Congruence Properties for a Timed Extension of the π-Calculus

Hiroaki Kuwabara Shoji Yuen Kiyoshi Agusa
Graduate School of Information Science, Nagoya University,
1 Furo-cho, Chikusa-ku, Nagoya-shi, Aichi, 464-8603, Japan
{kuwabara,yuen,agusa}@agusa.i.is.nagoya-u.ac.jp

Abstract

We propose equivalences and preorders with congruence properties for a timed extension of the π-calculus. We present a timed extension of syntax and basic operational semantics to the π-calculus. The derived timed bisimulation relations are shown to be non-input congruent. The timed bisimilarities equalize the bisimilar processes not only in actions but also in timing of the actions. For the purpose of modeling hard deadlines, we propose a more relaxed bisimulation, called delay time order *that relates a process behaves 'faster' in action to a process with the same communication capability. We show that the delay time orders are non-input congruent as well where the 'time-insensitive' composition is allowed. We illustrate our timed extension by a simple example of a streaming server and a player client where the network configuration may change dynamically.*

1. Introduction

Nowadays real-time systems are quite common in society in accordance with the rapid growth of the computer systems and network. An important requirement of real-time systems is *to react in the right timing*. The system is required not only to respond to the environment in time but also to wait till the appropriate moment. A real-time system is usually built as the composition of components with timing constraints, where the whole system is formed by concurrently running programs due to the reactive nature of the components. This feature makes the system analysis difficult since each component behavior depends on other components. The issues of timing constraints make the situation even worse. In order for such (concurrent) real-time system to correctly work, the close analysis of the behavior is essential and very carefully chosen tests are necessary.

To deal with such a complex analysis of real-time systems, we propose a timed extension of the π-calculus. π-calculus[6, 7, 8] is known as a powerful abstract computation model for concurrent systems. The π-calculus provides the various techniques based on the algebraic properties. If the equivalences for a sub-component are preserved in a context, it is possible to ensure the modularity for the sub-module not to break the behavior of the whole system. Following the semantics provided by the calculus, it is expected to improve the reliability of the whole system.

In this paper, the derived bisimulation equivalences are shown to be a non-input congruence. The congruence property ensures that adding time-out operations does not change the system behavior as long as the length of time for time-out is fixed, or it must have the identical binding relation for the time parameter in time-out operation.

The derived bisimulations equalize processes that identically behave in passage of time. To see if a process being composed by the composition operation satisfies the deadline specification as a single process, it is useful to define a 'relatively faster' relation without changing the communication capability. For this purpose, we define a bisimular relations called *delay time orders* that are shown to be non-input congruences as well with the restriction of 'time-insensitive composition' for the delay time orders. We demonstrate our timed extension by a simple example of a streaming server and a player client where the network for streaming may change dynamically under a timing constraint for delivering video streams from the server to the client.

The structure of this paper is as follows. In section 2, we define the syntax and operational semantics of timed π-calculus. Then, in section 3, we define bisimulation relations for timed π-calculus and show these relations is congruence for the restricted contexts. We define new order relations in section 4, and describe an example of a streaming system using timed π-calculus and our order relation in section 5. Next, in section 6, we discuss related works. Finally, in section 7, we make some concluding remarks.

2. π-Calculus with Time

We introduce a primitive for passage of discrete time to describe quantitative properties related to time and model time-out using a choice operator. A process may wait for a certain amount of time at least until the timer prefix reaches to zero. Due to the maximal progress property that τ-transition is prior to the time passage, the time-out operation is expressed by the combination of τ-action and choice.

2.1. Syntax

We introduce a prefix t indexed by natural number to the π-calculus. $t[\underline{m}]$, called time-passing action, means time waiting for m time units. t is not a name.

We assume $\mathcal{N}ame$ is an infinite set of names and $\mathcal{I}$ is a set of names showing natural numbers. So, $\mathcal{I} = \{\underline{0}, \underline{1}, \ldots\} \subset \mathcal{N}ame$ and $\underline{i}$ is a name which means natural number i. We write $\mathcal{N} = \mathcal{N}ame - \mathcal{I}$, $\overline{\mathcal{N}} = \{\overline{x} | x \in \mathcal{N}\}$ and $\mathcal{L} = \mathcal{N} \cup \overline{\mathcal{N}}$. $\widetilde{x}$ shows a list of names and x_i is i-th name of $\widetilde{x}$. $\mathcal{P}$ represents a set of the whole process expression of timed π-calculus. In this paper, we assume $x \in \mathcal{N}$, $m \in \mathcal{I}$, $n \in \mathcal{N}ame$, k is a natural number, all elements of $\widetilde{y}$ is included in $\mathcal{N}$, and all elements of $\widetilde{z}$ in $\mathcal{N}ame$.

Definition 1 A process expression P is defined by the following BNF-like format:

$$\begin{array}{rcl} \pi & ::= & x(\widetilde{y}) \mid \overline{x}\langle\widetilde{z}\rangle \mid \tau \mid t[n] \\ P & ::= & M \mid P_1 | P_2 \mid \boldsymbol{\nu} x\, P \mid !\, P \\ M & ::= & \mathbf{0} \mid \pi.P \mid M_1 + M_2 \end{array}$$

□

We write Act for the set of actions π and $\overline{x}(\widetilde{z})$ which means sending fresh names $\widetilde{z}$ via x.

Definition 2 For any $P = t[m].P'$, if $m \in \mathcal{N}$ and m is not bound by any input action, P is *inactive*, written $P\!\uparrow$. If $P\!\uparrow$ then

$$(x(\widetilde{y}).P)\!\uparrow, (\overline{x}\langle\widetilde{z}\rangle.P)\!\uparrow, (\tau.P)\!\uparrow, (t[n].P)\!\uparrow,$$
$$(P+Q)\!\uparrow, (P \mid Q)\!\uparrow, (\boldsymbol{\nu} z\, P)\!\uparrow, \text{ and } (!\, P)\!\uparrow$$

where $m \neq y_i$ for all i. When P is not inactive, P is *active*, written $P\!\downarrow$. □

If $P\!\uparrow$ holds, P cannot wait for time because time-passing action that is prefix of P or subprocess of P does not have natural number as argument. Activity of process is dynamically changed. For example, suppose $m \in \mathcal{N}$, $\overline{x}\langle m\rangle.\mathbf{0} \mid x(n).t[n].P$ is active since n in $t[n].P$ is bound by input action $x(n)$. However, this process evolves to $\mathbf{0} \mid t[m].P$ that is inactive.

Definition 3 In $x(z).P$ and $\boldsymbol{\nu} z\, P$, the scope of z is restricted in P. z is a *bound* name in this case. And a name is *free* if it is not bound. We write $\mathsf{fn}(P)$ for the set of names that are free in P, and $\mathsf{bn}(P)$ for bound in P. Suppose $n \in \mathcal{N}$, the set of free and bound names of an action α are defined as follow:

α	$x(y)$	$\overline{x}\langle z\rangle$	τ	$t[n]$
$\mathsf{fn}(\alpha)$	$\{x, y\}$	$\{x, z\}$	$\emptyset$	$\{n\}$
$\mathsf{bn}(\alpha)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

□

n of $t[n]$ is free name if $n \in \mathcal{N}$. We define substitution of names. Substitution to the elements of $\mathcal{I}$ is not permitted.

