A Theory of Testing for Soft Real-Time Processes

Rance Cleaveland¹
Dept. Computer Science
N.C. State University
Raleigh, NC 27695
rance@csc.ncsu.edu

Insup Lee†
Dept. Computer & Info. Science
U. Pennsylvania
Philadelphia, PA 19104
lee@cis.upenn.edu

Phil Lewis‡& Scott A. Smolka§
Dept. Computer Science
SUNY at Stony Brook
Stony Brook, NY 11794
{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,
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 \( \tau \), the internal action, and \( \chi \), the clock tick action, be distinguished actions not in Act. By \( \text{Act}^+ \) we denote the set \( \text{Act} \cup \{ \tau, \chi \} \), and let \( a, b, \ldots \) range over \( \text{Act}^+ \). Also, let \( \text{Const} \) be the set of agent constants with \( A, B, \ldots \) ranging over \( \text{Const} \). Finally, let \( \pi \) range over the real interval \([0, 1]\) and \( t \) range over the natural numbers. The following grammar defines the set \( \text{Proc} \) of PDP process expressions, with \( p, q, \ldots \) ranging over \( \text{Proc} \).

\[
p ::= \text{skip} \mid \text{stop} \mid A \mid a \mid p+q \mid p+q \mid p\parallel q \mid p/S \mid p\Delta t p
\]

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

Intuitively, the language constructs may be understood as follows. \( \text{skip} \) is the terminated process that halts immediately (although allows time to pass), while \( \text{stop} \) is the deadlocked process that can never halt nor perform any actions. \( A \) represents a “call” of process \( A \), while \( a \) engages in action \( a \) and terminates. \( +q \) is a choice construct such that \( p + q \) behaves like \( p \) with probability \( \pi \) and \( q \) with probability \( 1 - \pi \). \( p\parallel q \) first “executes” \( p \) to completion before invoking \( q \), while \( p/S \) denotes parallel composition: \( p\parallel q \) performs the actions of \( p \) and \( q \) in an interleaved manner, with synchronization required on \( \chi \) and the actions in \( S \). Interleaving commences with \( p \) with probability \( \pi \) 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 \( \tau \)). Finally, \( \Delta t \) 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.

The formal semantics of PDP captures the preceding intuitions using a termination predicate \( \text{Proc} \), and a transition probability function \( \mu : \text{Proc} \times \text{Act}^+ \times \text{Proc} \rightarrow [0, 1] \). Intuitively, \( p/S \) 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 \( \text{Proc} \) and \( \mu \) appear in Figures 1 and 2, respectively. In the latter, \( \equiv \) is used to denote syntactic identity of process expressions.

The definition of \( \mu(p\parallel q, a, r) \) 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 \( \sum_{p' \in \text{Proc}} \mu(p, a, p') \).

\[
\nu(p, q, S, \pi) = \sum_{a \in \text{Act} \cup \{ \chi \}} \mu(p, a) \cdot \mu(q, a)
\]

\[
+ \sum_{a \in \text{Act} \cup \{ \chi \}} (\pi \cdot \mu(p, a) + (1 - \pi) \cdot \mu(q, a))
\]

Figure 1: The termination predicate.
that some interaction occurs.

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

It is straightforward to show that \( \mu \), as defined above, satisfies the following condition: for any \( p \in \text{Proc} \),

\[
0 \leq \sum_{a \in \text{Act}^+} \sum_{p' \in \text{Proc}} \mu(p, a, p') \leq 1
\]

When this sum is 1, we say that \( p \) is stochastic, and substochastic otherwise. For example, \( a +_\tau b \) is stochastic while \( a +_\tau \text{ stop} \) and \( \text{ stop} \) are substochastic.

For technical reasons, we will restrict our attention to divergence-free PDP processes, i.e. those processes devoid of infinite \( \tau \)-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 \( \text{Act}^+ \). 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 \( \text{Test} \) to denote the set of PDP tests, and \( t \) to range over \( \text{Test} \).

