On the Schedulability of Real-Time Discrete-Event Systems*

Eleftherios Matsikoudis
University of California, Berkeley
ematsi@eecs.berkeley.edu

Christos Stergiou
University of California, Berkeley
chster@cs.berkeley.edu

Edward A. Lee
University of California, Berkeley
eal@eecs.berkeley.edu

ABSTRACT

We consider end-to-end latency specifications for hard real-time embedded systems. We introduce a discrete-event programming model generalizing such specifications, and address its schedulability problem for uniprocessor systems. This turns out to be rather idiosyncratic, involving complex, time-dependent release predicates and precedence constraints, quite unlike anything we have seen in the hard real-time computing literature. We prove the optimality of the earliest-deadline-first scheduling policy, and provide an algorithmic solution, reducing the schedulability problem to a reachability problem for timed automata.

1. INTRODUCTION

One of the defining features of embedded software, and one setting it apart from general-purpose computing, is its intimate relationship with the notion of time. Embedded programs are expected to interact with an inherently timed physical world, and are thus required to embrace time as part of their semantics. This is perhaps best exemplified by computer-control systems, which are required to sense the physical world and act upon it, all in a timely manner.

A computer-control system can be typically represented as a diagram of the form depicted in Figure 1(a). A sensor is producing measurements of the environment at certain instances of time. Each measurement is processed, and in response, an action on the environment is generated, and handed to an actuator responsible for carrying it through.

*This work was supported in part by the Center for Hybrid and Embedded Software Systems (CHESS) at UC Berkeley, which receives support from the National Science Foundation (NSF awards #0720882 (CSR-EHS: PRET), #1035672 (CPS: PTIDES), and #0931843 (ActionWebs)), the U. S. Army Research Lab (ARL #W911NF-11-2-0038), the Air Force Research Lab (AFRL), the Multiscale Systems Center (MuSyC), one of six research centers funded under the Focus Center Research Program, a Semiconductor Research Corporation program, and the following companies: Bosch, National Instruments, Thales, and Toyota.

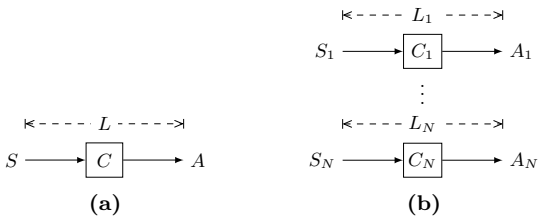

Figure 1. Simple path-latency specifications.

What is often critical is that each action be delivered to the actuator within a certain time interval beginning from the time of the corresponding measurement. This interval is often chosen based on the control algorithm in use, and the control designer will often try to enforce it by specifying a certain desired latency from a measurement at the sensor to a response to it from the actuator. The system engineer is then responsible for implementing the system in such a way that the latency requirement is met. But this is hardly rocket science. Assuming a latency specification L and a worst-case computation time W required for processing a measurement, the latency requirement can be met just as long as there are no more than $\lceil \frac{L}{W} \rceil$ measurements in any time window of size W.

It is worth noting here that the computation performed over different measurements may preserve a notion of state from one invocation to the next. Consequently, successive measurements have to be processed in the order that they are received, and cannot overlap in time.

Of course, the simple structure of Figure 1(a) may be replicated any number of times to produce systems with more than one sensor-to-actuator path, each with its own latency requirement, as depicted in Figure 1(b). The job of the engineer is a little harder now, because the computing resources of the system are now shared among the different paths, which must be scheduled in a way that ensures that all different latency requirements are satisfied. Still, this is well trodden territory. In the case of periodic or sporadic measurements, and assuming a single processing unit, one can rely on the classical hard real-time scheduling theory (e.g., see [2]), whereas for more complex input models, one may turn to more recent advances in algorithmic solutions (e.g., see [8]).

Things start to get interesting when one is considering merging paths from different sensors to a single actuator, as in Figure 2(a). If the latencies of the two paths are the same, nothing really changes; measurements at the two sensors may be thought of as jobs of the same task, and processed in the order received, with the exception of simultaneous

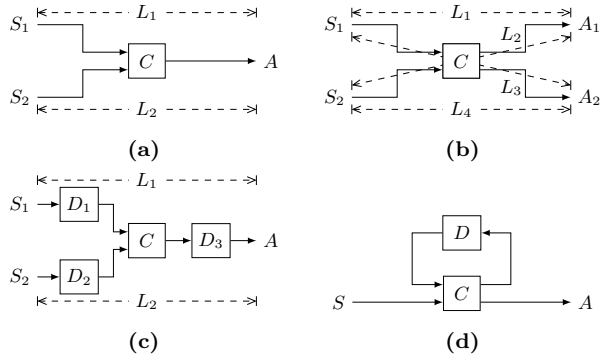

Figure 2. Delay versus latency specification examples.

measurements, which are to be treated as a single job. But what if one is inclined to specify different latencies for the two paths? How is the engineer supposed to execute such a specification? More importantly, what is the designer meant to specify? And is such a specification always meaningful?

Suppose that the latency specification from sensor S_1 to the actuator A is strictly larger than that from sensor S_2 to A; that is, $L_1 > L_2$. And suppose that S_1 produces a measurement m_1 at time t_1. Can m_1 be processed at t_1? If it is, and at some time t_2 between t_1 and $t_1 + (L_1 - L_2)$, S_2 produces a measurement m_2, then the actuation caused by m_1 will be performed after the actuation caused by m_2, even though m_1 was processed before m_2. But surely, this cannot be the intention of any rational designer wishing to merge the measurements of two different sensors through a common, possibly stateful processing component. In other words, the relative order of actuation times translates into a processing order on shared components. And therefore, assuming that m_2 can be produced at any time t_2 between t_1 and $t_1 + (L_1 - L_2)$, m_1 cannot be processed until it is possible to guarantee that $t_1 + L_1 \leq t_2 + L_2$, which is not until $t_1 + (L_1 - L_2)$.

But what if one tries to do the same on a more complex structure with two sensors and two actuators, arranged as in Figure 2(b)? Here, the designer might be tempted to specify four different latencies, one for each pair of sensor and actuator. However, not all possible choices make sense. For example, suppose that $L_1 > L_2$. Then, by the above reasoning, if S_1 produces m_1 at time t_1, and S_2 produces m_2 at time t_2 between t_1 and $t_1 + (L_1 - L_2)$, then m_1 must be processed after m_2. But if $t_1 + L_3 < t_2 + L_4$, then m_1 must be processed before m_2. And this is actually possible in the case that $L_3 - L_4 < L_1 - L_2$. Therefore, it must be the case that $L_3 - L_4 \geq L_1 - L_2$. But then $L_3 > L_4$, and by a symmetric argument, $L_1 - L_2 \geq L_3 - L_4$. So L_1, L_2, L_3, and L_4 must satisfy the equation $L_1 - L_2 = L_3 - L_4$ if the specification of the designer is to make any sense. What this means is that the designer is expected to come up with a real-time specification in terms of a number of dependent variables, a rather unappealing and impractical way to design systems, which is unlikely to scale well on yet more complex structures.

Luckily, there is a very natural solution to this problem. The way to go from dependent to independent variables is to forgo the notion of a path latency in favour of a notion of a link delay. For example, to return to the system in Figure 2(a), one may specify a delay D_1 on the link from S_1 to the processing component C, a delay D_2 on the link from S_2 to C, and a delay D_3 on the link from C to A, as in Figure 2(c), and by setting $D_1 = L_1 - L_2$, $D_2 = 0$, and $D_3 = L_2$, one can prescribe the same end-to-end latencies as before. And one can populate the links of the system in Figure 2(b) with any arbitrary choice of delays without any fear of ending up with nonsensical specifications. Once liberated from the notion of path latency, the designer may even begin to think of systems with feedback loops, as in Figure 2(d), where end-to-end latency specifications are at the very least incomplete, if not ambiguous.

Interestingly, it is now possible to come up with the same latency specification in more than one way. For example, in Figure 2(c), one may specify the same end-to-end latencies as before by setting $D_1 = L_1$, $D_2 = L_2$, and $D_3 = 0$. This raises the question of what the semantics of a link-delay specification is, and whether different delay specifications corresponding to identical latency specifications should be implemented differently by the system engineer.

One way to address this question is by making use of the above observation that the relative order of actuation times translates into a processing order on shared components. Of course, in order to resolve processing decisions at shared components deep inside the system, one would need to keep track of the paths traversed by the arriving tokens. And especially in the case of loops, this can become quite messy. But there is an equivalent, and we believe, more natural interpretation.

We may think of processing as a logical operation that takes no time, and transmission over a link as an operation that takes time equal to the delay associated with that link. For example, in the system of Figure 2(c), and according to the last delay specification above, a measurement at S_1 at time t is not available for processing at C until $t + L_1$, at which time it can be safely processed, along with any measurement arriving at that time through the link from S_2. This leads naturally to a programming model according to which programs are represented as block diagrams, and blocks correspond to components, or so-called "actors", consuming and producing time-stamped tokens, or so-called "events", conceptually ordered according to their time stamps. What we end up with is a discrete-event model of computation that is not used for modelling and simulation (e.g., see [6]), but for real-time programming (e.g., see [7]). Semantically then, different delay specifications correspond to different programs. However, the notion of time imparted by the time stamps is not a physical one, but a logical one. And while the designer is free to think of this as a timed programming model, and think of the components of the control algorithm and the physical world as living in the same space, and sharing the same temporal semantics, this is not required for the engineer. A measurement is initially time-stamped by the sensor with the actual time at which it was taken. It is then communicated between actors, which may increase the time stamp by some arbitrary but fixed amount before forwarding to the next actor, until it reaches an actuator, where the time stamp is interpreted as the actual time when actuation is to occur. It does not really matter when events are processed, as long as they are processed in time-stamp order. And the job of the engineer is to make sure that every event reaching an actuator is delivered before the time corresponding to its time stamp.

This amounts to a rather complex schedulability problem that has not been considered before in the hard real-time computing literature. A typical approach to schedulability

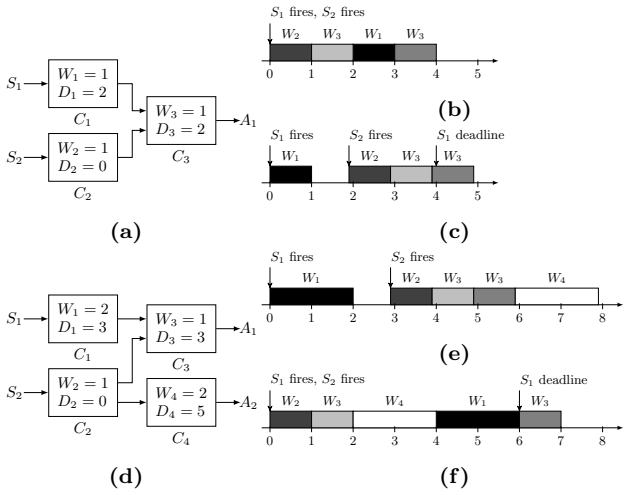

Figure 3. Maximizing processor demand (b), (f) versus waiting time (c), (e) as worst-case scenarios.

problems is to reduce the problem of scheduling every possible scenario to the problem of scheduling a single "worst-case" scenario. And that "worst-case" scenario typically corresponds in one way or another to a job-arrival pattern that maximizes processor demand over a certain time window. But in our case, there is another factor that may contribute to a bad scenario. And the two pull in opposite directions (see also [1]).

For example, consider the system in Figure 3(a), where each actor is annotated with a worst-case execution time and a delay added to the time stamps of processed events. Suppose that both sensors produce events at time 0, as in Figure 3(b). This is a scenario that maximizes processor demand, and yet is perfectly schedulable. Now, suppose that, instead, S_1 produces an event e_1 at time 0, whereas S_2 produces an event e_2 at time $2 - \epsilon$ for some small ϵ, as in Figure 3(c). Then C_1 will begin processing e_1 at time 0, and having a worst-case execution time $W = 1$ and a delay $D = 2$, will produce an event e_1' at time 1 with time stamp 2. But since C_2 has a delay 0, e_1' cannot be processed by C_3 until real time reaches the time stamp of that event, namely 2, lest there be another event produced by S_2 at some time t between 1 and 2, and thus, another one by C_2 with time stamp t. Thus, the system remains idle for almost a unit of time, in order to make sure that events are safely processed in time stamp order. And after C_2 produces e_2' in response to e_2, C_3 will have to process e_2' before e_1', since e_2' has a time stamp of $2 - \epsilon$ whereas e_1' a time stamp of 2. This causes the deadline of e_1' to be missed. The reverse situation is exhibited by the system of Figure 3(d), where instead, trying to maximize the time wasted waiting for events to become safe to process does not yield the worst-case scenario.