Definition 4 Substitution is a function from $\mathcal{N}$ to $\mathcal{N}ame$. Application of σ to P is recursively defined as follows:

$$\begin{array}{rcl} \mathbf{0}\sigma & \stackrel{def}{=} & \mathbf{0} \\ (x(\widetilde{y}).P)\sigma & \stackrel{def}{=} & x\sigma(\widetilde{y\sigma}).P\sigma \\ (\overline{x}\langle\widetilde{z}\rangle.P)\sigma & \stackrel{def}{=} & \overline{x\sigma}\langle\widetilde{z\sigma}\rangle.P\sigma \\ (\tau.P)\sigma & \stackrel{def}{=} & \tau.P\sigma \\ (t[n].P)\sigma & \stackrel{def}{=} & t[n\sigma].P\sigma \\ (P \mid Q)\sigma & \stackrel{def}{=} & P\sigma \mid Q\sigma \\ (P+Q)\sigma & \stackrel{def}{=} & P\sigma + Q\sigma \\ (\boldsymbol{\nu} x\, P)\sigma & \stackrel{def}{=} & \boldsymbol{\nu} x\, P\sigma \\ (!\, P)\sigma & \stackrel{def}{=} & !\, P\sigma \end{array}$$

where $x\sigma \in \mathcal{N}$. We assume the bound names are renamed appropriately to be different from the names of the substitutions. We also write $\{y_1, \ldots, y_n / x_1, \ldots, x_n\}$ for substitution σ. This means y_i is substituted for x_i. □

For example, $\overline{x}\langle\underline{5}\rangle.\mathbf{0} \mid x(n).t[n].P$ evolves to $\mathbf{0} \mid t[\underline{5}].P$. In this case, $\underline{5}$ is substituted for n.

We define structural congruence as follows.

Definition 5 Structural congruence, written $\equiv$, is the smallest congruence on processes containing $\equiv_\alpha$ and satisfying the following axioms:

$$\begin{array}{rcll} M_1 + (M_2 + M_3) & \equiv & (M_1 + M_2) + M_3 & \\ M_1 + M_2 & \equiv & M_2 + M_1 & \\ M_1 + \mathbf{0} & \equiv & M_1 & \\ P_1 \mid (P_2 \mid P_3) & \equiv & (P_1 \mid P_2) \mid P_3 & \\ P_1 \mid P_2 & \equiv & P_2 \mid P_1 & \\ P_1 \mid \mathbf{0} & \equiv & P_1 & \\ \boldsymbol{\nu} z\, \boldsymbol{\nu} w\, P & \equiv & \boldsymbol{\nu} w\, \boldsymbol{\nu} z\, P & \\ \boldsymbol{\nu} z\, \mathbf{0} & \equiv & \mathbf{0} & \\ \boldsymbol{\nu} z\, (P \mid Q) & \equiv & P \mid \boldsymbol{\nu} z\, Q & z \notin \mathsf{fn}(P) \\ !\, P & \equiv & P \mid !\, P & \end{array}$$

□

The time-passing action $t[\underline{i}]$ means to wait for i time units. For example, the process $t[\underline{10}].P$ evolves to P after 10 time units. The timeout is described using the time-passing action, non-deterministic choice and τ action, for example $a.P + t[\underline{5}].\tau.Q$. This process waits a for 5 time units at most before performing action a. If this process performs a within 4 time units then evolves to P. After 5 time units, if this process is able to perform a then it evolves to P or Q non-deterministically, otherwise becomes Q.

The intuitive meaning of other prefixes and operators are shown follows:

- (Input) $x(\widetilde{y})$: receive $\widetilde{y}$ via x.
- (Output) $\overline{x}\langle\widetilde{z}\rangle$: send $\widetilde{z}$ via x.
- (τ action) τ : internal action.
- (Choice) $P + Q$: choose and perform executable process P or Q. If both processes are executable choose one process non-deterministically.
- (Parallel composition) $P|Q$: P and Q run concurrently.
- (Restriction) $\boldsymbol{\nu} x\, P$: the scope of the name x is restricted to P.
- (Replication) $!\,P$: infinite parallel composition of P.

2.2. Operational Semantics by Labelled Transition

We define the labelled transition relation on $\mathcal{P}$ by the rules in Figure 1 for actions and the rules in Figure 2 for time-passing. $P\not\Downarrow$ and $Q\not\Downarrow$ hold in these rules except for ABORT. All elements appearing in $\widetilde{w}$ are included in $\mathcal{N}ame$.

$P \xrightarrow{\alpha} P'$ means that P evolves to P' by input/output when $\alpha \neq \tau$ or internal action when $\alpha = \tau$. $P \rightsquigarrow P'$ means P evolves to P' by time passing of 1 time unit. The $\mathrm{PAR_T}$ and $\mathrm{REP_T}$ rule show that τ transition is occurred prior to time passing.

3. Timed Bisimilarity

We define the bisimulation for the timed π-calculus. These relations are supposed to satisfy following properties:

- The sequence of input/output action of both processes is the same, and
- The timing that the process waits time and the length of time waiting are same.

Definition 6 *Timed strong bisimilarity* is the largest symmetric relation, $\sim_{\mathcal{T}}$, such that whenever $P \sim_{\mathcal{T}} Q$,

- $P\!\uparrow$ implies $Q\!\uparrow$,

$$\textsc{out}: \frac{}{\overline{x}\langle\widetilde{y}\rangle.P \xrightarrow{\overline{x}\langle\widetilde{y}\rangle} P} \qquad \textsc{input}: \frac{}{x(\widetilde{y}).P \xrightarrow{x(\widetilde{z})} P\{\widetilde{z}/\widetilde{y}\}} \qquad \textsc{tau}: \frac{}{\tau.P \xrightarrow{\tau} P}$$

$$\textsc{sum-l}: \frac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'} \qquad \textsc{par-l}: \frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}\ \mathsf{bn}(\alpha)\cap\mathsf{fn}(Q)=\emptyset$$

$$\textsc{sum-r}: \frac{Q \xrightarrow{\alpha} Q'}{P+Q \xrightarrow{\alpha} Q'} \qquad \textsc{par-r}: \frac{Q \xrightarrow{\alpha} Q'}{P \mid Q \xrightarrow{\alpha} P \mid Q'}\ \mathsf{bn}(\alpha)\cap\mathsf{fn}(P)=\emptyset$$

$$\textsc{comm-l}: \frac{P \xrightarrow{\overline{x}\langle\widetilde{y}\rangle} P' \quad Q \xrightarrow{x(\widetilde{y})} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \qquad \textsc{close-l}: \frac{P \xrightarrow{\overline{x}(\widetilde{y})} P' \quad Q \xrightarrow{x(\widetilde{y})} Q'}{P \mid Q \xrightarrow{\tau} \boldsymbol{\nu} z\,(P' \mid Q')}\ z \notin \mathsf{fn}(Q)$$

$$\textsc{comm-r}: \frac{P \xrightarrow{x(\widetilde{y})} P' \quad Q \xrightarrow{\overline{x}\langle\widetilde{y}\rangle} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \qquad \textsc{close-r}: \frac{P \xrightarrow{x(\widetilde{y})} P' \quad Q \xrightarrow{\overline{x}(\widetilde{y})} Q'}{P \mid Q \xrightarrow{\tau} \boldsymbol{\nu} z\,(P' \mid Q')}\ z \notin \mathsf{fn}(Q)$$

$$\textsc{res}: \frac{P \xrightarrow{\alpha} P'}{\boldsymbol{\nu} x\, P \xrightarrow{\alpha} \boldsymbol{\nu} x\, P'}\ \text{if } \alpha \notin \{x, \overline{x}\} \qquad \textsc{open}: \frac{P \xrightarrow{\overline{x}\langle z\rangle} P'}{\boldsymbol{\nu} z\, P \xrightarrow{\overline{x}\langle z\rangle} P'}\ z \neq x$$

