Real Time Process Algebra with Urgency Executing Policy

Wenbo Chen[1], Guang Zheng[1,*], Lian Li[1] and Jinzhao Wu[2]
[1]School of Information Science and Engineering,
Lanzhou University, Lanzhou, China 730000,
{chenwb, zhengguang, lil}@lzu.edu.cn
[2]School of Computer and Information Technology,
Beijing Jiaotong University,
Beijing, China, 100044,
[3]Chengdu Institute of Computer Applications Chinese Academy of Sciences,
Chengdu, China, 610054
himrwujz@yahoo.com.cn
*zhengguang@lzu.edu.cn

***Abstract*—Real time systems have a natural executing policy of urgency. However, real time process algebras of nowadays cannot specify this basic executing policy which limits their expressiveness. There is only one default policy called "maximal progress" in process algebras which is not enough to specify the behaviors of real time systems. Based on this, we propose a real time process algebra with urgency executing policy which can specify the behaviors of real time systems.**

***Key words:* process algebra, executing policy, real time, urgency.**

I. INTRODUCTION

There is little change in the composition of *non-deterministic choice* in the development of process algebras [14], [11], [1] under the assumption of maximal progress [3], [13], [6].

In PCCS (Probabilistic CCS)[2], [5], components under choice composition are equipped with probabilistic variables. These probabilistic variables make the decision among processes from non-deterministic to a more accurate form. In PEPA [7], [9], *maximal progress* is used. The system selects the action with shortest duration for the coming execution. Actions in PEPA are associated with exponential distributions which describe the actions' durations. *Race condition* is used in PEPA that governs the dynamic behavior of a model whenever more than one activity is enabled, which means only the "fastest" succeeds.

In real time systems [16], [4], all actions should be scheduled under the policy of *urgency*. The responsibility of the scheduling algorithm is to determine an order of execution of the real time tasks that is feasible, i.e. that meets timing requirements of those tasks. In the design of a real time system, the choice of an appropriate scheduling algorithm (or policy) may depend on several issues, e.g. the number of processors available in the system, their homogeneity or heterogeneity, the precedence relations among the application tasks, and the task synchronization methods.

Real time scheduling algorithms can be classified as either static or dynamic algorithms. A static scheduling algorithm is one in which a feasible schedule is computed off-line; one such algorithm typically requires a priori knowledge of the tasks characteristics. In contrast, a dynamic scheduling algorithm determines a feasible schedule at run time. Dynamic scheduling can adapt to changes in the environment. In this paper, we adopt dynamic scheduling algorithms called *urgency* executing policy. The dynamic scheduling approach has led to the development of a variety of preemptive scheduling policies. These include the Earliest Deadline First (EDF), and the Least Slack Time First (LSTF) policies.

With the EDF policy, the earlier the deadline of a task, the higher the priority assigned to that task. Instead, with the LSTF policy, the smaller the slack time (see below) of a task, the higher the priority value assigned to that task. The task slack time is defined as the difference between the amount of time needed from the current time value to the deadline of a task, and the amount of time that task requires to perform its computation.

We adopt the *urgency* executing policy like LSTF. We define parameter t as the time limitation before which action $a(t)$ should be executed successfully. What's more, we classify the choice composition into two groups. One group is internal choice $\oplus$, through which the system can decide which action is to be executed by its execution policy. Another group is external choice $\uplus$ which is the choice exposed to the environment. We also study the execution of parallel composition, through which the system can make its decisions under parallel composition.

In intelligent real time systems, they can perform some actions which can be called as the intelligent behaviors of

This work was partially supported by the National Natural Science Foundation of China (No. 90612016 and 60473095). China Next Generation Internet Cernet2 (No. 432353). National Key Basic Research Project of China (2004CB318000 and 2007CB310800), and The National High Technology Research and Development Program of China (No. 2007AA01Z143).

the systems. The system can evolve from one state to another by executing a series transitions labeled by actions.

In some cases, real time system may have several actions available for the coming execution and they are homogeneous. As to choosing which one, the system needs to make its decisions by algorithm under the *urgency* executing policy. We construct the system's action with structure $a(t)$, and parameter t represents the time left before a is executed.

Example 1.1 *There is a web server providing services for Internet games. For simplicity, we assume this server provides two games, one is* card, *and the other is* shooting. *During the interaction of clients and the server, the time restriction for card game is t_{card} and the time restriction for shooting is t_{shoot}. We set the $t_{card} = 60$ time units which is much greater than the $t_{shoot} = 3$. If action $a_1(60)$ and $b_1(3)$ are activated at the same time, then it is natural for us to require the server to serve the process with the shortest time restriction. Suppose that process P_{card} is loaded at the same time as process P_{shoot}, with the t_{shoot} shorter than t_{card}, then the server should execute the action $b(t_{shoot})$, which is shown in figure 1.*

The dashed lines connecting the two execution lines are time stamps, *which restrict the two lines with the same time scale. From this figure, we can see the execution policy of* urgency *in the system run.*

After defining the language of process algebra with urgency executing policies for real time systems, we show the operational semantics of the operators in this language. Then, we construct axioms for operators. As bisimulation relationships play an important role in the theory of process algebra, we define the strong bisimulation and weak bisimulation.

