A Theory of Testing for Soft Real-Time Processes

Rance Cleaveland*	Insup Lee†	Phil Lewis‡& Scott A. Smolka§
Dept. Computer Science	Dept. Computer & Info. Science	Dept. Computer Science
N.C. State University	U. Pennsylvania	SUNY at Stony Brook
Raleigh, NC 27695	Philadelphia, PA 19104	Stony Brook, NY 11794
rance@csc.ncsu.edu	lee@cis.upenn.edu	{pml,sas}@cs.sunysb.edu

Abstract

We present a semantic framework for soft real-time processes, i.e., processes that meet their deadlines most of the time. The key components of our framework are: (1) a specification language PDP (for Probabilistic and Discrete-time Processes) that incorporates both probabilistic and timing aspects of process behavior; (2) a formal operational semantics for PDP given as a recursively defined probability distribution function over process terms and atomic actions; and (3) a natural notion of a process passing a test with a certain probability, where a test is a process with the added capability of reporting success. By encoding deadlines as tests, the probability of a process passing a test may now be interpreted as the probability of the process meeting a deadline, thereby capturing the essence of soft real-time. A simple video frame transmission example illustrates our approach.

1 Introduction

In the literature on real-time systems (see, for example, [11]), a distinction is often drawn between hard and soft real-time systems. In a *hard real-time system*, all deadlines must be met, as the consequences of failing to meet a deadline can be devastating. Examples of such systems include process control systems for nuclear power plants and fly-by-wire avionics systems. In a *soft real-time system*, the consequences of failing to meet a deadline are not nearly as grave, and thus a certain percentage of missed deadlines can be tolerated. Examples of soft real-time systems include financial data delivery systems and the U.S. Postal Priority Mail system.

Recently, a large amount of research has been dedicated to the formal specification and verification of real-time systems, as it has been observed that such technology is likely to increase the safety and reliability of critical systems software. A number of these efforts, including [13, 14, 21, 19, 16, 2, 17, 20], can be viewed as extensions to the real-time setting of some process calculus or *process algebra*: an algebraic specification language for concurrent systems with a concomitant formal semantics. Prime examples of process algebras are CCS [12], CSP [9], ACP [3], and Acceptance Trees [8].

Most of the process-algebraic work on real-time has focused on the hard real-time case, although [1, 6, 7, 18] may be viewed as exceptions as they address both of the issues of time and probability. For example, [6, 7] present an *alternating model* of probabilistic computation—wherein a probabilistic choice is followed by a nondeterministic choice, and vice versa—augmented with the delay operator of ATP [16].

In [5], we presented a testing theory for probabilistic processes based on the classical testing theory for nondeterministic processes of De Nicola and Hennessy [15, 8]. Here we augment our probabilistic testing theory with a notion of discrete time to obtain a semantic framework for soft real-time systems. Specifically, we present a specification language PDP (for Probabilistic and Discrete-time Processes) that incorporates both probabilistic and timing aspects of process behavior. We also equip PDP with a formal operational semantics, given as a recursively defined probability distribution function over process terms and atomic actions. Within this setting, we define a natural notion of a process passing a test with a certain probability, where a test is a process with the added capability of reporting success. By encoding deadlines as tests,

*Research supported by NSF grants CCR-9120995 and CCR-9402807, ONR Young Investigator Award N00014-92-J-1582, NSF Young Investigator Award CCR-9257963, and AFOSR grant F49620-95-1-0508.

†Research supported in part by AFOSR Grant F49620-95-1-0508.

‡Research supported in part by AFOSR Grant F49620-95-1-0508.

§Research supported in part by NSF Grant CCR-9208585 and AFOSR Grants F49620-93-1-0250 and F49620-95-1-0508.

the probability of a process passing a test may now be interpreted as the probability of the process meeting a deadline, thereby capturing the essence of soft real-time. We illustrate this approach through a simple video frame transmission example.

The structure of the rest of the paper is as follows. Section 2 presents the syntax and semantics of PDP. Section 3 discusses our testing theory for PDP processes. Section 4 contains our video frame transmission example, and Section 5 our conclusions.