$$\textsc{rep-act}: \frac{P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P' \mid !P} \qquad \textsc{rep-comm}: \frac{P \xrightarrow{\overline{x}\langle\widetilde{y}\rangle} P' \quad P \xrightarrow{x(\widetilde{y})} P''}{!P \xrightarrow{\tau} (P' \mid P'') \mid !P}$$

$$\textsc{rep-close}: \frac{P \xrightarrow{\overline{x}(\widetilde{y})} P' \quad P \xrightarrow{x(\widetilde{y})} P''}{!P \xrightarrow{\tau} (\boldsymbol{\nu} z\,(P' \mid P'')) \mid !P}\ z \notin \mathsf{fn}(P)$$

$$\textsc{timeout}: \frac{P \xrightarrow{\alpha} P'}{t[\underline{0}].P \xrightarrow{\alpha} P'} \qquad \textsc{abort}: \frac{}{P \xrightarrow{abort}}\ \text{if } P\!\uparrow$$

Figure 1. The rules for actions

$$\mathrm{PASS_T}: \frac{}{t[\underline{n}].P \rightsquigarrow t[\underline{n-1}].P}\ \text{if } n>0 \qquad \mathrm{INACT_T}: \frac{}{\mathbf{0} \rightsquigarrow \mathbf{0}}$$

$$\mathrm{OUT_T}: \frac{}{\overline{x}\langle\widetilde{z}\rangle.P \rightsquigarrow \overline{x}\langle\widetilde{z}\rangle.P} \qquad \mathrm{IN_T}: \frac{}{x(\widetilde{y}).P \rightsquigarrow x(\widetilde{y}).P}$$

$$\mathrm{SUM_T}: \frac{P \rightsquigarrow P' \quad Q \rightsquigarrow Q'}{P+Q \rightsquigarrow P'+Q'} \qquad \mathrm{PAR_T}: \frac{P \rightsquigarrow P' \quad Q \rightsquigarrow Q'}{P \mid Q \rightsquigarrow P' \mid Q'}\ \text{if } P \mid Q \overset{\tau}{\not\rightarrow}$$

$$\mathrm{RES_T}: \frac{P \rightsquigarrow P'}{\boldsymbol{\nu} x\, P \rightsquigarrow \boldsymbol{\nu} x\, P'} \qquad \mathrm{REP_T}: \frac{P \rightsquigarrow P'}{!P \rightsquigarrow !P'}\ \text{if } P \mid P \overset{\tau}{\not\rightarrow}$$

$$\mathrm{STRUCT_T}: \frac{P \rightsquigarrow P'}{Q \rightsquigarrow Q'}\ \text{if } P \equiv Q,\ P' \equiv Q'$$

$$\mathrm{TIMEOUT_T}: \frac{P \rightsquigarrow P'}{t[\underline{0}].P \rightsquigarrow P'}$$

Figure 2. The rules for time passing

- $P \xrightarrow{\alpha} P'$ implies $Q \xrightarrow{\alpha} Q'$ for some Q' with $P' \sim_{\mathcal{T}} Q'$, and
- $P \rightsquigarrow P''$ implies $Q \rightsquigarrow Q''$ for some Q'' with $P'' \sim_{\mathcal{T}} Q''$. □

The timed strong bisimulation is a basis for equivalence relation between processes. If process P is timed strong bisimilar to process Q, P is not distinguishable from Q by the means of comparing their sequences of actions. The timed strong bisimulation treats internal action, visible action and time-passing action equally. For example, $t[\underline{1}].\overline{x}.\mathbf{0} \mid t[\underline{1}].y.\mathbf{0}$ is timed strong bisimilar to $t[\underline{1}].\overline{x}.y.\mathbf{0} + t[\underline{1}].y.\overline{x}.\mathbf{0}$. It is easy to check that $\mathcal{R}$ is a timed strong bisimulation, where

$$\begin{aligned}\mathcal{R} \;=\; \{&(t[\underline{1}].\overline{x}.\mathbf{0} \mid t[\underline{1}].y.\mathbf{0},\ t[\underline{1}].\overline{x}.y.\mathbf{0} + t[\underline{1}].y.\overline{x}.\mathbf{0}),\\ &(t[\underline{0}].\overline{x}.\mathbf{0} \mid t[\underline{0}].y.\mathbf{0},\ t[\underline{0}].\overline{x}.y.\mathbf{0} + t[\underline{0}].y.\overline{x}.\mathbf{0}),\\ &(\mathbf{0} \mid t[\underline{0}].y.\mathbf{0},\ y.\mathbf{0}),\\ &(t[\underline{0}].\overline{x}.\mathbf{0} \mid \mathbf{0},\ \overline{x}.\mathbf{0}),\end{aligned}$$

$$(\mathbf{0} \mid \mathbf{0},\ \mathbf{0})\}$$

The weak version of timed bisimilarity is defined by abstracting away τ-transitions.

Definition 7 Weak transition relations are defined as follows:

- $\longrightarrow_\tau$ is the reflexive and transitive closure of $\xrightarrow{\tau}$.
- $\xrightarrow{\alpha}_\tau$ is $\longrightarrow_\tau \xrightarrow{\alpha} \longrightarrow_\tau$ for any action α.
- $\rightsquigarrow_\tau$ is $\longrightarrow_\tau \rightsquigarrow \longrightarrow_\tau$. □

For convenience, we introduce the following notation: $\xrightarrow{\widehat{\tau}}_\tau$ is $\longrightarrow_\tau$, and $\xrightarrow{\widehat{\beta}}_\tau$ is $\xrightarrow{\beta}_\tau$ for any action β except τ.

Definition 8 *Timed weak bisimilarity* is the largest symmetric relation, $\approx_{\mathcal{T}}$, such that whenever $P \approx_{\mathcal{T}} Q$,

- $P\!\uparrow$ implies $Q\!\uparrow$,
- $P \xrightarrow{\alpha}_\tau P'$ implies $Q \xrightarrow{\widehat{\alpha}}_\tau Q'$ for some Q' with $P' \approx_{\mathcal{T}} Q'$, and
- $P \rightsquigarrow_\tau P''$ implies $Q \rightsquigarrow_\tau Q''$ for some Q'' with $P'' \approx_{\mathcal{T}} Q''$. □

Lemma 1 $\sim_{\mathcal{T}}$ and $\approx_{\mathcal{T}}$ are equivalence relations. □

We show $\sim_{\mathcal{T}}$ and $\approx_{\mathcal{T}}$ is preserved by all the operators except for input action.

