Predictability of Real-Time Systems: A Process-Algebraic Approach*

V. Natarajan † Rance Cleaveland
Department of Computer Science
North Carolina State University
Raleigh, NC 27695-8206, USA
{nvaidhy@eos, rance@csc}.ncsu.edu

Abstract

This paper presents a testing-based semantic preorder that relates real-time systems given in the process description language TPL on the basis of the predictability of their timing behavior. This predictability is measured in terms of the amount of variability present in processes' "activity-completion times." The semantic preorder is shown to coincide with an already existing, well-investigated implementation relation for TPL—-the $must$-preorder. The optimality of our relation is also established by means of a full abstraction result. An example is provided to illustrate the utility of this work.

1 Introduction

In the past decade and half, *process algebras* have emerged as a successful framework for system verification. In this framework, the correctness of an implementation is established by formulating its specification as a "high-level" system and then showing that the specification and the implementation are related by some *behavioral* (*implementation*) relation. If the implementation relation is a preorder, then the intuition is that the implementation is deemed "better" than the specification with respect to some criterion.

In general, behavioral preorders order processes on the basis of their external behavior. For untimed-systems, several such relations have been proposed in the literature, each of them sensitive to different aspects of system behavior, such as potential nondeterminism and divergence [4, 5, 17, 19, 20, 28], efficiency [1, 2, 18], spatial distribution [3, 6], etc. In the case of real-time systems, the existing behavioral preorders in the literature either try to capture the notion of "faster-than" [7, 16, 26, 27] or relate processes in terms of the nondeterminism they might exhibit during their execution [9, 15, 22, 30]. However, for timed systems, *predictability* is a more appropriate criterion for judging the merits of a system than speed is [24, 25]; more predictable components provide designers with better control in manipulating the timing characteristics of a system constructed using these components.

The aim of this paper is to develop a behavioral preorder that is sensitive to the relative predictability of real-time systems. We take as our starting point the *theory of testing* as devised by De Nicola and Hennessy [20] for untimed systems, which we augment with a notion of the amount of time a process may expend in passing a test. The rest of the paper is organized as follows. In the next section we give the syntax and semantics of the system description language TPL [12] and an overview of the theory of testing in the context of TPL [12]. In the following section, we present the definition of the predictability-sensitive semantic preorder and establish its congruence properties. This section also contains an alternative characterization to facilitate reasoning about systems using this preorder. In Section 4, the "optimality" of this preorder is demonstrated by means of a full-abstractness result. The penultimate section is devoted to an illustrative example, and the last section contains our conclusions and remarks on related research.

2 TPL and Testing Preorders

This section presents an exposition of the syntax and operational semantics of the language TPL [12] and an overview of the theory of testing [20].

2.1 Syntax and Semantics of TPL

The language we consider in this paper, TPL (Temporal Process Language) [12], is an extension of CCS [14]

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

† Current address: Networking Hardware Division, IBM Corporation, Research Triangle Park, NC 27709, USA.

for describing timed systems. In TPL, time is *discrete* and *deterministic*; time progresses in discrete steps, and systems cannot evolve to more than one state as a result of the passage of time. TPL systems also satisfy the *maximal progress* property (a process cannot idle if it can engage in an internal computation) and the *patience* property (a process can idle for arbitrary amount of time while awaiting an opportunity for computation).

Formally, let Λ denote a countably infinite set of labels; intuitively, Λ contains the "ports" over which processes may synchronize with their environment. Then the set Act_τ of *actions* may be defined by $Act_\tau = \Lambda \cup \overline{\Lambda} \cup \{\tau\}$, where $\overline{\Lambda} = \{ \ \overline{\lambda} \mid \lambda \in \Lambda\}$. Action $\lambda \in \Lambda$ may be thought of as representing the receipt of an input on port λ, while $\overline{\lambda} \in \overline{\Lambda}$ constitutes the depositing of an output on λ; $\tau \notin \Lambda$ represents an internal step. We use $Act = Act_\tau \setminus \{\tau\}$ to denote the set of all *external* or *visible* actions. The distinguished action σ, not in Act_τ, represents the passage of one unit of time; we define $Act_{\tau\sigma} = Act_\tau \cup \{\sigma\}$. In what follows we let $\alpha, \beta, \ldots$ range over Act_τ, $a, b, c, \ldots$ over Act and μ range over $Act_{\tau\sigma}$. We also extend $\bar{}$ to all non–τ, non–σ actions by taking $\overline{\overline{\lambda}} = \lambda$.

The terms of TPL may now be defined as follows.

$$t \ ::= \ nil \mid \Omega \mid \mu.t \mid t+t \mid t|t \mid t\backslash\lambda \mid t[R] \mid x \mid fix(x:t) \mid \lfloor t \rfloor(t)$$

Here R is a *relabeling*, a mapping from Act_τ to Act_τ preserving τ and $\bar{}$. We also assume that relabelings satisfy the following property: if R is a relabeling then the set $\{ \ a \in Act \mid a \neq R(a) \ \}$ is finite. We use Exp to stand for the set of all terms of the language, and we adopt the usual definitions for *free* and *bound* variables, *open* and *closed* terms, and *guarded recursion* [14]. We call the closed guarded terms *processes*; $\mathcal{P}$ represents the set of all processes and will be ranged over by $p, q, \ldots$.

The operational semantics of processes is given as a relation $\rightarrow \subseteq \mathcal{P} \times Act \times \mathcal{P}$; we write $p \xrightarrow{\mu} q$ in lieu of $\langle p, \mu, q\rangle \in \rightarrow$ and $p \xrightarrow{\mu}$ if $p \xrightarrow{\mu} q$ holds for some q. The formal definition is given inductively in Figure 1 and is taken from [12]. Intuitively, $p \xrightarrow{\alpha} q$ holds if p may engage in action α and thereafter behave like q. Similarly $p \xrightarrow{\sigma} q$ is true if p may "idle" for one unit of time and thereafter behave like q. Accordingly, $\mu.p$ may engage in μ and then behave like p. Because TPL is intended to have the patience property, *nil* and $a.p$ can idle indefinitely by performing the action σ. The operator $+$ constitutes a choice construct: $p + q$ may perform the non-delay actions of p or q and then behave like the process from which the action was chosen. However, because of time-determinism $p + q$ can idle only if both p and q are capable of idling. For example, if the process $a.nil + \sigma.b.nil$ idles for one unit of time then it would evolve to $a.nil + b.nil$. $\lfloor \ \rfloor(\ )$ is the timeout constructor: $\lfloor p \rfloor(q)$ behaves like p if it is possible for p to make progress within one unit of time, either by executing an internal step or by participating in a synchronization with another process. Otherwise $\lfloor p \rfloor(q)$ will idle for one unit of time by executing the σ action and thereafter behave like q. For example, the process $\lfloor \sigma.p \rfloor(q)$ will execute action σ and become the process q. Process $p \mid q$ represents the parallel composition of p and q, with concurrent execution modeled via interleaving and synchronization between p and q allowed if they are capable of complementary actions; the result of such a synchronization is a τ. Because of maximal progress, $p|q$ can idle only if no synchronization is possible between p and q and both p and q are capable of executing a delay action. The operator $\backslash\lambda$ defines a scope for actions using the port λ, while $p[R]$ relabels the actions of p in the appropriate manner. $fix(x:t)$ represents a recursively defined process that is a distinguished "solution" to the equation $x = t$.