This paper is organized as follows. Section 2 defines the language of process algebra with executing policies. Section 3 defines the operational semantics and the axioms for the operators. Section 4 proposes the equational theory including the expansion law, strong bisimulation and weak bisimulation. Section 5 concludes this paper.

II. Language

In this section, we will give out the definition of the language for the process algebra of real time systems.

Sequential operators as prefix, parallel, termination, deadlock and recurrence have little relationship with *urgency* executing policy. *Choice* composition is divided into two: the external choice $\uplus$ and internal choice $\oplus$.

The *internal choice* operator is very important and valuable in practise. It equips the real time systems under *urgency* policy with the ability to make feasible schedule of their executions.

The grammar of our PA is:

$$\begin{aligned} P ::\; = \;& 0 \mid \checkmark \mid \delta \mid a(t).P \mid P \uplus P \mid P \oplus P \mid P;P \mid \\ & P||_S P \mid P \setminus L \mid P[f] \mid X \mid fix(X = E) \end{aligned}$$

0 is the constant named *empty process* indicating inactive process capable of doing nothing. $\checkmark$ is the constant named *successful termination* indicating a process terminate successfully. δ is the constant named *deadlock* indicating unsuccessful termination of a process capable of doing nothing.

$a(t).P$ is action prefixing, only when the action $a(t)$ is executed, the system would behave process P. Parameter t indicates the time restriction for action a: a must be executed within time $t.P;Q$ is the sequential operator, only when the process P terminates successfully, then, process Q will get its turn to be executed.

$P \uplus P$ is the external choice composition for the interaction between system and environment. $P \oplus P$ is the internal choice which is decided by the system. By internal choice, system can select proper action for execution under the *urgency* executing policy.

$P||_S Q$ is the parallel operator. It represents a system in which components P and Q work together to perform activities in the set S. The set S is called the *synchronising* or *cooperation* set. Both components proceed independently with any activities whose types do not belong to the set S. Activities with action types in the set S are assumed to require the simultaneous involvement of both components. The resulting activity will have the same action type as the two contributing activities and a rate reflecting their rates. As to the rate, which is decided by action in the form of $a(t)$ or $a(p)$.

$P \setminus L$ is the hiding operator. It behaves as P except that any activities of types within the set L are *hidden*. meaning that their type is not visible outside the component upon completion. They appear as action typed τ as *internal* actions.

$P[f]$ is the relabelling operator. It behaves like P with actions of process P will be relabelled by function f.

X is a bound process variable. It is used in the definition of the recursive expression. $fix(X = E)$ is the recursive expression. We treat recursive expression as fixed-point to express the recursive process in the real world. For example, the expression $fix(X = a.X)$ represents a process that can perform infinite number of actions a.

Example 2.1 *Revisit example 1.1, we can get the expression of process a and b as: $a_1(60).a_2(60)||b_1(3).b_2(3).\cdots.b_n(3)$. As to the execution, after execute $b_1(3)$ just within the time limitation, the system will evolve into $a_1(57).a_2(60)||b_2(3).\cdots.b_n(3)$. After executing 19 b actions, the system evolves into $a_1(3).a_2(60)||b_{19}(3).\cdots.b_n(3)$. Then, the system executes $a_1(3)$ and $b_{19}(3)$ synchronously at time* 20 *and evolves into $a_2(60)||b_{21}(3).\cdots.b_n(3)$.*

This example shows that the language of real time process algebra with urgency executing policy can specify the behaviors of real time systems especially by the policy of EDF (Earliest Deadline First) and LSTF (Least Slack Time First).

III. Operational semantics and axioms

Operational semantics gives out the intuitive evolution rules of the system the process algebra concerned. There have many situations where intelligent system can make their own decisions, e.g. select an action (program) to be activated for the next execution.