Definition 9 A *non-input context* C is a context defined by following BNF-like format:

$$\begin{aligned} C[\cdot] \quad ::= \quad & [\cdot] \mid \overline{x}\langle\widetilde{z}\rangle.C[\cdot] \mid \tau.C[\cdot] \mid t[n].C[\cdot] \\ & \mid \overline{x}\langle\widetilde{z}\rangle.C[\cdot] + R \mid \tau.C[\cdot] + R \\ & \mid R + \overline{x}\langle\widetilde{z}\rangle.C[\cdot] \mid R + \tau.C[\cdot] \\ & \mid C[\cdot]|S \mid S|C[\cdot] \mid \boldsymbol{\nu} x\, C[\cdot] \mid !\,C[\cdot] \end{aligned}$$

where R is a guarded summation and S is an arbitrary process. □

Theorem 1 If $P \sim_{\mathcal{T}} Q$ then $C[P] \sim_{\mathcal{T}} C[Q]$ for any non-input context C.

proof : By induction on contexts. We prove the case $C[\cdot] = t[n].C'[\cdot]$. If n is included in $\mathcal{N}$ then $C[P]\!\uparrow$ and $C[Q]\!\uparrow$ hold. Suppose n is included in $\mathcal{I}$, written $\underline{n}$, and larger than 0. Then $C[P] \xrightarrow{\alpha}$ for any α. It is suffice to show that $t[\underline{n-1}].C'[P] \sim_{\mathcal{T}} t[\underline{n-1}].C'[Q]$. We show this by induction on n. If $n = 1$, $t[\underline{1}].C'[P] \rightsquigarrow t[\underline{0}].C'[P]$ and $t[\underline{1}].C'[Q] \rightsquigarrow t[\underline{0}].C'[Q]$. Now $t[\underline{1}].C'[P] \sim_{\mathcal{T}} t[\underline{1}].C'[Q]$ because $t[\underline{0}].C'[P] \sim_{\mathcal{T}} t[\underline{0}].C'[Q]$ by induction hypothesis on contexts and $t[\underline{0}].C'[P] \sim_{\mathcal{T}} C'[P]$. In the induction step, $t[\underline{n+1}].C'[P] \rightsquigarrow t[\underline{n}].C'[P]$ and $t[\underline{n+1}].C'[Q] \rightsquigarrow t[\underline{n}].C'[Q]$ with $t[\underline{n}].C'[P] \sim_{\mathcal{T}} t[\underline{n}].C'[Q]$ by induction hypothesis on n. So, $t[\underline{n+1}].C'[P] \sim_{\mathcal{T}} t[\underline{n+1}].C'[Q]$. □

Theorem 2 If $P \approx_{\mathcal{T}} Q$ then $C[P] \approx_{\mathcal{T}} C[Q]$ for any non-input context C.

proof : By induction on contexts. In the case $C[\cdot] = t[n].C'[\cdot]$, similar to theorem 1. □

The congruence for input prefix is not satisfied because the bisimilarity for any substitution is not hold. For example, if $m \in \mathcal{N}$, $t[m].\overline{a}.\mathbf{0}$ and $t[m].t[\underline{5}].\overline{a}.\mathbf{0}$ are bisimilar because both processes are inactive. However, $(t[m].\overline{a}.\mathbf{0})\{\underline{5}/m\}$ and $(t[m].t[\underline{5}].\overline{a}.\mathbf{0})\{\underline{5}/m\}$ are not bisimilar since the former needs 5 time units and the latter 10 to perform an action $\overline{a}$. We refine timed strong bisimulation to be preserved by input action[1].

Definition 10 P and Q are *timed strong full bisimilar*, $P \sim^c_{\mathcal{T}} Q$, if $P\sigma \sim_{\mathcal{T}} Q\sigma$ for every substitution σ. □

Lemma 2 Suppose $z \in \mathcal{N}$. If $P\{y/z\} \sim_{\mathcal{T}} Q\{y/z\}$ for any y included in $\mathsf{fn}(P) \cup \mathsf{fn}(Q) \cup \{z\} \cup \mathcal{I}$ then $x(z).P \sim_{\mathcal{T}} x(z).Q$. □

Theorem 3 $\sim^c_{\mathcal{T}}$ is a congruence.

proof: Suppose $P \sim^c_{\mathcal{T}} Q$. If $C \neq x(z).C'[\cdot]$ then $C[P] \sim^c_{\mathcal{T}} C[Q]$ since $\sim^c_{\mathcal{T}}$ is included in $\sim_{\mathcal{T}}$ and the theorem 1.

We show that $C[P] \sim^c_{\mathcal{T}} C[Q]$ by induction on contexts when $C = x(z).C'[\cdot]$. $C'[P] \sim^c_{\mathcal{T}} C'[Q]$ by induction hypothesis. By the definition 10, $C'[P]\sigma\{y/z\} \sim_{\mathcal{T}} C'[Q]\sigma\{y/z\}$ for any substitution σ and a name y included in $\mathsf{fn}(P) \cup \mathsf{fn}(Q) \cup \{z\} \cup \mathcal{I}$. $x(z).C'[P]\sigma \sim_{\mathcal{T}} x(z).C'[Q]\sigma$ by lemma 2. So $C[P] \sim^c_{\mathcal{T}} C[Q]$. □

If we want to show $x(z).P \sim_{\mathcal{T}} x(z).Q$ when $P \sim_{\mathcal{T}} Q$ holds, it is suffice to show that $P\{y/z\} \sim_{\mathcal{T}} Q\{y/z\}$ for any free names in P and Q and another name.

4. Delay Time Order Relation

Time constraints in the real-time systems are often given either by upper bound of time needed for some actions or by lower bound for delay time. In this section, we give an alternative order relation for 'relatively faster' processes based on the length of delay time where processes only differs in speed.

Definition 11 *Delay time order relation* is the largest relation, $\precsim_{\mathcal{T}}$, such that whenever $P \precsim_{\mathcal{T}} Q$,

$$\begin{aligned} P\!\uparrow \quad &\Longrightarrow \quad Q\!\uparrow \\ Q\!\uparrow \quad &\Longrightarrow \quad P\!\uparrow \\ P \xrightarrow{\alpha} P' \quad &\Longrightarrow \quad \exists Q'.\ Q \rightsquigarrow^* \xrightarrow{\alpha} Q' \wedge P' \precsim_{\mathcal{T}} Q' \\ P \rightsquigarrow P' \quad &\Longrightarrow \quad \exists Q'.\ Q \rightsquigarrow Q' \wedge P' \precsim_{\mathcal{T}} Q' \\ Q \xrightarrow{\alpha} Q' \quad &\Longrightarrow \quad \exists P'.\ P \xrightarrow{\alpha} P' \wedge P' \precsim_{\mathcal{T}} Q' \\ Q \rightsquigarrow Q' \quad &\Longrightarrow \quad P \precsim_{\mathcal{T}} Q' \vee \\ & \qquad\quad (\exists P'.\ P \rightsquigarrow P' \wedge P' \precsim_{\mathcal{T}} Q') \end{aligned}$$

1 Following [8] the definition of terms excludes the mixture of guarded sums and processes as in CCS. This makes the weak bisimilarity congruent for the choice operator.

□

If $P \precsim_{\mathcal{T}} Q$, both processes have equal transitions in the sense of the strong bisimulation, where P has less time-passing transitions than Q. When P can perform an action, Q either performs the same action or waits without losing the capability of the action. If P waits, Q must wait as well because P has less time-passing transitions than Q. By the same reason, if Q is able to execute some action, P must perform same action urgently. P may execute time-passing action if the relation between P and Q is hold, when Q performs time-passing action. Note that $\precsim_{\mathcal{T}}$ compares the relative length of time passage before an action is observed. The time passage after the action is counted for the following action.

Lemma 3 $P \rightsquigarrow^{n} P'$ implies $Q \rightsquigarrow^{n} Q'$ and $P' \precsim_{\mathcal{T}} Q'$ for some Q' if $P \precsim_{\mathcal{T}} Q$.

proof： By induction on n. □

Delay time order relation is preserved by performing any number of time-passing actions. We show $\precsim_{\mathcal{T}}$ is a preorder.