2 Syntax and Semantics of PDP

This section describes the syntax and semantics of PDP, our language for modeling soft (probabilistic) real-time systems. For simplicity we assume a discrete model of time, although this restriction can be relaxed without major difficulties.

Let Act be the set of atomic *actions* and let τ, the *internal* action, and χ, the *clock tick* action, be distinguished actions not in Act. By Act^+ we denote the set $\mathsf{Act} \cup \{\tau, \chi\}$, and let a, b, ... range over Act^+. Also, let Const be the set of *agent constants* with A, B, ... ranging over Const. Finally, let π range over the real interval $[0,1]$ and t range over the natural numbers. The following grammar defines the set Proc of PDP *process expressions*, with p, q, ... ranging over Proc.

$$\begin{array}{rcl} p & ::= & \mathsf{skip} \mid \mathsf{stop} \mid A \mid a \\ & \mid & p +_\pi q \mid p;q \mid p \,||_S^\pi\, p \mid p/S \mid P\ \Delta_t\ p \end{array}$$

In the above, π is constrained to satisfy $0 < \pi < 1$, while $S \subseteq \mathsf{Act}$. Additionally, we assume that for every process constant A there is a defining equation of the form $A \stackrel{\text{def}}{=} p$. Such equations provide a means for introducing recursion into the language.

Intuitively, the language constructs may be understood as follows. skip is the *terminated* process that halts immediately (although allows time to pass), while stop is the *deadlocked* process that can neither halt nor perform any actions. A represents a "call" of process A, while a engages in action a and terminates. $+_\pi$ is a choice construct such that $p +_\pi q$ behaves like p with probability π and q with probability $1-\pi$. $p;q$ first "executes" p to completion before invoking q, while $||_S^\pi$ denotes parallel composition: $p \,||_S^\pi\, q$ performs the actions of p and q in an interleaved manner, with synchronization required on χ and the actions in S. Interleaving commences with p with probability π and with q with probability $1-\pi$. p/S behaves like p but with the actions in S "hidden" (i.e. turned into the silent action τ.) Finally, Δ_t is a time-out operator; in $p\ \Delta_t\ q$, p is given t time units to terminate; if it fails to do so then q is invoked.

$$\begin{array}{rcl} & & \mathsf{skip}\surd \\ p\surd & \Rightarrow & p/S\surd, p\Delta_t q\surd \\ q\surd & \Rightarrow & p/0q\surd \\ p\surd, q\surd & \Rightarrow & p +_\pi q\surd, p;q\surd, p||_S^\pi q\surd \\ p\surd, C \stackrel{\text{def}}{=} p & \Rightarrow & C\surd \end{array}$$

Figure 1: The termination predicate.

The formal semantics of PDP captures the preceding intuitions using a *termination predicate* $\surd$, and a *transition probability function* $\mu : \mathsf{Proc} \times \mathsf{Act}^+ \times \mathsf{Proc} \to [0,1]$. Intuitively, $p\surd$ holds when p has successfully terminated, while $\mu(p,a,p')$ records the probability with which p may engage in action a and thereafter behave like p'. The formal definitions of $\surd$ and μ appear in Figures 1 and 2, respectively. In the latter, $\equiv$ is used to denote syntactic identity of process expressions.

The definition of $\mu(p||_S^\pi q, a, r)$ requires the use of a normalization factor $\nu(p,q,S,\pi)$ that is defined below. Here, and in what follows, we use $\mu(p,a)$ to stand for $\sum_{p' \in \mathsf{Proc}} \mu(p,a,p')$.

$$\nu(p,q,S,\pi) = \sum_{a \in S \cup \{\chi\}} \mu(p,a) \cdot \mu(q,a) + \sum_{a \notin S \cup \{\chi\}} (\pi \cdot \mu(p,a) + (1-\pi) \cdot \mu(q,a))$$