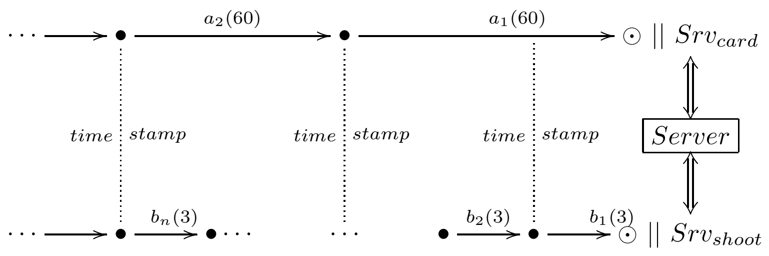

Fig. 1. Executing policy of *urgency*

Action prefix	$\dfrac{}{a(t).P \xrightarrow{a(t)} P}$	
Internal Choice	$\dfrac{P \xrightarrow{a(t)} P',\ Q \xrightarrow{a(t')} Q'}{P \oplus Q \xrightarrow{a(t)} P'}(t < t')$	$\dfrac{P \xrightarrow{a(t)} P', Q \xrightarrow{a(t')} Q'}{P \oplus Q \xrightarrow{a(t')} Q'}(t' < t)$
External Choice	$\dfrac{P \xrightarrow{a(t)} P'}{P \uplus Q \xrightarrow{a(t)} P'}$	$\dfrac{Q \xrightarrow{a(t)} Q'}{P \uplus Q \xrightarrow{a(t)} Q'}$
Parallel	$\dfrac{P \xrightarrow{a(t)} P'}{P\vert\vert_S Q \xrightarrow{a(t)} P'\vert\vert_S Q}(a \notin S)$	$\dfrac{Q \xrightarrow{a(t)} Q'}{P\vert\vert_S Q \xrightarrow{a(t)} P\vert\vert_S Q'}(a \notin S)$
	$\dfrac{P \xrightarrow{a(t')} P'\ Q \xrightarrow{a(t'')} Q'}{P\vert\vert_S Q \xrightarrow{\tau(t)} P'\vert\vert_S Q'}(a \in S)$	where $(t) = max(t', t'')$
Hiding	$\dfrac{P \xrightarrow{a(t)} P'}{P \setminus L \xrightarrow{a(t)} P' \setminus L}(a \notin L)$	$\dfrac{P \xrightarrow{a(t)} P'}{P \setminus L \xrightarrow{\tau} P' \setminus L}(a \in L)$
Relabelling	$\dfrac{P \xrightarrow{a(t)} P'}{P[f] \xrightarrow{f(a)(t)} P'[f]}$	
Recursion	$\dfrac{E\{fix(X = E)/X\} \xrightarrow{a(t)} E'}{fix(X = E) \xrightarrow{a(t)} E'}$	

TABLE I
THE OPERATIONAL SEMANTICS

A. Operational semantics

In this section, we will define the operational semantics of process algebra in Table 1. All deduction rules of the language listed in this table are in form of $a(t)$.

Action prefix: Process term $a(t).P$ will evolve to P by executing action $a(t)$ within time limitation t. In real time systems, $a(t)$ can be dropped if it can not be accomplished within time t.

Sequential composition: Sequential composition $P;Q$, when P is successfully terminated, Q gets its turn for execution. If there is a deadlock in P, Q can not be executed.

Internal action: In system runs, most actions are executed invisible and they are called as internal actions. We use τ to stand for them.

Internal choice: In real time systems, time plays the key role in the system run. All actions concerned should be activated and executed in the least time. Actions are equipped with time limitation t. System will activate another action when a timeout occurs.

Example 3.1 *For internal choice $\oplus$ under urgency policy, only parameter t is considered. System only select the action with the smallest t among action available for execution. There are also some changes in the operational semantics.*

In situation both P and Q have the same initial action a:

$$\frac{P \xrightarrow{a(t_1)} P',\ Q \xrightarrow{a(t_2)} Q'}{P \oplus Q \xrightarrow{a(t_1)} P'}\ (t_1 < t_2)$$

In case $t_1 = t_2$, then the choice would come to a non-deterministic for P and Q, and this is really hard to encounter. Under this situation, either process is OK under this policy.

External Choice: External choice $\uplus$ is an interface between system and the environment. System provides external choices for environment, and the environment can use them to accomplish its purposes. The external choice is completely non-deterministic for the system. thought there might have certain kind of probability for the actions in the long run, we assume that the system is not necessary to deal with it.

First, we explain the formula in For formula

$$\frac{P \xrightarrow{a(t)} P'}{P \uplus Q \xrightarrow{a(t)} P'}$$

the environment picks up the action $a(t)$ to execute among actions available provided by the system. As to the reason why the environment picks up the action $a(t)$, we leave it to the environment and not consider it here.

Termination predicator: We present $\checkmark$ to indicate the successful termination of a process which can be distinguished from deadlock (δ). $\checkmark$ indicates whether or not a process has a termination option. If so, then the result will be 0, and otherwise δ. For example, $\checkmark(P+0) = 0$ where as $\checkmark(P+Q) = \delta$.

Parallel composition: There are some changes in the parallel composition in this language with others.