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 \( a +_\tau b ; w \) 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 \( t, t_0, t_1, \ldots, t_n \in \text{Test} \) be PDP tests.
• **t** is said to be successful if \( \mu(t, w) > 0 \), and is terminal if \( \sum_{a \in \text{Act}} \mu(t, a) = 0 \).

• A computation of \( t \) is a (maximal) sequence of the form

\[
t = t_0 a_1 \pi_1 a_2 \pi_2 \ldots a_{n-1} \pi_{n-1} t_n
\]

where
- for all \( 0 \leq i \leq n-1 \), \( \mu(t, a_i, t_{i+1}) = \pi_i \);
- 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:
  - \( C_t \) is the set of all computations of \( t \).
  - \( S_t \) is the set of all successful computations of \( t \).

\( C_t \) and \( S_t \) are ranged over by \( C \).

• The computation probability distribution function \( \Pr : C_t \rightarrow [0, 1] \) is defined inductively as follows:

\[
\Pr(t) = 1 \\
\Pr(t \oplus_C C') = \pi \cdot \Pr(C')
\]

and is lifted to sets \( C \subseteq C_t \) of computations as follows:

\[
\Pr(C) = \sum_{C' \in C} \Pr(C')
\]

• \( \Pr_{\text{Succ}}(t) \), the success probability of \( t \), is given by

\[
\Pr_{\text{Succ}}(t) = \Pr(S_t).
\]

Intuitively, \( \Pr(C) \) is the probability that \( t \) 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 \( \sum_{C \in C_t} \Pr(C) = 1 \).

**Proof:** By a straightforward induction on the length of the longest computation in \( C_t \).

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 \( \pi \) a probability. Then the application of \( t \) to \( p \) is the parallel composition expression

\[
p \parallel t_{\text{Act}}
\]

The probability that \( p \) passes \( t \) is

\[
\Pr_{\text{Succ}}(p || t_{\text{Act}})
\]

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 \( \text{Act} \cup \{ \chi \} \). The probability \( \pi \) 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 \( \tau \)-free if \( \mu(p, \tau) = 0 \) and \( \forall a \in \text{Act} \), \( \forall p' \in \text{Proc}, \mu(p, a, p') > 0 \) implies \( p' \) is \( \tau \)-free. A \( \tau \)-free test is defined similarly. The following proposition states that the choice of \( \pi \) in the application of \( t \) to \( p \) is irrelevant if either \( p \) or \( t \) is \( \tau \)-free.

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

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 \( \sqsubseteq_{\text{prob}} \) on probabilistic processes wherein \( p \sqsubseteq_{\text{prob}} q \) 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 \( \sqsubseteq_{\text{prob}} \) 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 $\text{Sys}$ passes $T$—i.e. the measure of the set of successful computations of $\text{Sys} \parallel \text{Act} T$—we can determine the probability with which $\text{Sys}$ meets deadline $T$. Since $\text{Sys}$ (and, for that matter, $T$) are $\tau$-free, the choice of $\tau$ 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 $\text{Sys}$ 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 $+\tau$ and $\parallel \tau$ operators, and time by the $\Delta t$ 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 $p \parallel \text{Act} \tau t$, 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


\[\begin{align*}
\text{Sender} &= x^2; \text{frame}; \text{Sender} +.55 \ x^3; \text{Sender} \\
\text{Receiver} &= \text{frame}; \chi; \text{display}; \text{Receiver} +.3 \ \chi; \text{Receiver} \\
\text{Sys} &= (\text{Sender} \downarrow_{\text{frame}} \text{Receiver})/\{\text{frame}\} \\
T &= \text{fu}(\text{Display}, 6); \text{fu}(\text{Display}, 6); w \\
\text{Display} &= \text{display}; \text{skip} +.3 \ \chi; \text{Display}
\end{align*}\]

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