The behavior of $p \,||_S^\pi\, q$ can be viewed in terms of a coin-flipping protocol. p and q each possesses a coin that it flips to determine which transition it should execute, and the over-all system $p||_S^\pi q$ has an additional coin weighted in accordance with π. To determine a transition of $p \,||_S^\pi\, q$, the three coins are flipped simultaneously. If both p and q's coins indicate transitions labeled by the same action in $S \cup \{\chi\}$, then the system engages in this action. Otherwise, if the third coin indicates that p is allowed to proceed, and if p's chosen transition is labeled by an action not in $S \cup \{\chi\}$, then $p||_S^\pi q$ takes this transition, and symmetrically for q. Otherwise, the coin-flipping process is repeated. This repetition reflects are desire to model reactive systems, since if some interaction is possible then some interaction is guaranteed eventually to take place. It also explains the necessity for the normalization factor; it is used to condition the probabilities that synchronization or autonomous behavior occur by the probability

$$
\begin{array}{rcl}
\mu(a,b,r) & = & \begin{cases} 1 & \text{if } a=b \text{ and } r \equiv \mathsf{skip} \\ 0 & \text{otherwise} \end{cases} \\
\mu(\mathsf{skip},a,r) & = & \begin{cases} 1 & \text{if } a=\chi \text{ and } r \equiv \mathsf{skip} \\ 0 & \text{otherwise} \end{cases} \\
\mu(\mathsf{stop},a,r) & = & 0 \\
\mu(C,a,r) & = & \mu(p,a,r), \text{where } C \stackrel{\text{def}}{=} p \\
\mu(p +_\pi q,a,r) & = & \pi \cdot \mu(p,a,r) + (1-\pi)\cdot \mu(q,a,r) \\
\mu(p;q,a,r) & = & \begin{cases} \mu(p,a,p') & \text{if } r \equiv p';q \\ \mu(q,a,r) & \text{if } p\surd \\ 0 & \text{otherwise} \end{cases} \\
\mu(p||_S^\pi q,a,r) & = & \begin{cases} \frac{\mu(p,a,p')\cdot\mu(q,a,q')}{\nu(p,q,S,\pi)} & \text{if } r \equiv p'||_S^\pi q' \text{ and } a \in S \cup \{\chi\} \\ \frac{\pi\cdot\mu(p,a,p')+(1-\pi)\cdot\mu(q,a,q')}{\nu(p,q,S,\pi)} & \text{if } r \equiv p'||_S^\pi q' \text{ and } a \notin S \cup \{\chi\} \text{ and either} \\ & p \equiv p' \text{ or } q \equiv q' \\ 0 & \text{otherwise} \end{cases} \\
\mu(p/S,a,r) & = & \begin{cases} \mu(p,a,r) & \text{if } a \notin S \cup \{\tau\} \\ \sum_{a \in S\cup\{\tau\}} \mu(p,a,r) & \text{if } a = \tau \\ 0 & \text{otherwise} \end{cases} \\
\mu(p\Delta_t q,a,r) & = & \begin{cases} \mu(p,a,p') & \text{if } r \equiv p'\Delta_t q \text{ and } a \neq \chi \text{ and } t > 0 \\ \mu(p,\chi,p') & \text{if } r \equiv p'\Delta_{t-1} q \text{ and and } t > 0 \\ \mu(q,a,r) & \text{if } \neg(p\surd) \text{ and } t = 0 \\ 0 & \text{otherwise} \end{cases}
\end{array}
$$

Figure 2: The transition probability function μ.

that some interaction occurs.

It is also convenient to define the derived process *finished within*: $fw(p,i) \stackrel{\text{def}}{=} p\ \Delta_i\ \mathsf{stop}$. Intuitively, $fw(p,i)$ deadlocks if p fails to terminate within i time units.

It is straightforward to show that μ, as defined above, satisfies the following condition: for any $p \in \mathsf{Proc}$,

$$0 \leq \sum_{a \in \mathsf{Act}^+} \sum_{p' \in \mathsf{Proc}} \mu(p,a,p') \leq 1$$

When this sum is 1, we say that p is *stochastic*, and *substochastic* otherwise. For example, $a +_\pi b$ is stochastic while $a +_\pi \mathsf{stop}$ and stop are substochastic.