In action $a(t)$, rule

$$\frac{P \xrightarrow{a(t)} P'}{P||_S Q \xrightarrow{a(t)} P'||_S Q}(a \notin S)$$

indicates that process P will evolve to P' by executing $a(t)$ with $a \notin S$ which means there have no communication between process P and process Q, and the process Q remains unchanged. It is similar with the rule

$$\frac{Q \xrightarrow{a(t)} Q'}{P||_S Q \xrightarrow{a(t)} P||_S Q'}(a \notin S)$$

As to the rule

$$\frac{P \xrightarrow{a(t')} P',\ Q \xrightarrow{a(t'')} Q'}{P||_S Q \xrightarrow{\tau} P'||_S Q'}(a \in S)$$

there will have a communication action τ(which can also be expressed as $\tau(t)$).

As to the parameter of (t) bounded with action τ in the rule, we can also figure them out if necessary, the system will wait to the time either P or Q cannot wait any longer which is: $t = max(t', t'')$. When an action is turned into internal action τ, usually we do not care about the executing time.

A key issue in process algebra is how to model and deal with the models of concurrency and communications. Processes under parallel conmposition can perform actions and evolve into other states independently. They can also perform synchronous communications and then evolve into other states synchronously.

For parallel composition under *urgency* policy, we know that the synchronizing among processes $P_1||_S P_2||_S \cdots ||_S P_n$ has nothing to do with the position in the terms. System picks up the quickest two processes for synchronization if all the action of P_i $i = 1, 2, ..., n$ are waiting for it,

$$\frac{P_i \xrightarrow{a(t_i)} P_i',\ P_j \xrightarrow{a(t_j)} P_j'}{P_1||_S P_2||_S \cdots ||_S P_n \xrightarrow{\tau} P_1||_S \cdots P_i'||_S \cdots P_j'||_S \cdots ||_S P_n}$$

where $1 \leq i < j \leq n$ and $max(t_i, t_j) \leq (t_k \mid k \notin \{i, j\} \wedge 1 \leq k \leq n)$.

Hiding: Hiding term $P \setminus L$ behaves like P except that any activities of types within the set L. Actions in L are not visible outside the component upon execution and they are turned into internal action τ.

Deduction rule

$$\frac{P \xrightarrow{a(t)} P'}{P \setminus L \xrightarrow{a(t)} P' \setminus L}(a \notin L)$$

indicates process P will evolve to P' by executing action $a(t)$ within time t under condition $a \notin L$. The deduction rule

$$\frac{P \xrightarrow{a(t)} P'}{P \setminus L \xrightarrow{\tau} P' \setminus L}(a \in L)$$

indicates process P can evolve to P' by executing action τ (with parameters like $\tau(t)$) within time t under condition $a \in L$. There is no visible action observed outside when $a(t) \in L$ is executed but a sojourn time t.

We use a particular simple relabeling operator is of special importance, which is δ_H, the hiding operator. It can be taken as an indispensable feature in process algebra.

Relabeling: $P[f]$ is the relabeling operator. $P[f]$ behaves like P with its actions relabeled by function f. It does not change parameters bounded with actions. and just change the name of actions, so we can get the deduction rule in action form $a(t)$ by

$$\frac{P \xrightarrow{a(t)} P'}{P[f] \xrightarrow{f(a(t))} P'[f]}.$$

Recursion: The meaning of recursion operator is given by equation such as

$$E\{fix(X = E)/X\}$$

Process variable X is guarded in the expression E, and the system run of $E\{fix(X = E)/X\}$ can be taken as a fixed-point, which means the the final step of one cycle run of the system leads to the starting point of the system run.

The deduction rule of constant can be expressed in action form $a(t)$ like

$$\frac{E\{fix(X=E)/X\} \xrightarrow{a(t)} E'}{fix(X=E) \xrightarrow{a(t)} E'}$$

Recursive is of key importance in the theory of process algebra which can be used to model the circulatory behavior of systems. For example, $P ::= a.P \uplus b.P$ can represent a system that can execute either a or b, and then return to the initial state. In this section, we take recursive behavior as expressed as fixed point.

B. Axiomatizations

Providing sound and complete axiomatizations for various equivalence relations has been one of the major research topics in the development of process theories. A complete axiomatization not only allows us to reason about process behaviors by syntactic manipulation, but also helps to understand the properties of the operators used to build complex processes from simpler components.

Now, we list the axioms of the process algebra with *urgency* policy in table II.

IV. Equational theory

Equivalence relations have been used in classic[14], probabilistic[2], timed[4], [16] and stochastic[9] process algebras to compare components and to replace a component with another which exhibits an equivalent behavior, but has a simpler representation. This technique still applies in real time process algebra with urgency policy.

A. The expansion law

The expansion law shows us all the possible execution of concurrent systems. The nondeterminism and concurrency of the execution in complex systems can be showed clearly by this law.