Using the definition of $\rightarrow$, one may prove that TPL processes are time-deterministic (if $p \xrightarrow{\sigma} q_1$ and $p \xrightarrow{\sigma} q_2$ then q_1 and q_2 are syntactically identical) and satisfy the maximal progress assumption (if $p \xrightarrow{\tau}$ then $p \not\xrightarrow{\sigma}$).

The term Ω represents the "undefined" system and has no transitions. In order to capture the notion of undefinedness formally, we introduce the following.

Definition 2.1 *$\downarrow$ is the smallest predicate on closed terms satisfying the following.*

- $nil\downarrow, \alpha.p\downarrow, \sigma.p\downarrow$.
- $p\downarrow, q\downarrow \ \Rightarrow (p+q)\downarrow, (p|q)\downarrow, p[R]\downarrow, p\backslash\lambda\downarrow, \lfloor p \rfloor(q)\downarrow$.
- $t[fix(x:t)/x]\downarrow \ \Rightarrow fix(x:t)\downarrow$.

Let $p\uparrow$ if it is not the case that $p\downarrow$. Intuitively $p\uparrow$ means that p is an incomplete or partially defined system description. For example, the terms Ω, $a.nil + \Omega$ and $fix(x:x)$ are incomplete system descriptions.

We end this subsection by explaining some terminology and notation that will be used in the rest of this paper. If $s \in Act^*_{\tau\sigma}$ then $\hat{s}$ is the sequence obtained by deleting all the occurrences of τ from s, and the relation $\xrightarrow{s}$ is the obvious extension of $\xrightarrow{\mu}$. If $\beta \in Act \cup \{\sigma\}$ then $\stackrel{\beta}{\Rightarrow}$ is the relation $(\xrightarrow{\tau})^* \xrightarrow{\beta} (\xrightarrow{\tau})^*$. Intuitively, $p \stackrel{\beta}{\Rightarrow} q$ if p may engage in some internal computation, and then β, and then more internal computation and evolve to q. We may extend this definition to $\stackrel{s}{\Rightarrow}$ where $s \in (Act \cup \{\sigma\})^*$ in the natural fashion. If p is some process then $S(p) = \{ \ a \in Act \mid p \xrightarrow{a} \}$. $p \Uparrow$ iff $\exists q : \ p \stackrel{\epsilon}{\Rightarrow} q \wedge q\uparrow$ or $p(\xrightarrow{\tau})^\omega$. Intuitively $p\Uparrow$ if p can either autonomously evolve to a partially defined state or perform an infinite sequence of internal computation without ever interacting with its environment. $p \Downarrow$ if $p\Uparrow$ is not true. The symbol 1 denotes either σ or τ. A

$$
\begin{array}{lcl}
\alpha.t \xrightarrow{\alpha} t & & \\
s \xrightarrow{\alpha} s' & \Rightarrow & s+t \xrightarrow{\alpha} s', \quad t+s \xrightarrow{\alpha} s', \quad s|t \xrightarrow{\alpha} s'|t, \quad t|s \xrightarrow{\alpha} t|s', \quad s[S] \xrightarrow{S(\alpha)} s'[S] \\
s \xrightarrow{a} s', t \xrightarrow{\overline{a}} t' & \Rightarrow & s|t \xrightarrow{\tau} s'|t' \\
t \xrightarrow{\alpha} t', \alpha \neq \lambda, \overline{\lambda} & \Rightarrow & t\backslash\lambda \xrightarrow{\alpha} t'\backslash\lambda \\
t \xrightarrow{\alpha} t' & \Rightarrow & \mathit{fix}(x:t) \xrightarrow{\alpha} t'[\mathit{fix}(x:t)/x] \\
s \xrightarrow{\alpha} s' & \Rightarrow & \lfloor s \rfloor(t) \xrightarrow{\alpha} s' \\
a.t \xrightarrow{\sigma} a.t & & \\
\mathit{nil} \xrightarrow{\sigma} \mathit{nil} & & \\
\sigma.t \xrightarrow{\sigma} t & & \\
s \xrightarrow{\sigma} s', t \xrightarrow{\sigma} t' & \Rightarrow & s+t \xrightarrow{\sigma} s'+t' \\
s \not\xrightarrow{\tau} & \Rightarrow & \lfloor s \rfloor(t) \xrightarrow{\sigma} t \\
s \xrightarrow{\sigma} s', t \xrightarrow{\sigma} t', s|t \not\xrightarrow{\tau} & \Rightarrow & s|t \xrightarrow{\sigma} s'|t' \\
s \xrightarrow{\sigma} s' & \Rightarrow & s[R] \xrightarrow{\sigma} s'[R] \\
s \xrightarrow{\sigma} s' & \Rightarrow & s\backslash a \xrightarrow{\sigma} s'\backslash a \\
t \xrightarrow{\sigma} t' & \Rightarrow & \mathit{fix}(x:t) \xrightarrow{\sigma} t'[\mathit{fix}(x:t)/x]
\end{array}
$$

Figure 1. Operational semantics.

process p is *stable* if $p \not\xrightarrow{\tau}$ and $p \downarrow$. Note that the definition of stability slightly differs from the one usually found in the literature in that $p \downarrow$ is required. The set of *extended* integers is the set of integers augmented with two distinguished symbols ∞ and $-\infty$. The usual ordering $\leq$ on integers is also extended to include $x \leq \infty$ and $-\infty \leq x$ where x is any extended integer. Similarly the usual subtraction operator $-$ is also extended to include $\infty - x = \infty$ where x is any extended integer. Observe that we are taking $\infty - \infty$ as ∞. A *context* is simply a process with a 'hole', [ ], in it. These contexts will be written as $C[\,]$. By filling the hole in the context $C[\,]$ by a process p we get the well-defined process $C[p]$. Intuitively contexts represent systems with an empty "slot" in which components can be plugged in. If $\preceq$ is a preorder over terms then $\preceq^C$ will stand for the largest precongruence (with respect to TPL) contained in $\preceq$. Formally, $\preceq^C = \{ \langle p, q \rangle \mid \forall C[\,] : C[p] \preceq C[q] \}$.

