A Formalism for Real-Time Concurrent Object-Oriented Computing

Ichiro Satoh
satoh@mt.cs.keio.ac.jp

Mario Tokoro *
mario@mt.cs.keio.ac.jp

Department of Computer Science, Keio University
3-14-1, Hiyoshi, Kohoku-ku, Yokohama, 223, Japan
Tel:+81-45-560-1150 Fax:+81-45-560-1151

Abstract

We investigate a formal model for reasoning about real-time object-oriented computations. The model is an extension of CCS with the notion of time, called *RtCCS* (*Real-time Calculus of Communication Systems*). It can naturally model real-time concurrent objects as communicating processes and represent the timed properties of objects. We define two timed equivalences based on CCS's bisimulation and derive algebraic laws for reasoning about real-time processes. The equivalences provide a formal framework for analyzing the behavior and timing of real-time computations. Also, we define a sound and complete equational proof system for finite processes. Some examples in RtCCS are shown in order to demonstrate its usefulness.

1 Introduction

Real-time systems are increasingly being used in various areas such as factory automation, robotics, and multi-media systems. These systems have to interact and cooperate with many external elements which run in parallel, such as other computers, sensors, and actuators. The notion of concurrent object-oriented computing [24] is considered as a powerful method to design and to develop such real-time systems. This is because concurrent object-oriented systems consist of objects which are logically self-contained active entities that cooperate with each other. Concurrent objects can naturally model such active elements directly. Recently, some real-time systems and languages use this concept as their basis [9, 21].

Real-time systems have certain time constraints which must be satisfied. The correctness of a real-time system depends not only on the logical results of computation, but also on the time at which the results are produced. Therefore, the construction and debugging of real-time programs is far more complex and difficult than those of ordinary concurrent programs. We need the support of formal verification methods for reasoning about real-time programs. In the past few years several researchers have proposed such methods based on timed Petri nets, first order logics, and temporal logics. However, these frameworks are not always fit for concurrent object-oriented computing. The goal of this paper is to investigate a formal model to reason about real-time object-oriented computations.

Various formal models for concurrent object-oriented computation have previously been devised. Among these, process calculus [2, 11, 13] is a well-defined theory that can model naturally and easily concurrent objects as communicating pro-

*Also with Sony Computer Science Laboratory Inc. 3-14-13 Higashi-Gotanda, Shinagawa-ku, Tokyo, 141, Japan.

OOPSLA'92, pp. 315-326

cesses which may change their states when communicating with another agent. Also, process calculi have equivalence relations over processes which provide theoretical frameworks for analyzing the behavior of objects and for substituting objects.

In this paper we develop a process calculus which permits expressing and analyzing time constraints in real-time object-oriented computations. To do this, we extend an existing process calculus with the notion of time and introduce a timed process calculus called *RtCCS (Real-time Calculus of Communication Systems)*. It is an extension of Milner's CCS [11] with a minimal set of notions for time: a tick action and a timeout operator. The execution of tick actions corresponds to the passage of time; the timeout operator represents a behavior dependent on the passage of time. Furthermore, we develop theoretical equivalences over processes with time properties. In RtCCS, time independent properties of processes can be treated as usual CCS while the time dependent properties can be treated with the timed extensions. In order to demonstrate its effectiveness in a simpler form, our timed extension is developed for CCS. However, the same extension can be naturally adopted into π-calculus [13] which has more powerful mechanisms for modeling object-oriented computing.

In the next section, we discuss some of the related works. In Section 3, we define the syntax and the operational semantics of RtCCS. Section 4 defines two equivalence relations called timed strong equivalence and timed observation equivalence respectively, and studies their basic properties. In Section 5, we present some examples that demonstrate the usefulness of RtCCS. The final section contains some concluding remarks. In Appendix A, we define an equational proof system based on timed strong equivalence.

2 Related Research

Recently, many theoretical models for concurrent object-oriented computations have been explored in process calculi, such as CCS [11], π-calculus [13], and ACP [2]. In the process calculus paradigm, objects can be viewed as processes, interactions among objects can be seen as communications, and encapsulation can be modeled by the restriction of visible communications. In [8] a formal system based on the notion of actor-like object with asynchronous communication was investigated, and in [17] an executable notation for specifying concurrent object-oriented languages in a process calculus was explored. In [18] some requirements for a calculus suitable for concurrent objects is presented. In [15] the author presents a language based on a process calculus and analyzes the essential features of object-based concurrency. In [22] a semantics for POOL [1] is defined by the translation of the language constructs into π-calculus [13].

Many theoretical models for real-time computations have been explored in temporal logic, timed Petri nets, and denotational semantics frameworks based on linear history semantics. More recently, several studies developed temporal models based on process calculi such as CCS, ACP, and CSP [6, 5, 10, 16, 14, 23].

Milner's SCCS [10] is a calculus for synchronous processes based on the idea that each atomic action takes one unit of time. The concurrent agents of SCCS essentially proceed in lockstep and at every instant perform a single action; there is also an asynchronous operator. Since each object in an object-oriented computation proceeds at indeterminate relative speeds, SCCS does not fit our purposes.

TPA [6], timed CCS [23], and Temporal CCS [14] introduce delay operators into CCS. The delay operators represent the suspension of execution for a specified time, similarly to the *delay* command in Ada. In TPA and timed CCS, like in RtCCS, there are two assumptions: time advances only when communications are not possible and that actions are instantaneous. Once an action is enabled, it cannot be disabled. Therefore, they cannot express timeout behavior. On the other hand, Temporal CCS can represent timeout behavior but has no equivalent relation over processes based on observation of their behaviors. PTCCS [5] and ATP [16], like ours, have a timeout operator. PTCCS is

an extension of CCS with discrete time and probability. PTCCS is similar to our RtCCS. However, unlike ours, time stops inside the timeout operator of PTCCS. Therefore, the timeout operator of PTCCS cannot apply to a process with timed properties, and PTCCS cannot describe a timeout exception for such a process. The timeout operator of ATP is similar to ours. However, ATP has no notion of observation equivalence because ATP has no τ action, since it is not based on CCS.

3 RtCCS

In this section, we present the basic idea of RtCCS and provide its formal definition.

3.1 Time Extensions

We assume a conceptual global clock: time passes as the global clock performs *tick* actions. The tick action is a synchronous broadcast message over all processes. It is described as $\surd$. The advance of time can be represented as a sequence of tick actions and is viewed as discrete time. Also, we assume that all communications and internal actions take no time, and that complementary actions and internal actions are performed as soon as they become possible. Therefore, a process having a communication action ready must wait until other process becomes ready to communicate with it. Once both the partners of a communication are ready, they must perform the communication actions immediately, before the next tick $\surd$.

As mentioned previously, many real-time programming languages have timed operations such as delay and timeout. In order to be used as a framework for formal semantics of languages, RtCCS needs to represent behaviors dependent on the advancing of time. We introduce a special binary operator: $\langle\ ,\ \rangle_t$, called a *timeout* operator. As shown in Figure 1, $\langle P, Q\rangle_t$ denotes a process that after t time units becomes Q, unless P performs any actions prior to that. Intuitively $\langle P, Q\rangle_t$ behaves as process P if P can execute an initial transition within t units of time, whereas $\langle P, Q\rangle_t$ behaves as process Q if P does not perform any action within t units of time. That is to say, P and Q correspond to ordinary and timeout processes respectively, in practical programming.