Lemma 4 $\precsim_{\mathcal{T}}$ is a preorder.

proof： Reflexivity is clearly satisfied by definition. We show transitivity is satisfied. Suppose $P \precsim_{\mathcal{T}} Q \precsim_{\mathcal{T}} R$. So, $P\!\uparrow \Rightarrow Q\!\uparrow \Rightarrow R\!\uparrow$ and $R\!\uparrow \Rightarrow Q\!\uparrow \Rightarrow P\!\uparrow$ clearly. Let $P \xrightarrow{\alpha} P'$, we have $Q \rightsquigarrow^{*} \xrightarrow{\alpha} Q'$ with $P' \precsim_{\mathcal{T}} Q'$ since $P \precsim_{\mathcal{T}} Q$. Now $Q \rightsquigarrow^{n} Q'' \xrightarrow{\alpha} Q'$ for some n and Q'', so $R \rightsquigarrow^{n} R''$ and $Q'' \precsim_{\mathcal{T}} R''$ for some R'' since $Q \precsim_{\mathcal{T}} R$ and lemma 3. And we have $R'' \rightsquigarrow^{*} \xrightarrow{\alpha} R'$ with $Q' \precsim_{\mathcal{T}} R'$ for any Q'. So $R \rightsquigarrow^{*} \xrightarrow{\alpha} R'$ and $Q' \precsim_{\mathcal{T}} R'$. Let $R \rightsquigarrow R'$, then $Q \precsim_{\mathcal{T}} R'$ or $\exists Q'.\ Q \rightsquigarrow Q' \wedge Q' \precsim_{\mathcal{T}} R'$ since $Q \precsim_{\mathcal{T}} R$. If $Q \precsim_{\mathcal{T}} R'$ then $P \precsim_{\mathcal{T}} Q \precsim_{\mathcal{T}} R'$, otherwise $P \precsim_{\mathcal{T}} Q'$ or $\exists P'.\ P \rightsquigarrow P' \wedge P' \precsim_{\mathcal{T}} Q'$ for any Q'. If $P \precsim_{\mathcal{T}} Q'$ then $P \precsim_{\mathcal{T}} Q' \precsim_{\mathcal{T}} R'$, otherwise $P' \precsim_{\mathcal{T}} Q' \precsim_{\mathcal{T}} R'$. □

We show that the delay time order relation is preserved by all the operators except input action as well as the timed bisimulation. There is a restriction on the processes composed by parallel operator.

Definition 12 A *non-input and time-insensitive context* C is a context defined by following BNF-like format:

$$\begin{array}{rcl} C[\cdot] & ::= & [\cdot] \mid \overline{x}\langle\widetilde{z}\rangle.C[\cdot] \mid \tau.C[\cdot] \mid t[n].C[\cdot] \\ & & \mid \overline{x}\langle\widetilde{z}\rangle.C[\cdot] + R \mid \tau.C[\cdot] + R \\ & & \mid R + \overline{x}\langle\widetilde{z}\rangle.C[\cdot] \mid R + \tau.C[\cdot] \\ & & \mid C[\cdot]|S \mid S|C[\cdot] \mid \boldsymbol{\nu} x\, C[\cdot] \mid !\,C[\cdot] \end{array}$$

where R is a guarded summation and S does not include time-passing actions $t[n]$. □

Theorem 4 If $P \precsim_{\mathcal{T}} Q$ then $C[P] \precsim_{\mathcal{T}} C[Q]$ for any non-input and time-insensitive context C.

proof： We prove by induction on C. In the case $C[\cdot] = t[n].C'[\cdot]$, similar to theorem 1. We show the case $C[\cdot] = C'[\cdot] \mid R$. Suppose $\mathcal{R} = \{(C'[P] \mid R, C'[Q] \mid R) \mid P \precsim_{\mathcal{T}} Q\}$. $R \not\!\!\downarrow$ holds since R does not include time-passing actions. By the same reason, $R \rightsquigarrow R$, so $R \rightsquigarrow^{*} R$.

If $C[P]\!\uparrow$ then $C'[P]\!\uparrow$. If $C'[P]\!\uparrow$ then $C'[Q]\!\uparrow$ by induction hypothesis, otherwise clearly $C[Q]\!\uparrow$. And vice versa.

- Suppose $C'[P] \mid R \xrightarrow{\alpha} P' \mid R$ where $C'[P] \xrightarrow{\alpha} P'$. We have $C'[Q] \rightsquigarrow^{*} Q^{*} \xrightarrow{\alpha} Q'$ and $P' \precsim_{\mathcal{T}} Q'$ for some Q' and Q^{*} by hypothesis. Hence $C'[Q] \mid R \rightsquigarrow^{*} Q^{*} \mid R \xrightarrow{\alpha} Q' \mid R$, so $(P' \mid R, Q' \mid R) \in \mathcal{R}$.
- Suppose $C'[P] \mid R \xrightarrow{\tau} P' \mid R'$ where $C'[P] \xrightarrow{\alpha} P'$ and $R \xrightarrow{\overline{\alpha}} R'$. We have $C'[Q] \mid R \rightsquigarrow^{*} Q^{*} \mid R \xrightarrow{\tau} Q' \mid R'$ by hypothesis, so $(P' \mid R', Q' \mid R') \in \mathcal{R}$.
- Suppose $C'[P] \mid R \xrightarrow{\alpha} C'[P] \mid R'$. We have $C'[Q] \mid R \xrightarrow{\alpha} C'[Q] \mid R'$, so $(C'[P] \mid R', C'[Q] \mid R') \in \mathcal{R}$.
- Suppose $C'[Q] \mid R \rightsquigarrow Q'' \mid R$ where $C'[Q] \rightsquigarrow Q''$. By the induction hypothesis, $C'[P] \precsim_{\mathcal{T}} Q''$ or $\exists P''.\ C'[P] \rightsquigarrow P'' \wedge P'' \precsim_{\mathcal{T}} Q''$. If $C'[P] \precsim_{\mathcal{T}} Q''$ then $(C'[P] \mid R, Q'' \mid R) \in \mathcal{R}$, otherwise $C'[P] \mid R \rightsquigarrow P'' \mid R$ hence $(P'' \mid R, Q'' \mid R) \in \mathcal{R}$.
- Similar when $C'[P] \mid R \rightsquigarrow P'' \mid R$, $C'[Q] \mid R \xrightarrow{\alpha} Q' \mid R$, $C'[Q] \mid R \xrightarrow{\tau} Q' \mid R'$, and $C'[Q] \mid R \xrightarrow{\alpha} C'[Q] \mid R'$.

So $\mathcal{R} \subseteq \precsim_{\mathcal{T}}$. □

The delay time order relation is violated in the composition of a process which includes time-passing actions and an arbitrary process. For example, let $P = a.\mathbf{0}, Q = t[1].a.\mathbf{0}, R = t[2].b.\mathbf{0}$. Now $P \precsim_{\mathcal{T}} Q$ but $P \mid R \not\precsim_{\mathcal{T}} Q \mid R$ since $P \mid R \xrightarrow{a} \mathbf{0} \mid R$ and $Q \mid R \rightsquigarrow \xrightarrow{a} \mathbf{0} \mid t[1].b.\mathbf{0}$. $Q \mid R$ needs more time-passing actions for next input or output or τ action than $P \mid R$ when $P \precsim_{\mathcal{T}} Q$. If R has time-passing actions, the length of waiting time of R for next non-time-passing action decreases. Composing a process which includes time-passing actions is not possible because the delay time order relation have to be preserved after any transition occurred.

If $P \precsim_{\mathcal{T}} Q$ then P and Q have same sequences of non-time-passing actions. These sequences include τ actions invisible to the environment. We give another delay time order relation which focuses on only input and output actions.