Another source of complexity is the expressiveness of the proposed programming model. The ability to design systems with feedback loops comes with a cost, namely the possibility of unbounded accumulation of events circulating in these feedback loops, and thus, it is not immediately obvious that the schedulability problem is even decidable.

The purpose of this work is to address this schedulability problem for a uniprocessor system. Because of the above complications, it is unlikely that an analytical solution exists. Here, we consider an algorithmic solution. We provide a detailed formalization of the programming model, prove that the earliest-deadline-first scheduling policy is optimal, and show that the schedulability problem can be reduced to a finite-state reachability problem. Finally, we describe how to carry out this reduction using timed automata. The formalism of timed automata is not intrinsic to our solution, but rather, a convenient tool for seamlessly integrating system abstraction with different, and complex, input-event arrival models.

For lack of space, we omit all proofs; they can be found in [11].

2. EVENTS AND SIGNALS

We fix $\mathbb{T}$ to be the set $\mathbb{R}_{\geq 0}$ of all non-negative real numbers, and use it to represent our time domain. In order to formalize our notion of event, we further postulate an infinite set C of *channels*, and an infinite set V of values.

DEFINITION 2.1. *A* sort *is a non-empty finite subset of* C.

Sorts will serve to represent the interfaces of actors, and facilitate their composition into programs.

DEFINITION 2.2. *A* program event *of sort C is an ordered triple $\langle c, t^{\text{prog}}, v\rangle \in C \times \mathbb{T} \times \text{V}$.*

We write $\text{E}^{\text{prog}}(C)$ for the set of all program events of sort C.

Assume a program event $e = \langle c, t^{\text{prog}}, v\rangle$.

We write e for c, $\mathsf{time}^{\text{prog}}\, e$ for t^{prog}, and $\mathsf{val}\, e$ for v.

DEFINITION 2.3. *A* program signal *of sort C is a single-valued subset of $\text{E}^{\text{prog}}(C)$.*

We write $\text{S}^{\text{prog}}(C)$ for the set of all program signals of sort C, and $\text{S}^{\text{prog}}_{\text{fin}}(C)$ for the set of all finite program signals of sort C.

We will use signals to represent inputs and outputs of programs, as well as store the events circulating inside programs, much like calendar queues in typical discrete-event simulation implementations.

Here, we will be interested only in *discrete-event* signals, whose domain can be orderly embedded in the natural numbers.

We write $\text{S}^{\text{prog}}_{\text{DE}}(C)$ for the set of all discrete-event program signals of sort C.

3. ACTORS

An actor is a stateful, in general, component that interfaces with its environment through a set of input and a set of output channels, what we call its input and output sort respectively. Each time it fires, it consumes a set of events from its input sort, and produces a set of events at its output sort, possibly updating its state in the process.

DEFINITION 3.1. *An* input action *of sort C is a non-empty single-valued subset α of $\text{E}^{\text{prog}}(C)$ such that for every $e_1, e_2 \in \alpha$,*

$$\mathsf{time}^{\text{prog}}\, e_1 = \mathsf{time}^{\text{prog}}\, e_2.$$

We write $\text{IA}(C)$ for the set of all input actions of sort C.

Assume $\alpha \in \text{IA}(C)$.

We write $\mathsf{chan}\, \alpha$ for $\{\mathsf{chan}\, e \mid e \in \alpha\}$, and $\mathsf{time}^{\text{prog}}\, \alpha$ for the unique $t^{\text{prog}} \in \mathbb{T}$ such that for every $e \in \alpha$,

$$\mathsf{time}^{\text{prog}}\, e = t^{\text{prog}}.$$

$$\text{input}\ \frac{l \in \mathsf{lab}_{\text{in}}\, P}{\langle Q, \iota, \varepsilon\rangle \xrightarrow{\ l\ }_P \langle Q \cup \{l\}, \iota, \varepsilon\rangle} \qquad \text{output}\ \frac{l \in \mathsf{lab}_{\text{out}}\, P \quad l \in Q}{\langle Q, \iota, \varepsilon\rangle \xrightarrow{\ l\ }_P \langle Q \setminus \{l\}, \iota, \varepsilon\rangle}$$

$$\text{start}\ \frac{\langle A, \langle s, \alpha\rangle\rangle \in \mathsf{lab}_{\text{start}}\, P \quad \alpha \subseteq Q \quad \iota(A) = s}{\langle Q, \iota, \varepsilon\rangle \xrightarrow{\langle A, \langle s, \alpha\rangle\rangle}_P \langle Q \setminus \alpha, \iota \setminus \{\langle A, s\rangle\}, \varepsilon \cup \{\langle A, \langle s, \alpha\rangle\rangle\}\rangle}$$

$$\text{finish}\ \frac{\langle A, \alpha, s\rangle \in \mathsf{lab}_{\text{finish}}\, P \quad \mathrm{f}^A(\varepsilon(A)) = \alpha \quad \mathrm{u}^A(\varepsilon(A)) = s}{\langle Q, \iota, \varepsilon\rangle \xrightarrow{\langle A, \alpha, s\rangle}_P \langle Q \cup \alpha, \iota \cup \{\langle A, s\rangle\}, \varepsilon \setminus \{\langle A, \varepsilon(A)\rangle\}\rangle}$$

Figure 4. Program transition rules.

Input actions restrict the possible input behaviours of actors, and are central to our perception of a time stamp as a logical time instant: an actor is perceived to fire at the unique logical time instant associated with the corresponding input action.

DEFINITION 3.2. *An* output action *of sort C is a single-valued subset α of $\mathrm{E}^{\text{prog}}(C)$ such that for any $e_1, e_2 \in \alpha$ such that $\mathsf{chan}\, e_1 = \mathsf{chan}\, e_2$,*

$$\mathsf{time}^{\text{prog}}\, e_1 = \mathsf{time}^{\text{prog}}\, e_2.$$

We write $\mathrm{OA}(C)$ for the set of all output actions of sort C.

Unlike input actions, the events of an output action need not share the same time stamp. This is not inconsistent with our perception of time stamps as logical time instants. Output actions can be understood operationally: they schedule events in logical time according to their time stamps. What might seem strange, then, is why output actions are constrained to carry no more than one event per channel. This is a technical constraint that will simplify our analysis, without sacrificing any expressiveness, as can be easily shown.

DEFINITION 3.3. *An* actor *is an ordered sextuple $\langle S, s_{\text{init}}, C_{\text{in}}, C_{\text{out}}, f, u\rangle$ such that the following are true:*

1. *S is a non-empty set;*
2. *$s_{\text{init}} \in S$;*
3. *C_{in} is a sort;*
4. *C_{out} is a sort;*
5. *f is a function from $S \times \mathrm{IA}(C_{\text{in}})$ to $\mathrm{OA}(C_{\text{out}})$;*
6. *u is a function from $S \times \mathrm{IA}(C_{\text{in}})$ to S.*

Assume an actor $A = \langle S, s_{\text{init}}, C_{\text{in}}, C_{\text{out}}, f, u\rangle$.

We write S^A for S, $\mathrm{s}^A_{\text{init}}$ for s_{init}, C^A_{in} for C_{in}, $\mathrm{C}^A_{\text{out}}$ for C_{out}, f^A for f, and u^A for u.

We now identify the class of actors that we will consider in this work.

We say that A is *output-homogeneous* if and only if for every $\langle s, \alpha\rangle \in \mathrm{S}^A \times \mathrm{IA}(\mathrm{C}^A_{\text{in}})$,

$$\{\mathsf{chan}\, e \mid e \in \mathrm{f}^A(\langle s, \alpha\rangle)\} = \mathrm{C}^A_{\text{out}}.$$

An output-homogeneous actor is simply an actor that, when fired, produces a single event at each channel in its output sort. This is of course consistent with the way we described the programming model in the introduction but as it will be apparent later on, it is also a necessary condition in order to be able to statically determine event deadlines.

Assume an output-homogeneous actor A.

We say that A is *constant-delay* if and only if for every $c \in \mathrm{C}^A_{\text{out}}$, there is $\delta \in \mathbb{Q}_{\geq 0}$ such that for every $\langle s, \alpha\rangle \in \mathrm{S}^A \times \mathrm{IA}(\mathrm{C}^A_{\text{in}})$, and every $e \in \mathrm{f}^A(\langle s, \alpha\rangle)$ such that $\mathsf{chan}\, e = c$,

$$\mathsf{time}^{\text{prog}}\, e = \mathsf{time}^{\text{prog}}\, \alpha + \delta.$$

Constant-delay output-homogeneous actors are *causal* actors that produce events at fixed, non-negative logical time distances from the logical time instance associated with the events consumed.

Assume a constant-delay output-homogeneous actor A.

We write $\mathsf{delay}\, A$ for a function from $\mathrm{C}^A_{\text{out}}$ to $\mathbb{Q}_{\geq 0}$ such that for every $c \in \mathrm{C}^A_{\text{out}}$, every $\langle s, \alpha\rangle \in \mathrm{S}^A \times \mathrm{IA}(\mathrm{C}^A_{\text{in}})$, and every $e \in \mathrm{f}^A(\langle s, \alpha\rangle)$ such that $\mathsf{chan}\, e = c$,

$$\mathsf{time}^{\text{prog}}\, e = \mathsf{time}^{\text{prog}}\, \alpha + (\mathsf{delay}\, A)(c).$$

4. PROGRAMS

For every actor A_1 and A_2, we say that A_1 and A_2 are *compatible* if and only if $\mathrm{C}^{A_1}_{\text{in}} \cap \mathrm{C}^{A_2}_{\text{in}} = \emptyset$ and $\mathrm{C}^{A_1}_{\text{out}} \cap \mathrm{C}^{A_2}_{\text{out}} = \emptyset$.

DEFINITION 4.1. *A* program *is a non-empty finite set of pairwise compatible constant-delay output-homogeneous actors.*

Assume a program P.

We write $\mathsf{chan}\, P$ for $\bigcup\{\mathrm{C}^A_{\text{in}}, \mathrm{C}^A_{\text{out}} \mid A \in P\}$, C^P_{in} for $\bigcup\{\mathrm{C}^A_{\text{in}} \mid A \in P\} \setminus \bigcup\{\mathrm{C}^A_{\text{out}} \mid A \in P\}$, and $\mathrm{C}^P_{\text{out}}$ for $\bigcup\{\mathrm{C}^A_{\text{out}} \mid A \in P\} \setminus \bigcup\{\mathrm{C}^A_{\text{in}} \mid A \in P\}$.

We use a labelled transition system to formalize all possible executions of P.

DEFINITION 4.2. *A* state *of P is an ordered triple $\langle Q, \iota, \varepsilon\rangle$ such that the following are true:*

1. *$Q \in \mathrm{S}^{\text{prog}}_{\text{fin}}(\mathsf{chan}\, P)$;*
2. *there is a partition $\{P_{\text{idle}}, P_{\text{exec}}\}$ of P such that the following are true:*
   (a) *ι is a function from P_{idle} such that for any $A \in P_{\text{idle}}$, $\iota(A) \in \mathrm{S}^A$;*
   (b) *ε is a function from P_{exec} such that for any $A \in P_{\text{exec}}$, $\varepsilon(A) \in \mathrm{S}^A \times \mathrm{IA}(\mathrm{C}^A_{\text{in}})$.*

We write $\mathsf{state}\, P$ for the set of all states of P.

A state $\langle Q, \iota, \varepsilon\rangle$ of P captures the composite state of P at some particular time instant during its execution: Q represents the set of all events circulating inside the program, ι the set of all idle actors, along with their state, and ε the set of all executing actors, along with their state and input events processed.

We write $\mathsf{state}_{\text{init}}\, P$ for a state $\langle Q, \iota, \varepsilon\rangle$ of P such that the following are true:

1. Q is the empty program signal;
2. ι is a function from P such that for every $A \in P$,
$$\iota(A) = \mathrm{s}^A_{\text{init}};$$
3. ε is the empty function.

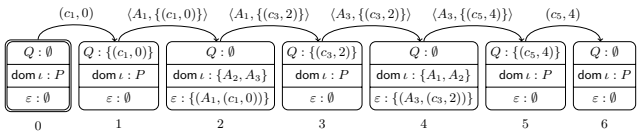

Figure 5. An example of a finite execution of the program of Figure 3(a), where event values and actor states have been omitted.

The labels of the transition system that we will associate with P represent the possible actions of P. P can either receive an event from its environment, have an idle actor start processing an input action, have a processing actor finish its execution and produce an output action, or send an event to its environment.