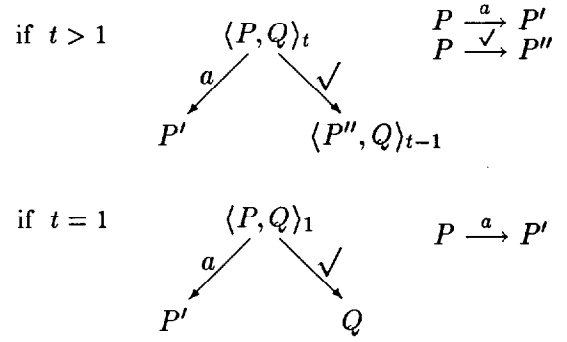


Figure 1: The behavior of the timeout operator

3.2 The RtCCS Language

In this subsection, we present the syntax and the operational semantics of RtCCS.

3.2.1 Notation and Syntax of RtCCS

We presuppose that $\mathcal{A}$ is a set of communication action names and $\overline{\mathcal{A}}$ the set of co-names. Let a,b,... range over $\mathcal{A}$ and $\overline{a}$,$\overline{b}$,... over $\overline{\mathcal{A}}$. An action $\overline{a}$ is the complementary action of a, and $\overline{\overline{a}} \equiv a$. Let τ denote an internal action, and $\surd$ a tick action. Finally let $Act \equiv \mathcal{A} \cup \overline{\mathcal{A}} \cup \{\tau\}$ ranged over by α,β,..., and $Act_T \equiv Act \cup \{\surd\}$ ranged over by μ, ν,....

Definition 1 The set $\mathcal{E}$ of RtCCS expressions ranged over by $E, E_1, E_2, F, \ldots$ is defined recursively by the following abstract syntax.

E	::=	$\mathbf{0}$	(Deadlock Process)
		X	(Process Variable)
		$\alpha.E$	(Action Prefix)
		$E_1 + E_2$	(Summation)
		$E_1 \mid E_2$	(Composition)
		$E[f]$	(Relabeling)
		$E \setminus L$	(Restriction)
		$\mathbf{rec}\ X : E$	(Recursion)
		$\langle E_1, E_2\rangle_t$	(Timeout)

where t is a natural number, $f \in Act \to Act$ and $L \subseteq Act$. We assume that $f(\tau) = \tau$, $f(\surd) = \surd$, and that X is always *guarded* [1] [11].

We denote the set of closed expressions by $\mathcal{P}(\subseteq \mathcal{E})$, ranged over $P, Q, \ldots$. ∎

The syntax of RtCCS is essentially the same as that of CCS, except for the newly introduced tick action $\surd$ and timeout operator. Intuitively, the meaning of process constructions are as follows: **0** represents a deadlocked or terminated process; $\alpha.E$ performs an action α and then behaves like E; $E_1 + E_2$ is the process which may behave as E_1 or E_2; $E_1|E_2$ represents processes E_1 and E_2 executing concurrently; $E[f]$ behaves like E but with the actions relabeled by function f; $E \setminus L$ behaves like E but with actions in $L \cup \bar{L}$ prohibited; $\mathbf{rec}\, X : E$ binds the free occurrences of X in E but we shall often use the more readable notation $X \stackrel{\text{def}}{=} E$ instead.

3.2.2 Operational Semantics of RtCCS

RtCCS is a labeled transition system $\langle\ \mathcal{E},\ Act_T,\ \{\ \stackrel{\mu}{\longrightarrow}\ \mid \mu \in Act_T\ \}\ \rangle$ where $\stackrel{\mu}{\longrightarrow}$ is a transition relation ($\stackrel{\mu}{\longrightarrow} \subseteq \mathcal{E} \times \mathcal{E}$). The transition relation $\stackrel{\mu}{\longrightarrow}$ is defined by structural induction and is the smallest relation given by the rules in Figure 2.

The $\longrightarrow$ relation does not distinguish between observable and unobservable actions. In order to reflect that τ is not visible, we define two transition relations due to the unobservability of τ.

Definition 2

(i) $P \stackrel{\mu}{\Longrightarrow} Q \stackrel{\text{def}}{=} P(\stackrel{\tau}{\longrightarrow})^* \stackrel{\mu}{\longrightarrow} (\stackrel{\tau}{\longrightarrow})^* Q$

(ii) $P \stackrel{\hat{\mu}}{\Longrightarrow} Q \stackrel{\text{def}}{=} P(\stackrel{\tau}{\longrightarrow})^* \stackrel{\mu}{\longrightarrow} (\stackrel{\tau}{\longrightarrow})^* Q$ if $\mu \neq \tau$ and otherwise $P(\stackrel{\tau}{\longrightarrow})^* Q$. ∎

Also, we define an operation to denote a process capable of infinite internal computation.

Definition 3 $E\uparrow$ if $\exists E'$: $E(\stackrel{\tau}{\longrightarrow})^\omega E'$ ∎

[1] X is *guarded* in E if each occurrence of X is only within some subexpressions $\alpha.E'$ in E; c.f. *unguarded* expressions, e.g. $\mathbf{rec}\, X : X$ or $\mathbf{rec}\, X : X + E$.

$$\mathbf{ACT_0}: \quad \frac{-}{\alpha.E \stackrel{\alpha}{\longrightarrow} E}$$

$$\mathbf{ACT_1}: \quad \frac{-}{a.E \stackrel{\surd}{\longrightarrow} a.E}$$

$$\mathbf{ACT_2}: \quad \frac{-}{\mathbf{0} \stackrel{\surd}{\longrightarrow} \mathbf{0}}$$

$$\mathbf{SUM_0}: \quad \frac{E \stackrel{\alpha}{\longrightarrow} E'}{E + F \stackrel{\alpha}{\longrightarrow} E'}$$

$$\mathbf{SUM_1}: \quad \frac{F \stackrel{\alpha}{\longrightarrow} F'}{E + F \stackrel{\alpha}{\longrightarrow} F'}$$

$$\mathbf{SUM_2}: \quad \frac{E \stackrel{\surd}{\longrightarrow} E',\ F \stackrel{\surd}{\longrightarrow} F'}{E + F \stackrel{\surd}{\longrightarrow} E' + F'}$$

$$\mathbf{COM_0}: \quad \frac{E \stackrel{\alpha}{\longrightarrow} E'}{E|F \stackrel{\alpha}{\longrightarrow} E'|F}$$

$$\mathbf{COM_1}: \quad \frac{F \stackrel{\alpha}{\longrightarrow} F'}{E|F \stackrel{\alpha}{\longrightarrow} E|F'}$$

$$\mathbf{COM_2}: \quad \frac{E \stackrel{a}{\longrightarrow} E',\ F \stackrel{\bar{a}}{\longrightarrow} F'}{E|F \stackrel{\tau}{\longrightarrow} E'|F'}$$

$$\mathbf{COM_3}: \quad \frac{E \stackrel{\surd}{\longrightarrow} E',\ F \stackrel{\surd}{\longrightarrow} F',\ E|F \not\stackrel{\tau}{\longrightarrow}}{E|F \stackrel{\surd}{\longrightarrow} E'|F'}$$

$$\mathbf{RES_0}: \quad \frac{E \stackrel{\alpha}{\longrightarrow} E',\ \alpha, \bar{\alpha} \notin L}{E \setminus L \stackrel{\alpha}{\longrightarrow} E' \setminus L}$$

$$\mathbf{RES_1}: \quad \frac{E \stackrel{\surd}{\longrightarrow} E'}{E \setminus L \stackrel{\surd}{\longrightarrow} E' \setminus L}$$