2.2 A Theory of Testing

The traditional testing framework of [8, 20, 10] defines preorders that relate processes on the basis of their responses to tests. In this setting a test is any process whose sort contains a distinguished *success* action w. To determine a process' response(s) to a test one must examine the computations that result when the test is "applied" to the process. These intuitions may be formalized in the TPL framework as follows [11, 12].

Definition 2.2

1. *A* configuration *has form $p|t$, where p is a process and t is a test. $p|t$ is* successful *if $t \xrightarrow{w}$.*
2. *A* computation *of $p|t$ is a maximal (potentially infinite) sequence $p|t = p_0|t_0 \xrightarrow{1} p_1|t_1 \xrightarrow{1} \cdots$; it is* successful *if some $p_i|t_i$ is successful and $p_j|t_j \downarrow$ for all $j < i$. The set of all computations of $p|t$ is denoted by $\mathcal{C}(p,t)$.*
3. *p* guarantees *t if every computation of $p|t$ is successful.*

In terms of the ability of a process to guarantee a test, the following preorder may now be defined.

Definition 2.3 *$p \sqsubseteq_{must} q$ if for every test t such that p guarantees t, q guarantees t as well.*

The $must$-preorder is proposed as an implementation relation and extensively investigated in [12]: full operational, denotational and axiomatic characterizations are provided.

3 Testing for Predictability

In this section we develop a semantic preorder based on the theory of testing that will help us to reason about the relative predictability of real-time systems. Further we show a surprising result: this relation actually coincides with the $must$-preorder given above.

Real-time systems are often non-deterministic; the time they take to accomplish a certain "activity" cannot necessarily be predicted accurately in advance. Such behavior is undesirable, as it complicates system designers' efforts to ensure that their systems exhibit the desired timing behavior. From this viewpoint a system is better than another if it has less variability in its activity-completion times (i.e. is more *predictable*) than the latter. For example, the system $p = a.\sigma.\sigma.b$ should be more predictable than $q = a.\sigma.b + a.\sigma.\sigma.b$ as there is less certainty in q of when the action b is enabled after the execution of the action a. In order to formalize these intuitions and define a behavioral preorder that is sensitive to the predictability of the systems, we first to quantify the variability in the times a process needs to pass a test. The following definition proposes one such measure.

Definition 3.1 *Let p be a process and t be a test, and let $c \equiv p|t = p_0|t_0 \xrightarrow{1} p_1|t_1 \xrightarrow{1} p_2|t_2 \xrightarrow{1} \cdots$ be a computation of $p|t$. Then the nonnegative-integer-valued function dur and the extended-integer-valued functions $sl, \mathcal{D}_w, \mathcal{D}_b$, and $\mathcal{D}$ are defined as follows.*

1. $dur(c, i) =$
$$\begin{cases} 0 & \text{if } i = 0 \\ dur(c, i-1) + 1 & \text{if } i > 0, p_{i-1}|t_{i-1} \xrightarrow{\sigma} p_i|t_i \\ dur(c, i-1) & \text{otherwise} \end{cases}$$

2. $sl(c) = \inf\{\, dur(c, i) \mid t_i \xrightarrow{w} \wedge (j < i \Rightarrow p_j|t_j{\downarrow}) \,\}$
3. $\mathcal{D}_w(p, t) = \sup\{\, sl(c) \mid c \in \mathcal{C}(p, t) \,\}$.
4. $\mathcal{D}_b(p, t) = \inf\{\, sl(c) \mid c \in \mathcal{C}(p, t) \,\}$.
5. $\mathcal{D}(p, t) = \mathcal{D}_w(p, t) - \mathcal{D}_b(p, t)$.

$dur(c, i)$ counts the number of σ-transitions present in the prefix of the computation c up to the configuration $p_i|t_i$. The *success length* of the computation, $sl(c)$, is the number of σ-transitions $p|t$ performs in reaching the first successful configuration while not passing through a partially defined configuration. If this computation is unsuccessful then the success length is ∞, as $\inf \emptyset = \infty$. $\mathcal{D}_w(p, t)$ is the supremum of the success lengths of the computations of $p|t$, while $\mathcal{D}_b(p, t)$ is the infimum. $\mathcal{D}(p, t)$ is the *range* of the success lengths of the computations of $p|t$.

The intuition underlying these definitions is the following. We wish to think of tests as activities that processes should complete by interacting with them. A computation c of $p|t$ thus represents an attempt by the process p to complete the activity t. Since σ stands for the passage of one unit of time, $dur(c, i)$ represents the time required for the computation c to proceed up to the configuration $p_i|t_i$. As the completion of the activity is signaled by the enabling of the action w, $sl(c)$ represents one possible execution time for the process p to complete t. Then $\mathcal{D}_w(p, t)$ is a measure of the worst-case performance of p on t, while $\mathcal{D}_b(p, t)$ represents the best-case performance of p on t. Finally $\mathcal{D}(p, t)$ measures the "dispersion" present in the execution times for the process p to complete the activity t.

We have chosen the number $\mathcal{D}(p, t)$ as our measure of "predictability" for the following reason. Given a sample (set of values), several measures of its dispersion have been proposed in statistics, and two of them are widely used: *range* and *variance*. Of these two measures, range is more sensitive to the extreme values present in the sample than variance is. The systems considered in this paper are *hard* real-time systems; the inability to meet deadlines for such systems generally give raise to catastrophic consequences. In general, the ability of a system to meet or miss its deadlines depends upon its latest activity-completion times. Also, finishing a certain activity too soon could be as disastrous as missing a deadline [23]. This is the reason for adopting the *range* of the activity-completion times as the measure of predictability with which a process completes an activity.

The following lemma states some properties of the function $\mathcal{D}$.

Lemma 3.2 *Let p be a process and t be a test.*

1. $\mathcal{D}(p, t) \geq 0$.
2. *p guarantees t* $\Leftrightarrow \mathcal{D}(p, t) < \infty$.

A preorder which relates processes on the basis of their predictability can be defined as follows.

Definition 3.3 $p \sqsubseteq_{pred} q$ *if* $\forall t :\ \mathcal{D}(q, t) \leq \mathcal{D}(p, t)$.

So $p \sqsubseteq_{pred} q$ if q exhibits less variable that p in completing *any* test; thus q is "more predictable" than p.

Alternative characterization. In order to facilitate reasoning about processes using this preorder, we would like an alternative characterization that refers only to the two processes in question and does not mention tests explicitly. We also want to prove that the $must$-preorder and the $predictability$-preorder coincide. We achieve both of these goals by showing that the alternative characterization of $must$-preorder given in [12] is also a characterization of $predictability$-preorder. We begin by recalling the definition of the notion of *barb* [12].