We write $\mathsf{lab}_{\text{in}}\, P$ for $\mathrm{E}^{\text{prog}}(\mathrm{C}_{\text{in}}^P)$, $\mathsf{lab}_{\text{start}}\, P$ for

$$\{\langle A, \langle s, \alpha\rangle\rangle \mid A \in P,\ s \in \mathrm{S}^A, \text{ and } \alpha \in \mathrm{IA}(\mathrm{C}_{\text{in}}^A)\},$$

$\mathsf{lab}_{\text{finish}}\, P$ for

$$\{\langle A, \alpha, s\rangle \mid A \in P,\ \alpha \in \mathrm{OA}(\mathrm{C}_{\text{out}}^A), \text{ and } s \in \mathrm{S}^A\},$$

$\mathsf{lab}_{\text{out}}\, P$ for $\mathrm{E}^{\text{prog}}(\mathrm{C}_{\text{out}}^P)$, and $\mathsf{lab}\, P$ for

$$\mathsf{lab}_{\text{in}}\, P \cup \mathsf{lab}_{\text{start}}\, P \cup \mathsf{lab}_{\text{finish}}\, P \cup \mathsf{lab}_{\text{out}}\, P.$$

We write $\longrightarrow_P$ for a ternary relation between $\mathsf{state}\, P$, $\mathsf{lab}\, P$, and $\mathsf{state}\, P$ defined by the rules in Figure 4.

We write $\langle Q_1, \iota_1, \varepsilon_1\rangle \xrightarrow{l}{}^{\text{in}}_P \langle Q_2, \iota_2, \varepsilon_2\rangle$ if and only if $\langle Q_1, \iota_1, \varepsilon_1\rangle \xrightarrow{l}_P \langle Q_2, \iota_2, \varepsilon_2\rangle$ and $l \in \mathsf{lab}_{\text{in}}\, P$, and similarly for $\mathsf{lab}_{\text{start}}\, P$, $\mathsf{lab}_{\text{finish}}\, P$, and $\mathsf{lab}_{\text{out}}\, P$.

Definition 4.3. *An* execution *of P is a non-empty sequence E^{prog} such that the following are true:*

1. $E_0^{\text{prog}} = \mathsf{state}_{\text{init}}\, P$;
2. *one of the following is true:*
   (a) *E^{prog} is finite, and the following are true:*
      i. *for any $n < (|E^{\text{prog}}| - 1)/2$,*
      $$E_{2n}^{\text{prog}} \xrightarrow{E_{2n+1}^{\text{prog}}}_P E_{2n+2}^{\text{prog}};$$
      ii. *for every l and $\langle Q, \iota, \varepsilon\rangle$, if*
      $$E_{|E^{\text{prog}}|-1}^{\text{prog}} \xrightarrow{l}_P \langle Q, \iota, \varepsilon\rangle$$
      then $l \in \mathsf{lab}_{\text{in}}\, P$;
   (b) *E^{prog} is infinite, and for every $n \in \mathbb{N}$,*
   $$E_{2n}^{\text{prog}} \xrightarrow{E_{2n+1}^{\text{prog}}}_P E_{2n+2}^{\text{prog}}.$$

Figure 5 displays an example of a finite execution of the program of Figure 3(a), where event values and actor states have been omitted.

Assume an execution E^{prog} of P. We write $\mathsf{in}\, E^{\text{prog}}$ for

$$\{E_{2n+1}^{\text{prog}} \mid E_{2n+1}^{\text{prog}} \in \mathsf{lab}_{\text{in}}\, P\},$$

and $\mathsf{out}\, E^{\text{prog}}$ for

$$\{E_{2n+1}^{\text{prog}} \mid E_{2n+1}^{\text{prog}} \in \mathsf{lab}_{\text{out}}\, P\}.$$

Clearly, our definition of execution is too liberal. In particular, it allows for executions where an actor processes its inputs out of time-stamp order, what is at odds with the intended role of time stamps as a logical notion of time.

We say that E^{prog} is *safe* if and only if every actor in P processes its inputs in time-stamp order (see Appendix A for a formal definition).

Informally, an execution is safe just as long as every actor processes its input events in time-stamp order. Notice that safety does not constrain a program to process every single event in time-stamp order, only that each actor does so.

Another worrisome type of execution allowed by our definition is that of one where an event in the program remains unprocessed indefinitely, an actor is not given the opportunity to finish processing, or an output event of the program is never sent to the environment.

We say that E^{prog} is *actor-fair* if and only if every actor in P is eventually given the opportunity to start and finish processing (see Appendix A for a formal definition).

We say that E^{prog} is *output-fair* if and only if every output event of the program is eventually sent to the environment (see Appendix A for a formal definition).

We say that E^{prog} is *fair* if and only if E^{prog} is actor-fair and output-fair.

We say that E^{prog} is *correct* if and only if E^{prog} is safe and fair.

Correct executions of a program constitute a semantic specification of valid implementations of that program.

Theorem 4.4. *For every correct execution E_1^{prog} and E_2^{prog} of P such that $\mathsf{in}\, E_1^{\text{prog}} = \mathsf{in}\, E_2^{\text{prog}}$, $\mathsf{out}\, E_1^{\text{prog}} = \mathsf{out}\, E_2^{\text{prog}}$.*

Theorem 4.4 formally characterizes the determinacy of discrete-event programs. It asserts that as long as the logical notion of time in a program is not violated, in the rather loose sense of the safety property, and each party involved is given the opportunity to make progress, the behaviour of the program is determinate with respect to input and output signals.

The question remains whether a program has correct executions at all, that is, other than the trivial one. Clearly, a program that contains cycles of zero logical-time delay cannot possibly have an execution where an event enters the cycle without violating either safety or fairness. Here, we exclude such ill-behaved programs from consideration.

We say that P is *well defined* if and only if for every non-empty $X \subseteq P$, there is $A \in X$ such that for every $A' \in X$,

$$(\mathsf{delay}\, A')^{-1}(0) \cap (\mathrm{C}_{\text{in}}^A) = \emptyset.$$

Assume a well defined program P.

Theorem 4.5. *For any non-Zeno $s^{\text{prog}} \in \mathrm{S}_{\text{DE}}^{\text{prog}}(\mathrm{C}_{\text{in}}^P)$, there is a correct execution E^{prog} of P such that $\mathsf{in}\, E^{\text{prog}} = s^{\text{prog}}$.*

Theorem 4.5 guarantees that any program free of zero logical-time delay cycles will have a correct execution for every possible discrete-event and non-Zeno input signal it is presented with.

We write $\mathsf{delay}\, P$ for a function from $\mathsf{chan}\, P \times \mathsf{chan}\, P$ to $\mathbb{Q}_{\geq 0} \cup \{\infty\}$ such that for every $c_1, c_2 \in \mathsf{chan}\, P$, $(\mathsf{delay}\, P)(c_1, c_2)$ is the minimum incurred delay among all possible paths from c_1 to c_2 in P (formal definition omitted).

5. SYSTEMS

Programs deal exclusively in logical time. There is nothing in their semantics that bears any relevance to physical time

$$\text{input}\ \frac{\langle Q_1, \iota_1, \varepsilon_1\rangle \xrightarrow{l}{}^{\text{in}}_{P} \langle Q_2, \iota_2, \varepsilon_2\rangle \quad \mathsf{time}^{\text{prog}}\, l = t^{\text{sys}}}{\langle Q_1, \iota_1, \varepsilon_1, \rho, \pi, t^{\text{sys}}\rangle \xrightarrow{l}_{\langle P,R\rangle} \langle Q_2, \iota_2, \varepsilon_2, \rho, \pi, t^{\text{sys}}\rangle}$$

$$\text{output}\ \frac{\langle Q_1, \iota_1, \varepsilon_1\rangle \xrightarrow{l}{}^{\text{out}}_{P} \langle Q_2, \iota_2, \varepsilon_2\rangle \quad \mathsf{time}^{\text{prog}}\, l = t^{\text{sys}}}{\langle Q_1, \iota_1, \varepsilon_1, \rho, \pi, t^{\text{sys}}\rangle \xrightarrow{l}_{\langle P,R\rangle} \langle Q_2, \iota_2, \varepsilon_2, \rho, \pi, t^{\text{sys}}\rangle}$$

$$\text{start}\ \frac{\langle Q_1, \iota_1, \varepsilon_1\rangle \xrightarrow{\langle A,\langle s,\alpha\rangle\rangle}{}^{\text{start}}_{P} \langle Q_2, \iota_2, \varepsilon_2\rangle \quad r \in R(A) \quad \rho_2 = \rho_1 \cup \{\langle A, r\rangle\}}{\langle Q_1, \iota_1, \varepsilon_1, \rho_1, \pi, t^{\text{sys}}\rangle \xrightarrow{\langle\langle A,\langle s,\alpha\rangle\rangle, r\rangle}_{\langle P,R\rangle} \langle Q_2, \iota_2, \varepsilon_2, \rho_2, \pi, t^{\text{sys}}\rangle}$$

$$\text{finish}\ \frac{\langle Q_1, \iota_1, \varepsilon_1\rangle \xrightarrow{\langle A,\alpha,s\rangle}{}^{\text{finish}}_{P} \langle Q_2, \iota_2, \varepsilon_2\rangle \quad \rho_1(A) = 0 \quad \rho_2 = \rho_1 \setminus \{\langle A, 0\rangle\}}{\langle Q_1, \iota_1, \varepsilon_1, \rho_1, A, t^{\text{sys}}\rangle \xrightarrow{\langle A,\alpha,s\rangle}_{\langle P,R\rangle} \langle Q_2, \iota_2, \varepsilon_2, \rho_2, \mathtt{NULL}, t^{\text{sys}}\rangle}$$

$$\text{context-switch}\ \frac{l \in \{\mathtt{NULL}\} \cup \mathsf{dom}\, \varepsilon}{\langle Q, \iota, \varepsilon, \rho, \pi, t^{\text{sys}}\rangle \xrightarrow{l}_{\langle P,R\rangle} \langle Q, \iota, \varepsilon, \rho, l, t^{\text{sys}}\rangle}$$

Figure 6. System transition rules.

$$\text{idle}\ \frac{\pi = \mathtt{NULL}}{\langle Q, \iota, \varepsilon, \rho, \pi, t^{\text{sys}}\rangle \xrightarrow{d}_{\langle P,R\rangle} \langle Q, \iota, \varepsilon, \rho, \pi, t^{\text{sys}} + d\rangle} \qquad \text{busy}\ \frac{\pi \neq \mathtt{NULL} \quad d \leq \rho_1(\pi) \quad \rho_2(A) = \begin{cases} \rho_1(A) - d & \text{if } A = \pi; \\ \rho_1(A) & \text{otherwise;} \end{cases}}{\langle Q, \iota, \varepsilon, \rho_1, \pi, t^{\text{sys}}\rangle \xrightarrow{d}_{\langle P,R\rangle} \langle Q, \iota, \varepsilon, \rho_2, \pi, t^{\text{sys}} + d\rangle}$$

Figure 7. System time transition rules.

at all. But if we are to use them as executable real-time specifications, we need to determine how exactly these two different notions of time relate to one another.

DEFINITION 5.1. *A* system *is an ordered pair $\langle P, R\rangle$ such that the following are true:*

1. *P is a well-defined program;*
2. *R is a function from P to $\mathscr{P}_{\geq 1}\, \mathbb{T}$,[1] such that for every $A \in P$, $0 < \inf R(A) \leq \sup R(A)$ and $\sup R(A) \in \mathbb{Q}_{\geq 0}$.*

Assume a system $\langle P, R\rangle$. $\langle P, R\rangle$ is understood as a program P running on a given uniprocessor platform. The function R captures the computation-time requirements of each actor in P on that platform. We use a timed labelled transition system to formalize all possible executions of $\langle P, R\rangle$.

DEFINITION 5.2. *A* state *of $\langle P, R\rangle$ is an ordered sextuple $\langle Q, \iota, \varepsilon, \rho, \pi, t^{\text{sys}}\rangle$ such that the following are true:*

1. *$\langle Q, \iota, \varepsilon\rangle \in \mathsf{state}\, P$;*
2. *ρ is a function from $\mathsf{dom}\, \varepsilon$ to $\mathbb{T}$ such that for any $A \in \mathsf{dom}\, \varepsilon$, there is $r \in R(A)$ such that $\rho(A) \leq r$;*
3. *$\pi \in \{\mathtt{NULL}\} \cup \{A \mid A \in \mathsf{dom}\, \varepsilon \text{ and } 0 < \rho(A)\}$;*
4. *$t^{\text{sys}} \in \mathbb{T}$.*

We write $\mathsf{time}^{\text{sys}}\, \langle Q, \iota, \varepsilon, \rho, \pi, t^{\text{sys}}\rangle$ for t^{sys}.

A state $\langle Q, \iota, \varepsilon, \rho, \pi, t^{\text{sys}}\rangle$ of $\langle P, R\rangle$ augments the state of P with information on the actual state of the platform at some particular time instant during the execution of P. Specifically, ρ captures the remaining computation time of any processing actor, π represents the actor running on the processor at that time instant, and t^{sys} stands for that time instant. We write $\mathsf{state}\, \langle P, R\rangle$ for the set of all states of $\langle P, R\rangle$.