$$\mathbf{REL}: \quad \frac{E \stackrel{\mu}{\longrightarrow} E'}{E[f] \stackrel{f(\mu)}{\longrightarrow} E'[f]}$$

$$\mathbf{REC}: \quad \frac{E\{\mathbf{rec}\, X : E/X\} \stackrel{\mu}{\longrightarrow} E'}{\mathbf{rec}\, X : E/X \stackrel{\mu}{\longrightarrow} E'}$$

$$\mathbf{TIME_0}: \quad \frac{E \stackrel{\alpha}{\longrightarrow} E',\ t > 0}{\langle E, F\rangle_t \stackrel{\alpha}{\longrightarrow} E'}$$

$$\mathbf{TIME_1}: \quad \frac{E \stackrel{\surd}{\longrightarrow} E',\ t > 0}{\langle E, F\rangle_t \stackrel{\surd}{\longrightarrow} \langle E', F\rangle_{t-1}}$$

$$\mathbf{TIME_2}: \quad \frac{F \stackrel{\mu}{\longrightarrow} F'}{\langle E, F\rangle_0 \stackrel{\mu}{\longrightarrow} F'}$$

Figure 2: Inference Rules of RtCCS

Example 1 Here we give some simple examples of RtCCS expressions.

(1) $a.\langle \bar{b}.P, \mathbf{0}\rangle_t$ After performing an input action a, it behaves as P if an output action $\bar{b}$ is executed within t units of time, otherwise terminates.

(2) $a.\langle \mathbf{0}, \bar{b}.P\rangle_t$ After performing a, it is idle for t time units and then behaves as $\bar{b}.P$.

(3) $a.(\langle \mathbf{0}, \bar{b}.\mathbf{0}\rangle_d | P)$ After performing a, it behaves as P but $\bar{b}$ becomes available after d time units. Note that $\langle \mathbf{0}, \bar{b}.\mathbf{0}\rangle_d | P$ represents a process which behaves as P and sends $\bar{b}$ as an asynchronous output communication with transmission delay d time units.

Remarks

1. The fundamental aspects of concurrent object-oriented computing are modeled as the primitive constructions of RtCCS as follows:

   - Objects as processes.
   - Concurrency among objects as parallel composition.
   - Message passing as communication between processes.
   - Encapsulation in objects as restriction and relabeling.

   Also, we capture the fundamental aspects of real-time computing delayed processing and timeout handling, through the timeout operator of RtCCS.

2. An external action cannot be performed before its partner action in another process is ready to communicate. While a process having an external action is waiting for its partner action, it must perform $\surd$ actions. If a process has an executable communication (including τ), it must perform the communication immediately, instead of idling.

4 Timed Bisimilarity

In this section we present two equivalences over processes, which are extensions of CCS's bisimulation with the notion of time. The equivalences provide a formal framework for analyzing the behavior and timing of real-time object-oriented computations.

4.1 Timed Strong Equivalence

Definition 4 A binary relation $\mathcal{S} \subseteq \mathcal{P} \times \mathcal{P}$ is a *timed strong bisimulation* if $(P, Q) \in \mathcal{S}$ implies, for all $\mu \in Act_T$,

(i) $\forall P' \colon P \xrightarrow{\mu} P' \supset \exists Q' \colon Q \xrightarrow{\mu} Q' \wedge (P', Q') \in \mathcal{S}$.

(ii) $\forall Q' \colon Q \xrightarrow{\mu} Q' \supset \exists P' \colon P \xrightarrow{\mu} P' \wedge (P', Q') \in \mathcal{S}$.

We let "$\sim_T$" denote the largest timed strong bisimulation, and call P and Q *timed strongly equivalent* if $P \sim_T Q$. ∎

Intuitively, if P and Q are timed strongly equivalent, they seem indistinguishable from one another in behavior and timing. We show some algebraic properties of the equivalence:

Proposition 1

1. $P + Q \sim_T Q + P$
2. $P + (Q + R) \sim_T (P + Q) + R$
3. $P + P \sim_T P$
4. $P + \mathbf{0} \sim_T P$

Proposition 2

1. $P|Q \sim_T Q|P$
2. $P|(Q|R) \sim_T (P|Q)|R$
3. $P|\mathbf{0} \sim_T P$
4. $(P + Q) \setminus L \sim_T P \setminus L + Q \setminus L$
5. $(\alpha.P) \setminus L \sim_T \begin{cases} \mathbf{0} & \text{if } \alpha \in L \cup \bar{L} \\ \alpha.P \setminus L & \text{otherwise} \end{cases}$
6. $(P + Q)[f] \sim_T P[f] + Q[f]$
7. $(\alpha.P)[f] \sim_T f(\alpha).P[f]$

Proposition 3

1. $\langle \alpha.P, \alpha.P\rangle_t \sim_T \alpha.P$

2. $\langle \alpha.P, \langle \alpha.P, Q\rangle_t\rangle_s \sim_{\mathcal{T}} \langle \alpha.P, Q\rangle_{s+t}$
3. $\langle \tau.P + Q, R\rangle_t \sim_{\mathcal{T}} \tau.P + Q \qquad (t > 0)$
4. $\langle P, Q\rangle_t \setminus L \sim_{\mathcal{T}} \langle P \setminus L, Q \setminus L\rangle_t$
5. $\langle P, Q\rangle_t[f] \sim_{\mathcal{T}} \langle P[f], Q[f]\rangle_t$
6. $\langle P, Q + R\rangle_t \sim_{\mathcal{T}} \langle P, Q\rangle_t + \langle P, R\rangle_t$
7. $\langle P + Q, R\rangle_t \sim_{\mathcal{T}} \langle P, R\rangle_t + \langle Q, R\rangle_t$
8. $\langle\langle P, Q\rangle_s, R\rangle_t \sim_{\mathcal{T}} \begin{cases} \langle P, R\rangle_t & (s \geq t) \\ \langle P, \langle Q, R\rangle_{t-s}\rangle_s & (s < t) \end{cases}$

Proof. All the laws may be proved by exhibiting appropriate timed strong bisimulations. The hardest is (8) of **Proposition 3** in the case of $(s < t)$ which we prove as follows:

We need to show that $\mathcal{S}$ is a timed strong bisimulation, where $\{ (\langle\langle P_1, P_2\rangle_s, P_3\rangle_t, \langle P_1, \langle P_2, P_3\rangle_{t-s}\rangle_s) \mid 0 \leq s < t, P_1, P_2, P_3 \in \mathcal{P}\}$. So let $\langle\langle P_1, P_2\rangle_s, P_3\rangle_t \xrightarrow{\mu} Q_1$; it is enough to find Q_2 such that $\langle P_1, \langle P_2, P_3\rangle_{t-s}\rangle_s \xrightarrow{\mu} Q_2$ and $(Q_1, Q_2) \in \mathcal{S}$. There are two cases:

Case 1 $\mu \in Act$ and $P_1 \xrightarrow{\mu} P_1'$. By **TIME$_0$**, $Q_1 \equiv P_1'$, and $Q_2 \equiv P_1'$, and we know $Q_1 \equiv Q_2$.

Case 2 $\mu = \surd$.

Case 2.1 $s = 0$. We assume that $P_2 \xrightarrow{\surd} P_2'$. By **TIME$_2$**, $Q_1 \equiv \langle P_2', P_3\rangle_{t-1}$ and $Q_2 \equiv \langle P_2', P_3\rangle_{t-1}$, hence $Q_1 \equiv Q_2$.