Definition 3.4 *The set of* barbs *is smallest set satisfying the following conditions, where $a \in Act$ and $A \subseteq Act$ is finite.*

1. *Ω, A are barbs.*
2. *If $\underline{b}$ is a barb then $a\underline{b}$ is a barb.*
3. *If $\underline{b}$ is a barb then $A\sigma\underline{b}$ is a barb.*

Informally barbs are finite sequences of the form "$s_1 A_1 \sigma s_2 A_2 \sigma \cdots s_k A_k$" or "$s_1 A_1 \sigma s_2 A_2 \sigma \cdots s_k \Omega$" where each $s_i \in Act^*$ and A_i is a finite subset of *Act*; Ω denotes undefined behavior. Intuitively, a barb represents

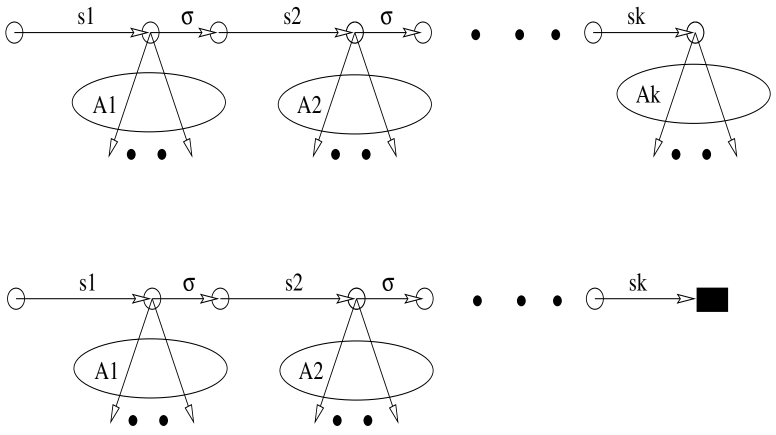

Figure 2. Pictorial representation of barbs

a "view" of an execution of a process, in a manner that will be made precise later. A pictorial representation of barbs is presented in Figure 2. Now we define a preorder $<<$ on the set of barbs [12].

Definition 3.5 *The relation $<<$ is the smallest relation satisfying the following conditions.*

1. $\Omega << \underline{b}$.
2. $A \subseteq B \Rightarrow A << B$.
3. $\underline{b}_1 << \underline{b}_2 \Rightarrow a\underline{b}_1 << a\underline{b}_2$.
4. $\underline{b}_1 << \underline{b}_2 \wedge A \subseteq B \Rightarrow A\sigma\underline{b}_1 << B\sigma\underline{b}_2$.

To obtain an intuitive feeling for this ordering, Figure 3 pictorially explains the situation when $s_1 A_1 \sigma s_2 A_2 \sigma \cdots s_k A_k \quad << \quad s_1 B_1 \sigma s_2 B_2 \sigma \cdots s_k B_k$. Figure 4 does the same when $s_1 A_1 \sigma s_2 A_2 \sigma \cdots s_k \Omega$ $<< s_1 B_1 \sigma s_2 B_2 \sigma \cdots s_k B_k \cdots$.

We now explain how to associate a set of barbs (or "execution views") to a process.

Definition 3.6 *Let p be a process. $Barb(p)$ is the smallest set of barbs satisfying the following conditions.*

1. $\Omega \in Barb(p)$ *if* $p\Uparrow$.
2. $A \in Barb(p)$ *if* $\exists q : \; p \stackrel{\epsilon}{\Rightarrow} q \wedge q$ *is stable* $\wedge$ $S(q) = A$.
3. $a\underline{b} \in Barb(p)$ *if* $\exists q : \; p \stackrel{a}{\Rightarrow} q \wedge \underline{b} \in Barb(q)$.
4. $A\sigma\underline{b} \in Barb(p)$ *if* $\exists q_1, q_2 : \; p \stackrel{\epsilon}{\Rightarrow} q_1 \wedge$ q_1 *is stable* $\wedge\, S(q_1) = A \wedge q_1 \stackrel{\sigma}{\rightarrow} q_2 \wedge \underline{b} \in Barb(q_2)$.

Intuitively, if $s_1 A_1 \sigma s_2 A_2 \sigma \cdots s_k A_k \in Barb(p)$ then p can "execute" the sequence $s_1 \sigma s_2 \sigma \cdots \sigma s_k$, and there are "intermediate states" $p_1, \cdots, p_k$, each stable, such that $p \stackrel{s_1}{\Rightarrow} p_1 \stackrel{\sigma s_2}{\Rightarrow} \cdots \stackrel{s_k}{\Rightarrow} p_k$ and $A_i = S(p_i)$. Similarly, if $s_1 A_1 \sigma s_2 A_2 \sigma \cdots s_k \Omega \in Barb(p)$ then p can "execute" $s_1 \sigma s_2 \sigma \cdots \sigma s_k$, and there are "intermediate states" $p_1, \cdots, p_{k-1}$, each stable, and p_k such that $p \stackrel{s_1}{\Rightarrow} p_1 \stackrel{\sigma s_2}{\Rightarrow} \cdots \stackrel{s_k}{\Rightarrow} p_k$, $A_i = S(p_i)$ and $p_k \Uparrow$.

Now we present the main theorem of this section, which establishes the connection between the $must$-preorder and the $predictability$-preorder; it also provides an alternative characterization of the $predictability$-preorder.

Theorem 3.7 *The following are equivalent.*

1. $p \mathrel{\underset{\sim}{\sqsubset}}_{pred} q$.
2. $p \mathrel{\underset{\sim}{\sqsubset}}_{must} q$.
3. $(\underline{b}_1 \in Barb(q) \Rightarrow \exists \underline{b}_2 \in Barb(p) : \; \underline{b}_2 << \underline{b}_1)$.

Proof. $(1 \Rightarrow 2)$: Let $p \mathrel{\underset{\sim}{\sqsubset}}_{pred} q$ and t be a test such that p guarantees t. We prove that q guarantees t as well. By Lemma 3.2 we have that $\mathcal{D}(p,t)$ is finite and nonnegative. From Definition 3.3, $\mathcal{D}(q,t) \leq \mathcal{D}(p,t)$ and so $\mathcal{D}(q,t)$ is also finite and nonnegative. By applying Lemma 3.2 once again we get q guarantees t which is the required result.

$(2 \Rightarrow 3)$: See [12]

$(3 \Rightarrow 1)$: We want to prove that $\mathcal{D}(q,t) \leq \mathcal{D}(p,t)$ for any test t. Fix a test t. Suppose p guarantees t is *not* true. Then by Lemma 3.2, we can conclude that $\mathcal{D}(p,t)$ is ∞ which immediately implies that $\mathcal{D}(q,t) \leq \mathcal{D}(p,t)$ irrespective of the value of $\mathcal{D}(q,t)$.

