プログラミング 6－8
(1996. 3. 26)

Predicative Verification of Real-time Communicating Processes

Kanako Shinohara Shoji Yuen Toshiki Sakabe Yasuyoshi Inagaki

Department of Information Engineering,
Nagoya University
Furo-cho, Chikusa-ku, Nagoya, 464-01, Japan
sinohara@sakabe.nuie.nagoya-u.ac.jp
{yuen,sakabe,inagaki}@nuie.nagoya-u.ac.jp

Abstract

We present a verification method of real-time communicating processes with the synchronous communication mechanism based on a predicative specification. The predicative specification is originally proposed by Hehner for asynchronous communicating processes[1], where a communicating process is translated into a first-order predicate formula.

The predicative specification is inherently "proof-oriented" in that the predicate calculus directly provides the framework for proof. To verify that a process P satisfies a specification S is reduced to the proof of $[\![P]\!] \Rightarrow S$ where $[\![P]\!]$ is a predicative specification of P where S is given as a first-order predicate formula.

In this paper, we propose a proof technique as a verification method where a specification is also given by a process. To verify that a program P should satisfy a specification Q, i.e., to prove $[\![P]\!] \Rightarrow [\![Q]\!]$, we decompose the proof into smaller parts using the fact that a specification formula derived from a process can be decomposed into the input part and the output part. For this propose, we define a "normalized" specification formula in which inputs and outputs are separated. Finally, we show a checking algorithm for our verification method following the structure of normalized formulas.

実時間通信プロセスの述語的仕様記述に対する検証法

篠原 加奈子 結縁 祥治 坂部 俊樹 稲垣 康善

名古屋大学 工学部 情報工学科
〒464-01 名古屋市千種区不老町

概 要

本稿では，Hehner によって提案されている通信プロセスの述語的意味論を，実時間制約をもつ通信プロセスの枠組に拡張し，述語的意味に基づく検証法を提案する．本稿における実時間通信プロセスは，同期的に通信メカニズムを持ち，通信ポートにおける通信は時間的な制約を受ける．

述語的意味論では，以下のように一階述語論理の形式的証明手法によってシステムの検証を行う．プロセス P の述語的意味を表す論理式を $[\![P]\!]$ とする．P に要求された仕様 S を一階述語論理の論理式で与えれば，P が S を満たすかどうかという検証は，$[\![P]\!] \Rightarrow S$ を一階述語論理の枠組で証明することに帰着できる．

本稿では，仕様もプロセス式で与えられる場合に注目する．実現プロセス式 P が仕様プロセス式 Q を満たすかどうかの検証、すなわち、$[\![P]\!] \Rightarrow [\![Q]\!]$ の証明手法を提案する．プロセス P の述語的意味を表す論理式 $[\![P]\!]$ は，それに対する入力を表す論理式とその出力を表す論理式とに分割することができる．この点に着目し，$[\![P]\!]$ をそれと論理的等価な標準形に変換し，その構造に従って，証明全体を小さく簡単な部分に分割して行うアルゴリズムを与える．

1 Introduction

Recently many formal models that incorporate "time" have been proposed based on the communicating process framework [4][7].In those models, many approaches have been taken to reason communicating processes that interact with the environment with time constraints.

In view of a communicating process as a model of concurrent programs, we aim to model the reliability for program with time constraints in a formal framework. For this purpose, we realize a communicating process as a first-order predicate formula where its behaviors are specified as the interpretation of the formula. The framework of first-order predicate calculus provides the direct method to provide a "proof" that a program satisfies a specification. This "proof-oriented" approach is originally taken in [1][2] for the asynchronous communication mechanism. We extended the approach previously in [8] for real-time communicating process with the synchronous communication mechanism. In this paper, we further investigate a verification method for this approach.

In [2], the predicative specification of communicating process with time constraints is introduced, where the communicating mechanism is totally asynchronous, thus a process that requires a value on an input channel must wait arbitrarily long until any value is available on the channel. But in the real situation, an input value is required to be enabled when a process requires it in such a situation like providing a password at logging into a machine, where a machine must "time-out" and notify the failure if a user does not provide a correct password in time. With this observation, we take a view that a communication is synchronous in nature to specify real-time communications.