Case 2.2 $s > 0$. We assume that $P_1 \xrightarrow{\surd} P_1'$. By **TIME$_1$**, $Q_1 \equiv \langle\langle P_1', P_2\rangle_{s-1}, P_3\rangle_{t-1}$, and $Q_2 \equiv \langle P_1', \langle P_2, P_3\rangle_{t-s}\rangle_{s-1}$, and clearly $(Q_1, Q_2) \in \mathcal{S}$.

By a symmetric argument, we complete the proof. ∎

RtCCS, like CCS, is based on the concept of interleaving and can reduce concurrent processes to sequential processes in terms of nondeterminism, for example, $\alpha.\mathbf{0}|\beta.\mathbf{0} \sim_{\mathcal{T}} \alpha.\beta.\mathbf{0} + \beta.\alpha.\mathbf{0}$.

Corollary 1 *The Expansion Laws*

1. Let $P \equiv \sum_{i\in I} \alpha_i.P_i$ and $Q \equiv \sum_{j\in J} \beta_j.Q_j$;

$$P|Q \sim_{\mathcal{T}} \sum_{i\in I} \alpha_i.(P_i|Q) + \sum_{j\in J} \beta_j.(P|Q_j) + \sum_{\alpha_i=\bar{\beta}_j} \tau.(P_i|Q_j)$$

2. Let $P \equiv \langle \sum_{i\in I} \alpha_i.P_i, P'\rangle_1$ and $Q \equiv \sum_{j\in J} \beta_j.Q_j$;

$$P|Q \sim_{\mathcal{T}} \langle \sum_{i\in I} \alpha_i.(P_i|Q) + \sum_{j\in J} \beta_j.(P|Q_j) + \sum_{\alpha_i=\bar{\beta}_j} \tau.(P_i|Q_j), P'|Q\rangle_1$$

3. Let $P \equiv \langle \sum_{i\in I} \alpha_i.P_i, P'\rangle_1$ and $Q \equiv \langle \sum_{j\in J} \beta_j.Q_j, Q'\rangle_1$;

$$P|Q \sim_{\mathcal{T}} \langle \sum_{i\in I} \alpha_i.(P_i|Q) + \sum_{j\in J} \beta_j.(P|Q_j) + \sum_{\alpha_i=\bar{\beta}_j} \tau.(P_i|Q_j), P'|Q'\rangle_1$$

From the laws in Proposition 1, 2, 3, and Corollary 1, we can develop an equational proof system based on timed strong equivalence. The system is given in appendix A.

Proposition 4 (Congruence)
Timed strong equivalence $\sim_{\mathcal{T}}$ is preserved by all operators.

Proof. We show a proof only for the timeout operator; the proofs for the other operations are similar to the proofs of [11]. We will prove that $\langle P_1, Q\rangle_t \sim_{\mathcal{T}} \langle P_2, Q\rangle_t$ where $P_1, P_2, Q \in \mathcal{P}$ and $P_1 \sim_{\mathcal{T}} P_2$. It is enough to show that $\mathcal{S} = \{(\langle P_1, Q\rangle_t, \langle P_2, Q\rangle_t) : P_1 \sim_{\mathcal{T}} P_2, t \geq 0\}$ is a timed strong bisimulation. We assume $\langle P_1, Q\rangle_t \xrightarrow{\mu} R$. There are two cases:

Case 1 $\mu \in Act$ and $P_1 \xrightarrow{\mu} P_1'$. Then because $P_1 \sim_{\mathcal{T}} P_2$, we have $\exists P_2' : P_2 \xrightarrow{\mu} P_2'$ with $P_1' \sim_{\mathcal{T}} P_2'$. Hence $\langle P_1, Q\rangle_t \xrightarrow{\mu} P_1'$, $\langle P_2, Q\rangle_t \xrightarrow{\mu} P_2'$, and clearly $P_1' \sim_{\mathcal{T}} P_2'$.

Case 2 $\mu = \surd$.

Case 2.1 $t = 0$. Obvious.

Case 2.2 $t > 0$, $P_1 \xrightarrow{\mu} P_1'$, and $\langle P_1, Q\rangle_t \xrightarrow{\mu} \langle P_1', Q\rangle_{t-1}$. Then because $P_1 \sim_{\mathcal{T}} P_2$, we have $\exists P_2' : P_2 \xrightarrow{\mu} P_2'$ with $P_1' \sim_{\mathcal{T}} P_2'$. Hence $\langle P_2, Q\rangle_t \xrightarrow{\mu} \langle P_2', Q\rangle_{t-1}$ and $(\langle P_1', Q\rangle_{t-1}, \langle P_2', Q\rangle_{t-1}) \in \mathcal{S}$.

By a symmetric argument, we complete the proof. The proof for the other case of the timeout operator, i.e. $\langle Q, P_1\rangle_t \sim_{\mathcal{T}} \langle Q, P_2\rangle_t$ where $P_1 \sim_{\mathcal{T}} P_2$, is similar. ∎

By this proposition we guarantee that if two objects are timed strongly equivalent, the objects are substitutable for each other.

4.2 Timed Observation Equivalence

The timed strong equivalence has several useful algebraic properties but gives no special status to the internal action τ which indeed should not be observed. We present a weaker equivalence which reflects the observation of behaviors and timing in the computation.

Definition 5 A binary relation $\mathcal{S} \subseteq \mathcal{P} \times \mathcal{P}$ is a *timed weak bisimulation* if $(P,Q) \in \mathcal{S}$ implies, for all $\mu \in Act_T$,

(i) $\forall P'\colon P \stackrel{\mu}{\Longrightarrow} P' \supset \exists Q'\colon Q \stackrel{\widehat{\mu}}{\Longrightarrow} Q' \wedge (P',Q') \in \mathcal{S}$.

(ii) $\forall Q'\colon Q \stackrel{\mu}{\Longrightarrow} Q' \supset \exists P'\colon P \stackrel{\widehat{\mu}}{\Longrightarrow} P' \wedge (P',Q') \in \mathcal{S}$.

Let "$\approx_T$" denote the largest timed weak bisimulation, and call P and Q *timed observation equivalent* if $P \approx_T Q$. ∎

Intuitively, if P and Q are timed observation equivalent, each action of P must be matched by a sequence of actions of Q with the same visible contents and timing, and conversely. The equivalence can equate objects that are not distinguishable by the observable behavior and the timing in the computations.

Proposition 5

1. $\tau.P \approx_T P$
2. $\alpha.\tau.P \approx_T \alpha.P$
3. $\tau.P + P \approx_T \tau.P$
4. $\alpha.(\tau.P + Q) \approx_T \alpha.(\tau.P + Q) + \alpha.Q$
5. $\langle \tau.P, Q\rangle_t \approx_T P \qquad (t > 0)$
6. $\langle P, \tau.Q\rangle_t \approx_T \langle P, Q\rangle_t$
7. $\langle \alpha.P, \tau.\langle \alpha.P, Q\rangle_s\rangle_t \approx_T \langle \alpha.P, Q\rangle_{s+t}$

Proof. Easy application of Definition 5. ∎

Like a weak equivalence in CCS, $\approx_T$ is not a congruence relation. However, we can define a timed observation congruence relation based on the timed observation equivalence.

Definition 6 P and Q are *timed observation congruent* if for all $\mu \in Act_T$

(i) $\forall P'\colon P \stackrel{\mu}{\Longrightarrow} P' \supset \exists Q'\colon Q \stackrel{\mu}{\Longrightarrow} Q' \wedge P' \approx_T Q'$.