Definition 4.1 The expansion law under *urgency*
Let $P \equiv (P_1||_S \cdots ||_S P_n)$, with $n \geq 1$. Then

$$\begin{aligned} P \sim & \sum_{\oplus} \{a(t_{min}).(P_1||_S \cdots ||_S P_i'||_S \cdots ||_S P_n) \mid \\ & P_i \xrightarrow{a} P_i', a(t) \notin S\} \\ & + \sum_{\uplus} \{a.(P_1||_S \cdots ||_S P_i'||_S \cdots ||_S P_n) \mid \\ & P_i \xrightarrow{a} P_i', a \notin S\} \\ & + \sum \{\tau.(P_1||_S \cdots ||_S P_i'||_S \cdots ||_S P_j' \cdots ||_S P_n) \mid \\ & P_i \xrightarrow{l} P_i', P_j \xrightarrow{l} P_j', i<j, l \in S\} \end{aligned}$$

This policy is somehow like "*maximum progress*" in probabilistic process algebras, PEPA and stochastic process algebras. They all willing to execute more and more processes within restricted time.

Under internal choice composition, action $a(t)$ is executed among P_i $(1 \leq i \leq n)$. System under urgency policy selects the one with the shortest time restriction for execution which is $a(t_{min})$. But for external choice composition, it is out of the control of the system and it is non-deterministic. For internal action τ between two processes, it can not be observed outside, but we have reason to believe that the system under urgent policy will force its components to behave in the manner of urgency.

B. Recursiveness

It is very common for system to have recursive behaviors. We will study the equivalent relationship about recursive behavior in our language.

Definition 4.2 *For strong bisimulation relationships under execution policies as $\sim_U$, let E and F contain variables $\widetilde{X}$ at most. Then $E \sim_U F$ if, for all indexed sets $\widetilde{P}$ of processes, $E\{\widetilde{P}/\widetilde{X}\} \sim_U F\{\widetilde{P}/\widetilde{X}\}$.*

We shall also use $\widetilde{E} \sim_U \widetilde{F}$ to mean component-wise congruence between $\widetilde{E}$ and $\widetilde{F}$.

Proposition 4.3 *If $\widetilde{A} \overset{def}{=} \widetilde{P}$, then $\widetilde{A} \sim_U \widetilde{P}$*

Proof: By the operational semantics of *Congruence*, we see that for each i, A_i and P_i have exactly the same derivatives, and the result follows directly. □

Now we are ready to show that $\sim_U$ is preserved by recursive definition.

Proposition 4.4 *Let $\widetilde{E}$ and $\widetilde{F}$ contain variables $\widetilde{X}$ at most. Let $\widetilde{A} \overset{def}{=} \widetilde{E}\{\widetilde{A}/\widetilde{X}\}$, $\widetilde{B} \overset{def}{=} \widetilde{F}\{\widetilde{B}/\widetilde{X}\}$ and $\widetilde{E} \sim_U \widetilde{F}$. Then $\widetilde{A} \sim_U \widetilde{B}$.*

Proof We shall deal only with the case of single recursion equations, thus replacing $\widetilde{E}, \widetilde{F}, \widetilde{A}, \widetilde{B}$ by E, F, A, B. So assume $E \sim_U F$, $A \overset{def}{=} E\{A/X\}$, and $B \overset{def}{=} F\{B/X\}$. It will be enough to show that $\mathcal{S}$ is a strong bisimulation up to $\sim$, where

$$\begin{aligned} \mathcal{S} = & \{(G\{A/X\}, G\{B/X\}) \mid \\ & G \text{ contains at most the variable } X\} \end{aligned}$$

For then, by taking $G \equiv X$, it follows that $A \sim B$.

To show this, it will be enough to prove that
If $G\{A/X\} \xrightarrow{a} P'$ then, for some Q' and Q'',

$$G\{B/X\} \xrightarrow{a} Q'' \sim_U Q' \text{ with } (P',Q') \in \mathcal{S}$$

We shall prove the above formula by transition induction, on the depth of the inference by which the action $G\{A/X\} \xrightarrow{a} P'$ is inferred. We argue by cases on the form of G:

Case 1 $G \equiv X$.

Then $G\{A/X\} \equiv A$, so $A \xrightarrow{a} P'$, hence also

$$E\{A/X\} \xrightarrow{a} P'$$

by a shorter inference. Hence, by induction

$$E\{B/X\} \xrightarrow{a} Q'' \sim_U Q', \text{ with } (P',Q') \in \mathcal{S}.$$

But $E \sim F$, so

$$F\{B/X\} \xrightarrow{a} Q''' \sim_U Q'$$

and since

$$B \overset{def}{=} F\{B/X\},$$

For simplicity, we use $+$ to stands for $\{\oplus, \uplus\}$ in general.