The final goal of this paper is to provide a systematic verification method for our predicative specification to deduce an implication between specifications of processes. This kind of proof is useful to verify a step of refinement. We take the advantage that the specifications of processes are restricted in the form. Intuitively a process is specified to reach one of specified states offering specified outputs if a specified pattern of inputs are provided, therefore we can convert a specification formula into "a normalized specification" where inputs and outputs are syntactically separated. We show the algorithm to prove normalized specification formulas by *decomposition.*

The rest of the paper is structured as follows. section 2 reviews the timed predicative specification shown in [8] with some modifications. Section 3 proposes the proof technique and the algorithm. Section 4 concludes the paper.

2 Predicative Specification

As in the untimed predicative specification [1], an interpretation for a communication channel is a trace of values where each value is stamped with the time to happen. This *timed trace* is similar to that appearing in [6] or [2].

Let the set of non-negative real numbers be denoted by $\mathbf{R}^{\geq 0}$, then a time domain is $\mathbf{R}^{\infty} = \mathbf{R}^{\geq 0} \cup \{\infty_{\mathcal{T}}\}$ where $\infty_{\mathcal{T}}$ is a constant that designates the "unbounded time", where $r < \infty_{\mathcal{T}}$ for all $r \in \mathbf{R}^{\geq 0}$. Over time domain $\mathbf{R}^{\infty}$, we assume the operation extended with $\infty_{\mathcal{T}} + r = r + \infty = \infty_{\mathcal{T}}$ and $\infty_{\mathcal{T}} - r = \infty_{\mathcal{T}}$.

Given a set of values $\mathcal{V}$ and a time domain $\mathbf{R}^{\infty}$, $\langle v, t \rangle \in \mathcal{V} \times \mathbf{R}^{\infty}$ is called a *timed value.* $\langle v, t \rangle$ is intended to mean a communication or a update involved in value v at time t. We shall call t the *time-stamp* of timed value $\langle v, t \rangle$ with the projections with postfix notation *.ts* and *.val* respectively:for timed value $v_t = \langle v, t \rangle$, $v_t.val = v$ and $v_t.ts = t$. The *time shift operation* denoted by $v_t \backslash t'$, is defined as: $v_t \backslash t' = \langle v_t.val, v_t.ts \dot{-} t' \rangle$ Intuitively, $v_t \backslash t'$ generates the timed value relative to time t'.

A operation *non-negative subtraction* $\dot{-}$ over $\mathbf{R}^{\geq 0}$ is defined by:

$$r_1 \dot{-} r_2 = \begin{cases} r_1 - r_2 & \text{if } r_1 \geq r_2 \\ 0 & \text{otherwise} \end{cases}$$

First we define a history sequence, called a *timed history*, that is designated for an observation for a communication channel.

Definition 1 *Given a set of values $\mathcal{V}$, a* timed history *h is a (possibly infinite) sequence of timed values for $\mathcal{V}$. We denote (i+1)-th timed value of h by $h[i]$. A timed history has the following conditions:*

1. *$h[0].ts \geq 0$;*
2. *$i < j$ implies $h[i].ts < h[j].ts$;*
3. *for all $t \in \mathbf{R}^{\geq 0}$ there exists i such that $h[i].ts \geq t$ if h is infinite.*

For timed histories, we use the following notation throughout the paper:

- The projection functions for timed history h; $h.val[i] = h[i].val$ and $h.ts[i] = h[i].ts$
- The time shift operator for a timed history h; $h \backslash t = h'$ such that $h'[i].val = h[i].val$ and $h'[i].ts = h[i].ts \dot{-} t$
- We write $\#h$ for the length of h if h is finite. Otherwise $\#h = \infty$, where ∞ is an extra constant that means the "infinite length".

- $\#h = n$ if $h[i].ts < \infty_T$ for all $i < n$ and $h[j] = \langle \bot, \infty_T \rangle$ if $j \geq n$, where $\bot$ is a distinguished value for "undefined".
- We write $\frown$ for concatenation of timed histories. If $\#h = \infty$, $h \frown h' = h$
- $h \precsim h'$ if for some h_0 $h' = h_0 \frown h$, and $h \leq h'$ if for some h_0, $h' = h \frown h_0$

Timed values and timed histories constitute the domain for models of specification formulas of real-time communicating processes. A process is specified through a specification formula that satisfies the intended observation of timed values and timed histories. We next discuss the specification formulas.

Before giving the specifications, we define a structure over time for time constraints of programs.