(ii) $\forall Q'\colon Q \stackrel{\mu}{\Longrightarrow} Q' \supset \exists P'\colon P \stackrel{\mu}{\Longrightarrow} P' \wedge P' \approx_T Q'$.

(iii) $P \uparrow$ iff $Q \uparrow$.

We write $P =_T Q$ if P and Q are timed observation congruent. ∎

Proposition 6 If $P \approx_T Q$ and both processes are stable [2], then $P =_T Q$.

Proof. Direct from Definition 5 and 6. ∎

Proposition 7 For all $P,Q \in \mathcal{P}$, if $P \sim_T Q$ then $P =_T Q$, and if $P =_T Q$ then $P \approx_T Q$.

Proof. By Definition 4 to 6, clearly $=_T$ lies between $\sim_T$ and $\approx_T$. ∎

Intuitively, if $P \approx_T Q$ and both have no τ-derivative, then $P =_T Q$. Therefore, we can easily prove the following properties.

Proposition 8

1. $\alpha.\tau.P =_T \alpha.P$
2. $\tau.P + P =_T \tau.P$
3. $\alpha.(\tau.P + Q) =_T \alpha.(\tau.P + Q) + \alpha.Q$
4. $\langle P, \tau.Q\rangle_t =_T \langle P, Q\rangle_t \qquad (t > 0)$
5. $\langle \alpha.P, \tau.\langle \alpha.P, Q\rangle_s\rangle_t =_T \langle \alpha.P, Q\rangle_{s+t}$

Proof. Use Proposition 5 and 6. ∎

Proposition 9 (Congruence)
Timed observation congruence $=_T$ is preserved by all operators.

Proof. Same as Proposition 4. ∎

$=_T$ is a congruence relation and very close to timed observation equivalence. We grantee that if two objects are timed observation congruent, they are substitutable for each other even though their internal implementations are different.

[2] P is stable if $P \stackrel{\tau}{\longrightarrow} P'$ is impossible for all P'.

Remarks

It seems that we can always assume that if $P \sim Q$ (or $P \approx Q$) holds in CCS, then $P \sim_T Q$ (or $P \approx_T Q$) holds in RtCCS. There is however a counterexample: unguarded recursion expressions, e.g., **rec** $X : X$ expression. This is because in RtCCS such expressions are not allowed. However, we think that such expressions are unrealizable in object-oriented computing.

5 Examples

In this section we present two examples which illustrate the expressiveness and utility of RtCCS.

Example 2 *A Timer Object*

We describe a timer object T in RtCCS as follows:

$$T \stackrel{\text{def}}{=} start.\langle stop.T, \overline{timeout}.T\rangle_t$$

Upon reception of a request *start* to start the timer, it receives the amount t of $\surd$ and then sends $\overline{timeout}$. If it accepts a request *stop*, the timer goes back to its initial state T.

Example 3 *Verification of Timed Systems*

In order to illustrate how to describe and verify systems in RtCCS we describe the Alternating Bit Protocol for OSI data link layer. There are five components: the sender, the receiver, the two unreliable channels, and the timer. To find out duplicate messages or acknowledgments, messages and acknowledgements are sent tagged with bits 0 and 1 alternately.

- Upon reception of a request to send a message (*accept*), the sender sends it tagged with an appropriate bit b ($\overline{send_{S,b}}$) to the channel. It then waits for an appropriate acknowledgement ($ack_{R,b}$). If the acknowledgement cannot be received within a specified period of time, the sender retransmits the message.

- The two channels are unreliable. They may occasionally lose a message or an acknowledgement.

- The receiver waits for a message ($send_{R,b}$ or $send_{R,\widehat{b}}$, where $\widehat{b}$ is the complement of b) and then it sends an acknowledgement to the channel ($\overline{ack_{R,b}}$ or $\overline{ack_{R,\widehat{b}}}$). If it accepts a message tagged with an appropriate bit, it sends the message to the environment ($\overline{deliver}$).

The sender S has the following definition.

$$\begin{aligned}
S_b &\stackrel{\text{def}}{=} accept.S'_b \\
S'_b &\stackrel{\text{def}}{=} \overline{send_{S,b}}.\overline{start}.S''_b \\
S''_b &\stackrel{\text{def}}{=} ack_{S,b}.\overline{stop}.S_{\widehat{b}} + ack_{S,\widehat{b}}.S''_b + timeout.S'_b \\
S_{\widehat{b}} &\stackrel{\text{def}}{=} accept.S'_{\widehat{b}} \\
S'_{\widehat{b}} &\stackrel{\text{def}}{=} \overline{send_{S,\widehat{b}}}.\overline{start}.S''_{\widehat{b}} \\
S''_{\widehat{b}} &\stackrel{\text{def}}{=} ack_{S,\widehat{b}}.\overline{stop}.S_b + ack_{S,b}.S''_{\widehat{b}} + timeout.S'_{\widehat{b}}
\end{aligned}$$

The timer T has the following definition, where t is the timeout time.

$$T \stackrel{\text{def}}{=} start.\langle stop.T, \overline{timeout}.T\rangle_t$$

The channel C_1 has the following definition, where s is the transmission time for a message.

$$\begin{aligned}
C_1 \stackrel{\text{def}}{=}\ & send_{S,b}.\langle \mathbf{0}, \overline{send_{R,b}}.C_1 + \tau.C_1\rangle_s \\
& + send_{S,\widehat{b}}.\langle \mathbf{0}, \overline{send_{R,\widehat{b}}}.C_1 + \tau.C_1\rangle_s
\end{aligned}$$

The channel C_2 has the following definition, where a is the transmission time for an acknowledgement.

$$\begin{aligned}
C_2 \stackrel{\text{def}}{=}\ & ack_{R,b}.\langle \mathbf{0}, \overline{ack_{S,b}}.C_2 + \tau.C_2\rangle_a \\
& + ack_{R,\widehat{b}}.\langle \mathbf{0}, \overline{ack_{S,\widehat{b}}}.C_2 + \tau.C_2\rangle_a
\end{aligned}$$

The receiver R has the following definition.

$$\begin{aligned}
R_b \stackrel{\text{def}}{=}\ & send_{R,b}.\overline{deliver}.\overline{ack_{R,b}}.R_{\widehat{b}} \\
& + send_{R,\widehat{b}}.\overline{ack_{R,\widehat{b}}}.R_b \\
R_{\widehat{b}} \stackrel{\text{def}}{=}\ & send_{R,\widehat{b}}.\overline{deliver}.\overline{ack_{R,\widehat{b}}}.R_b \\
& + send_{R,b}.\overline{ack_{R,b}}.R_{\widehat{b}}
\end{aligned}$$

We present the behavior of the whole protocol under the assumption that $s + a < t$. The protocol is constructed by the parallel composition of

the sender, the receiver, the two channels, and the timer, as follows:

$$(S_b|T|C_1|C_2|R_b) \setminus L$$

where $L \stackrel{\text{def}}{=} \{send_{S,b}, send_{S,\widehat{b}}, send_{R,b}, send_{R,\widehat{b}}, ack_{S,b}, ack_{S,\widehat{b}}, ack_{R,b}, ack_{R,\widehat{b}}, start, stop, timeout\}$

By using timed bisimilarity, we transform $(S_b|T|C_1|C_2|R_b)\setminus L$ into a timed observation equivalent expression ABP, which is a sequential process:

$$(S_b|T|C_1|C_2|R_b) \setminus L \approx_{\mathcal{T}} ABP \quad where$$

$$\begin{aligned} ABP &\stackrel{\text{def}}{=} accept.ABP' \\ ABP' &\stackrel{\text{def}}{=} \tau.\langle \mathbf{0}, \overline{deliver}.\sum_{i\geq 0}\langle \mathbf{0}, ABP\rangle_{i\times t+a}\rangle_s \\ &\quad + \tau.\langle \mathbf{0}, ABP'\rangle_t \end{aligned}$$

We present the proof of this transformation in Appendix B.