We write $\mathsf{state}_{\text{init}}\, \langle P, R\rangle$ for a state $\langle Q, \iota, \varepsilon, \rho, \pi, t^{\text{sys}}\rangle$ of $\langle P, R\rangle$ such that the following are true:

1. $\langle Q, \iota, \varepsilon\rangle = \mathsf{state}_{\text{init}}\, P$;
2. ρ is the empty function;
3. $\pi = \mathtt{NULL}$;
4. $t^{\text{sys}} = 0$.

The discrete actions of a system are as in the case of its associated program, with the addition of a type of action meant to represent the scheduling decisions of the platform.

We write $\mathsf{lab}_{\text{start}}\, \langle P, R\rangle$ for

$$\{\langle\langle A, \langle s, \alpha\rangle\rangle, r\rangle \mid \langle A, \langle s, \alpha\rangle\rangle \in \mathsf{lab}_{\text{start}}\, P \text{ and } r \in R(A)\},$$

$\mathsf{lab}_{\text{prog}}\, \langle P, R\rangle$ for

$$\mathsf{lab}_{\text{in}}\, P \cup \mathsf{lab}_{\text{start}}\, \langle P, R\rangle \cup \mathsf{lab}_{\text{finish}}\, P \cup \mathsf{lab}_{\text{out}}\, P,$$

$\mathsf{lab}_{\text{sch}}\, \langle P, R\rangle$ for $\{\mathtt{NULL}\} \cup P$, and $\mathsf{lab}\, \langle P, R\rangle$ for

$$\mathsf{lab}_{\text{prog}}\, \langle P, R\rangle \cup \mathsf{lab}_{\text{sch}}\, \langle P, R\rangle.$$

Notice that the labels related to an actor starting to process an input action are augmented with a rational number representing the amount of computation time that will be required for processing that input action.

We write $\longrightarrow_{\langle P,R\rangle}$ for a ternary relation between $\mathsf{state}\, \langle P, R\rangle$, $\mathsf{lab}\, \langle P, R\rangle$, and $\mathsf{state}\, \langle P, R\rangle$ defined by the rules in Figure 6.

There are a few observations that need to be made here.

First, input and output actions of a system bind program time to system time and system time to program time respectively. One may think of an input channel of a program as connected to an ideal sensor that will time-stamp its measurements with the exact physical time at which they were made, and instantaneously deliver them to the program, and an output channel as connected to an ideal actuator that will actuate the environment instantaneously at the exact physical time dictated by the time stamp of the corresponding output event.

Second, every time an actor starts processing, a requirement for computation-time necessary to perform its processing is chosen nondeterministically, and the actor becomes available for execution. But it is not executed until the system decides to allocate the platform's processor to it. Once it starts executing, it can be preempted and resumed arbitrarily according to the scheduling decisions of the system.

[1] For every set A, we write $\mathscr{P}_{\geq 1}\, A$ for the set of all nonempty subsets of A.

And third, a processing actor finishes exactly when it has been allocated processor time equal to its corresponding computation-time requirement, at which point it is taken off the processor.

System-time progress is modeled using a different type of transition labelled with the amount of physical time elapsed.

We write $\dashrightarrow_{\langle P,R\rangle}$ for a ternary relation between $\mathsf{state}\,\langle P,R\rangle$, $\mathbb{T}$, and $\mathsf{state}\,\langle P,R\rangle$ defined by the rules in Figure 7.

When the processor of the platform is free, there is no actor executing, and unless there is an actor available for processing that the system decides to execute, system time may progress arbitrarily. But when the processor is allocated to a processing actor, system time cannot progress more than the remaining computation time of that actor, for at that point, a discrete transition corresponding to that actor finishing must occur.

DEFINITION 5.3. *An* execution *of $\langle P,R\rangle$ is an infinite sequence E^{sys} such that the following are true:*

1. $E^{\text{sys}}_0 = \mathsf{state}_{\text{init}}\,\langle P,R\rangle$;
2. *for every $n \in \mathbb{N}$, one of the following is true:*
   (a) $E^{\text{sys}}_{2n} \xrightarrow{E^{\text{sys}}_{2n+1}}_{\langle P,R\rangle} E^{\text{sys}}_{2n+2}$;
   (b) $E^{\text{sys}}_{2n} \overset{E^{\text{sys}}_{2n+1}}{\dashrightarrow}_{\langle P,R\rangle} E^{\text{sys}}_{2n+2}$;
3. *for every $t^{\text{sys}} \in \mathbb{T}$, there is n such that*
$$t^{\text{sys}} \leq \mathsf{time}^{\text{sys}}\, E^{\text{sys}}_{2n};$$
4. *for any n such that*
$$\mathsf{time}^{\text{sys}}\, E^{\text{sys}}_{2n} = \mathsf{time}^{\text{sys}}\, E^{\text{sys}}_{2n+2},$$
if $E^{\text{sys}}_{2n+3} \in \mathsf{lab}_{\text{in}}\,\langle P,R\rangle$, then $E^{\text{sys}}_{2n+1} \in \mathsf{lab}_{\text{in}}\,\langle P,R\rangle$.

A system execution is always infinite, for even if the environment ceases to produce input stimuli, system time will continue to progress, and in fact, diverge, as required by the third clause of the definition. What the fourth clause amounts to is giving input transitions a higher priority, and is necessitated by the idealization choices of our formalization, as will soon become clear.

Assume an execution E^{sys} of $\langle P,R\rangle$. We write $\mathsf{prog}\,E^{\text{sys}}$ for the unique program execution underlying E^{sys} (formal definition omitted). Here, we think of prog as a projection operator from system to program executions.

Figure 8 shows a prefix of an execution of the system of Figure 3(a), whose underlying program execution is that of Figure 5, where, again, event values and actor states have been omitted.

Just as was the case with program executions, system executions are too general. What we want is that the logical-time specification of our programs prescribe the physical-time behaviour of our systems. And for that to be the case, we need to make sure that program time, as expressed through time stamps of events, maintains its role as a logical notion of time. To do so we have to limit ourselves to system executions whose underlying program executions are safe. But here we must further guarantee that the resulting system specification will exclude non-causal implementations that clairvoyantly execute actors without ever violating program safety, correctly guessing the environment's future behaviour. And having bound program time to system time at the input edges of a program, we can use the structure and state of the program to determine at any given time, whether it is safe for an actor to process an input event or not.

We say that A is *ready* in $\langle Q,\iota,\varepsilon,\rho,\pi,t^{\text{sys}}\rangle$ if and only if $A \in \mathsf{dom}\,\iota$, there is $\alpha \in \text{IA}(\text{C}^A_{\text{in}})$ such that $\alpha \subseteq Q$, and the following are true:

1. $t^{\text{sys}} \geq \mathsf{time}^{\text{prog}}\,\alpha - (\mathsf{delay}\,P)(\text{C}^P_{\text{in}}, \text{C}^A_{\text{in}})$;
2. for any $e \in Q \setminus \alpha$,
$$\mathsf{time}^{\text{prog}}\,e > \mathsf{time}^{\text{prog}}\,\alpha - (\mathsf{delay}\,P)(\mathsf{chan}\,e, \text{C}^A_{\text{in}});$$
3. for any A' and $\langle s',\alpha'\rangle$ such that $\varepsilon(A') = \langle s',\alpha'\rangle$,
$$\mathsf{time}^{\text{prog}}\,\alpha' > \mathsf{time}^{\text{prog}}\,\alpha - (\mathsf{delay}\,P)(\text{C}^{A'}_{\text{in}}, \text{C}^A_{\text{in}}).$$

Informally, an actor is ready just as long as there is an event to process, it is impossible for a new event arriving at a sensor to eventually cause an event of smaller or equal time stamp as the event to be processed, and the same is true for any other event already circulating or being processed inside the program. Notice that the inequality in the first clause is non-strict, as opposed to those in the second and third one. This reflects our idealization that allows a system to instantaneously make a scheduling decision that takes into account the input status at that same time instant, and is the reason for giving input transitions a higher priority in system executions. Moreover, this definition of when an actor is ready to process an event is input agnostic, i.e., it makes no assumptions on the input model. It is easy to see how the safe to process test can be made more efficient, in terms of how long an actor should wait for new events, in the case for example of a periodic or sporadic input model where for some time windows it is guaranteed that no new input events will arrive.

We say that E^{sys} is *actor-safe* if and only if for every $A \in P$, and any n, $\langle Q,\iota,\varepsilon,\rho,\pi,t^{\text{sys}}\rangle$, $\langle s,\alpha\rangle$, and r such that
$$E^{\text{sys}}_{2n} = \langle Q,\iota,\varepsilon,\rho,\pi,t^{\text{sys}}\rangle$$
and
$$E^{\text{sys}}_{2n+1} = \langle\langle A,\langle s,\alpha\rangle\rangle, r\rangle,$$
A is ready in E^{sys}_{2n}.

Of course, we are interested in system executions that meet the physical-time constraints implied by the logical-time specification of our programs.

We say that E^{sys} is *output-safe* if and only if for any $c \in \text{C}^P_{\text{out}}$, and every n and $\langle Q,\iota,\varepsilon,\rho,\pi,t^{\text{sys}}\rangle$ such that
$$E^{\text{sys}}_{2n} = \langle Q,\iota,\varepsilon,\rho,\pi,t^{\text{sys}}\rangle,$$
if there is e in Q such that $\mathsf{chan}\,e = c$, then
$$t^{\text{sys}} \leq \mathsf{time}^{\text{prog}}\,e.$$

If we think of time stamps of events reaching the output channels of a program as actuator deadlines, then a system execution is output-safe just as long as no actuator deadline is ever missed.

We say that E^{sys} is *safe* if and only if E^{sys} is actor-safe and output-safe.

THEOREM 5.4. *If E^{sys} is safe, then $\mathsf{prog}\,E^{\text{sys}}$ is safe.*

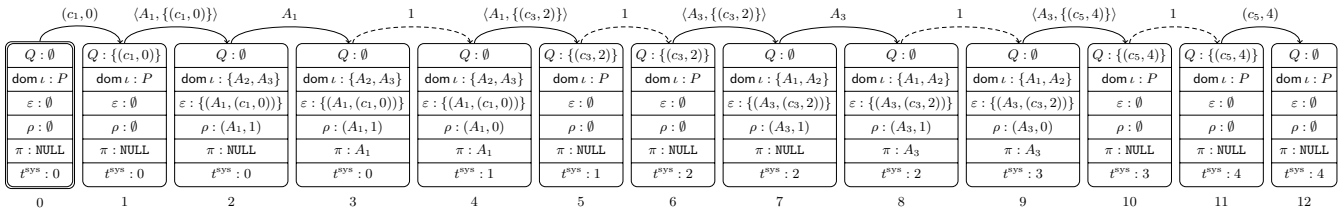

Figure 8. A prefix of an execution of the system of Figure 3(a) corresponding to the program execution of Figure 5, where event values and actor states have been omitted.

We say that E^{sys} is *fair* if and only if $\mathsf{prog}\, E^{\text{sys}}$ is actor-fair.

We say that E^{sys} is *correct* if and only if E^{sys} is safe and fair.

THEOREM 5.5. *If E^{sys} is correct, then $\mathsf{prog}\, E^{\text{sys}}$ is correct.*

6. SCHEDULABILITY

By Theorem 4.5, every well defined program will be able to execute correctly when presented with any discrete-event input signal. For a system, this is clearly not the case. The problem is to decide whether a given system will be able to execute correctly when presented with any discrete-event input signal from some given class of such signals.

First we need to address a technical issue. Since computation-time requirements of actors are chosen nondeterministically, we need to be able to quantify system executions of the same underlying program executions over all possible computation-time-requirement choices made.

For every $A \in P$, we write $\mathsf{trace}^A_{\text{start}}\, E^{\text{prog}}$ for $(\mathsf{filter}\, \mathsf{lab}^{\text{A}}_{\text{start}}\, \text{P})\, E^{\text{prog}}$, and $\mathsf{trace}^{\text{A}}_{\text{start}}\, E^{\text{sys}}$ for $(\mathsf{filter}\, \mathsf{lab}^{\text{A}}_{\text{start}}\, \langle \text{P}, \text{R} \rangle)\, E^{\text{sys}}$.

PROPOSITION 6.1. *For every $A \in P$, there is $O \in \mathscr{S}\, R(A)$ such that*

$$\mathsf{trace}^A_{\text{start}}\, E^{\text{sys}} = \mathsf{zip}\, \langle \mathsf{trace}^A_{\text{start}}\, \mathsf{prog}\, E^{\text{sys}}, O \rangle.$$

We write $(\text{RO}\, \langle P, R \rangle)(E^{\text{prog}})$ for a set such that for every $O \in (\text{RO}\, \langle P, R \rangle)(E^{\text{prog}})$, O is a function from P such that for every $A \in P$, $O(A) \in \mathscr{S}_{|\mathsf{trace}^A_{\text{start}}\, E^{\text{prog}}|}\, R(A)$.