Now consider the possibility that p guarantees t is true. It must be case that q guarantees t is also true, as Condition 2 and Condition 3 are equivalent [12]. Now we will show that if c is some arbitrary computation of $q|t$ then there exists some computation c' of $p|t$ such that $sl(c') = sl(c)$. This would imply that $\mathcal{D}_w(p,t) \geq$

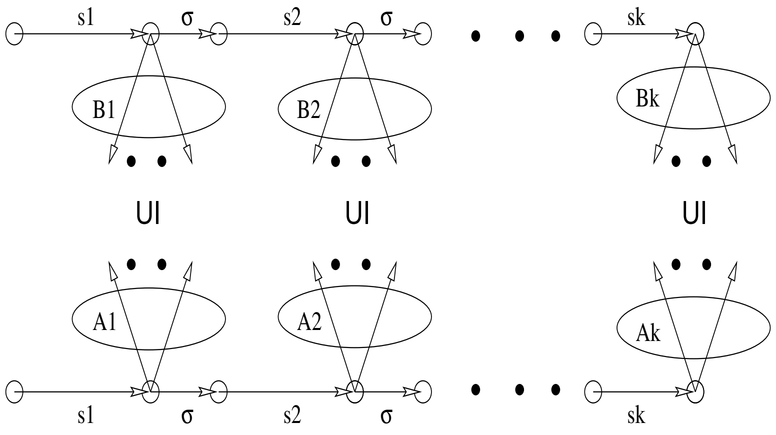

Figure 3. $s_1A_1\sigma s_2A_2\sigma\cdots s_kA_k << s_1B_1\sigma s_2B_2\sigma\cdots s_kB_k$

$\mathcal{D}_w(q,t)$ and $\mathcal{D}_b(p,t) \leq \mathcal{D}_b(q,t)$. Consequently we can infer that $\mathcal{D}(q,t) \leq \mathcal{D}(p,t)$.

To prove the above statement, let c be a computation of $q|t$ as below:

$$q|t \equiv q_0|t_0 \xrightarrow{1} q_1|t_1 \xrightarrow{1} q_2|t_2 \xrightarrow{1} \cdots$$

Using Condition 3, it is possible to devise a sequence of states $p_0, p_1, \cdots$ with $p_0 = p$, in such a way that the p_i's can be "zipped-up" with the t_i's to produce a computation c' of $p|t$ [12]. Since both p and q guarantee the test t, the computations c and c' are successful. Further the success lengths of c and c' must be the same as both use the same sequence of test states. This is the required result.

□

Corollary 3.8 $p \sqsubseteq\!\!\!\!\sim_{pred}\ q$ *if and only if for every* t *such that* p *guarantees* t, $\mathcal{D}_w(p,t) \geq \mathcal{D}_w(q,t)$ *and* $\mathcal{D}_b(p,t) \leq \mathcal{D}_b(q,t)$.

Intuitively, the above corollary states that when $p \sqsubseteq\!\!\!\!\sim_{pred}\ q$, any activity that the system p is guaranteed to complete will also be completed by q "no sooner" and "no later" than p.

Congruences. In order for a semantic preorder to be useful for verification purposes, it is imperative that the preorder be preserved under all process contexts. Such a *congruence* facilitates hierarchical reasoning; properties of a large system can be inferred from those of its components. In view of the above theorem the predictability-preorder also enjoys all the congruence properties of the $must$-preorder.

Lemma 3.9 *Let* $p \sqsubseteq\!\!\!\!\sim_{pred} q$ *and* r *be some process. Then the following are true.*

1. $\alpha.p \sqsubseteq\!\!\!\!\sim_{pred} \alpha.q$.
2. $p|r \sqsubseteq\!\!\!\!\sim_{pred} q|r$.
3. $p[R] \sqsubseteq\!\!\!\!\sim_{pred} q[R]$.
4. $p\backslash\lambda \sqsubseteq\!\!\!\!\sim_{pred} q\backslash\lambda$.

Due to the preemptive power of the action τ, the predictability-preorder is not preserved under the operator $+$ and $\lfloor\ \rfloor(\ )$. This problem can be remedied in the usual fashion as follows.

Definition 3.10 *Let* $p \sqsubseteq\!\!\!\!\sim^{+}_{pred} q$ *if for every closed term* r, $p + r \sqsubseteq\!\!\!\!\sim_{pred} q + r$.

We extend $\sqsubseteq\!\!\!\!\sim^{+}_{pred}$ to open terms in the usual way. The following theorem asserts that the preorder $\sqsubseteq\!\!\!\!\sim^{+}_{pred}$ is preserved under all TPL contexts. (Recall that $\sqsubseteq\!\!\!\!\sim^{C}_{pred}$ is the largest precongruence with respect to TPL contained in $\sqsubseteq\!\!\!\!\sim_{pred}$).

Theorem 3.11 *Let* t *and* u *be terms. Then* $t \sqsubseteq\!\!\!\!\sim^{+}_{pred} u \Leftrightarrow t \sqsubseteq\!\!\!\!\sim^{C}_{pred} u$

Proof. The relation $\sqsubseteq\!\!\!\!\sim^{+}_{pred}$ and $\sqsubseteq\!\!\!\!\sim^{+}_{must}$ are the same. It is shown in [12] that the relation $\sqsubseteq\!\!\!\!\sim^{+}_{must}$ is the largest precongruence contained in $\sqsubseteq\!\!\!\!\sim_{must}$. But we have established in Theorem 3.7 that $\sqsubseteq\!\!\!\!\sim_{pred}$ and $\sqsubseteq\!\!\!\!\sim_{must}$ coincide. Consequently, we have proved that $\sqsubseteq\!\!\!\!\sim^{+}_{pred}$ and $\sqsubseteq\!\!\!\!\sim^{+}_{must}$ are identical.

