conference on System Sciences*, volume I, pages 329–338, Hawaii, January 1993.

[10] Gilles Kahn. The semantics of a simple language for parallel programming. In *Information Processing 74: Proceedings of IFIP Congress 74*, pages 471–475, Stockholm, Sweden, August 1974. North-Holland.

[11] J.-L. Lassez, V. L. Nguyen, and E. A. Sonnenberg. Fixed point theorems and semantics: A folk tale. *Information Processing Letters*, 14(3):112–116, May 1982.

[12] Edward A. Lee and David G. Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9):1235–1245, September 1987.

[13] Edward A. Lee and Thomas M. Parks. Dataflow process networks. *Proceedings of the IEEE*, 83(5):773–801, May 1995.

[14] Willem C. Mallon and Jan Tijmen Udding. Building finite automata from DI specifications. In *Proceedings of Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, pages 184–193, San Diego, California, March 1998.

[15] John O'Leary and Geoffrey Brown. Synchronous emulation of asynchronous circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 16(2):205–209, February 1997.

[16] Ad Peeters and Kees van Berkel. Synchronous handshake circuits. In *Proceedings of the Seventh International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 86–95, Salt Lake City, Utah, March 2001.

[17] Jan Tijmen Udding. A formal model for defining and classifying delay-insensitive circuits and systems. *Distributed Computing*, 1(4):197–204, 1986.

[18] Kees van Berkel. *Handshake Circuits: An Intermediary Between Communicating Processes and VLSI*. PhD thesis, Eindhoven University of Technology, The Netherlands, May 1992.

[19] Kees van Berkel. *Handshake Circuits: An Asynchronous Architecture for VLSI Programming*. Cambridge University Press, 1993.

[20] Kees van Berkel, Joep Kessels, Marly Roncken, Ronald Raeijs, and Frits Schalij. The VLSI-programming language Tangram and its translation into handshake circuits. In *Proceedings of European Design Automation (EDAC)*, pages 384–389, Amsterdam, The Netherlands, February 1991.

[21] Kees van Berkel and Martin Rem. VLSI programming of asynchronous circuits for low power. In G. Birtwistle and A. Davis, editors, *Asynchronous Digital Circuit Design*, Workshops in Computing, pages 151–210. Springer-Verlag, 1995.

[22] Jan L. A. Van de Snepscheut. *Trace Theory and VLSI Design*, volume 200 of *Lecture Notes in Computer Science*. Springer-Verlag, 1985.