Definition 2 *Let the set of non-negative rational numbers be denoted by* $\mathbf{Q}^{\geq 0}$. *Given* $q_1, q_2 \in \mathbf{Q}^{\geq 0}$ *and* $q_3 \in \mathbf{Q}^{\geq 0} \cup \{\infty_T\}$ *such that* $q_1 \leq q_2$ *and* $q_1 \leq q_3$, *a* time interval *is one of the following forms:* $\emptyset$, $[q_1, q_2]$, $[q_1, q_3)$, $(q_1, q_2]$ *and* (q_1, q_3). *We usually use* I *for either form of time intervals and* $\mathcal{I}$ *for the set of all time intervals.*

2.1 Specification formulas and implementability

A communication process may have program variables X, communication channels Γ and time variables T_v. A The behavior for each variable is specified by its initial state and its final state. Γ is supposed to be partitioned into two disjoint sets Γ_{in} and Γ_{out}. Γ_{in} stands for the set of input channels and Γ_{out} stands for output channels. We usually use c or $c_1, c_2, \dots$ for input channels, d or $d_1, d_2, \dots$ for output channels in the rest of the paper. We write $\vec{v}$ for a vector of variables $\langle v_1, v_2, \cdots, v_n \rangle$, where v_i is a free variable.

Then, a *specification formula* has the following types of free variables;

1. $\grave{X}$: A set of initial computation variables, ranged over by $\grave{x}, \grave{x}_1, \grave{x}_2, \cdots$.
2. $\acute{X}$: A set of final computation variables, ranged over by $\acute{x}, \acute{x}_1, \acute{x}_2, \cdots$.
3. $\grave{\Gamma}$: A set of initial channel variables, ranged over by $\grave{c}, \grave{d}, \cdots$.
4. $\acute{\Gamma}$: A set of final channel variables, ranged over by $\acute{c}, \acute{d}, \cdots$.
5. T_v: A set of time variables, ranged over by $t, t_1, t_2, \cdots$.

For $\grave{X}$ and $\acute{X}$, timed values are assigned, for $\grave{\Gamma}$ and $\acute{\Gamma}$, timed histories are assigned, and for T_v, nonnegative real numbers are assigned. We call a variable marked by "`" *initial* and a variable marked by "´" *final.*

First we argue the *implementability of a specification* [2] as the untimed framework [1], we have introduce the "chaos" specification to ensure the consistency of the initial/final relation. The *chaos specification* is defined as:

$$K =_{def} \vec{\grave{x}}.ts = 0 \wedge \vec{\acute{c}} \precsim \vec{\grave{c}} \wedge \vec{\grave{d}} \leq \vec{\acute{d}}$$

Following [1], we claim that K is the least specification with respect to determinacy. Therefore only a specification formula that implies K is admitted as a specification of a process. Since every specification $[\![P]\!]$ for process P must be implementable, the following implication must hold.

$$\forall \vec{\grave{v}}, \vec{\acute{v}}, \vec{t} \; [\; [\![P]\!] \Rightarrow K \;]$$

2.2 Real-time Communicating Processes

In this section, we define a language for real-time communicating processes which is a direct extension of Hehner's language [1] annotated by time constraints for communications.

2.2.1 Syntax of Processes

- **Sublanguage for values** :
Given operations Σ_v and a set of computation variables X, a *value expressions over* X, ranged over by e, is a well-formed term built from Σ_v and $\{x.val \mid x \in X\}$. A ground expression, i.e. with no variable, is assumed to be evaluated to some $v \in \mathcal{V}$.

- **Language for processes:**
Let $X, \Gamma, \mathcal{T}_v$ and $\mathcal{I}$ be a set of computation variables, communication variables, time variables and time intervals respectively. We write e for a expression over X and b for a boolean expression over X. Let $x, x_1, x_2 \in X$, $c, c_1, c_2 \in \Gamma_{in}$, $d \in \Gamma_{out}$, $I, I_1, I_2 \in \mathcal{I}$ and m is a timed expression generated over $\mathbf{R}^{\geq 0}$ and T_v. A pair of an input channel c and an output channel d, written as $c \leftarrow d$, is called a *communication channel* and let Com be a set of communication channels.