We think of every member of $(\text{RO}\, \langle P, R \rangle)(E^{\text{prog}})$ as an oracle, capturing the nondeterministic choices of computation-time requirements for each actor over a given system realization of the program execution E^{prog}.

DEFINITION 6.2. *A* model *is an ordered pair $\langle \langle P, R \rangle, I \rangle$ such that the following are true:*

1. *$\langle P, R \rangle$ is a well defined system;*
2. *$I \subseteq \text{S}^{\text{prog}}(\text{C}^P_{\text{in}})$.*

Assume a model $\langle \langle P, R \rangle, I \rangle$. We say that $\langle \langle P, R \rangle, I \rangle$ is *schedulable* if and only if for any $s^{\text{prog}} \in I$, every correct execution E^{prog} of P such that

$$\mathsf{in}\, E^{\text{prog}} = s^{\text{prog}},$$

and every $O \in (\text{RO}\, \langle P, R \rangle)(E^{\text{prog}})$, there is a correct execution E^{sys} of $\langle P, R \rangle$ such that

$$\mathsf{in}\, \mathsf{prog}\, E^{\text{sys}} = s^{\text{prog}},$$

and for every $A \in P$,

$$\mathsf{trace}^A_{\text{start}}\, E^{\text{sys}} = \mathsf{zip}\, \langle \mathsf{trace}^A_{\text{start}}\, E^{\text{prog}}, O(A) \rangle.$$

A convenient tool in approaching schedulability problems is the identification of a particular scheduling strategy that is optimal, in the sense that if a system is schedulable, then it is also schedulable under that particular strategy. For uniprocessor platforms, the earliest-deadline-first (EDF) strategy has been proven optimal over a variety of different schedulability problems. To make it applicable here, we first need to formalize a notion of deadline for the events circulating inside the program of a system.

We write $\mathsf{deadline}\, P$ for a function from $\bigcup \{\text{IA}(\text{C}^A_{\text{in}}) \mid A \in P\}$ to $\mathbb{T}$ such that for every $\alpha \in \bigcup \{\text{IA}(\text{C}^A_{\text{in}}) \mid A \in P\}$,

$$(\mathsf{deadline}\, P)(\alpha) = \mathsf{time}^{\text{prog}}\, \alpha + (\mathsf{delay}\, P)(\mathsf{chan}\, \alpha, \text{C}^P_{\text{out}}).$$

In other words, an event's deadline is the earliest time stamp of any event eventually caused by the former that reaches an actuator.

We say that E^{sys} is *earliest-deadline-first* if and only if for every $A \in P$, the following are true:

1. for any n and $\langle Q, \iota, \varepsilon, \rho, \pi, t^{\text{sys}} \rangle$ such that
$$E^{\text{sys}}_{2n} = \langle Q, \iota, \varepsilon, \rho, \pi, t^{\text{sys}} \rangle,$$
and any d such that A is ready in $\langle Q, \iota, \varepsilon, \rho, \pi, t^{\text{sys}} + d \rangle$, if $E^{\text{sys}}_{2n+1} \in \mathbb{T}$, then $E^{\text{sys}}_{2n+1} \leq d$;
2. for any n, $\langle Q, \iota, \varepsilon, \rho, \pi, t^{\text{sys}} \rangle$, and $\langle s, \alpha \rangle$ such that $E^{\text{sys}}_{2n} = \langle Q, \iota, \varepsilon, \rho, \pi, t^{\text{sys}} \rangle$, and
$$\pi \notin \{A \mid \text{there is } \langle s, \alpha \rangle \text{ such that } \varepsilon(A) = \langle s, \alpha \rangle, \text{ and for every } A' \text{ and } \langle s', \alpha' \rangle \text{ such that } \varepsilon(A') = \langle s', \alpha' \rangle, (\mathsf{deadline}\, P)(\alpha) \leq (\mathsf{deadline}\, P)(\alpha')\},$$
if $E^{\text{sys}}_{2n+1} \in \mathbb{T}$, then $E^{\text{sys}}_{2n+1} = 0$;
3. for any n and $\langle Q, \iota, \varepsilon, \rho, \pi, t^{\text{sys}} \rangle$ such that
$$E^{\text{sys}}_{2n} = \langle Q, \iota, \varepsilon, \rho, \pi, t^{\text{sys}} \rangle,$$
$A \in \mathsf{dom}\, \varepsilon$, and $\rho(A) = 0$, if $E^{\text{sys}}_{2n+1} \in \mathbb{T}$, then $E^{\text{sys}}_{2n+1} = 0$.

In other words, a system execution is earliest-deadline-first just as long as it is eager with respect to start and finish transitions, and at each time instant, allocates the processor to the actor processing the events with the smallest deadline.

PROPOSITION 6.3. *If E^{sys} is earliest-deadline-first, then E^{sys} is fair.*

THEOREM 6.4. *If E^{sys} is earliest-deadline-first, then E^{sys} is correct if and only if E^{sys} is safe.*

THEOREM 6.5. *For any $s^{\text{prog}} \in I$, every execution E^{prog} of P such that*

$$\mathsf{in}\, E^{\text{prog}} = s^{\text{prog}},$$

and every $O \in (\mathrm{RO}\,\langle P, R\rangle)(E^{\mathrm{prog}})$*, if there is a correct execution* E^{sys} *of* $\langle P, R\rangle$ *such that*

$$\mathsf{in\ prog}\, E^{\mathrm{sys}} = s^{\mathrm{prog}},$$

and for every $A \in P$*,*

$$\mathsf{trace}^{A}_{\mathrm{start}}\, E^{\mathrm{sys}} = \mathsf{zip}\,\langle \mathsf{trace}^{A}_{\mathrm{start}}\, E^{\mathrm{prog}}, O(A)\rangle,$$

then there is a correct earliest-deadline-first execution E^{sys} *of* $\langle P, R\rangle$ *such that*

$$\mathsf{in\ prog}\, E^{\mathrm{sys}} = s^{\mathrm{prog}},$$

and for every $A \in P$*,*

$$\mathsf{trace}^{A}_{\mathrm{start}}\, E^{\mathrm{sys}} = \mathsf{zip}\,\langle \mathsf{trace}^{A}_{\mathrm{start}}\, E^{\mathrm{prog}}, O(A)\rangle.$$

Theorem 6.5 states the optimality of the EDF scheduling strategy for our systems.

7. DECIDABILITY

In the previous section we showed that in order to decide if a model is schedulable we can narrow our search in the EDF executions of the system. Furthermore, since the properties of actor-safe and EDF system executions are local properties, we can construct a transition system whose traces are prefixes of actor-safe EDF system executions. The question of whether a non output-safe, actor-safe, and EDF execution exists, is equivalent to whether a state in which an event misses its deadline is reachable. Is the reachability of a deadline miss state decidable? At first the answer seems negative since the state space of a system is infinite. However, since we are only interested in the schedulability of the system we can abstract away part of its state.

First, we abstract all states that contain any events that have missed their deadline under a new state called `error`.

Note that we have constrained the actors in a program to be output homogeneous and constant delay. Effectively this means that the timing properties of the executions of a system, and thus its schedulability, do not depend on the actor states or on the event values, and thus, those can also be abstracted away.

Next, intuitively it should be the case that in correct executions the number of events in any state of the program could not grow unboundedly. Events are associated with an execution time requirement and a deadline, and thus the accumulation of too many execution requirements should conclusively lead the system to a deadline miss. One complication that arises in our programs is the fact that the deadline of an event in a channel does not only depend on that channel but also on the path that the event has followed to reach that channel. Specifically, the deadline of an event is a function of its timestamp which in principle could grow unboundedly (e.g. if the event circles around a program loop). However, what is really of interest in order to bound the number of events, is the relative deadline which does not only depend on the timestamp but rather on the difference between the timestamp and the current system time. That difference can be shown to be bounded in all correct executions for all events in every channel of the program. The lower bound naturally follows the definition of deadline. The upper bound, which claims that the timestamp of an event in a channel cannot grow too much relatively to system time, is a consequence of safety and the requirement that actors are ready.

Theorem 7.1. *For every correct execution* E^{sys} *of* $\langle P, R\rangle$*, every* n *and* $\langle Q, \iota, \varepsilon, \rho, \pi, t^{\mathrm{sys}}\rangle$ *such that* $E^{\mathrm{sys}}_{2n} = \langle Q, \iota, \varepsilon, \rho, \pi, t^{\mathrm{sys}}\rangle$*, and any* $e \in Q$*,*

$$-\,(\mathsf{delay}\ P)(\mathsf{chan}\ e, \mathrm{C}^{P}_{\mathrm{out}}) \leq \mathsf{time}^{\mathrm{prog}}\, e - t^{\mathrm{sys}} \leq (\mathsf{delay}\ P)(\mathrm{C}^{P}_{\mathrm{in}}, \mathsf{chan}\ e).$$

Theorem 7.2. *For every correct execution* E^{sys} *of* $\langle P, R\rangle$*, every* n *and* $\langle Q, \iota, \varepsilon, \rho, \pi, t^{\mathrm{sys}}\rangle$ *such that*

$$E^{\mathrm{sys}}_{2n} = \langle Q, \iota, \varepsilon, \rho, \pi, t^{\mathrm{sys}}\rangle,$$

every $A \in P$*, and every* $c \in \mathrm{C}^{A}_{\mathrm{in}} \cup \mathrm{C}^{A}_{\mathrm{out}}$*,*

$$|\{e \mid e \in Q \text{ and } \mathsf{chan}\ e = c\}| \leq ((\mathsf{delay}\ P)(\mathrm{C}^{P}_{\mathrm{in}}, c) + (\mathsf{delay}\ P)(c, \mathrm{C}^{P}_{\mathrm{out}}))/\inf R(A) + 1.$$

Theorem 7.2 allows to further abstract under the `error` state all those system states that have channels with more events than the specified bound.

Furthermore, note that both system time and event logical times can grow unboundedly. However, it is possible instead of using system time to timestamp new events at the input transitions, to track the relative time that an event is in the program. This alternative involves associating a *timer* and a *delay* with each event. At input transitions, the timer and the delay are set to zero. At time transitions, all the events's timers are increased by the time elapsed. At every finish transition, the delay of each event in the output action is set equal to the sum of the delay of the input action and the delay of the corresponding output channel of the actor. It is easy to see that the difference between the delay and the timer is always equal to the difference between the logical time of the event and system time. Since all operations of the EDF transition system only involve differences between logical time and system time, this alternative scheme, which tracks the relative time an event is in the program, can replace system time and event logical times. Moreover, because of Theorem 7.1, it is possible to keep timers and delays of events bounded. Since their difference is bounded, we can set a limit for the value of a timer, and exactly when the timer crosses the limit, reset it and subtract the value of the limit from the corresponding delay so that their difference stays the same.

So far, we have argued that the part of the EDF transition system that handles events has a bounded discrete state and uses timers that are linearly compared and reset. Therefore, that can be implemented with a timed automaton. What remains is to see whether the part of the system that deals with actor execution can also be implemented using finite state and "clocks".

We write $\mathsf{worst}\ R$ for a function from P to $\mathscr{P}_{\geq 1}\,\mathbb{T}$ such that for every $A \in P$, $(\mathsf{worst}\ R)(A) = \{\sup R(A)\}$.

Theorem 7.3. $\langle\langle P, R\rangle, I\rangle$ *is schedulable if and only if* $\langle\langle P, \mathsf{worst}\ R\rangle, I\rangle$ *is schedulable.*

In the EDF transition system, actor execution is tracked using the ρ function. At the start transition of an actor A, $\rho(A)$ is set equal to a value in $R(A)$, which from Theorem 7.3 can be fixed to $\sup R(A)$, and the ρ value of the actor that is executing decreases as time elapses at time transitions. An actor completes its execution when its ρ value is equal to zero. Note that this is equivalent to setting $\rho(A)$ to zero

initially and increase it until it reaches $\sup R(A)$. To show that this functionality can be implemented with clocks that do not freeze when an actor is not executing, we add in the state a value that tracks the total preemption time of an actor. When an actor A is first assigned the processor, $\rho(A)$ is set to zero, and at time transitions all ρ values are increased by the elapsed time. When an actor A is preempted by another actor B, $\sup R(B)$ is added to the preemption time of A. An actor A finishes executing when its ρ value is equal to the sum of $\sup R(A)$ and its preemption time. The scheme above is correct since, in EDF, when B preempts A, B will finish executing before A executes again. Lastly, note that an actor is added to ρ's domain when the actor is first allocated the processor and not at the start transition. This is because any delay that follows the allocation time can be accounted for precisely, whereas the interval between start time and allocation time cannot.