□

4 Full Abstraction

In this section we wish to prove the optimality of the $\sqsubseteq\!\!\!\!\sim^{+}_{pred}$-precongruence in the sense that it only distinguishes processes that an observer can tell apart. Such a

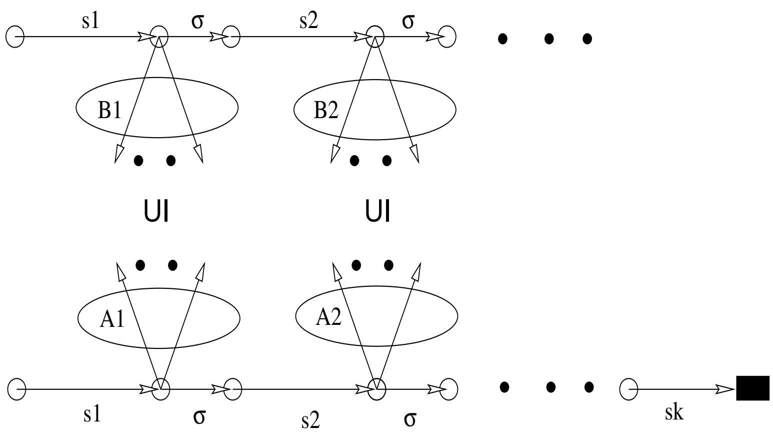

Figure 4. $s_1A_1\sigma s_2A_2\sigma\cdots s_k\Omega << s_1B_1\sigma s_2B_2\sigma\cdots s_kB_k\cdots$

result is typically achieved in the following manner [29]. First we decide what are the "observables of interest" for a process. Such a choice naturally induces a preorder $\preceq$ on processes: $p \preceq q$ if for all contexts $C[\,]$ the observables of $C[q]$ are "better" than the observables of $C[p]$. Second we show that $\underset{\sim}{\sqsubset}^{+}_{pred}$ and $\preceq$ are the same. Informally this means that $\underset{\sim}{\sqsubset}^{+}_{pred}$ does not make any unnecessary distinctions, and we say $\underset{\sim}{\sqsubset}^{+}_{pred}$ is *fully abstract* with respect to the "observables of interest".

Since we are interested in the timing behavior of the processes, the time required to obtain a response when prompted for an interaction by the environment is a possible candidate for an observable of interest. But processes are nondeterministic in general, and the time to obtain a response for a prompt may vary. As the capability of a process to meet or miss its deadlines depends on its worst-case behavior, we take the "maximum time to obtain response" to be the observable of interest for a process. Now we define a relation $\underset{\sim}{\sqsubset}_{ready}$ such that $\underset{\sim}{\sqsubset}^{C}_{ready}$ will play the role of $\preceq$ in the previous paragraph. Then we will prove that $\underset{\sim}{\sqsubset}^{C}_{ready}$ and $\underset{\sim}{\sqsubset}^{+}_{pred}$ are the same. The proof technique employed for establishing this result is inspired by the work of Main [13].

Definition 4.1 *Let p be a process, m be a natural number, and $a \in Act$.*

1. *p is* responsive *if p is stable and $\exists a :\ p \xrightarrow{a}$.*
2. *$p \Uparrow$ if $\exists q :\ p(\xrightarrow{1})^{*}q \wedge q \Uparrow$; $p \Downarrow$ if it is not the case that $p \Uparrow$.*
3. *p is m-responsive if $p \Downarrow$ and $(p(\overset{\sigma^i}{\Rightarrow})p' \wedge p'$ is not responsive $\Rightarrow \exists j, q :\ i + j \leq m \wedge p' \overset{\sigma^j}{\Rightarrow} q \wedge q$ is responsive).*
4. *p is* ready *if it is m-responsive for some m.*
5. *$p \underset{\sim}{\sqsubset}_{ready} q$ if for every m, p is m-responsive implies q is m-responsive too.*

Intuitively, a process is responsive if it is guaranteed to respond immediately to some request for interaction from its environment. Note that an unstable process is not guaranteed to respond to an external stimulus immediately. The reason is that an unstable process is either a partially defined system or one that is capable of executing an internal step. Any partially defined system has "unpredictable" behavior and so might not respond to an external stimulus. Similarly a system that is capable of an internal action can ignore the external stimulus and execute the internal step. So the process $a + \tau.b$, for example, is not responsive. However, we do consider the process $a + \sigma.b$ to be responsive. This decision is a consequence of the maximal progress assumption; the system will not idle if it has a possibility of making progress by interacting with its environment.

If $p \Uparrow$, then there exists a possibility that after the passage of some amount of time p will not respond to an external request, for example, by executing an infinite sequence of internal actions. A process is m-responsive if it is inevitable that it will evolve to a responsive state within m units of time. In other words, a response to *some* external stimulus is guaranteed within m steps. Accordingly when $p \underset{\sim}{\sqsubset}_{ready} q$ it means that in the worst case, q will respond to an external stimulus "no later than" p.

Now we show that the largest precongruence contained in this relation is actually $\underset{\sim}{\sqsubset}^{C}_{pred}$.

Theorem 4.2 $p \underset{\sim}{\sqsubset}^{C}_{ready} q \Leftrightarrow p \underset{\sim}{\sqsubset}^{C}_{pred} q$.

Proof. Let $p \underset{\sim}{\sqsubset}^{C}_{ready} q$. We would like to prove that $p \underset{\sim}{\sqsubset}^{C}_{pred} q$. In view of Theorem 3.7, it is sufficient to prove that $p \underset{\sim}{\sqsubset}^{C}_{must} q$. If $C[\,]$ is an arbitrary TPL context, we want to prove that $C[p] \sqsubseteq_{must} C[q]$. Let $\underline{b}$ be a barb of $C[q]$. Our aim is to prove the existence of a barb $\underline{b'} \in C[p]$ such that $\underline{b'} << \underline{b}$.

The proof of this statement hinges on the following fact. Given any barb $\underline{b}$ and a pair of processes p and q it is possible to construct a test t_{bpq} such that the following statements hold. If $\underline{b} \in Barb(p)$ then p does *not* guarantee t_{pqb}. Conversely, if p does not guarantee t_{bpq}, then there exists some barb $\underline{b'} \in Barb(p)$ such that $\underline{b'} << \underline{b}$.

Using this fact we can infer that $C[q]$ does not guarantee $t_{C[p]C[q]b}$. For brevity, we will refer to this test as t_b. Let S_{pq} stand for $Sort(C[p]) \cup Sort(C[q])$ and $D_b[\,]$ stand for the context $([\,]|t_b)\backslash S_{pq}$. Then it is not difficult to see that $D_b[C[q]]$ is not ready. As $p \sqsubseteq^C_{ready} q$ it follows that $D_b[C[p]] \sqsubseteq_{ready} D_b[C[q]]$. So it must be the case that $D_b[C[p]]$ is not ready as well. By examining the structure of the context $D_b[\,]$, it can be inferred that $C[p]$ does not guarantee t_b. Again, by using the fact presented in the previous paragraph, we deduce the existence of barb $\underline{b'} \in C[p]$ such that $\underline{b'} << \underline{b}$. This concludes the proof of the statement that $p \sqsubseteq^C_{ready} q \Rightarrow p \sqsubseteq^C_{pred} q$.