$P + Q = Q + P$	**A1**	$a \cdot \tau \cdot P = a \cdot P$	**T1**
$P + (Q + R) = (P + Q) + R$	**A2**	$\tau \cdot P = \tau \cdot P + P$	**T2**
$P + P = P$	**A3**	$P \cdot (\tau \cdot Q + R) = P \cdot (\tau \cdot Q + R) + P \cdot Q$	**T3**
$(P + Q) \cdot R = P \cdot R + Q \cdot R$	**A4**		
$(P \cdot Q) \cdot R = P \cdot (Q \cdot R)$	**A5**	$P\|\|_S 0 = P$	**C1**
$P + \delta = P$	**A6**	$P\|\|_S \delta = P$	**C2**
$\delta \cdot P = \delta$	**A7**	$P\|\|_S Q = Q\|\|_S P$	**C3**
$P + 0 = P$	**A8**	$(P + Q)\|\|_S R = P\|\|_S R + Q\|\|_S R$	**C4**
$P \cdot 0 = P$	**A9**	$R\|\|_S (P + Q) = R\|\|_S P + R\|\|_S Q$	**C5**
$0 \cdot P = P$	**A10**	$p[f] = p$ if $p = \{0, \checkmark, \delta\}$	**L1**
$\delta_H(\tau) = \tau$	**H0**	$p[f] = f(p)$	**L2**
$\delta_H(a) = a$ if $a \notin H$	**H1**	$P[id] = P$	**L3**
$\delta_H(a) = \delta$ if $a \in H$	**H2**	$(P + Q)[f] = P[f] + Q[f]$	**L4**
$\delta_H(P + Q) = \delta_H(P) + \delta_H(Q)$	**H3**	$(P \cdot Q)[f] = (P[f]) \cdot (Q[f])$	**L5**
$\delta_H(P \cdot Q) = \delta_H(P) \cdot \delta_H(Q)$	**H4**	$P[f][g] = P[f \circ g]$	**L6**
$\delta_H \delta_K(P) = \delta_{H \cup K}(P)$	**H5**	$(P\|\|_S Q)[f] = (P[f])\|\|_{S[f]}(Q[f])$	**L7**
$P \oplus (Q \uplus R) = (P \oplus Q) \uplus (P \oplus R)$	**D1**	$P \uplus (Q \oplus R) = (P \uplus Q) \oplus (P \uplus R)$	**D2**
$fix(X = E) = E\{fix(X = E)/X\}$			**R1**
If $F = E\{F/X\}$ then $F = fix(X = E)$, with X is guarded in E			**R2**
$fix(X = X + E) = fix(X = E)$			**R3**
$fix(X = \tau \cdot X + E) = fix(X = \tau \cdot E)$			**R4**
$fix(X = \tau \cdot (X + E) + F) = fix(X = \tau + X + E + F)$			**R5**

TABLE II
THE AXIOMS OF PROCESS ALGEBRA WITH EXECUTING POLICY

$$G\{B/X\} \equiv B \xrightarrow{a} Q''' \sim_U Q', \text{ with } (P', Q') \in \mathcal{S}$$

are required.

Case 2 $G \equiv a \cdot G'$.

Then

$$G\{A/X\} \equiv a \cdot G'\{A/X\}$$

so

$$P' \equiv G'\{A/X\}$$

also

$$G\{B/X\} \equiv a \cdot G'\{B/X\} \xrightarrow{a} G'\{B/X\}$$

and clearly

$$(G'\{A/X\}, G'\{B/X\} \in \mathcal{S})$$

as required.

Case 3 $G \equiv G_1 \oplus G_2$ and $G \equiv G_1 \uplus G_2$.

This is simpler than the following case, and we omit the proof.

Case 4 $G \equiv G_1 ||_S G_2$.

Then

$$G\{A/X\} \equiv G_1\{A/X\} ||_S G_2\{A/X\}.$$

There are three cases for the action

$$G\{A/X\} \xrightarrow{a} P',$$

according to whether it arises from one or other component alone or from a communication. We shall treat only the case in which $a \in S$, and

$$G_1\{A/X\} \xrightarrow{a} P_1', \; G_2\{A/X\} \xrightarrow{a} P_2'$$

where $P' \equiv P_1' ||_S P_2'$. Now each component action has a shorter inference, so by induction

$$G_1\{A/X\} \xrightarrow{a} Q_1'' \sim_U Q_1', \text{ with } (P_1', Q_1') \in \mathcal{S},$$

$$G_2\{A/X\} \xrightarrow{a} Q_2'' \sim_U Q_2', \text{ with } (P_2', Q_2') \in \mathcal{S}$$

Hence, setting $Q' \equiv Q_1' ||_S Q_2'$ and $Q'' \equiv Q_1'' ||_S Q_2''$,

$$G\{B/X\} \equiv G_1\{B/X\} ||_S G_2\{B/X\} \xrightarrow{\tau} Q'' \sim_U Q'$$

It remains to show that $(P', Q') \in \mathcal{S}$. But

$$(P_i', Q_i') \in \mathcal{S} (i = 1, 2)$$

so for some H_i, $P_i' \equiv H_i\{A/X\}$ and $Q_i' \equiv H_i\{B/X\} (i = 1, 2)$; thus if we set $H \equiv H_1 ||_S H_2$ we have

$$(P', Q') \equiv (H\{A/X\}, H\{B/X\}) \in \mathcal{S}.$$

Case 5 $G \equiv G_1 \setminus L$, or $G_1[R]$

These cases are simpler than **Case 4**, and we omit the proof.