With actor execution, all parts of the system have been shown to require finite discrete state and continuous variables that behave like clocks. Hence, the EDF transition system can be implemented as a timed automaton. We use timed automata with deadlines and priorities, as introduced in [5], because they simplify modeling. The complete definition can be found in Appendix B. We write $\mathsf{TADP}\,\langle\langle P, R\rangle, b\rangle$ for the resulting timed automaton with deadlines and priorities that simulates safe EDF executions for system $\langle P, R\rangle$, where b is the chosen limit of the event timers.

In a timed automaton implementation of the system, the inputs also have to be described using a timed automaton. An *input model* of sort C is a timelock-free TADP (see [4]) such that the label set of the automaton is a subset of $L \subseteq C \cup \{\tau\}$, and in any run of the automaton, for each time instant, and every $c \in C$, there can only be one transition with label c (see Appendix B for a formal definition).

Assume an input model IM of sort C. We write $\mathsf{sig}^{\text{prog}}\, IM$ for a subset of $\mathrm{S}^{\text{prog}}(C)$ such that for every $s^{\text{prog}} \in \mathrm{S}^{\text{prog}}(C)$, $s^{\text{prog}} \in \mathsf{sig}^{\text{prog}}\, IM$ if and only if there is a run of IM that simulates the event arrival times of s^{prog} (see Appendix B for a formal definition).

We say that $\mathsf{TADP}\,\langle\langle P, R\rangle, b\rangle \mid\mid IM$ is *safe* if and only if for every s, $\langle \texttt{error}, s\rangle$ is not reachable in $\mathsf{TADP}\,\langle\langle P, R\rangle, b\rangle \mid\mid IM$.

THEOREM 7.4. *The following are equivalent: (1) for every $s^{\text{prog}} \in \mathsf{sig}^{\text{prog}}\, IM$, there is a safe earliest-deadline-first execution E^{sys} of $\langle P, \mathsf{worst}\, R\rangle$ such that $\mathsf{in}\, E^{\text{sys}} = s^{\text{prog}}$; (2) for every $b > 0$, $\mathsf{TADP}\,\langle\langle P, R\rangle, b\rangle \mid\mid IM$ is safe.*

Theorem 7.4, along with Theorem 6.4 and 6.5, establishes the decidability of the schedulability problem.

8. RELATED WORK

The programming model described in the paper has been implemented in a framework called Ptides [7]. Ptides also employs clock synchronization algorithms and network delay bounds in order to uniformly support the same discrete-event semantics across distributed embedded platforms.

Synchronous languages (see [3]) have also been used for programming real-time systems. However, their approach is different than the one presented here, in that latencies arise from the implementation instead of being part of the programming abstraction.

Our programming model belongs in the family of logically execution time based models which was pioneered by Giotto [10]. A significant difference between Giotto and our case is the fact that the former is time-triggered. That distinction is moderated with xGiotto [9], which is an event-triggered extension of Giotto that, however, does not allow for the specification of relative deadlines on events.

Furthermore, timed automata have been used before for testing schedulability of real-time systems. [8] presents a real-time system model, called task automata, where asynchronous processes are bound to timed automata locations, thereby allowing for considerable expressiveness in the task arrival and dependency patterns. Our work is inspired by that work. Schedulability there is checked algorithmically via reduction to a decidable subclass of suspension automata. In contrast, the encoding of preemption times in the discrete state allows us to use regular timed automata.

Lastly, on the topic of schedulability of real-time systems, recent advances in pseudo-polynomial schedulability algorithms, culminating in the digraph model [12], are pushing the boundary of the expressiveness of such techniques with the ability to model different job types and conditional execution. However, all these models rely on the task independence assumption (see [1]) to gain tractability. Specifically, they cannot accurately model systems where executions of different tasks depend on each other, as is the case in our programming model.

9. REFERENCES

[1] S. Baruah, D. Chen, S. Gorinsky, and A. Mok. Generalized multiframe tasks. *Real-Time Systems*, 17(1):5–22, 1999.

[2] S. K. Baruah, A. K. Mok, and L. E. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *RTSS 11th*, December 1990.

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

[4] S. Bornot, G. Gößler, and J. Sifakis. On the Construction of Live Timed Systems. In *TACAS'00*.

[5] S. Bornot, J. Sifakis, and S. Tripakis. Modeling urgency in timed systems. In *Compositionality: The Significant Difference*, LNCS 1536. 1998.

[6] C. Cassandras and S. Lafortune. *Introduction to discrete event systems*, volume 11. Kluwer academic publishers, 1999.

[7] J. Eidson, E. Lee, S. Matic, S. Seshia, and J. Zou. Distributed real-time software for cyber-physical systems. *Proceedings of the IEEE*, 100(1):45–59, 2012.

[8] E. Fersman, P. Krcal, P. Pettersson, and W. Yi. Task automata: Schedulability, decidability and undecidability. *Information and Computation*, 2007.

[9] A. Ghosal, T. A. Henzinger, C. M. Kirsch, and M. A. A. Sanvido. Event-driven programming with logical execution times. In *HSCC*. Springer, 2004.

[10] T. A. Henzinger, B. Horowitz, and C. M. Kirsch. Giotto: A time-triggered language for embedded programming. In *EMSOFT'01*. Springer-Verlag, 2001.

[11] E. Matsikoudis, C. Stergiou, and E. A. Lee. On the schedulability of real-time discrete-event systems (extended version). Available from `http://chess.eecs.berkeley.edu/pubs/1001.html`.

[12] M. Stigge, P. Ekberg, N. Guan, and W. Yi. The Digraph Real-Time Task Model. In *RTAS*, 2011.

APPENDIX

A. PROGRAM EXECUTION SAFETY AND FAIRNESS

We say that E^{prog} is *safe* if and only if for every $A \in P$, and any n_1, $\langle Q_1, \iota_1, \varepsilon_1\rangle$, and $\langle s_1, \alpha_1\rangle$ such that

$$E^{\text{prog}}_{2n_1} = \langle Q_1, \iota_1, \varepsilon_1\rangle$$

and

$$\varepsilon_1(A) = \langle s_1, \alpha_1\rangle,$$

and any n_2, $\langle Q_2, \iota_2, \varepsilon_2\rangle$, and $\langle s_2, \alpha_2\rangle$ such that

$$E^{\text{prog}}_{2n_2} = \langle Q_2, \iota_2, \varepsilon_2\rangle$$

and

$$\varepsilon_2(A) = \langle s_2, \alpha_2\rangle,$$

if $n_1 < n_2$, then $\mathsf{time}^{\text{prog}}\,\alpha_1 < \mathsf{time}^{\text{prog}}\,\alpha_2$.

We say that E^{prog} is *actor-fair* if and only if for every $A \in P$, the following are true:

1. for any n and $\langle Q, \iota, \varepsilon\rangle$ such that
$$E^{\text{prog}}_{2n} = \langle Q, \iota, \varepsilon\rangle,$$
if $A \in \mathsf{dom}\,\iota$, and there is $e \in Q$ such that $\mathsf{chan}\,e \in \mathrm{C}^A_{\text{in}}$, then there is n', $\langle Q', \iota', \varepsilon'\rangle$, s', and α' such that $n < n'$,
$$E^{\text{prog}}_{2n'} = \langle Q', \iota', \varepsilon'\rangle,$$
$A \in \mathsf{dom}\,\varepsilon'$,
$$\varepsilon'(A) = \langle s', \alpha'\rangle,$$
and $e \in \alpha'$;
2. for any n and $\langle Q, \iota, \varepsilon\rangle$ such that
$$E^{\text{prog}}_{2n} = \langle Q, \iota, \varepsilon\rangle,$$
if $A \in \mathsf{dom}\,\varepsilon$, then there is n' and $\langle Q', \iota', \varepsilon'\rangle$ such that $n < n'$,
$$E^{\text{prog}}_{2n'} = \langle Q', \iota', \varepsilon'\rangle,$$
and $A \in \mathsf{dom}\,\iota'$.

We say that E^{prog} is *output-fair* if and only if for every $c \in \mathrm{C}^P_{\text{out}}$, and any n and $\langle Q, \iota, \varepsilon\rangle$ such that

$$E^{\text{prog}}_{2n} = \langle Q, \iota, \varepsilon\rangle,$$

if there is $e \in Q$ such that

$$\mathsf{chan}\,e = c,$$

then there is n' such that $n \le n'$ and

$$E^{\text{prog}}_{2n'+1} = e.$$

B. TIMED AUTOMATON DEFINITION

We postulate a non-empty class $\mathbb{X}$ of *clock symbols*.

Assume a non-empty subset X of $\mathbb{X}$.

Definition B.1. *An X-valuation is a function from X to $\mathbb{T}$.*

We write $\mathrm{V}(X)$ for the set of all X-valuations.

Definition B.2. *The set $\Gamma(X)$ of clock constraints γ is defined by the grammar*

$$\gamma := x \le r \mid r \le x \mid x_1 - x_2 \le r \mid r \le x_1 - x_2 \mid \neg\gamma \mid \gamma_1 \wedge \gamma_2$$

where x, x_1, and x_2 are clocks in X, and $r \in \mathbb{Q}_{\ge 0}$.

A member γ of $\Gamma(X)$ is called an X-constraint.

For every X-valuation v and every X-constraint γ, we say that v *satisfies* γ, and write $\models \gamma(v)$, if and only if the formula is true when every clock x in γ is replaced with $v(x)$.

Definition B.3. *A* timed automaton with deadlines and priorities *(*TADP*) is an ordered sextuple $\langle S, s_{\text{init}}, L, X, T, \preceq\rangle$ such that the following are true:*

1. *S is a finite set;*
2. *$s_{\text{init}} \in S$;*
3. *L is a finite set;*
4. *X is a subset of $\mathbb{X}$;*
5. *T is a subset of $S \times \Gamma(X) \times \Gamma(X) \times L \times \mathscr{P}\,X \times S$ such that for every $\langle s_1, \gamma, \delta, l, U s_2\rangle \in T$ and every $v \in \mathrm{V}(X)$, if $\models \delta(v)$, then $\models \gamma(v)$;*
6. *$\preceq$ is an order on L.*

Assume a TADP $\langle S, s_{\text{init}}, L, X, T, \preceq\rangle$.

We write $\longrightarrow_{\langle S, s_{\text{init}}, L, X, T, \preceq\rangle}$ for a ternary relation between $S \times \mathrm{V}(X)$, L, and $S \times \mathrm{V}(X)$ such that for every $\langle s_1, v_1\rangle \in S \times \mathrm{V}(X)$, $l \in L$, and $\langle s_2, v_2\rangle \in S \times \mathrm{V}(X)$,

$$\longrightarrow_{\langle S, s_{\text{init}}, L, X, T, \preceq\rangle} (\langle s_1, v_1\rangle, l, \langle s_2, v_2\rangle)$$

if and only if there is γ, δ, and U such that the following are true:

1. $\langle s_1, \gamma, \delta, l, U, s_2\rangle \in T$ and $\models \gamma(v_1)$, and for every γ', δ', l', U', and s_2' such that $\langle s_1, \gamma', \delta', l', U', s_2'\rangle \in T$ and $\models \gamma'(v_1)$, $l' \preceq l$;
2. for every $x \in X$, $v_2(x)$ is equal to 0 if $x \in U$ and $v_1(x)$ otherwise.

We write $\langle s_1, v_1\rangle \xrightarrow{l}_{\langle S, s_{\text{init}}, L, X, T, \preceq\rangle} \langle s_2, v_2\rangle$ if and only if $\longrightarrow_{\langle S, s_{\text{init}}, L, X, T, \preceq\rangle} (\langle s_1, v_1\rangle, l, \langle s_2, v_2\rangle)$.

We write $\dashrightarrow_{\langle S, s_{\text{init}}, L, X, T, \preceq\rangle}$ for a ternary relation between $S \times \mathrm{V}(X)$, $\mathbb{T}$, and $S \times \mathrm{V}(X)$ such that for every $\langle s_1, v_1\rangle \in S \times \mathrm{V}(X)$, $d \in \mathbb{T}$, and $\langle s_2, v_2\rangle \in S \times \mathrm{V}(X)$,

$$\dashrightarrow_{\langle S, s_{\text{init}}, L, X, T, \preceq\rangle} (\langle s_1, v_1\rangle, d, \langle s_2, v_2\rangle)$$

if and only if the following are true:

1. $s_1 = s_2$, and for every γ, δ, l, U, and s_2' such that $\langle s_1, \gamma, \delta, l, U, s_2'\rangle \in T$, and every $d' < d$, $\not\models \delta(v_1')$, where v_1' is an X-valuation such that for every $x \in X$,
$$v_1'(x) = v_1(x) + d'.$$
2. for every $x \in X$,
$$v_2(x) = v_1(x) + d.$$

We write $\langle s_1, v_1\rangle \xdashrightarrow{d}_{\langle S, s_{\text{init}}, L, X, T, \preceq\rangle} \langle s_2, v_2\rangle$ if and only if $\dashrightarrow_{\langle S, s_{\text{init}}, L, X, T, \preceq\rangle} (\langle s_1, v_1\rangle, d, \langle s_2, v_2\rangle)$.