For technical reasons, we will restrict our attention to *divergence-free* PDP processes, i.e. those processes devoid of infinite τ-computations (see Section 3 for a formal definition of computation).

3 Testing PDP Processes

We define a testing framework for PDP processes, based on our probabilistic testing preorders of [5], that will allow us to model soft real-time systems. Let w be a distinguished *success* action not in Act^+. A *test* is simply a PDP process that can additionally execute w. For technical reasons (see Definition 3.1), we assume that tests are finite, i.e., definable without recursion. We use Test to denote the set of PDP tests, and t to range over Test.

We next show how to associate a "success probability" with a test t. Intuitively, this will be the sum of the probabilities of the computation paths of t that lead t to a state where w can be performed. For example, it is easy to see that the success probability of $a +_\pi b; w$ is $1-\pi$. We will later show how to apply a test t to a process p using PDP's parallel composition operator. The result of this application will itself be a test t' and the success probability of t' can be viewed as the probability by which p passes t.

Definition 3.1 *In the following, let* $t, t_0, t_1, \ldots, t_n \in \mathsf{Test}$ *be PDP tests.*

- *t is said to be* successful *if $\mu(t,w) > 0$, and is* terminal *if $\sum_{a \in \mathsf{Act}^+ \cup \{\tau\}} \mu(t,a) = 0$.*

- *A* computation *of t is a (maximal) sequence of the form*

$$t = t_0 \xrightarrow{a_0,\pi_0} t_1 \xrightarrow{a_1,\pi_1} \ldots \xrightarrow{a_{n-1},\pi_{n-1}} t_n$$

  where

  - *for all $0 \leq i \leq n-1$, $\mu(t_i, a_i, t_{i+1}) = \pi_i$;*
  - *for all $0 \leq i \leq n-1$, t_i is not successful; and*
  - *either t_n is successful or terminal.*

  Such a computation is successful *if t_n is successful.*

- *The following notations are used for sets of computations:*

  - *$\mathcal{C}_t$ is the set of all computations of t.*
  - *$\mathcal{S}_t$ is the set of all successful computations of t.*

  $\mathcal{C}_t$ and $\mathcal{S}_t$ are ranged over by C.

- *The* computation probability distribution function $\mathrm{Pr} : \mathcal{C}_t \rightarrow [0,1]$ *is defined inductively as follows:*

$$\mathrm{Pr}(t) = 1$$
$$\mathrm{Pr}(t \xrightarrow{a,\pi} C') = \pi \cdot \mathrm{Pr}(C')$$

  and is lifted to sets $\mathcal{C} \subseteq \mathcal{C}_t$ of computations as follows:

$$Pr(\mathcal{C}) = \sum_{C \in \mathcal{C}} Pr(C)$$

- *$Pr_{Succ}(t)$, the* success probability *of t, is given by $Pr_{Succ}(t) = Pr(\mathcal{S}_t)$.*

Intuitively, $\mathrm{Pr}(C)$ is the probability that test t engages in computation C. The success probability of t is then the cumulative probability of its successful computations. Note that our assumption that tests are finite ensures that all maximal sequences satisfying the conditions laid out for computations are finite.

The following proposition establishes that Pr is indeed a probability distribution on computations.

Proposition 1 *Let t be a non-terminal PDP test. Then $\sum_{C \in \mathcal{C}_t} \mathrm{Pr}(C) = 1$.*

Proof: By a straightforward induction on the length of the longest computation in $\mathcal{C}_t$. ■

As alluded to above, our real interest is in defining the probability that a PDP process p passes a PDP test t. This is accomplished as follows.

Definition 3.2 *Let p be a PDP process, t a PDP test, and π a probability. Then the* application *of t to p is the parallel composition expression*

$$p||^{\pi}_{\mathsf{Act}}t$$

The probability that p passes t *is*

$$Pr_{Succ}(p||^{\pi}_{\mathsf{Act}}t)$$

Thus, in our setup, the application of a test t to a process p is nothing more than the parallel composition of p and t in which synchronization is required on all actions in $\mathsf{Act} \cup \{\chi\}$. The probability π specifies the relative frequency with which internal execution steps in the process take precedence over those in the test.