Case 6 $G \equiv C$, a process Constant with associated definition $C \overset{def}{=} R$. Then, since X does not occur, $G\{A/X\}$ and $G\{B/X\}$ are identical with C and hence both have a-derivative P'; clearly

$$(P', P') \equiv (P'\{A/X\}, P'\{B/X\}) \in \mathcal{S}$$

$\square$

The recursion, represented by the definition of Constants, is the only feature of the calculus which gives us processes with the power to computer infinitely; more than that, it gives our calculus the full power of Turing machines or any other basis for computation, so we should expect to spend some effort in showing that it behaves properly.

C. Strong bisimulation

We want to introduce the strong bisimulation for process algebra with *urgency* executing policy.

Definition 4.5 *A binary relation $\mathcal{S} \subseteq \mathcal{P} \times \mathcal{P}$ over processes with action in form of $a(t)$ is a* strong bisimulation *under policy of* urgency *if $(P, Q) \in S_U$ implies, for all $a(t) \in Act$,*

- *Whenever $P \xrightarrow{a(t_{min})} P'$ then, for some Q', $Q \xrightarrow{a(t_{min})} Q'$ and $(P', Q') \in S_U$;*
- *Whenever $Q \xrightarrow{a(t_{min})} Q'$ then, for some P', $P \xrightarrow{a(t_{min})} P'$ and $(P', Q') \in S_U$.*

We use $a(t_{min})$ to denote the action whose parameter t is the smallest. By doing this, system can select the most *urgency* actions available for the next execution.

In the sequential composition, action prefix and so on, there have no other actions available for choice, parameter t has no competitors and it is naturally the smallest.

We use $P \sim_U Q$ to denote processes P and Q in the relationship of strong bisimulation under policy of urgency.

The strong bisimulation under policy of *urgency* can be used to compare two or more processes dealing with real time systems. Real time systems have their actions scheduled dynamically during its executions.

Actions being executed within time limitations. If an action is successfully terminated within the time limitations, the system just moves on to execute the following actions. Otherwise, the timer guarding the time limitation will be triggered, and the system arranges another action for execution as designed.

Definition 4.6 *P and Q are strongly equivalent or strongly bisimilar written as $P \sim_{(t)} Q$ w.r.t. $a(t)$ under execution policies of urgency if $(P, Q) \in \mathcal{S}$ for some strong bisimulation $\mathcal{S}$ under one of the execution policy. This may be equivalently expressed as follows:*

$$\sim_U = \cup\{\mathcal{S} | \mathcal{S} \text{ is a strong bisimulation under policy of } \textit{urgency}\}.$$

D. Weak bisimulation

We have introduced strong bisimulation. Every action α in a process must be matched by an α action of the other - even every internal action τ. But in some cases, we should loosen the requirement. If the observer does not has the ability of observing the internal action τ, this yields a weaker notion of bisimulation called *weak bisimulation.* More precisely, we merely require that each internal action τ can just be omitted from the process. For example, $\tau.\alpha.\tau$ can be denoted by $\hat{\alpha}$ and transition $\overset{\hat{\alpha}}{\Rightarrow}$.

Definition 4.7 *A binary relation $\mathcal{S}_U \subseteq \mathcal{P} \times \mathcal{P}$ over processes with action in form of $a(t)$ is a weak bisimulation under policy of* urgency, *if $(P, Q) \in S_U$ implies, for all $a(t) \in Act$,*

- *Whenever* $P \xrightarrow{a(t_{min})} P'$ *then, for some* Q', $Q \overset{\widehat{a(t_{min})}}{\Longrightarrow} Q'$ *and* $(P', Q') \in S_U$*;*
- *Whenever* $Q \xrightarrow{a(t_{min})} Q'$ *then, for some* P', $P \overset{\widehat{a(t_{min})}}{\Longrightarrow} P'$ *and* $(P', Q') \in S_U$*.*

where the $\overset{\widehat{a(t_{min})}}{\Longrightarrow}$ *means* $\overset{\widehat{\tau^n}}{\Longrightarrow}\xrightarrow{a(t_{min})}\overset{\widehat{\tau^n}}{\Longrightarrow}$*, and* $\widehat{\tau^n} = 0$ *(the empty sequence).*

By analogy with our treatment of strong equivalence with $\sim_U$, we write this formally as follows using $\approx_U$ to stand for *weak bisimulation* (also as *observation equivalence*) under policy *urgency*.