Now we would like to prove $p \sqsubseteq^C_{ready} q$ assuming $p \sqsubseteq^C_{pred} q$. Clearly if we show that the preorder $\sqsubseteq_{ready}$ contains the preorder $\sqsubseteq_{must}$ then we are done. Let $p \sqsubseteq_{must} q$ and p be m-responsive. Our goal is to show that q is also m-responsive. Let T_{pq} be the set $Sort(p) \cup Sort(q) \cup \overline{Sort(p)} \cup \overline{Sort(q)}$. Consider the class of tests t_i defined inductively as follows.

$$t_0 = \lfloor \sum_{a \in T_{pq}} a.w.nil \rfloor (nil) \qquad t_{i+1} = \sigma.t_i \quad i \geq 0$$

It can be shown that p guarantees t_m and since $p \sqsubseteq_{must} q$ it must be the case that q guarantees t_m. By analyzing the structure of the test t_m, one can show that q is m-responsive too . This concludes the proof of $p \sqsubseteq^C_{pred} q \Rightarrow p \sqsubseteq^C_{ready} q$.

□

5 An Example: Client-Server System

In this section we illustrate the utility of the *predictability* preorder in system verification by means of a simple example paradigmatic of many real-time systems. The system we consider consists of a *client* trying to complete an assignment with the help of the services provided by a *server*. The exact value of the time required by the server to process a request from the client is not known, though its upper and lower bounds are. An abstract specification of such a composite system consisting a client and a server is given by the process *Spec* defined below.

$$Spec \Leftarrow \sum_{i=2}^{i=5} start.\sigma^i.\overline{finish}.Spec$$

Intuitively the above specification states that the time required to finish the assignment is either 2,3,4 or 5 units. Now we will construct systems $Impl_1$, $Impl_2$ and $Impl_3$ and examine their relationship with *Spec* in terms of the *predictability* preorder. All of the above systems will be built with the following process Cl acting as the client process.

$$Cl \Leftarrow start.\overline{req}.res.\overline{finish}.Cl$$

The client process is asked to perform an assignment by means of the action *start*; it then requests the assistance of a server through the action *req* and awaits reply. After receiving the result through the port *res*, it signals the completion of the assignment by executing the action *finish* and reverts back to its initial state. Now we present different "implementations" of the server process, each having different timing characteristics.

$$Ser_1 \Leftarrow req.(\tau.\sigma^2.\overline{res}.Ser_1 + \tau.\sigma^4.\overline{res}.Ser_1)$$

$$Ser_2 \Leftarrow req.(\tau.\sigma.\overline{res}.Ser_1 + \tau.\sigma^4.\overline{res}.Ser_1)$$

$$Ser_3 \Leftarrow req.(\tau.\sigma^6.\overline{res}.Ser_1)$$

These server processes accept the request from the client through the port *req* and report the result back to the client through the port *res*. To process a request Ser_1 takes either 2 or 4 units of time, Ser_2 takes either 1 or 4 units of time and Ser_3 takes 6 units of time. When these server processes are composed with the client process in the following manner we get three putative implementations of the specification (Figure 5).

$$Impl_1 \Leftarrow (Ser_1|Cl)\backslash\{req, res\}$$

$$Impl_2 \Leftarrow (Ser_2|Cl)\backslash\{req, res\}$$

$$Impl_3 \Leftarrow (Ser_3|Cl)\backslash\{req, res\}$$

In $Impl_1$, given an assignment it will be completed in either 2 or 4 units of time. So it appears that $Impl_1$ has less variability in its assignment-completion times than *Spec*. Indeed it is possible to show that $Spec \sqsubseteq^+_{pred} Impl_1$ by making use of the algebraic characterization of $\sqsubseteq^+_{must}$ given in [12] and Theorem 3.7. This substantiates our intuition that $Impl_1$ is more predictable than *Spec*.

The system $Impl_2$, on the other hand, can complete an assignment in 1 or 4 units of time. It has the same amount of variability as that of *Spec* in finishing this assignment and even completes the assignment faster than *Spec* sometimes. However it is not case that $Impl_2$ is

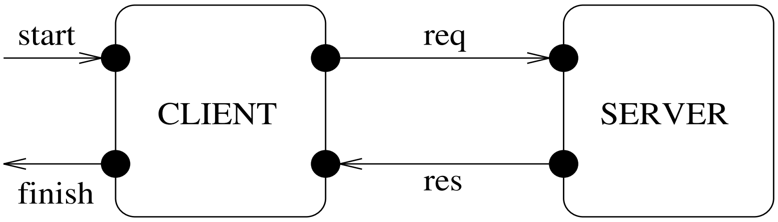

Figure 5. Client-Server Architecture

better than *Spec* with respect to $\sqsubseteq_{pred}^{C}$. The context $D[\ ] = (\overline{\mathit{start}}.\sigma.\lfloor \mathit{finish.nil} \rfloor (w.\mathit{nil}) | [\ ]) \backslash \{\mathit{start}, \mathit{finish}\}$ can distinguish them; $D[Spec] \not\sqsubseteq_{pred} D[Impl_2]$. Intuitively, this context can initiate an assignment for the client but is not prepared to receive the result of the assignment after 1 unit of time. This system can work successfully with the *Spec* but not with the $Impl_2$. Thus finishing an activity too soon can also be undesirable.

Now let us examine the relationship between $Impl_3$ and $Spec$. The behavior of this system is quite predictable but it is still not the case that *Spec* $\sqsubseteq_{pred}$ $Impl_3$. The context $C[\ ] = (\overline{\mathit{start}}.\sigma^5.\lfloor \mathit{finish}.w.\mathit{nil} \rfloor (\mathit{nil}) | [\ ]) \backslash \{\mathit{start}, \mathit{finish}\}$ can distinguish them; $C[Spec] \not\sqsubseteq_{pred} C[Impl_3]$. This context demonstrates the existence of a system that requires the result of the assignment exactly 5 units after the initiation. *Spec* can satisfy this requirement but $Impl_3$ cannot as it is slower.

Thus the above example illustrates the sensitivity of the *predictability* preorder to both the worst-case and best-case timing characteristics of real-time systems.

6 Conclusions

In this paper we presented a process-algebraic framework, founded on the paradigm of testing, for reasoning about the predictability of real-time systems. Under this framework, if a process p is less than a process q then q is at least as predictable as p; given any activity t, q would finish it "no later than" and "no sooner than" p. We developed this framework by relating systems on the basis of the variability found in their activity-completion times. Further we showed that the $predictability$ preorder is same as the $must$ preorder [12]. The benefits of such an identification are two-fold. Firstly, the algebraic characterization and congruence properties of $must$ preorder obtained in [12] are applicable to the $predictability$ preorder. In the other direction, the full abstractness result given here for the $predictability$ preorder is applicable for the $must$ preorder. The full abstractness result also reaffirms the naturalness and optimality of the $predictability$ preorder by showing that it is the largest reasonable precongruence respecting the notion of "faster time to obtain response". We also provided a simple example to illustrate the usefulness of this approach.