Definition B.4. *A* run *of $\langle S, s_{\text{init}}, L, X, T, \preceq\rangle$ is an infinite sequence R such that the following are true:*

1. *there is $v_{\text{init}} \in V(X)$ such that for every $x \in X$, $v_{\text{init}}(x) =$ and $R_0 = \langle s_{\text{init}}, v_{\text{init}}\rangle$;*
2. *for every $n \in \mathbb{N}$, one of the following is true:*
   (a) $R_{2n} \xrightarrow{R_{2n+1}}_{\langle S, s_{\text{init}}, L, X, T, \preceq\rangle} R_{2n+2}$;
   (b) $R_{2n} \xdashrightarrow{R_{2n+1}}_{\langle S, s_{\text{init}}, L, X, T, \preceq\rangle} R_{2n+2}$;

We say that s is *reachable* in $\langle S, s_{\text{init}}, L, X, T, \preceq \rangle$ if and only if there is a run R of $\langle S, s_{\text{init}}, L, X, T, \preceq \rangle$, an X-valuation v, and n such that $R_n = \langle s, v \rangle$.

Assume a run R of $\langle S, s_{\text{init}}, L, X, T, \preceq \rangle$.

We write $\mathsf{lapse}\, R$ for a function from $\mathbb{N} \times \mathbb{N}$ to $\mathbb{T}$ such that for every $n_1, n_2 \in \mathbb{N}$,

$$(\mathsf{lapse}\, R)(n_1, n_2) = \sum \{R_{2n+1} \mid n_1 \leq n < n_2 \text{ and } R2n' + 1 \in \mathbb{T}\}.$$

We say that R is *divergent* if and only if for every $t \in \mathbb{T}$, there is $n \in \mathbb{N}$ such that

$$t \leq (\mathsf{lapse}\, R)(0, n).$$

We say that $\langle S, s_{\text{init}}, L, X, T, \preceq \rangle$ is *timelock-free* if and only if for every run R of $\langle S, s_{\text{init}}, L, X, T, \preceq \rangle$, and every $n \in \mathbb{N}$, there is a divergent run R' of $\langle S, s_{\text{init}}, L, X, T, \preceq \rangle$ such that for any $n' < n$,

$$R'_{2n'} = R_{2n'}$$

and

$$R'_{2n'+1} = R_{2n'+1}.$$

Assume $b \in \mathbb{N}$.

We write $\mathrm{R}_{\mathrm{AD}}(b)$ for a function from $\mathsf{chan}\, P$ to $\mathscr{P}_{\text{fin}}\, \mathbb{Q}_{\geq 0}$ such that for every $c \in \mathsf{chan}\, P$,

$$\mathrm{R}_{\mathrm{AD}}(b)(c) = \{i \cdot g \mid i \in \mathbb{Z} \text{ and } -(\mathsf{delay}\, P)(c, \mathrm{C}^P_{\text{out}}) \leq i \cdot g \leq (\mathsf{delay}\, P)(\mathrm{C}^P_{\text{in}}, c) + b \cdot g\},$$

where $g = \mathrm{GCD}(\{(\mathsf{delay}\, A)(c) \mid A \in P \text{ and } c \in \mathrm{C}^A_{\text{out}}\})$.

For a channel c, the set $\mathrm{R}_{\mathrm{AD}}(b)(c)$ is the domain of the accumulated delay values of events in c.

We write $\mathrm{b}_{\text{queue}}$ for a function from $\mathsf{chan}\, P$ to $\mathbb{N}$ such that for every $c \in \mathsf{chan}\, P$ and every $A \in P$,

$$\mathrm{b}_{\text{queue}}(c) = \begin{cases} \left\lfloor \frac{(\mathsf{delay}\, P)(\mathrm{C}^P_{\text{in}}, c) + (\mathsf{delay}\, P)(c, \mathrm{C}^P_{\text{out}})}{(\sup R)(A)} \right\rfloor & \text{if } c \in \mathrm{C}^A_{\text{in}}; \\ \left\lfloor \frac{(\mathsf{delay}\, P)(\mathrm{C}^P_{\text{in}}, c) + (\mathsf{delay}\, P)(c, \mathrm{C}^P_{\text{out}})}{(\sup R)(A)} \right\rfloor + 1 & \text{if } c \in \mathrm{C}^A_{\text{out}} \cap \mathrm{C}^P_{\text{out}}. \end{cases}$$

We write R_{PD} for a function from P to $\mathscr{P}_{\text{fin}}\, \mathbb{Q}_{\geq 0}$ such that for every $A \in P$ and every $r \in \mathbb{Q}_{\geq 0}$, $r \in \mathrm{R}_{\mathrm{PD}}(A)$ if and only if there is a subset P' of $P \setminus \{A\}$, and a function f from P' to $\mathbb{N}$ such that

$$\sum \{f(A') \cdot (\sup R)(A') \mid A' \in P'\} \leq (\mathsf{delay}\, P)(\mathrm{C}^P_{\text{in}}, \mathrm{C}^A_{\text{in}}) + (\mathsf{delay}\, P)(\mathrm{C}^A_{\text{in}}, \mathrm{C}^P_{\text{out}})$$

and

$$r = \sum \{(f(A') + 1) \cdot (\sup R)(A') \mid A' \in P'\}.$$

For every actor A, $\mathrm{R}_{\mathrm{PD}}(A)$ is the set of possible values for the preemption delay of A.

For every $n \in \mathbb{N}$, we fix a distinct clock symbol $\mathtt{x}_n$, and for every actor A, a distinct clock symbol $\mathtt{x}_A$.

$\mathtt{x}_n$ are the clocks assigned to events and $\mathtt{x}_A$ are the clocks that track the execution of actors.

We write $\mathsf{TADP}\, \langle \langle P, R \rangle, b \rangle$ for a TADP $\langle S, s_{\text{init}}, L, X, T, \preceq \rangle$ such that the following are true:

1. S is the set of all s such that one of the following is true:
   (a) s is a ordered quadruple $\langle Q, \varepsilon, \rho, \pi \rangle$ such that the following are true:
      i. Q is a function from $\mathsf{chan}\, P$ such that for every $c \in \mathsf{chan}\, P$, $Q(c) \in \mathscr{S}_{\leq \mathrm{b}_{\text{queue}}(c)}(X \times \mathrm{R}_{\mathrm{AD}}(b)(c))$;
      ii. there is a subset P_{exec} of P such that ε is a function from P_{exec} such that for any $A \in P_{\text{exec}}$,
      $$\varepsilon(A) \in X \times \bigcup \{\mathrm{R}_{\mathrm{AD}}(b)(c) \mid c \in \mathrm{C}^A_{\text{in}}\};$$
      iii. there is a subset P_{run} of $\mathsf{dom}\, \varepsilon$ such that ρ is a function from P_{run} such that for any $A \in P_{\text{run}}$,
      $$\rho(A) \in \mathrm{R}_{\mathrm{PD}}(A);$$
      iv. $\pi \in \{\mathtt{NULL}\} \cup P$;
   (b) $s = \mathtt{error}$;
2. s_{init} is an ordered quadruple $\langle Q_{\text{init}}, \varepsilon_{\text{init}}, \rho_{\text{init}}, \pi_{\text{init}} \rangle$ such that Q_{init} is a function from $\mathsf{chan}\, P$ such that for every $c \in \mathsf{chan}\, P$, $Q_{\text{init}}(c) = \langle\ \rangle$, $\varepsilon_{\text{init}}$ is the empty function, ρ_{init} is the empty function, and $\pi_{\text{init}} = \mathtt{NULL}$;
3. $L = (\mathrm{C}^P_{\text{in}}) \cup \{\tau\} \cup (\mathrm{C}^P_{\text{out}}) \cup \{\mathtt{miss}\}$;
4. $X = \{\mathtt{x}_n \mid n \in \mathbb{N} \text{ and } n < \sum \{\mathrm{b}_{\text{queue}}(c) \mid c \in \mathsf{chan}\, P\}\} \cup \{\mathtt{x}_A \mid A \in P\}$;
5. T is a subset of $S \times \Gamma(X) \times \Gamma(X) \times L \times \mathscr{P}\, X \times S$ such that one of the following is true:
   (a) for every $\langle Q_1, \varepsilon_1, \rho_1, \pi_1 \rangle \in S$, every $\gamma \in \Gamma(X)$, every $\delta \in \Gamma(X)$, every $l \in L$, every $U \in \mathscr{P}\, X$, and every $\langle Q_2, \varepsilon_2, \rho_2, \pi_2 \rangle \in S$,
   $$\langle \langle Q_1, \varepsilon_1, \rho_1, \pi_1 \rangle, \gamma, \delta, l, U, \langle Q_2, \varepsilon_2, \rho_2, \pi_2 \rangle \rangle \in T$$
   if and only if one of the following is true:
      i. there is $c \in \mathrm{C}^P_{\text{in}}$ and i such that the following are true:
         A. $|Q_1(c)| < \mathrm{b}_{\text{queue}}(c)$;
         B. $i = \min\{j \mid$ for every $c' \in \mathsf{chan}\, P$, every δ, and any $n < |Q_1(c')|$, $Q_1(c')(n) \neq \langle \mathtt{x}_j, \delta \rangle$, and for any $A \in \mathsf{dom}\, \varepsilon_1$ and every δ, $\varepsilon_1(A) \neq \langle \mathtt{x}_j, \delta \rangle\}$;
         C. $\gamma = \mathtt{true}$;
         D. $\delta = \mathtt{false}$;
         E. $l = c$;
         F. $U = \{\mathtt{x}_i\}$;
         G. for every $c' \in \mathsf{chan}\, P$,
         $$Q_2(c') = \begin{cases} Q_1(c') \cdot \langle \langle \mathtt{x}_i, 0 \rangle \rangle & \text{if } c' = c; \\ Q_1(c') & \text{otherwise;} \end{cases}$$
         H. $\varepsilon_2 = \varepsilon_1$;
         I. $\rho_2 = \rho_1$;
         J. $\pi_2 = \pi_1$;
      ii. there is A, $I \subseteq \mathrm{C}^A_{\text{in}}$, $c \in I$, and $\langle x, d \rangle$ such that the following are true:
         A. for every $c' \in I$, $|Q_1(c')| > 0$, and $\mathsf{head}\, Q_1(c) = \langle x, d \rangle$;
         B. $A \notin \mathsf{dom}\, \varepsilon_1$;
         C. $\gamma = \gamma_{\text{action}} \wedge \gamma_{\text{ready}}$, where the following are true:

(1) $\gamma_{\text{action}} = \bigwedge\{d' - x' = d - x$
there is $c' \in I$ such that
$\mathsf{head}\, Q_1(c') = \langle x', d' \rangle\}$;

(2) $\gamma_{\text{ready}} = \gamma_1 \wedge \gamma_2 \wedge \gamma_3$, where the following are true:
$\gamma_1 = x \geq d - (\mathsf{delay}\, P)(\mathrm{C}^P_{\text{in}}, \mathrm{C}^A_{\text{in}})$;
$\gamma_2 = \bigwedge\{d' - x' + (\mathsf{delay}\, P)(c', \mathrm{C}^A_{\text{in}})$
$> d - x \mid c' \notin I$
and $\mathsf{head}\, Q(c') = \langle x', d' \rangle\}$;
$\gamma_3 = \bigwedge\{d' - x'$
$+(\mathsf{delay}\, P)(\mathrm{C}^{A'}_{\text{in}}, \mathrm{C}^A_{\text{in}})$
$> d - x \mid$
$\varepsilon_1(A') = \langle x', d' \rangle\}$;

D. $\delta = \gamma$;
E. $l = \tau$;
F. $U = \emptyset$;
G. for every $c' \in \mathsf{chan}\, P$,

$$Q_2(c') = \begin{cases} \mathsf{tail}\, Q_1(c') & \text{if } c' \in I; \\ Q_1(c') & \text{otherwise;} \end{cases}$$

H. $\varepsilon_2 = \varepsilon_1 \cup \{\langle A, \langle x, d \rangle\rangle\}$;
I. $\rho_2 = \rho_1$;
J. $\pi_2 = \pi_1$;

iii. there is $\langle x, d \rangle$ such that the following are true:
A. for any $c \in (\mathrm{C}^{\pi_1}_{\text{out}} \setminus \mathrm{C}^P_{\text{out}})$, $|Q_1(c)| < \mathrm{b}_{\text{queue}}(c)$;
B. $\varepsilon_1(\pi_1) = \langle x, d \rangle$;
C. $\gamma = \mathtt{x}_{\pi_1} = (\sup R)(\pi_1) + \rho_1(\pi_1)$;
D. $\delta = \gamma$;
E. $l = \tau$;
F. $U = \emptyset$;
G. for every $c \in \mathsf{chan}\, P$,

$$Q_2(c) = \begin{cases} Q_1(c) \cdot \langle\langle x, d + (\mathsf{delay}\, \pi_1)(c) \rangle\rangle \\ \qquad\qquad \text{if } c \in \mathrm{C}^{\pi_1}_{\text{out}}; \\ Q_1(c) \qquad \text{otherwise;} \end{cases}$$