A *real-time communicating process* P is defined by the following BNF:

$$\begin{aligned} P ::= \; & \mathbf{skip}(m) \mid x :=_I e \mid d!_I e \mid c?_I x \to P \\ & \mid P_1 \; ; \; P_2 \mid P_1 \parallel_{Com} P_2 \mid P_1 \; \mathbf{or} \; P_2 \\ & \mid [\; c_1?_{I_1} x_1 \to P_1 \;\Box\; c_2?_{I_2} x_2 \to P_2 \;] \\ & \mid \mathbf{if} \; b \; \mathbf{then} \; P_1 \; \mathbf{else} \; P_2 \end{aligned}$$

The intuitive meanings of process terms are as follows:

· **skip**(m) is the empty operation but m time units past.

- $x :=_I e$ is an assignment of value e into computation value x in I.
- $d!_I e$ emits value e via port d in I
- $c?_I x \rightarrow P$ receives a value to variable x via port c in I and becomes P in which variable x is replaced by the value received.
- $P_1; P_2$ is the sequential composition that P_2 starts immediately after P_1 gets inactive.
- $P_1 \parallel_{Com} P_2$ is the parallel composition of P_1 and P_2 connected by Com. As a syntactic restriction, P_1 and P_2 share no program variable nor port variable.
- P_1 **or** P_2 is a nondeterministic choice.
- $[\ c_1?_{I_1}x \rightarrow P \ \Box\ c_2?_{I_2}x \rightarrow P\ ]$ is the external choice. If any value is available on either of c_1 in I_1 or c_2 in I_2, the process takes the value and choose the branch.
- **if** b **then** P_1 **else** P_2 is the conditional choice.

2.2.2 Translation

We write $f^{\vec{v}}_{\vec{e}}$ for the expression obtained by replacing each occurrences of $\vec{v}$ with the corresponding expression $\vec{e}$. In the specification, t_ω is a distinguished time variable to represent the termination time of the process. We consider a process is *terminated* at t if no more communication is enabled after t.

- **Skip** $\mathbf{skip}(m)$

$$\stackrel{\mathrm{def}}{=}\ (\vec{v}\backslash m = \grave{\vec{v}}) \wedge (t_\omega = m) \wedge K$$

- **Assignment** $x :=_I e$

$$\stackrel{\mathrm{def}}{=}\ \exists t\ [\ t \in I \wedge \vec{v} = \grave{\vec{v}}^{\ \grave{x}}_{\langle \grave{e},t\rangle} \wedge t_\omega = t\ ] \wedge K$$

where $\grave{e} = e^{\vec{v}}_{\grave{\vec{v}}.val}$

- **Output** $d!_I e$

$$\stackrel{\mathrm{def}}{=}\ \exists t\ [\ \vec{v} = \grave{\vec{v}}^{\ \grave{d}}_{\grave{d}\frown\langle \grave{e},t\rangle}\ \wedge t \in I \wedge t_\omega = t\ ] \wedge K$$

- **Input** Let Shift(P,t) be the shorthand for $P^{\grave{\vec{v}}\ \ \vec{v}\ \ \ t_\omega}_{\grave{\vec{v}}\backslash t\ \vec{v}\backslash t\ t_\omega \dot{-} t}$.Shift$(P,t)$ is the process all timed variables of which are shifted backward by t.

$$c?_I x \rightarrow P \stackrel{\mathrm{def}}{=} [\ (\#\grave{c} > 0 \wedge \grave{c}[0].ts \in I) \Rightarrow \mathrm{Shift}(P, \grave{c}[0].ts)^{\grave{x}\ \ \ \ \grave{c}}_{\grave{c}[0]\ \grave{c}[1\ldots]}\ ] \wedge K$$

- **Sequential composition**

$$P;Q \stackrel{\mathrm{def}}{=} [\ \neg\forall\vec{v}[P \Leftrightarrow K] \Rightarrow \exists\dot{\vec{v}}\exists\dot{t}\ [\ P^{\vec{v}\ \ t_\omega}_{\dot{\vec{v}}\ \ \dot{t}} \wedge \mathrm{Shift}(Q,\dot{t})^{\grave{\vec{v}}}_{\dot{\vec{v}}}\ ]\ ] \wedge K$$

- **Parallel Composition** $P \parallel_{Com} Q$

$$\stackrel{\mathrm{def}}{=} \exists\acute{\vec{d}},\ \vec{c}\ [\ P^{t_\omega}_{t_1} \wedge Q^{t_\omega}_{t_2}\ ]^{\grave{\vec{c}}\ \grave{\vec{d}}}_{\acute{\vec{d}}\ \lambda} \wedge t_\omega = max(t_1,t_2)$$

where $Com = \{c_i \leftarrow d_i \mid 1 \leq i \leq n\}$

- **Deterministic Choice**
  if b **then** P **else** Q

$$\stackrel{\mathrm{def}}{=} (\grave{b}.val \wedge P) \vee (\neg\grave{b}.val \wedge Q)$$