Theorem 4.8 $\sim_U \Rightarrow \approx_U$

Proof: straight forward. □

V. Conclusions

Process algebras have been studied over 20 years. Researchers have extended them in many aspects, e.g., PEPA [9], SPAs [8], [10], probabilistic PAs [12], [17], [15], [18] and so on. Among all of them, there is one common executing policy: *maximum progress*. This policy is useful but not enough to specify the behaviors of real time systems.

In this paper, we introduced *urgency* executing policy into process algebra to specify the behaviors of real time systems. Urgency executing policy, together with the default policy of *maximal progress*, forms executing policy for real time systems. As for action structure, we use $a(t)$ to stand for action a with parameter t as time limitation. Action $a(t)$ must be executed within limitation t, or, it will be terminated by a timer and the system will activate another action as designed.

We defined the operational semantics and axioms for operators in the language. As per the operational semantics and axioms, we defined equivalent relationship of strong bisimulation and weak bisimulation, thus forming a theory of process algebra for real time systems with execution policy of urgency.

References

[1] J. A. Bergstra and J.W. Klop, *Algebra of Communitating Processes with Abstraction*, TCS 37,1, pp. 77-121, 1985.

[2] P.R. DArgenio, B. Jeannet, H.E. Jensen, and K.G. Larsen, *Reachability analysis of probabilistic systems by successive refinements*, PAPM-PROBMIV 2001, Aachen, Germany (L. de Alfaro and S. Gilmore, eds.), LNCS, vol. 2165, Springer-Verlag, 2001, pp. 29C56.

[3] Colin Fidge. *Proceedings of the Fifth International Conference on Formal Description Techniques*. edited by M. Diaz and R. Groz, Lannion, France, 13C16 October 1992, pp. 355C370.

[4] Harald Fecher , *A Real-Time Process Algebra with Open Intervals and Maximal Progress*, Nordic Journal of Computing, Fall 2001.

[5] A. Giacalone, C.-C. Jou and S.A. Smolka. *Algebraic reasoning for probabilistic concurrent systems*. In M. Broy and C.B. Jones, eds, Proc. of the Working Conf. on Programming Concepts and Methods, pages 443–458. North-Holland, 1990.

[6] M. Majster-Cederbaum, J. Wu. *Towards Action Refinement for True Concurrent Real Time*. Acta Informatica, 39(8), pp. 531-577, 2003.

[7] Holger Hermanns, Ulrich Herzog and Joost-Pieter Katoen, *Process algebra for performance evaluation*, Theoretical Computer Science, 274, pp 43-87, 2002.

[8] H. Hermanns and M. Rettelbach. *Syntax, Semantics, Equivalences, and Axioms for MTIPP*. In U. Herzog and M. Rettelbach, editors, Proc. of the 2nd Workshop on Process Algebras and Performance Modelling, Erlangen-Regensberg, July 1994. IMMD, Universitat Erlangen-Nurnberg.

[9] Jane Hillston, *A Compositional Approach to Performance Modelling*, 1996, Cambridge University Press.

[10] Jane Hillston, *Process algebras for Quantitative Analysis*, 2005, Proceedings of the 20^{th} Annual Symposium on Logic in Computer Science (LICS'05).

[11] C.A.R. Hoare. *Communicating Sequential Process*, Prentice-Hall, 1985.

[12] C-C. Jou and S.A. Smolka. *Equivalences, Congruences and Complete Axiomatizations of Probabilistic Processes*. In J.C.M. Baeten and J.W. Klop, editors, CONCUR'90, volume 458 of LNCS, pages 367-383. Springer-Verlag, August 1990.

[13] J. Markovski and E.P. de Vink, *Embedding Real Time in Stochastic Process Algebras*, Lecture Notes in Computer Science, 4054, pp. 47-62, 2006.

[14] Robin Milner, *Communication and Concurrency*, Prentice Hall, 1989.

[15] M. Mislove, J. Ouaknine and J. Worrell. *Axioms for Probability and Nondeterminism*. In Proc. EXPRESS'03, ENTCS 91(3), 2003.

[16] Fabio Panzieri and Renzo Davoli, *Real Time Systems: A Tutorial*, Performance/SIGMETRICS Tutorials, pp. 435-462, 1993.

[17] M.I.A. Stoelinga and F.W. Vaandrager. *A testing scenario for probabilistic automata*. In J.C.M. Baeten, J.K. Lenstra, J. Parrow, and G.J. Woeginger, editors, Proceedings 30 ICALP, volume 2719 of Lecture Notes in Computer Science, pages 407-418. Springer-Verlag, 2003.

[18] Daniele Varacca, Glynn Winskel. *Distributing probability over nondeterminism* Mathematical Structures in Computer Science archive Volume 16 , Issue 1 (February 2006) Pages: 87 - 113, 2006 ISSN:0960-1295, Cambridge University Press.