By using the simpler expression, we can not only investigate the behavior of the protocol but also temporal properties, such as the minimal and maximal delay for delivering a message. With specifications based on RtCCS, we can derive the behavior and timing of whole systems from the description of its components. Timed bisimilarities allows us to analyze systems through timed and behaviorally equivalent processes which may have substantially less complexity in their structure.

6 Conclusion

We have defined a formal model called RtCCS for real-time object-oriented computations. RtCCS is an extension of Milner's CCS by introducing a tick action and a timeout operator. It can formally describe the aspects of time dependence in real-time systems and enjoy many pleasant properties of CCS. We have introduced two timed equivalences based on CCS's bisimulation: timed strong equivalence and timed observation equivalence. The equivalences provide a formal framework for analyzing the behavior and timing of real-time computations.

We have presented examples that demonstrate the expressiveness and the proof method of RtCCS. Furthermore, we have given a sound and complete equational proof system based on timed strong equivalence in appendix.

As an application of RtCCS, we are presently developing on the semantics of real-time object-oriented languages, such as DROL [21]. The semantics is defined by translation rules from the syntactical constructions of the languages into RtCCS process expressions, also used as a static verification model for real-time programs. Details are presented in [19].

We are also interested in investigating whether an axiomatization based on observation congruence [12] can be applied to RtCCS, and whether other process equivalences such as distributed bisimulation [3] and testing equivalence [4] can be adopted by RtCCS.

Acknowledgements

We are grateful to P. Wegner, and K. Takashio for useful discussions. We would like to thank A. Togashi, and B. Jonnson for suggestions at the early stage of this work. We thank V. Vasconcelos, T. Minohara, and K. Honda for comments on an earlier version of this paper.

References

[1] America, P., de Bakker, J., Kok, J., and Rutten, J., *Operational Semantics of a Parallel Object-Oriented Language*, In proceedings of ACM POPL, 1987.

[2] Beaten, J. C. M, and Bergstra, J. A, *Process Algebra*, Cambridge University Press 1990.

[3] Castellani, H., Hennessy, M., *Distributed Bisimulation*, Journal of ACM, Vol.36, No.4, 1989.

[4] de Nicola, R. and Hennessy, M., *Testing equivalence for processes*, Theoretical Computer Science, Vol.34, 1984.

[5] Hansson, H. and Jonsson, B., *A Calculus of Communicating Systems with Time and Probabilities*,

In proceedings of the 11th IEEE Real-Time Systems Symposium, 1990.

[6] Hennessy, M. and Regan, T., *A Temporal Process Algebra*, Technical Report 2/90, University of Sussex, 1990

[7] Hoare, C. A. R., *Communicating Sequential Processes*, Prentice Hall, 1985.

[8] Honda, K., and Tokoro, M., *An Object Calculus for Asynchronous Communication*, In proceedings of ECOOP'91, LNCS 512, 1991.

[9] Ishikawa, Y., Tokuda, H., and Mercer, C. W., *Object-Oriented Real-Time Language Design: Construction for Timing Constraints*, In proceedings of ECOOP/OOPSLA'90, 1990.

[10] Milner, R., *Calculi for Synchrony and Asynchrony*, Theoretical Computer Science, Vol.25, 1983.

[11] Milner, R., *Communication and Concurrency*, Prentice Hall, 1989.

[12] Milner, R., *A Complete Axiomatization for Observational Congruence of Finite Behavior*, Information and Computation, Vol.81, 1989.

[13] Milner, R., Parrow, J., and Walker, D., *A Calculus of Mobile Processes Part 1 & 2*, Technical report ECS-LFCS-89-85 & 86, University of Edinburgh, 1989.

[14] Moller, F., and Tofts, C., *A Temporal Calculus of Communicating Systems*, In proceedings of CONCUR'90, LNCS 458, 1990.

[15] Najm, E., and Stefani, J. B., *Object-Based Concurrency: A Process Calculus Analysis*, In proceedings of TAPSOFT'91, LNCS 493, 1991.

[16] Nicollin, X., and Sifakis, J., *The Algebra of Timed Process ATP: Theory and Applications*, IMAG Technical Report, RT-C26, 1990.

[17] Nierstrasz, O. M., and Papathomas, M., *Viewing Objects as Patterns of Communicating Agents*, In proceedings of ECOOP/OOPSLA'90, 1990.

[18] Nierstrasz, O. M., *Towards an Object Calculus*, In proceedings of ECOOP'91 Workshop on Concurrent Object-Based Concurrent Computing, LNCS 612, 1992.

[19] Satoh, I., and Tokoro, M., *A Formal Description and Verification for Parallel Computing with Timed Constraints*, In proceeding of Joint Symposium Parallel Processing'92, June, 1992. (in Japanese)

[20] Satoh, I., and Tokoro, M., *Timed Process Calculus Semantics for A Real-Timed Concurrent Object-Oriented Language*, Technical Report, Dept. Computer Science, Keio University, February, 1992.

[21] Takashio, K., and Tokoro, M., *DROL: An Object-Orinted Programming Language for Distributed Real-time Systems*, In proceedings of ACM OOPSLA'92, October, 1992.

[22] Walker, D., *π-Calculus Semantics of Object-Oriented Programming Languages*, In proceedings of Theoretical Aspects of Computer Software, LNCS 526, 1991.

[23] Yi, W., *CCS + Time = an Interleaving Model for Real Time Systems*, In proceedings of Automata, Languages and Programming'91, LNCS 510, 1991.

[24] Yonezawa, A., and Tokoro, M., editors, *Object-Oriented Concurrent Programming*, MIT Press, 1987.

Appendix A

In this appendix, we develop a sound and complete equational proof system based on timed strong equivalence. We first restrict our attention to the non-recursive finite process expressions. The equational system for RtCCS is defined in Figure 3. We shall use "=" to denote derivability in our equational system, and "$\equiv$" to denote syntactical identity.

It can be easily shown that the equational processes in Figure 3 are timed strongly equivalent. Therefore, we get the following result.

Theorem 1 (Soundness) If $P = Q$ where $P, Q \in \mathcal{P}$, then $P \sim_T Q$.

Proof. Obvious according to the propositions 1 to 3, and Corollary 1. ∎

Next, we want to show that the system is complete. We can accomplish this by using the following normal forms.

Definition 7 A *normal form* ($\mathcal{NF}$) is defined inductively by:

(i) A process is in normal form if it is in $\mathcal{NF}_\Sigma$ or $\mathcal{NF}_{\langle\rangle}$.

(ii) A process is in $\mathcal{NF}_\Sigma$ if it is of the form $\sum_{0 \le i \le n} \alpha_i.P_i$ where each P_i is in $\mathcal{NF}$.