- **Nondeterministic Choice**
  P **or** $Q \stackrel{\mathrm{def}}{=} P \vee Q$
- **Input Choice**

$$\begin{aligned}
&[\ c_1?_{I_1}x \rightarrow P \ \Box\ c_2?_{I_2}y \rightarrow Q\ ]\\
&\stackrel{\mathrm{def}}{=}[\ (\ \#\grave{c}_1 = 0 \vee \grave{c}_1[0].ts \notin I_1\ )\\
&\quad \wedge(\ \#\grave{c}_2 = 0 \vee \grave{c}_2[0].ts \notin I_2\ ) \wedge K\ ]\\
&\vee[\ \#\grave{c}_1 > 0 \wedge \grave{c}_1[0].ts \in I_1 \wedge \grave{c}_1[0].ts \leq \grave{c}_2[0].ts\\
&\quad \wedge\mathrm{Shift}(P, \grave{c}_1[0].ts)^{\grave{x},\ \ \ \ \grave{c}_1}_{\grave{c}_1[0],\grave{c}_1[1\ldots]}\ ]\\
&\vee[\ \#\grave{c}_2 > 0 \wedge \grave{c}_2[0].ts \in I_2 \wedge \grave{c}_2[0].ts \leq \grave{c}_1[0].ts\\
&\quad \wedge\mathrm{Shift}(Q, \grave{c}_2[0].ts)^{\grave{y},\ \ \ \ \grave{c}_2}_{\grave{c}_2[0],\grave{c}_2[1\ldots]}\ ]
\end{aligned}$$

2.2.3 Specification Examples

In the following examples, we interpret L_{in} as the port that accepts a password, S_{out} as the port that displays a prompt, and P_{wd} as a person who is giving a password "#" to login.

Example 1 *The person can always give the password and login to the machine with the prompt on the screen.*

$$\begin{aligned}
&[\ L_{in}?_{I_1}x \rightarrow S_{out}!_I\,''\%''\ \parallel_{\{L_{in}\leftarrow P_{wd}\}}\ P_{wd}!_{I_2}\,''\#''\ ]\\
&\qquad \textit{where } I_1 = [0,30] \textit{ and } I_2 = [0,20]\\
&\stackrel{\mathrm{def}}{=}\ \exists\acute{L}_{in}\exists\acute{P}_{wd}[\ (L_{in}?_{I_1}x \rightarrow S_{out}!_I\,''\%'')^{t_1}_{t_\omega}\\
&\qquad\qquad \wedge(P_{wd}!_{I_2}\,''\#'')^{t_2}_{t_\omega}\ ]^{\grave{L}_{in}\ \grave{P}_{wd}}_{\acute{P}_{wd}\ \lambda}\\
&\qquad \wedge(t_\omega = max(t_1,t_2))\\
&=\ \exists t\ [\ \acute{x} = \langle''\#'',t\rangle \wedge \acute{S}_{out} = \grave{S}_{out} \frown ''\,\%''\\
&\qquad\qquad \wedge t \in [0,20] \wedge t_2 = t\ ]\\
&\qquad \wedge t_1 \in [0,30] \wedge t_\omega = max(t_1,t_2) \wedge K
\end{aligned}$$

Example 2 *The person may fail to give the password in time. Then, the login procedure results in meaningless.*

$$\begin{aligned}
&[\ L_{in}?_{I_3}x \rightarrow S_{out}!_I\,''\%''\ \parallel_{\{L_{in}\leftarrow P_{wd}\}}\ P_{wd}!_{I_4}\,''\#''\ ]\\
&\qquad \textit{where } I_3 = [0,30] \textit{ and } I_4 = [0,60]\\
&\stackrel{\mathrm{def}}{=}\ \exists\acute{L}_{in}\exists\acute{P}_{wd}[\ (L_{in}?_{I_3}x \rightarrow S_{out}!_I\,''\%'')^{t_1}_{t_\omega}\\
&\qquad\qquad \wedge(P_{wd}!_{I_4}\,''\#'')^{t_2}_{t_\omega}\ ]^{\grave{L}_{in}\ \grave{P}_{wd}}_{\acute{P}_{wd}\ \lambda}\\
&\qquad \wedge(t_\omega = max(t_1,t_2))\\
&\textit{(Consider } \acute{P}_{wd} = \langle''\#'',t\rangle \textit{ for some } 30 < t \leq 60\textit{)}\\
&=\ K
\end{aligned}$$

3 Verification Method

From now on, we focus on the specification formulas derived from processes by the predicative specification. Viewing a process also as a specification, it is important to prove that a process satisfies another process in the situation to prove a step of "refinement" [3]. In this section we investigate a systematic proof technique by decomposing a specification formula into smaller parts of input/output formulas.