H. $\varepsilon_2 = \varepsilon_1 \setminus \{\langle A, \varepsilon_1(A) \rangle\}$;
I. $\rho_2 = \rho_1 \setminus \{\langle A, \rho_1(A) \rangle\}$;
J. $\pi_2 = \mathtt{NULL}$;

iv. there is $c \in \mathrm{C}^P_{\text{out}}$ and $\langle x, d \rangle$ such that the following are true:
A. $\mathsf{head}\, Q_1(c) = \langle x, d \rangle$;
B. $\gamma = x = d$;
C. $\delta = \gamma$;
D. $l = c$;
E. $U = \emptyset$;
F. for every $c' \in \mathsf{chan}\, P$,

$$Q_2(c') = \begin{cases} \mathsf{tail}\, Q_1(c') & \text{if } c' = c; \\ Q_1(c') & \text{otherwise;} \end{cases}$$

G. $\varepsilon_2 = \varepsilon_1$;
H. $\rho_2 = \rho_1$;
I. $\pi_2 = \pi_1$;

v. there is A and $\langle x, d \rangle$ such that the following are true:
A. $\varepsilon_1(A) = \langle x, d \rangle$;
B. $A \notin \mathsf{dom}\, \rho_1$;
C. $\pi_1 = \mathtt{NULL}$;
D. $\gamma = \bigwedge\{(\mathsf{deadline}\, P)(A, \langle x, d \rangle) \leq$
$(\mathsf{deadline}\, P)(A', \langle x', d' \rangle) \mid$
$\varepsilon_1(A') = \langle x', d' \rangle\}$,
where

$$(\mathsf{deadline}\, P)(A, \langle x, d \rangle) = d - x + (\mathsf{delay}\, P)(\mathrm{C}^A_{\text{in}}, \mathrm{C}^P_{\text{out}});$$

E. $\delta = \gamma$;
F. $l = \tau$;
G. $U = \{\mathtt{x}_A\}$;
H. $Q_2 = Q_1$;
I. $\varepsilon_2 = \varepsilon_1$;
J. ρ_2 is a function from $\mathsf{dom}\, \rho_1 \cup \{A\}$ such that for every $A' \in \mathsf{dom}\, \rho_1 \cup \{A\}$,

$$\rho_2(A') = \begin{cases} 0 & \text{if } A' = A; \\ \rho_1(A') + (\sup R)(A) & \text{otherwise;} \end{cases}$$

K. $\pi_2 = A$;

vi. there is A and $\langle x, d \rangle$ such that the following are true:
A. $\varepsilon_1(A) = \langle x, d \rangle$;
B. $A \notin \mathsf{dom}\, \rho_1$;
C. $\pi_1 \neq \mathtt{NULL}$;
D. $\gamma = (\mathsf{deadline}\, P)(A) <$
$(\mathsf{deadline}\, P)(\pi_1, \varepsilon_1(\pi_1)) \wedge$
$\bigwedge\{(\mathsf{deadline}\, P)(A, \langle x, d \rangle) \leq$
$(\mathsf{deadline}\, P)(A', \langle x', d' \rangle) \mid$
$\varepsilon_1(A') = \langle x', d' \rangle\}$,
where

$$(\mathsf{deadline}\, P)(A, \langle x, d \rangle) = d - x + (\mathsf{delay}\, P)(\mathrm{C}^A_{\text{in}}, \mathrm{C}^P_{\text{out}});$$

E. $\delta = \gamma$;
F. $l = \tau$;
G. $U = \{\mathtt{x}_A\}$;
H. $Q_2 = Q_1$;
I. $\varepsilon_2 = \varepsilon_1$;
J. ρ_2 is a function from $\mathsf{dom}\, \rho_1 \cup \{A\}$ such that for every $A' \in \mathsf{dom}\, \rho_1 \cup \{A\}$,

$$\rho_2(A') = \begin{cases} 0 & \text{if } A' = A; \\ \rho_1(A') + (\sup R)(A) & \text{otherwise;} \end{cases}$$

K. $\pi_2 = A$;

vii. there is A and $\langle x, d \rangle$ such that the following are true:
A. $\varepsilon_1(A) = \langle x, d \rangle$;
B. $A \in \mathsf{dom}\, \rho_1$;
C. $\pi_1 = \mathtt{NULL}$;
D. $\gamma = \bigwedge\{(\mathsf{deadline}\, P)(A, \langle x, d \rangle) \leq$
$(\mathsf{deadline}\, P)(A, \langle x', d' \rangle) \mid$
$\varepsilon_1(A') = \langle x', d' \rangle\}$,
where

$$(\mathsf{deadline}\, P)(A, \langle x, d \rangle) = d - x + (\mathsf{delay}\, P)(\mathrm{C}^A_{\text{in}}, \mathrm{C}^P_{\text{out}});$$

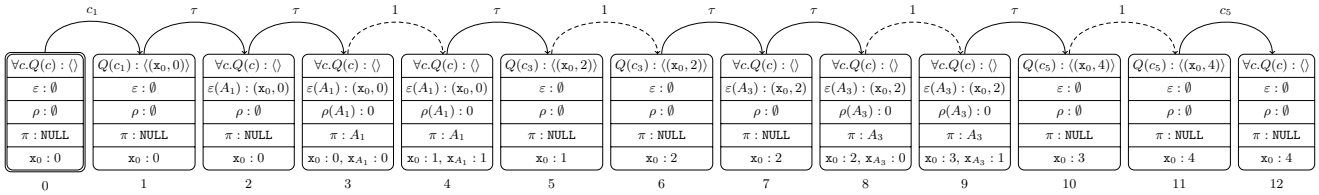

Figure 9. Example TADP execution.

E. $\delta = \gamma$;
F. $l = \tau$;
G. $U = \emptyset$;
H. $Q_2 = Q_1$;
I. $\varepsilon_2 = \varepsilon_1$;
J. $\rho_2 = \rho_1$;
K. $\pi_2 = A$;

viii. there is x and $g = \mathrm{GCD}(\{(\mathsf{delay}\, A)(c) \mid A \in P \text{ and } c \in \mathrm{C}^A_{\mathrm{out}}\})$; such that the following is true:
A. $\gamma = x = b \cdot g$;
B. $\delta = \gamma$;
C. $l = \tau$;
D. $U = \{x\}$;
E. for every $c \in \mathsf{chan}\, P$, any $n < |Q_1(c)|$, and every d,

$$Q_2(c)(n) = \begin{cases} \langle x, d - b \cdot g\rangle \\ \qquad \text{if } Q_1(c)(n) = \langle x, d\rangle; \\ Q_1(c)(n) \qquad \text{otherwise;} \end{cases}$$

F. for any $A \in \mathsf{dom}\, \varepsilon_1$ and every d,

$$\varepsilon_2(A) = \begin{cases} \langle x, d - b \cdot g\rangle & \text{if } \varepsilon_1(A) = \langle x, d\rangle; \\ \varepsilon_1(A) & \text{otherwise;} \end{cases}$$

G. $\rho_2 = \rho_1$;
H. $\pi_2 = \pi_1$;

(b) for every $\langle Q, \varepsilon, \rho, \pi\rangle \in S$, every $\gamma \in \Gamma(X)$, every $\delta \in \Gamma(X)$, every $l \in L$, and every $U \in \mathscr{P}\, X$,

$$\langle\langle Q, \varepsilon, \rho, \pi\rangle, \gamma, \delta, l, U, \mathtt{error}\rangle \in T$$

if and only if one of the following is true:

i. there is $c \in \mathrm{C}^P_{\mathrm{in}}$ such that the following are true:
A. $|Q(c)| = \mathrm{b}_{\mathrm{queue}}(c)$;
B. $\gamma = \mathtt{true}$;
C. $\delta = \gamma$;
D. $l = c$;
E. $U = \emptyset$;

ii. there is $c \in (\mathrm{C}^\pi_{\mathrm{out}} \setminus \mathrm{C}^P_{\mathrm{out}})$ such that the following are true:
A. $|Q(c)| = \mathrm{b}_{\mathrm{queue}}(c)$;
B. $\gamma = \mathtt{x}_\pi = (\sup R)(\pi) + \rho(\pi)$;
C. $\delta = \gamma$;
D. $l = \tau$;
E. $U = \emptyset$;

iii. there is c and $\langle x, d\rangle$ such that the following are true:
A. there is n such that $Q(c)(n) = \langle x, d\rangle$;
B. $\gamma = x \geq d + (\mathsf{delay}\, P)(c, \mathrm{C}^P_{\mathrm{out}})$
C. $\delta = \gamma$;
D. $l = \mathtt{miss}$;
E. $U = \emptyset$;

iv. there is A and $\langle x, d\rangle$ such that the following are true
A. $\varepsilon(A) = \langle x, d\rangle$;
B. $\gamma = x \geq d + (\mathsf{delay}\, P)(\mathrm{C}^A_{\mathrm{in}}, \mathrm{C}^P_{\mathrm{out}})$
C. $\delta = \gamma$;
D. $l = \mathtt{miss}$;
E. $U = \emptyset$;

6. $\preceq$ is the least order on L such that the following are true:
(a) for every $c \in \mathrm{C}^P_{\mathrm{in}}$, $\tau \preceq c$;
(b) for every $c \in \mathrm{C}^P_{\mathrm{out}}$, $c \preceq \tau$ and $\mathtt{miss} \preceq c$.

Definition B.5. *An* input model *of sort C is a timelock-free TADP $\langle S, s_{\mathrm{init}}, L, X, T, \preceq\rangle$ such that the following are true:*

1. *$L \subseteq C \cup \{\tau\}$;*
2. *for every $c \in C$, every run R of $\langle S, s_{\mathrm{init}}, L, X, T, \preceq\rangle$, and every n_1 and n_2 such that $R(n_1) = c$ and $R(n_2) = c$,*

$$0 < (\mathsf{lapse}\, R)(n_1, n_2).$$

Assume an input model IM of sort C.

We write $\mathsf{sig}^{\mathrm{prog}}\, IM$ for a subset of $\mathrm{S}^{\mathrm{prog}}(C)$ such that for every $s^{\mathrm{prog}} \in \mathrm{S}^{\mathrm{prog}}(C)$, $s^{\mathrm{prog}} \in \mathsf{sig}^{\mathrm{prog}}\, IM$ if and only if there is a run R of IM such that

$$\{\langle c, t^{\mathrm{prog}}\rangle \mid \text{there is } v \text{ such that } \langle c, t^{\mathrm{prog}}, v\rangle \in s^{\mathrm{prog}}\} = \{\langle R_{2n+1}, (\mathsf{lapse}\, R)(0, n)\rangle \mid n \in \omega \text{ and } R_{2n+1} \in C\}.$$

Figure 9 shows a prefix of a run of a $\mathsf{TADP}\, \langle\langle P, R\rangle, b\rangle$ where $\langle P, R\rangle$ is the system of Figure 3(a). The run corresponds to the system execution of Figure 8.

Remark about priority of input transitions: The correctness of $\mathsf{TADP}\, \langle\langle P, R\rangle, b\rangle \mid\mid IM$ depends on correctly implementing the input priority property described earlier in Definition 5.3(4). Specifically, in order to guarantee that start transitions are restricted to actors that are ready, it is necessary that, for every time instant, any input transition happens before other transitions of the timed automaton. We focus on a specific channel $c \in \mathrm{C}^P_{\mathrm{in}}$, and distinguish between two cases: the input model of c can be described with a deterministic timed automaton or not. In the former case, the transitions of IM with label c will be eager, and a higher priority, $\tau \preceq c$, is sufficient to guarantee the input priority property. In the latter case, the IM will include a transition

with label c that is delayable or lazy, i.e. it is not necessarily taken as soon as its guard becomes true. Assigning priority $\tau \preceq c$ will block every transition τ as soon as and for as long as that guard is true. Therefore, the implementation of the input priority property that uses $\preceq$ for inputs that correspond to non-deterministic transitions of the IM, is incorrect. Notice that a per time instant instead of global priority is required. That notion of priority can be implemented with the help of an extra clock x_p. The clock is reset in every transition with label τ, and the guard $x_p > 0$ is conjucted to the guards of all IM transitions with label c. This combination guarantees that for each time instant, a τ transition cannot be followed by an input transition c.