(iii) A process is in $\mathcal{NF}_{\langle\rangle}$ if it is of the form $\langle P, Q \rangle_1$ where P is in $\mathcal{NF}_\Sigma$ and Q is in $\mathcal{NF}$. ∎

$$
\begin{array}{rclrcl}
P+Q & = & Q+P & P+(Q+R) & = & (P+Q)+R \\
P+P & = & P & P+\mathbf{0} & = & P \\
(P+Q)\setminus L & = & P\setminus L+Q\setminus L & (\alpha.P)\setminus L & = & \begin{cases} \mathbf{0} & \text{if } \alpha\in L\cup\bar{L} \\ \alpha.P\setminus L & \text{otherwise} \end{cases} \\
(P+Q)[f] & = & P[f]+Q[f] & (\alpha.P)[f] & = & f(\alpha).P[f] \\
\langle \alpha.P,\alpha.P\rangle_t & = & \alpha.P & \langle \tau.P+Q,R\rangle_t & = & \tau.P+Q \quad (t>0) \\
\langle P,Q\rangle_0 & = & Q & \langle\langle P,Q\rangle_s,R\rangle_t & = & \begin{cases} \langle P,R\rangle_t & (s\geq t) \\ \langle P,\langle Q,R\rangle_{t-s}\rangle_s & (s<t) \end{cases} \\
\langle P,Q+R\rangle_t & = & \langle P,Q\rangle_t+\langle P,R\rangle_t & \langle P+Q,R\rangle_t & = & \langle P,R\rangle_t+\langle Q,R\rangle_t \\
\langle P,Q\rangle_t[f] & = & \langle P[f],Q[f]\rangle_t & \langle P,R\rangle_t\setminus L & = & \langle P\setminus L,Q\setminus L\rangle_t \\
P|Q & = & Q|P & P|(Q|R) & = & (P|Q)|R \\
P|\mathbf{0} & = & P & & & \\
P|Q & = & \multicolumn{4}{l}{\textstyle\sum_{\alpha_i=\bar{\beta}_j}\tau.(P_i|Q_j)+\sum_{i\in I}\alpha_i.(P_i|Q)+\sum_{j\in J}\beta_j.(P|Q_j)} \\
 & & \multicolumn{4}{l}{\text{where } P\equiv\textstyle\sum_{i\in I}\alpha_i.P_i \text{ and } Q\equiv\sum_{j\in J}\beta_j.Q_j} \\
P|Q & = & \multicolumn{4}{l}{\langle\textstyle\sum_{\alpha_i=\bar{\beta}_j}\tau.(P_i|Q_j)+\sum_{i\in I}\alpha_i.(P_i|Q)+\sum_{j\in J}\beta_j.(P|Q_j)\,,\,P'|Q\rangle_1} \\
 & & \multicolumn{4}{l}{\text{where } P\equiv\langle\textstyle\sum_{i\in I}\alpha_i.P_i,P'\rangle_1 \text{ and } Q\equiv\sum_{j\in J}\beta_j.Q_j} \\
P|Q & = & \multicolumn{4}{l}{\langle\textstyle\sum_{\alpha_i=\bar{\beta}_j}\tau.(P_i|Q_j)+\sum_{i\in I}\alpha_i.(P_i|Q)+\sum_{j\in J}\beta_j.(P|Q_j)\,,\,P'|Q'\rangle_1} \\
 & & \multicolumn{4}{l}{\text{where } P\equiv\langle\textstyle\sum_{i\in I}\alpha_i.P_i,P'\rangle_1 \text{ and } Q\equiv\langle\sum_{j\in J}\beta_j.Q_j,Q'\rangle_1}
\end{array}
$$

Figure 3: Equational Proof System based on $\sim_{\mathcal{T}}$

Proposition 10 For each $P\in\mathcal{P}$, there exists a normal form P' such $P=P'$.

Proof. By structural induction on P. ∎

Lemma 1

1. If $P\sim_{\mathcal{T}}Q$ where P is in $\mathcal{NF}_{\Sigma}$ and Q is in $\mathcal{NF}$, then Q is in $\mathcal{NF}_{\Sigma}$.
2. If $P\sim_{\mathcal{T}}Q$ where P is in $\mathcal{NF}_{\langle\rangle}$ and Q is in $\mathcal{NF}$, then Q is in $\mathcal{NF}_{\langle\rangle}$.

Proof. The structure of $\mathcal{NF}_{\Sigma}$ processes cannot be transformed by $\surd$ and any $\mathcal{NF}_{\langle\rangle}$ process transforms its structure by $\surd$. ∎

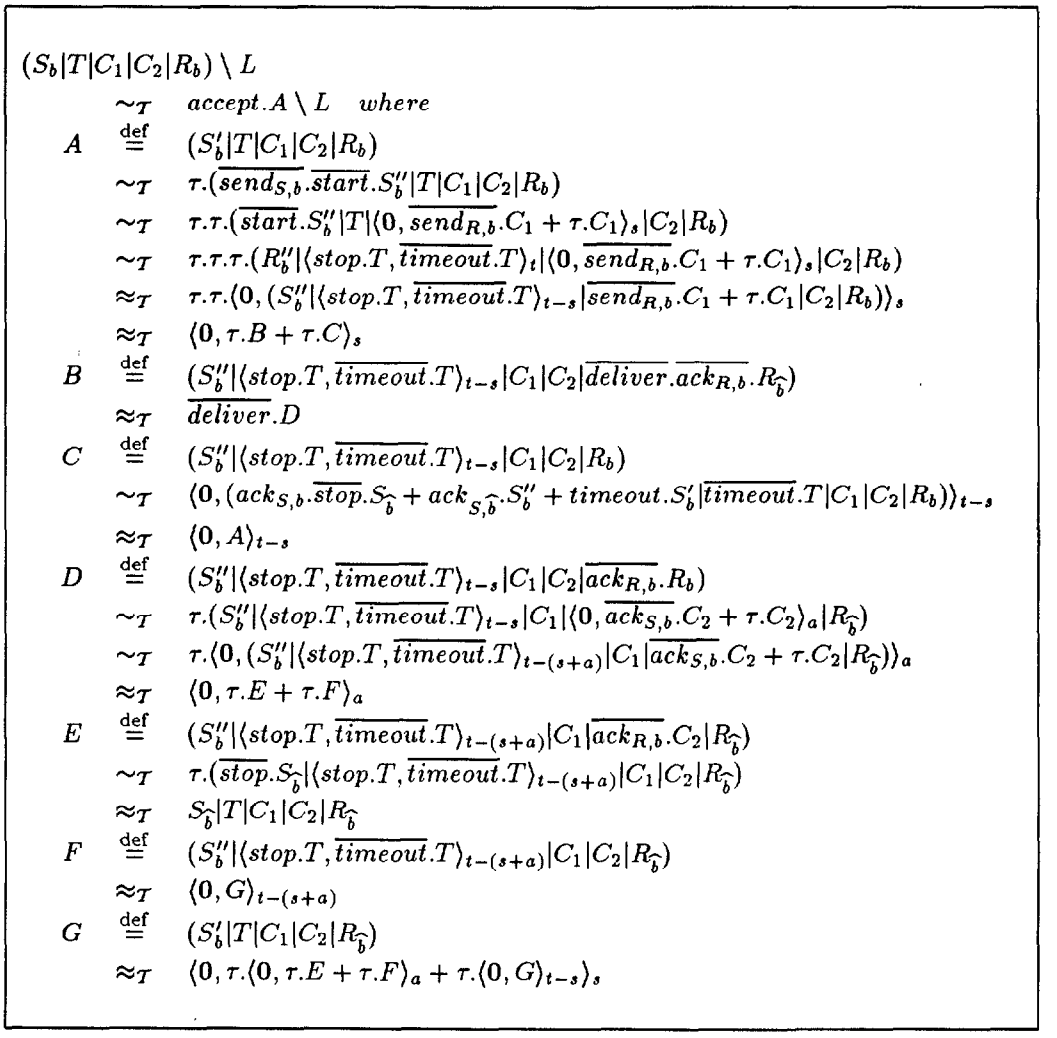
Theorem 2 (Completeness) If $P\sim_{\mathcal{T}}Q$ where $P,Q\in\mathcal{P}$, then $P=Q$.