We convert a specification formula into a composition of *condition formulas* and *assertion formulas*. Intuitively, condition formulas indicate input condition for process P and assertion formulas indicate output from process P when the condition is satisfied.

Definition 3 *A* condition term *is a term the variables of which are only initial. An* atomic condition formula *is of either form of (c-1) or (c-2):*

(c-1) $M_1 = M_2$, $M_1 < M_2$ *or* $M_1 > M_2$ *where* M_1 *and* M_2 *are condition terms.*

(c-2) $M.ts \in I$ *where* M *is a condition term.*

An atomic assertion formula *is in the either form of (a-1), (a-2) or (a-3):*

(a-1) K

(a-2) $\acute{v} = M$ *or* $t = M$ *where* v *is either of a program variable, an input channel or output channel and* M *is a term and* t *is a time variable*

(a-3) $t \in M$ *where* t *is a time variable and* M *is a term*

Definition 4

(i) *Every atomic condition (resp. assertion) formula is a condition (resp. assertion) formula.*

(ii) *If* α, β *are condition (resp. assertion) formulas, then so are* $\neg\alpha$ *and* $\alpha \wedge \beta$.

Definition 5 (Normal Form) *A* normal form *is a disjunction form:* $\vee_i[\ CF_i \wedge AF_i\ ]$ *where* CF_i*'s are condition formulas and* AF_i*'s are assertion formulas.*

Theorem 6 *For any process* P*, there exists a normal form that is logically equivalent to* $[\![P]\!]$.

For notational convenience, we write $NF(P)$ for the normal form of the specification formula of P, $CF(P)_i$'s for the condition formulas of $NF(P)$ and $AF(P)_i$'s for the assertion formulas of $NF(P)$.

Lemma 7

$$NF(P) = \bigvee_i\ [\ CF(P)_i \wedge AF(P)_i\ ]$$
$$NF(Q) = \bigvee_j\ [\ CF(Q)_j \wedge AF(Q)_j\ ]$$

Then, if

$$\forall\vec{v}\ \vec{\acute{v}}\ \vec{t}\,[\ NF(P) \Rightarrow NF(Q)\ ]$$

$$\forall i\exists j\ [\ (CF(P)_i \Rightarrow CF(Q)_j) \wedge (AF(P)_i \Rightarrow AF(Q)_j)\ ]$$

Next we show the algorithm to see if $[\![P]\!] \Rightarrow [\![Q]\!]$ by proving $NF(P) \Rightarrow NF(Q)$. By decomposing $NF(P)$ into $CF(P)$ and $AF(P)$, we can construct a checking algorithm. In the algorithm, we use the fact that checking a implication of condition formulas is easier than checking a implication of assertion formulas from the definition. In the algorithm, first we check

$$CF(P)_i \Rightarrow CF(Q)_j$$

and only when this implication is satisfied, the procedure may proceed to check:

$$AF(P)_i \Rightarrow AF(Q)_j$$

Checking Algorithm

Let $CF(P)_i, CF(Q)_j$ be as following formulas:

$$CF(P)_i = \bigwedge_k F(P)_{ik}$$
$$CF(Q)_j = \bigwedge_l F(Q)_{jl}$$

where $F(P)_{ik}, F(Q)_{jl}$ are atoms.

I. set $i = 1$.

[step I-0]

Set $S^{(0)}(P)_i = \{\ CF(Q)_j \mid \forall j.CF(Q)_j\ \}$

[step I-(k+1)] (for $k \geq 0$)

If $S^{(k)}(P)_i$ has an empty formula $\emptyset_c$, then goto **step II.** Otherwise, set

$\bar{S}^{(k)}(P)_i$
$= \{CF(Q)_j \mid \forall j\exists l\ [\ CF(Q)_j \in S^{(k)}(P)_i$ with $\underline{F(Q)_{jl}}$ where $\underline{F(P)_{ik} \Rightarrow F(Q)_{jl}}\ ]\}$

If $\bar{S}^{(k)}(P)_i$ is the empty set, terminate this algorithm with the result

"$NF(P)$ not satisfy $NF(Q)$".