A PDP process p is said to be *τ-free* if $\mu(p,\tau) = 0$ and $\forall a \in \mathsf{Act}^+$, $\forall p' \in \mathsf{Proc}$, $\mu(p,a,p') > 0$ implies p' is τ-free. A τ-free test is defined similarly. The following proposition states that the choice of π in the application of t to p is irrelevant if either p or t is τ-free.

Proposition 2 *Let p be a PDP process and t a PDP test. If p or t is τ-free then for arbitrary $\pi, \pi' \in [0,1]$, $Pr_{Succ}(p||^{\pi}_{\mathsf{Act}}t) = Pr_{Succ}(p||^{\pi'}_{\mathsf{Act}}t)$.*

Defining testing in this manner yields a framework that is exactly analogous to the classical testing theory (for nondeterministic processes) of Hennessy and De Nicola [8, 15]: synchronization is required on all visible actions, while internal actions occur autonomously (i.e. in an interleaved fashion). In [5], the definition of probabilistic testing is slightly different in that synchronization of internal actions is permitted.

In [5] we go on to define a testing preorder $\sqsubseteq_{prob}$ on probabilistic processes wherein $p \sqsubseteq_{prob} q$ if the probability by which p passes an arbitrary test t is no more than the probability by which q passes t. An interesting property of $\sqsubseteq_{prob}$ is that the natural testing preorder induced on nondeterministic processes by the definition of testing in [5] is identical to the testing preorder of [8, 15]. In this sense, the internal-move synchronization permitted in [5] is insignificant.

It is worth noting that in contrast to [10] and [4], tests in our setup are probabilistic. This is again consistent with the approach forged in [8, 15] where processes and tests are "compatible", i.e. structurally identical. We may thus conclude that the theory of testing proposed here for PDP processes bears close connections with the classical testing theory of Hennessy and De Nicola.

4 Example: Video Frame Transmission

To illustrate our approach to analyzing soft real-time systems, we present a simple video frame transmission example. The system we model consists of two component processes, *Sender* and *Receiver*. Every two time units, *Sender* generates and transmits a new video frame to *Receiver*. To model the possibility of lost frames due e.g. to a lossy communication medium, *Sender* may, with a small probability, choose to transmit nothing at all at the two-unit mark.

Receiver executes an infinite loop in which it receives a frame from *Sender*, processes it, and displays the processed frame. Processing takes one time unit. To prevent *Receiver* from entering a "time deadlock" with *Sender*, *Receiver* may, with a small probability, simply idle for one time unit during a loop iteration.

The deadline we wish the system to meet is that it displays two consecutive frames, each within six time units. This deadline is modeled as test T. Since, under normal circumstances, a video frame transmission system can tolerate a certain level of lost frames, it eminently qualifies as a soft real-time system.

The formal specification of the system is given by the PDP code of Figure 3.

By computing the probability that Sys passes T—i.e. the measure of the set of successful computations of $Sys||^{\pi}_{\mathsf{Act}}T$—we can determine the probability with which Sys meets deadline T. Since Sys (and, for that matter, T) are τ-free, the choice of π is irrelevant (see Proposition 2). A fairly simple calculation shows this success probability to be 0.994875.

We can treat the example more symbolically by letting p, q, r be the probabilities that *Sender* drops a frame, *Receiver* idles, and *Display* idles, respectively. Then the probability with which Sys passes T is $1 - 2p^2 - p^3$.

5 Conclusions

To permit the modeling and analysis of soft real-time systems, we have put forth the language PDP. Probability is captured in PDP through the $+_{\pi}$ and $||^{\pi}_{S}$ operators, and time by the Δ_t operator. The video frame transmission example has illustrated that this combination of operators, along with the "deadlines as tests" perspective, is potentially quite useful.