Proof. Assume that $P\sim_{\mathcal{T}}Q$. By proposition 10, we can find normal form processes P' and Q' where $P=P'$ and $Q=Q'$. By lemma 1, both P' and Q' must be in $\mathcal{NF}_{\Sigma}$ or both P' and Q' must be in $\mathcal{NF}_{\langle\rangle}$. Hence we have that $P=Q$. ∎

Appendix B

In this appendix we present a rather detailed description of the behavior and timing of the Alternating Bit Protocol as given in section 5 under the assumption that $s+a<t$. The whole protocol is represented as a parallel composition of the five component as follows:

$$(S_b|T|C_1|C_2|R_b)\setminus L$$



$$
\begin{array}{lll}
(S_b|T|C_1|C_2|R_b)\setminus L & & \\
 & \sim_T & accept.A\setminus L \quad where \\
A & \stackrel{\text{def}}{=} & (S'_b|T|C_1|C_2|R_b) \\
 & \sim_T & \tau.(\overline{send_{S,b}}.\overline{start}.S''_b|T|C_1|C_2|R_b) \\
 & \sim_T & \tau.\tau.(\overline{start}.S''_b|T|\langle \mathbf{0}, \overline{send_{R,b}}.C_1+\tau.C_1\rangle_s|C_2|R_b) \\
 & \sim_T & \tau.\tau.\tau.(R''_b|\langle stop.T, \overline{timeout}.T\rangle_t|\langle \mathbf{0}, \overline{send_{R,b}}.C_1+\tau.C_1\rangle_s|C_2|R_b) \\
 & \approx_T & \tau.\tau.\langle \mathbf{0}, (S''_b|\langle stop.T, \overline{timeout}.T\rangle_{t-s}|\overline{send_{R,b}}.C_1+\tau.C_1|C_2|R_b)\rangle_s \\
 & \approx_T & \langle \mathbf{0}, \tau.B+\tau.C\rangle_s \\
B & \stackrel{\text{def}}{=} & (S''_b|\langle stop.T, \overline{timeout}.T\rangle_{t-s}|C_1|C_2|\overline{deliver}.\overline{ack_{R,b}}.R_{\widehat{b}}) \\
 & \approx_T & \overline{deliver}.D \\
C & \stackrel{\text{def}}{=} & (S''_b|\langle stop.T, \overline{timeout}.T\rangle_{t-s}|C_1|C_2|R_b) \\
 & \sim_T & \langle \mathbf{0}, (ack_{S,b}.\overline{stop}.S_{\widehat{b}}+ack_{S,\widehat{b}}.S''_b+timeout.S'_b|\overline{timeout}.T|C_1|C_2|R_b)\rangle_{t-s} \\
 & \approx_T & \langle \mathbf{0}, A\rangle_{t-s} \\
D & \stackrel{\text{def}}{=} & (S''_b|\langle stop.T, \overline{timeout}.T\rangle_{t-s}|C_1|C_2|\overline{ack_{R,b}}.R_b) \\
 & \sim_T & \tau.(S''_b|\langle stop.T, \overline{timeout}.T\rangle_{t-s}|C_1|\langle \mathbf{0}, \overline{ack_{S,b}}.C_2+\tau.C_2\rangle_a|R_{\widehat{b}}) \\
 & \sim_T & \tau.\langle \mathbf{0}, (S''_b|\langle stop.T, \overline{timeout}.T\rangle_{t-(s+a)}|C_1|\overline{ack_{S,b}}.C_2+\tau.C_2|R_{\widehat{b}})\rangle_a \\
 & \approx_T & \langle \mathbf{0}, \tau.E+\tau.F\rangle_a \\
E & \stackrel{\text{def}}{=} & (S''_b|\langle stop.T, \overline{timeout}.T\rangle_{t-(s+a)}|C_1|\overline{ack_{R,b}}.C_2|R_{\widehat{b}}) \\
 & \sim_T & \tau.(\overline{stop}.S_{\widehat{b}}|\langle stop.T, \overline{timeout}.T\rangle_{t-(s+a)}|C_1|C_2|R_{\widehat{b}}) \\
 & \approx_T & S_{\widehat{b}}|T|C_1|C_2|R_{\widehat{b}} \\
F & \stackrel{\text{def}}{=} & (S''_b|\langle stop.T, \overline{timeout}.T\rangle_{t-(s+a)}|C_1|C_2|R_{\widehat{b}}) \\
 & \approx_T & \langle \mathbf{0}, G\rangle_{t-(s+a)} \\
G & \stackrel{\text{def}}{=} & (S'_b|T|C_1|C_2|R_{\widehat{b}}) \\
 & \approx_T & \langle \mathbf{0}, \tau.\langle \mathbf{0}, \tau.E+\tau.F\rangle_a+\tau.\langle \mathbf{0}, G\rangle_{t-s}\rangle_s
\end{array}
$$

Figure 4: Expansion of $(S_b|T|C_1|C_2|R_b)\setminus L$

where $L \stackrel{\text{def}}{=} \{send_{S,b}, send_{R,b}, send_{S,\widehat{b}}, send_{R,\widehat{b}}, ack_{S,b}, ack_{R,b}, ack_{S,\widehat{b}}, ack_{R,\widehat{b}}, start, stop, timeout\}$.

Using the timed strong equivalence and timed observation equivalence, we derive an expansion of $(S_b|T|C_1|C_2|R_b)\setminus L$ as shown in Figure 4. Expression A follows directly from Corollary 1, while $A \approx_T \langle \mathbf{0}, \tau.B+\tau.C\rangle_s$ is derived from Proposition 5. Derivations of expressions B to G are similar to that of A. From Figure 4, we can transform $(S_b|T|C_1|C_2|R_b)\setminus L$ into a timed observation equivalent expression P as follows:

$$(S_b|T|C_1|C_2|R_b)\setminus L \quad \approx_T \quad P \quad \text{where}$$

$$
\begin{array}{lll}
P & \stackrel{\text{def}}{=} & accept.A \\
A & \stackrel{\text{def}}{=} & \langle \mathbf{0}, \tau.\overline{deliver}.\langle \mathbf{0}, \tau.P+\tau.\langle \mathbf{0}, G\rangle_{t-(s+a)}\rangle_a \\
 & & \quad +\tau.\langle \mathbf{0}, A\rangle_{t-s}\rangle_s \\
G & \stackrel{\text{def}}{=} & \langle \mathbf{0}, \tau.\langle \mathbf{0}, \tau.A+\tau.\langle \mathbf{0}, G\rangle_{t-(s+a)}\rangle_a \\
 & & \quad +\tau.\langle \mathbf{0}, G\rangle_{t-s}\rangle_s
\end{array}
$$

Furthermore, since it can be easily shown that $P \approx_T ABP$ as given in section 5, we can conclude that ABP is timed observation equivalent to $(S_b|T|C_1|C_2|R_b)\setminus L$.