Definition 13 *Weak delay time order relation* is the largest relation, $\precapprox_{\mathcal{T}}$, such that whenever $P \precapprox_{\mathcal{T}} Q$,

$$\begin{array}{rcl} P\!\uparrow & \Longrightarrow & Q\!\uparrow \\ Q\!\uparrow & \Longrightarrow & P\!\uparrow \\ P \xrightarrow{\alpha}_{\tau} P' & \Longrightarrow & \exists Q'.\ Q \rightsquigarrow_{\tau}^{*} \xrightarrow{\widehat{\alpha}}_{\tau} Q' \wedge P' \precapprox_{\mathcal{T}} Q' \\ P \rightsquigarrow_{\tau} P' & \Longrightarrow & \exists Q'.\ Q \rightsquigarrow_{\tau} Q' \wedge P' \precapprox_{\mathcal{T}} Q' \\ Q \xrightarrow{\alpha}_{\tau} Q' & \Longrightarrow & \exists P'.\ P \xrightarrow{\widehat{\alpha}}_{\tau} P' \wedge P' \precapprox_{\mathcal{T}} Q' \end{array}$$

$$Q \rightsquigarrow_\tau Q' \implies P \precsim_{\mathcal{T}} Q' \vee (\exists P'.\ P \rightsquigarrow_\tau P' P' \precsim_{\mathcal{T}} Q')$$

□

Lemma 5 $\precsim_{\mathcal{T}}$ is a preorder. □

Theorem 5 If $P \precsim_{\mathcal{T}} Q$ then $C[P] \precsim_{\mathcal{T}} C[Q]$ for any non-input and time-insensitive context C.

proof： By the induction on contexts. In the case $C[\cdot] = t[n].C'[\cdot]$, similar to theorem 1. □

Weak delay time order relation is not preserved by input actions like the strong case. To show $P \precsim_{\mathcal{T}} Q$, it is suffice to show $P' \precsim_{\mathcal{T}} Q'$ where P' and Q' are obtained by omitting common parallel composite processes and prefixes except input from P and Q respectively since theorem 5.

5. Example

In this section, we describe a simple video streaming system. This system consists of a server that delivers video and a player that receives and plays video. A pair of routers connect server and player. The routers work alternatively in order to provide video frames to the player. The player attaches to one of the routers and gets video frames. If the player does not receive next video frame in time, the player switches to another router. This switching takes a given time to start delivering video frames after the player connects to the alternative router. The player must decompress received video frames before playing because every video frames are compressed.

The server $Server$ is described as follows:

$$Server \stackrel{def}{=} (\boldsymbol{\nu} g)\,(\overline{g} \mid !\,(g.\overline{v}\langle video\rangle.\overline{g})).$$

$Server$ keeps sending video via name v. A name $video$ means one video frame. Every video frames differ for each other, but we represent them by an identical name $video$ for simplicity.

Two routers $Router_i$ are described as follows:

$$\begin{aligned} Router_1 &\stackrel{def}{=} !\,attach(l).(\boldsymbol{\nu} u, g) \\ &\quad (\overline{l}\langle \underline{2}, u, rel_1\rangle.v(video).\overline{u}\langle video\rangle.\overline{g} \\ &\quad \mid !\,(g.v(video).\overline{u}\langle video\rangle.\overline{g} + g.rel_1)). \\ Router_2 &\stackrel{def}{=} !\,attach(l).(\boldsymbol{\nu} u, g) \\ &\quad (\overline{l}\langle \underline{4}, u, rel_2\rangle.v(video).\overline{u}\langle video\rangle.\overline{g} \\ &\quad \mid !\,(g.v(video).\overline{u}\langle video\rangle.\overline{g} + g.rel_2)). \end{aligned}$$

$Router_i$ receives connection requests of player via $attach$ and sends a time that is needed to start delivering, a name u that is a channel for sending video frames, and a name rel_i that is used to disconnect player. After that, $Router_i$ receives a video frame from $Server$ via v and sends it to the player via new name u. When $Router_i$ fails to receive a video non-deterministically, $Router_i$ waits to be disconnected by the player. In this example, the length of time needed before delivering of $Router_1$ is 2 time units and $Router_2$ is 4.

The player $Player$ is described as follows:

$$\begin{aligned} Player \stackrel{def}{=}\ & (\boldsymbol{\nu} l, g, h) \\ & (\overline{attach}\langle l\rangle.\overline{h} \mid !\,h.l(n, v, r).t[n].\overline{g}\langle v, r\rangle \\ & \mid !\,g(v, r). \\ & \quad (v(video).t[\underline{1}].\overline{play}\langle video\rangle.\overline{g}\langle v, r\rangle \\ & \quad + t[\underline{1}].\tau.\overline{attach}\langle l\rangle.\overline{r}.\overline{h})). \end{aligned}$$

$Player$, first, requests to connect to one of the routers via $attach$ and receives via l (1) the time needed to start delivering video, (2) a name v for receiving video, and (3) r used to disconnect router. After that, $Player$ waits for the time amount received via l and waits for delivering video frames from connected $Router$. On receiving a video frame, $Player$ decompresses a compressed video frame for 1 time unit and plays it. If $Player$ cannot receive next video frame within 1 time unit, it requests to connect to another router via $attach$ again and disconnect attached router via r.

The whole system $System$ is described as follows:

$$\begin{aligned} System \stackrel{def}{=}\ & (\boldsymbol{\nu} attach, v, rel_1, rel_2) \\ & (Server \mid Router_1 \mid Router_2 \mid Player). \end{aligned}$$

We illustrate an execution trace.

$$\begin{aligned} & System \\ \xrightarrow{\tau}{}^{6} \rightsquigarrow^{2}\ & (\boldsymbol{\nu} attach, v, rel_1, rel_2, g_1, g_2, g_3, u, l, h) \\ & (\overline{v}\langle video\rangle.\overline{g_1} \mid !\,g_1.\overline{v}\langle video\rangle.\overline{g_1} \mid Router_1 \\ & \mid \overline{u}\langle video\rangle.\overline{g_2} \mid !\,(g_2.\cdots) \mid Router_2 \\ & \mid t[\underline{0}].\overline{g_3}\langle u, rel_1\rangle \mid !\,h.\cdots \mid !\,g_3(v, r).\cdots) \\ \xrightarrow{\tau}{}^{3} \rightsquigarrow\ & (\boldsymbol{\nu} attach, v, rel_1, rel_2, g_1, g_2, g_3, u, l, h) \\ & (\overline{v}\langle video\rangle.\overline{g_1} \mid !\,g_1.\overline{v}\langle video\rangle.\overline{g_1} \mid Router_1 \\ & \mid rel_1 \mid Router_2 \mid !\,h.\cdots \\ & \mid t[\underline{0}].\overline{play}\langle video\rangle.\overline{g_3}\langle u, rel_1\rangle \mid !\,g_3(v, r).\cdots) \\ \xrightarrow{\overline{play}} \xrightarrow{\tau}\ & (\boldsymbol{\nu} attach, v, rel_1, rel_2, g_1, g_2, g_3, u, l, h) \\ & (\overline{v}\langle video\rangle.\overline{g_1} \mid !\,g_1.\overline{v}\langle video\rangle.\overline{g_1} \mid Router_1 \\ & \mid rel_1 \mid Router_2 \mid !\,h.\cdots \\ & \mid (u(video).\cdots.\overline{g_3}\langle u, rel_1\rangle + t[1].\cdots.\overline{h}) \\ & \mid !\,g_3(v, r).\cdots) \\ \rightsquigarrow \xrightarrow{\tau}\ & (\boldsymbol{\nu} attach, v, rel_1, rel_2, g_1, g_2, g_3, u, l, h) \\ & (\overline{v}\langle video\rangle.\overline{g_1} \mid !\,g_1.\overline{v}\langle video\rangle.\overline{g_1} \mid Router_1 \\ & \mid rel_1 \mid Router_2 \mid !\,h.\cdots \\ & \mid \overline{attach}\langle l\rangle.\overline{rel_1}.\overline{h} \mid !\,g_3(v, r).\cdots) \\ \xrightarrow{\tau}{}^{6} \rightsquigarrow^{4}\ & (\boldsymbol{\nu} attach, v, rel_1, rel_2, g_1, g_2, g_3, u, l, h) \\ & (\overline{v}\langle video\rangle.\overline{g_1} \mid !\,g_1.\overline{v}\langle video\rangle.\overline{g_1} \mid Router_1 \\ & \mid \overline{u}\langle video\rangle.\overline{g_2} \mid !\,(g_2.\cdots) \mid Router_2 \\ & \mid t[\underline{0}].\overline{g_3}\langle u, rel_2\rangle \mid !\,h.\cdots \mid !\,g_3(v, r).\cdots) \end{aligned}$$