Regarding directions for future work, an obvious choice is to devise an algorithm for automatically computing the success probability of a PDP expression of the form $p||^{\pi}_{\mathsf{Act}}t$, in the case where this system is finite-state. Moreover, when p is itself a composition of processes, it would be highly advantageous to compute the success probability *compositionally*; i.e., without first computing p.

Code generation is another area worth investigating, that is, the generation of executable code from PDP expressions. Besides being able to generate a working implementation of a soft real-time system, one could generate a test suite (from the PDP tests used in the analysis of the system) for the implementation.

References

[1] R. Alur, C. Courcoubetis, and D. Dill. Model checking for probabilistic real-time systems. In *Proceedings of ICALP '91*, Barcelona, Spain, July 1991.

[2] Jos C. M. Baeten and Jan A. Bergstra. Real time process algebra. *Formal Aspects of Computing*, 3(2), 1991.

[3] J. A. Bergstra and J. W. Klop. Process algebra for synchronous communication. *Information and Computation*, 60:109–137, 1984.

[4] I. Christoff. Testing equivalences for probabilistic processes. Technical Report DoCS 90/22, Ph.D. Thesis, Department of Computer Science, Uppsala University, Uppsala, Sweden, 1990.

[5] R. Cleaveland, S. A. Smolka, and A. E. Zwarico. Testing preorders for probabilistic processes. In *Proceedings of the 19th ICALP*, July 1992.

[6] H. A. Hansson and B. Jonsson. A calculus for communicating systems with time and probabilities. In *Proceedings of the 11th IEEE Symposium on Real-Time Systems*, 1990.

[7] Hans A. Hansson. *Time and Probability in Formal Design of Distributed Systems*, volume 1 of *Real –Time Safety Critical Systems*. Elsevier, Uppsala University, Department of Computer Systems, Uppsala, Sweden, 1994.

[8] M. C. B. Hennessy. *Algebraic Theory of Processes*. MIT Press, Boston, Mass., 1988.

[9] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, London, 1985.

[10] K. G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, September 1992.

$$
\begin{array}{lcl}
Sender & = & \chi^2\,;\, frame\,;\, Sender +_{.95} \chi^2\,;\, Sender \\
Receiver & = & frame\,;\, \chi\,;\, display\,;\, Receiver +_{.9} \chi\,;\, Receiver \\
Sys & = & (Sender ||^{.5}_{\{frame\}} Receiver)/\{frame\} \\
 & & \\
T & = & fw(Display, 6)\,;\, fw(Display, 6)\,;\, w \\
Display & = & display\,;\, skip +_{.9} \chi\,;\, Display
\end{array}
$$

Figure 3: PDP code for the video frame transmission system.

[11] J. Lee and S. H. Son. Performance of concurrency control algorithms for real-time database systems. In *Performance of Concurrency Control Mechanisms in Centralized Database Systems*, pages 429–460. Prentice-Hall, 1996.

[12] R. Milner. *Communication and Concurrency*. International Series in Computer Science. Prentice Hall, 1989.

[13] Faron Moller and Chris Tofts. A temporal calculus of communicating systems. In *Proceedings of CONCUR'90*. LNCS 458, 1990.

[14] Faron Moller and Chris Tofts. Behavioural abstraction in tccs. In *Proceedings of 19th ICALP*. LNCS 623, 1992.

[15] R. De Nicola and M. C. B. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1983.

[16] Xavier Nicollin, J.-L. Richier, Joseph Sifakis, and J. Voiron. ATP: an algebra for timed processes.

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

[18] R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, MIT, Cambridge, Massachusetts, 1996.

[19] Kārlis Čerāns. Ctr: A calculus of timed refinement. pages 516–530.

[20] Kārlis Čerāns, Jens Christopher Godskesen, and Kim Guildstrand Larsen. Timed modal specification — theory and tools. In *Proceedings of CAV'93*. LNCS 697, 1993.

[21] Wang Yi. *A Calculus of Real Time Systems*. PhD thesis, Chalmers University of Technology, 1991.