The literature on process-algebraic approaches to specification and verification of real-time systems is quite extensive, and [21] presents a survey of existing results as of the beginning of this decade. It is clear that our work is built on the foundations laid by the research reported in [12]. All of the other existing work either provide a bisimulation-based semantic theory for analyzing the potential nondeterminism the systems might exhibit during their execution [9, 15, 22, 30] or develop a framework for reasoning about the relative speeds of the systems [7, 16]. In a non-interleaving semantic framework, testing-based "faster-than" relations have been proposed in [26, 27]. None of them attempt to relate the systems on the basis of their predictability and to our knowledge, this paper is the first one to make such a contribution.

As we remarked in Section 3 this paper considers hard real-time systems in which finishing a certain activity too soon can be as catastrophic as finishing it too late. Consequently, we adopted the range of activity-completion times as the measure of predictability with which a process completes an activity. This measure is sensitive to the earliest and latest activity-completion times, and it does not depend on the "sample size" i.e., the total number of possible completion times. But it is insensitive to the dispersion present in the intermediate values. On the other hand, *variance* is a measure that reflects the dispersion present in the whole sample more accurately but its sensitivity to extreme value diminishes as sample size increases. Thus this measure would be appropriate for modeling *soft* real-time systems (the class of real-time time systems for which occasionally missing a deadline is acceptable). As future work, we intend to investigate this issue of designing an algebraic framework for reasoning about soft real-time systems.

References

[1] S. Arun-Kumar and M. Hennessy. An efficiency pre-

order for processes. *Acta Informatica*, 29(8):737–760, 1992.

[2] S. Arun-Kumar and V. Natarajan. Conformance: A precongruence close to bisimilarity. In *Proceedings of the International Workshop on Structures in Concurrency Theory, STRICT '95*, Workshops in Computing Series, Berlin, May 1995. Springer-Verlag.

[3] G. Boudol, I. Castellani, M. Hennessy, and A. Kiehn. A theory of processes with localities. In *CONCUR' 92*, volume 630 of *LNCS*, pages 108–122. Springer-Verlag, 1992.

[4] E. Brinksma, A. Rensink, and W. Vogler. Fair testing. In *CONCUR '95*, volume 962 of *LNCS*, pages 313–328. Springer-Verlag, 1995.

[5] S. Brookes, C. Hoare, and A. Roscoe. A theory of communicating sequential processes. *Journal of the Association for Computing Machinery*, 31(3):560–599, July 1984.

[6] I. Castellani and M. Hennessy. Distributed bisimulations. *JACM*, 10(4):887–911, 1989.

[7] R. Cleaveland and A. Zwarico. A theory of testing for real time. In *Sixth Annual Symposium on Logic in Computer Science (LICS '91)*, pages 110–119, Amsterdam, July 1991. IEEE Computer Society Press.

[8] R. De Nicola. Extensional equivalences for transition systems. *Acta Informatica*, 1987.

[9] R. Gerber and I. Lee. CCSR: A calculus for communicating shared resources. In J. Baeten and J. Klop, editors, *CONCUR '90*, volume 458 of *Lecture Notes in Computer Science*, pages 263–277, Amsterdam, Aug. 1990. Springer-Verlag.

[10] M. Hennessy. *Algebraic Theory of Processes*. MIT Press, Boston, 1988.

[11] M. Hennessy. Timed process algebras: A tutorial. In *Proceedings of International Summer School on* Process Design Calculi*, Martoberdorf*. 1992.

[12] M. Hennessy and T. Regan. A process algebra for timed systems. *Information and Computation*, 117:221–239, 1995.

[13] M. Main. Trace, failure and testing equivalences for communicating processes. *International Journal of Parallel Programming*, 16(5):383–400, 1987.

[14] R. Milner. *Communication and Concurrency*. Prentice-Hall, London, 1989.

[15] F. Moller and C. Tofts. A temporal calculus of communicating systems. In *CONCUR '90*, number 458 in LNCS, pages 401–415. Springer-Verlag, 1990.

[16] F. Moller and C. Tofts. Relating processes with respect to speed. In J. Baeten and J. Groote, editors, *CONCUR '91*, volume 527 of *Lecture Notes in Computer Science*, pages 424–438, Amsterdam, Aug. 1991. Springer-Verlag.

[17] V. Natarajan. *Degrees of Delay: Semantic theories for priority, efficiency, fairness and predictability in process algebras*. PhD thesis, North Carolina State University, 1996.

[18] V. Natarajan and R. Cleaveland. An algebraic theory of process efficiency. To appear in the Proceedings of Eleventh Annual Symposium on Logic in Computer Science (LICS '96).

[19] V. Natarajan and R. Cleaveland. Divergence and fair testing. In Z. Fülöp and F. Gécseg, editors, *Automata, Languages and Programming (ICALP '95)*, volume 944 of *Lecture Notes in Computer Science*, pages 648–659, Szeged, Hungary, July 1995. Springer-Verlag.

[20] R. D. Nicola and M. Hennessy. Testing equivalence for processes. *Theoretical Computer Science*, 34:83–133, 1984.

[21] X. Nicollin and J. Sifakis. An overview and synthesis on timed process algebras. In *REX Workshop on Real-Time: Theory in Practice*, number 600 in LNCS. Springer-Verlag, 1991.

[22] X. Nicollin and J. Sifakis. The algebra of timed processes, ATP:theory and applications. *Information and Computation*, 114:131–178, 1994.

[23] C. Park and A. Shaw. Experiments with a program timing tool based on source-level timing schema. *IEEE Computer*, 24(5):48–57, May 1991.

[24] J. Stankovic. Misconceptions about real-time computing. *IEEE Computer*, 21(10):10–19, October 1988.

[25] J. Stankovic and K. Ramamritham. What is predictability of real-time systems? *Journal of Real-Time Systems*, 2:247–254, December 1990.

[26] W. Vogler. Timed testing of concurrent systems. In *ICALP '93*, volume 700 of *LNCS*. Springer-Verlag, 1993.

[27] W. Vogler. Faster asynchronous systems. In *CONCUR '95*, volume 962 of *LNCS*, pages 299–313. Springer-Verlag, 1995.

[28] D. Walker. Bisimulations and divergence in CCS. *Information and Computation*, 85(2):202–241, 1990.

[29] G. Winskel. *The Formal Semantics of Programming Languages*. MIT Press, 1993.

[30] W. Yi. Real-time behavior of asynchronous agents. In *CONCUR '90*, volume 458 of *LNCS*, pages 502–520. Springer-Verlag, 1990.