$$
\begin{array}{ll}
\xrightarrow{\tau}^{5}\rightsquigarrow & (\boldsymbol{\nu} attach, v, rel_1, rel_2, g_1, g_2, g_3, u, l, h) \\
& (\overline{v}\langle video\rangle.\overline{g_1} \mid !\, g_1.\overline{v}\langle video\rangle.\overline{g_1} \mid Router_1 \\
& \mid \overline{u}\langle video\rangle.\overline{g_2} \mid !\,(g_2.\cdots) \mid Router_2 \\
& \mid t[\underline{0}].\overline{play}\langle video\rangle.\overline{g_3}\langle u, rel_2\rangle \mid !\, h.\cdots \\
& \mid !\, g_3(v, r).\cdots) \\
\xrightarrow{\overline{play}} & \cdots
\end{array}
$$

This trace shows following behavior:

1. The player connected to $Router_1$, received video and played it.
2. Timeout was occurred while the player is waiting next video frame.
3. The player connected to $Router_2$ and got next video frame and played it.

The delay from playing a video frame represented by $\overline{play}$ to playing next frame is shortest when the player gets next frame through same router. In this case, 1 time unit delay is occurred. The longest delay is 6 time units when time-out is occurred and the player connects $Router_2$ instead of $Router_1$. These assertions are checked by showing follows:

$$(\boldsymbol{\nu} v)\,(Server \mid !\, v(video).t[\underline{1}].\overline{play}\langle video\rangle) \precsim_{\mathcal{T}} System$$

$$System \precsim_{\mathcal{T}} (\boldsymbol{\nu} v)\,(Server \mid !\, v(video).t[\underline{6}].\overline{play}\langle video\rangle)$$

By theorem 5, it is suffice to show

$$
\begin{array}{c}
!\, v(video).t[\underline{1}].\overline{play}\langle video\rangle \\
\precsim_{\mathcal{T}} \quad Router_1 \mid Router_2 \mid Player
\end{array}
$$

and

$$
\begin{array}{c}
Router_1 \mid Router_2 \mid Player \\
\precsim_{\mathcal{T}} \quad !\, v(video).t[\underline{6}].\overline{play}\langle video\rangle
\end{array}
$$

because $Server$ is 'time-insensitive'.

6. Related Work

Lee and Žic proposed the πRT-calculus[3] which is a real-time extension of the π-calculus. They introduce the timeout operator $\overset{t}{\triangleright}$ to the π-calculus. This calculus is similar to our timed π-calculus in respect of following features:

- Time is discrete.
- Actions and time passing are separated.
- Time is treated as name and transmitted on links.

The syntax and semantics of πRT-calculus are defined, however relations between processes, for instance bisimilarity, are not defined.

Berger and Honda extended π-calculus by introducing timers, message loss, sites and so on and described the two phase commit protocol[4]. The timer is a process $\mathsf{timer}^t(Q, R)$. The behavior is defined by the time stepper function and operational semantics. Applying the time stepper function to process means time passing. This calculus do not have an action meaning time passing. One time unit passes when a process perform an input or output or τ action.

Chen extended π-calculus with dense time[2]. He defines a weak bisimulation relation and constructs a complete proof system for weak congruence. This weak bisimulation is preserved by choice operator unlike our weak bisimulation. He investigates equivalence relations between processes, however dose not propose order relations for 'faster' processes.

In the context of CCS, the notion of 'faster' processes has been proposed[9, 10, 11]. Satoh proposed a prebisimulation for timed calculus. The orders by Hennessy-Kumar and Natarajan-Cleaveland count τ-transition, where smaller number of τs is faster. Our approach is following these previous literature extended for the π-calculus aiming at characterizing more dynamic software behavior.

7. Conclusion

In this paper, we proposed a timed extension of the π-calculus. The derived bisimilarities are shown to be a non-input congruence. The derived timed bisimulation equivalences characterize the equal behavior both in actions and in timing of the actions. Next, we defined the delay time order relations in order to relate processes only different in speed. If an implementation is proved to be faster than the specification, the implementation is proved to satisfy the deadline constraints following the classical way to prove the implementation[5]. In our timed extension, the time-out operation is modeled by the choice with the τ-prefix. The non-input congruence property ensures that adding the time-out operation does not break the delay time order provided the time-amount for time-out is fixed. In this respect, the weak delay time order relation is a useful foundation for proving the correctness in practical use. The weak delay time order relation is congruent for contexts whose composition is 'time-insensitive' in the sense that time-passing actions do not change the status of composing processes. This restriction is not generally a big obstacle in the server/client systems because a server generally is time-insensitive where it process the requests from clients at any time as shown by the example.

For the future work, besides the relative speed of processes, it is often useful to consider the relation where a process accepts an event for a longer time than the other. A process that accepts an event between 5 second and 15 second after the start-up is more 'generous' than a process that accepts the event between 7 second and 12 second after the start-up. The delay time order relation is violated when the

length of time either process waits for becomes shorter or longer than another process with progression. It is why this relation is not enough to use for verifying the real-time systems. We are interested in methods to verify using the testing equivalence[1] and so on.

References

[1] M. Boreale and R. D. Nicola. Testing Equivalence for Mobile Processes. *Information and Computation*, 120:279–303, 1995.

[2] J. Chen. A Proof System for Weak Congruence in Timed π Calculus. Technical report, Laboratoire d'Informatique Fondamentale d'Orl'eans, Universit'e d'Orl'eans, France, 2004.

[3] J. Y. Lee and J. Žic. On Modeling Real-time Mobile Processes. In *Proceedings of the twenty-fifth Australasian conference on Computer science*, pages 139–147. Australian Computer Society, Inc., 2002.

[4] M.Berger and K.Honda. The Two-Phase Commitment Protocol in an Extended π-Calculus. In *Preliminary Proceedings of EXPRESS '00*, 2000.

[5] R. Milner. *Communication and Concurrency*. Cambridge University Press, 1988.

[6] R. Milner. *Communication and Mobile Systems: the π-Calculus*. Cambridge University Press, 1999.

[7] R. Milner, J. Parrow, and D. Walker. A Calculus of mobile processes, Part I/II. *Information and Computation*, 100:1–77, 1992.