Otherwise for the same $F(Q)_{jl}$ as the underlined part, set

$S^{(k+1)}(P)_i$
$= \{CF(Q)_j \mid \forall j[\ CF(Q)_j \in \bar{S}^{(k)}(P)_i,$ of which $F(Q)_{jl}$ is eliminated $]\}$ and go on to **step I-(k+2)**.

II. An empty formula means that all atoms are satisfied with $CF(P)_i$, i.e., $CF(Q)_j = F(P)_{i1} \wedge \cdots \wedge F(P)_{i,k-1}$
Then check $AF(P)_i \Rightarrow AF(Q)_j$ in the first-order predicate calculus.
If "$AF(P)_i \Rightarrow AF(Q)_j$ is valid" then $i = i + 1$ and goto **step I-0.**
Otherwise, terminate this algorithm with the result "$NF(P)$ not satisfy $NF(Q)$".

Proof Example

$$P = [\ c_1?_{I_1}x \rightarrow \mathbf{skip}(0)\ \Box\ c_2?_{I_2}y \rightarrow \mathbf{skip}(0)\ ]$$
$$Q = \ c_1?_{I_1}x \rightarrow \mathbf{skip}(0)\ \mathbf{or}\ c_2?_{I_2}y \rightarrow \mathbf{skip}(0)$$

We prove $[\![\ P\,]\!] \Rightarrow [\![\ Q\,]\!]$.

At first, we transform a specification formula into the normal form as follows: (underlined parts are condition formulas)

$NF(P)$

$$\begin{aligned}
= &\ [\underline{\#\grave{c}_1 = 0 \wedge \#\grave{c}_2 = 0} \wedge K] \\
&\vee[\underline{\#\grave{c}_1 = 0 \wedge \grave{c}_2[0].ts \notin I_2} \wedge K] \\
&\vee[\underline{\grave{c}_1[0].ts \notin I_1 \wedge \#\grave{c}_2 = 0} \wedge K] \\
&\vee[\underline{\grave{c}_1[0].ts \notin I_1 \wedge \grave{c}_2[0].ts \notin I_2} \wedge K]
\end{aligned}$$

$$\vee[\underline{\#\grave{c}_1 > 0 \wedge \grave{c}_1[0].ts \in I_1} \wedge \vec{v} = \vec{\grave{v}}\,{}^{\grave{x}}_{\grave{c}_1[0]}{}^{\grave{c}_1}_{\grave{c}_1[1\ldots]}]$$
$$\vee[\underline{\#\grave{c}_2 > 0 \wedge \grave{c}_2[0].ts \in I_2} \wedge \vec{v} = \vec{\grave{v}}\,{}^{\grave{y}}_{\grave{c}_2[0]}{}^{\grave{c}_2}_{\grave{c}_2[1\ldots]}]$$

$NF(Q)$
$$= [\underline{\#\grave{c}_1 = 0} \wedge K]$$
$$\vee[\underline{\grave{c}_1[0].ts \notin I_1} \wedge K]$$
$$\vee[\underline{\#\grave{c}_2 = 0} \wedge K]$$
$$\vee[\underline{\grave{c}_2[0].ts \notin I_2} \wedge K]$$
$$\vee[\underline{\#\grave{c}_1 > 0 \wedge \grave{c}_1[0].ts \in I_1} \wedge \vec{v} = \vec{\grave{v}}\,{}^{\grave{x}}_{\grave{c}_1[0]}{}^{\grave{c}_1}_{\grave{c}_1[1\ldots]}]$$
$$\vee[\underline{\#\grave{c}_2 > 0 \wedge \grave{c}_2[0].ts \in I_2} \wedge \vec{v} = \vec{\grave{v}}\,{}^{\grave{y}}_{\grave{c}_2[0]}{}^{\grave{c}_2}_{\grave{c}_2[1\ldots]}] .$$

Then we prove $NF(P) \Rightarrow NF(Q)$ as follows:

Proof:
I. set $i = 1$, $CF(P)_1 = \#\grave{c}_1 = 0 \wedge \#\grave{c}_2 = 0$
[step I-0] set
$$S^{(0)}(P)_1 = \{\ CF(Q)_j \mid \forall j.CF(Q)_j\ \}$$
[step I-1] $S^{(0)}(P)_1$ has no empty formula $\emptyset_c$, then set
$$\bar{S}^{(0)}(P)_1 = \{CF(Q)_1\} = \{\#\grave{c}_1 = 0\}$$
$\bar{S}^{(0)}(P)_1$ is not empty, then set
$$S^{(1)}(P)_1 = \{\emptyset_c\}$$
then goto **step I-2**.
[step I-2] $S^{(1)}(P)_1$ has an empty formula $\emptyset_c$, then goto **II**.
II. from $AF(P)_1 = AF(Q)_1 = K$, clearly
$$AF(P)_1 \Rightarrow AF(Q)_1$$
then $i = 2$ and goto **step I-0**. $\cdots$

In the same way, we prove that
$$\forall i \exists j\ [\ (CF(P)_i \Rightarrow CF(Q)_j) \wedge (AF(P)_i \Rightarrow AF(Q)_j)\ ]$$
Then, $NF(P) \Rightarrow NF(Q)$. ■

Next we check $NF(Q) \Rightarrow NF(P)$.

Proof:
I set $i = 1$, $CF(Q)_1 = \#\grave{c}_1 = 0$
[step I-0] set
$$S^{(0)}(Q)_1 = \{\ CF(P)_i \mid \forall i.CF(P)_i\ \}$$
[step I-1] $S^{(0)}(Q)_1$ has no empty formula $\emptyset_c$, then set
$$\bar{S}^{(0)}(Q)_1 = \{CF(P)_1\} = \{\#\grave{c}_1 = 0 \wedge \#\grave{c}_2 = 0\}$$
$\bar{S}^{(0)}(Q)_1$ is not empty, then set
$$S^{(1)}(Q)_1 = \{\#\grave{c}_2 = 0\}$$
then goto **step I-2**.
[step I-2] $S^{(1)}(Q)_1$ has no empty formula, and this algorithm terminates.
There is no $CF(P)_i$ satisfied by $CF(Q)_1$, i.e., $NF(Q) \Rightarrow NF(P)$ is not satisfied. ■

4 Concluding Remarks

We have presented a predicative specification for real-time communicating processes with the synchronous communicating mechanism, the earlier version of which is appeared in [8]. The translation from a process to a first order predicate formula is compositional, especially communications are specified basically as the conjunction of formulas. Next, we focused on specification formulas derived from processes and gave the proof technique by decomposing a specification formula into the input part and the output part.

The notion of time stamp appears generally in the specifications with time. In our framework, the time domain is the non-negative real numbers, but no underlying nature of time is assumed other than density. Our communication model is synchronous, but not fully compatible with the usual process algebras like [4][6] in that the communication channel may be seen as a queue with time constraint where no message is lost even if the time constraint is expired. In that case, the specification will report the failure.

Our communication model is more suitable to report a "failure" of communication. Thus, the usefulness of the specification will be exhibited by the application to the situation where strict reliability is required. We need further investigation in this respect.

References

[1] E.C.R. Hehner: "Predicative Programming Part 1 & 2", *Communication of ACM vol 27*,pp.134–151, (1984)

[2] E.C.R. Hehner: *A Practical Theory of Programming*, Springer-Verlag, (1993)

[3] D. Scholefield,H. Zedan,He Jifeng., " A Predicative Semantics for the Refinement of Real-Time Systems", LNCS 802,pp.230–249. (1994)

[4] Liang Chen: "Timed Processes: Models, Axioms and Decidability", Ph.D Thesis, CST-101-93, University of Edinburgh, (1993)

[5] E.C.R.Hehner, L.E.Gupta and A.J.Malton, "Predicative Methodology", *Acta Informatica 23*, pp.487–505. (1986)

[6] Rajeev Alur, David Dill: "Automata For Modeling Real-time Systems", LNCS 443, pp.322–335. (1990)

[7] U. Holmer, K. Larsen and W. Yi: "Deciding Properties of Regular Real Timed Processes", LNCS 575, pp.443–453, (1991)

[8] S. Yuen, K. Shinohara, T. Sakabe and Y. Inagaki: "Predicative Specification for Real-time Communicating Processes",12th Conference Proceedings Japan Society for Software Science and Technology, pp.181–184, (1995)