[8] D. Sangiorgi and D. Walker. *The π-calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.

[9] S.Arun-Kumar and M. Hennessy. An Efficiency Preorder for Processes. *Acta Informatica*, 29(8):737–760, 1992.

[10] I. Satoh and M. Tokoro. Asynchrony and Real-time in Distributed Computing. In *Proceedings of Parallel Symbolic Computing Workshop*, volume 748 of *LNCS*, pages 318–330. Springer, 1993.

[11] V.Natarajan and R. Cleaveland. An Algebraic Theory of Process Efficiency. In *LICS '96*, pages 63–72. IEEE Computer Society Press, 1996.

A. The Details of Trace in Example

In section 5, we illustrate an outline of execution trace of $System$ that is a simple video streaming system. We show the details of some steps of this trace in this section.

Firstly, communication via g in $Server$ occurs.

$System$
$\xrightarrow{\tau}$ $(\boldsymbol{\nu} attach, v, rel_1, rel_2, g_1)$
$(\overline{v}\langle video\rangle.\overline{g_1} \mid !\, g_1.\overline{v}\langle video\rangle.\overline{g_1}$
$\mid Router_1 \mid Router_2 \mid Player)$

$Router_1$ and $Player$ communicate via $attach$.
$\xrightarrow{\tau}$ $(\boldsymbol{\nu} attach, v, rel_1, rel_2, g_1, g_2, g_3, u, l, h)$
$(\overline{v}\langle video\rangle.\overline{g_1} \mid !\, g_1.\overline{v}\langle video\rangle.\overline{g_1}$
$\mid \overline{l}\langle \underline{2}, u, rel_1\rangle.v(video).\overline{u}\langle video\rangle.\overline{g_2}$
$\mid !(g_2.v(video).\overline{u}\langle video\rangle.\overline{g_2} + g_2.rel_1)$
$\mid Router_1 \mid Router_2$
$\mid \overline{h} \mid !\, h.l(n, v, r).t[n].\overline{g_3}\langle v, r\rangle$
$\mid !\, g_3(v, r).(v(video).t[\underline{1}].\overline{play}\langle video\rangle.\overline{g_3}\langle v, r\rangle$
$+t[\underline{1}].\tau.\overline{attach}\langle l\rangle.\overline{r}.\overline{h}))$

Communication via h occurs.
$\xrightarrow{\tau}$ $(\boldsymbol{\nu} attach, v, rel_1, rel_2, g_1, g_2, g_3, u, l, h)$
$(\overline{v}\langle video\rangle.\overline{g_1} \mid !\, g_1.\overline{v}\langle video\rangle.\overline{g_1}$
$\mid \overline{l}\langle \underline{2}, u, rel_1\rangle.v(video).\overline{u}\langle video\rangle.\overline{g_2}$
$\mid !(g_2.v(video).\overline{u}\langle video\rangle.\overline{g_2} + g_2.rel_1)$
$\mid Router_1 \mid Router_2$
$\mid l(n, v, r).t[n].\overline{g_3}\langle v, r\rangle \mid !\, h.l(n, v, r).t[n].\overline{g_3}\langle v, r\rangle$
$\mid !\, g_3(v, r).(v(video).t[\underline{1}].\overline{play}\langle video\rangle.\overline{g_3}\langle v, r\rangle$
$+t[\underline{1}].\tau.\overline{attach}\langle l\rangle.\overline{r}.\overline{h}))$

Communication via l occurs.
$\xrightarrow{\tau}$ $(\boldsymbol{\nu} attach, v, rel_1, rel_2, g_1, g_2, g_3, u, l, h)$
$(\overline{v}\langle video\rangle.\overline{g_1} \mid !\, g_1.\overline{v}\langle video\rangle.\overline{g_1}$
$\mid v(video).\overline{u}\langle video\rangle.\overline{g_2}$
$\mid !(g_2.v(video).\overline{u}\langle video\rangle.\overline{g_2} + g_2.rel_1)$
$\mid Router_1 \mid Router_2$
$\mid t[\underline{2}].\overline{g_3}\langle u, rel_1\rangle \mid !\, h.l(n, v, r).t[n].\overline{g_3}\langle v, r\rangle$
$\mid !\, g_3(v, r).(v(video).t[\underline{1}].\overline{play}\langle video\rangle.\overline{g_3}\langle v, r\rangle$
$+t[\underline{1}].\tau.\overline{attach}\langle l\rangle.\overline{r}.\overline{h}))$

Communication via v occurs.
$\xrightarrow{\tau}$ $(\boldsymbol{\nu} attach, v, rel_1, rel_2, g_1, g_2, g_3, u, l, h)$
$(\overline{g_1} \mid !\, g_1.\overline{v}\langle video\rangle.\overline{g_1}$
$\mid \overline{u}\langle video\rangle.\overline{g_2}$
$\mid !(g_2.v(video).\overline{u}\langle video\rangle.\overline{g_2} + g_2.rel_1)$
$\mid Router_1 \mid Router_2$
$\mid t[\underline{2}].\overline{g_3}\langle u, rel_1\rangle \mid !\, h.l(n, v, r).t[n].\overline{g_3}\langle v, r\rangle$
$\mid !\, g_3(v, r).(v(video).t[\underline{1}].\overline{play}\langle video\rangle.\overline{g_3}\langle v, r\rangle$
$+t[\underline{1}].\tau.\overline{attach}\langle l\rangle.\overline{r}.\overline{h}))$

Communication via g_1 occurs.
$\xrightarrow{\tau}$ $(\boldsymbol{\nu} attach, v, rel_1, rel_2, g_1, g_2, g_3, u, l, h)$
$(\overline{v}\langle video\rangle.\overline{g_1} \mid !\, g_1.\overline{v}\langle video\rangle.\overline{g_1}$
$\mid \overline{u}\langle video\rangle.\overline{g_2}$
$\mid !(g_2.v(video).\overline{u}\langle video\rangle.\overline{g_2} + g_2.rel_1)$
$\mid Router_1 \mid Router_2$
$\mid t[\underline{2}].\overline{g_3}\langle u, rel_1\rangle \mid !\, h.l(n, v, r).t[n].\overline{g_3}\langle v, r\rangle$
$\mid !\, g_3(v, r).(v(video).t[\underline{1}].\overline{play}\langle video\rangle.\overline{g_3}\langle v, r\rangle$
$+t[\underline{1}].\tau.\overline{attach}\langle l\rangle.\overline{r}.\overline{h}))$

Wait 2 time units.
$\rightsquigarrow^2$ $(\boldsymbol{\nu} attach, v, rel_1, rel_2, g_1, g_2, g_3, u, l, h)$
$(\overline{v}\langle video\rangle.\overline{g_1} \mid !\, g_1.\overline{v}\langle video\rangle.\overline{g_1}$
$\mid \overline{u}\langle video\rangle.\overline{g_2}$
$\mid !(g_2.v(video).\overline{u}\langle video\rangle.\overline{g_2} + g_2.rel_1)$
$\mid Router_1 \mid Router_2$
$\mid t[\underline{0}].\overline{g_3}\langle u, rel_1\rangle \mid !\, h.l(n, v, r).t[n].\overline{g_3}\langle v, r\rangle$
$\mid !\, g_3(v, r).(v(video).t[\underline{1}].\overline{play}\langle video\rangle.\overline{g_3}\langle v, r\rangle$
$+t[\underline{1}].\tau.\overline{attach}\langle l\rangle.\overline{r}.\overline{h}))$
$\xrightarrow{\tau}$ $\cdots$