Stochastic Process Algebra with Value-Passing and Weak Time Restrictions

Guang Zheng[1,2], Jinzhao Wu[3,4], and Aiping Lu[2,*]
[1] Information Science and Engineering School, Lanzhou University,
Lanzhou 730000, China, Email: forzhengguang@163.com
[2] Institute of Basic Research in Clinical Medicine, China Academy of Chinese Medical Sciences,
Dongzhimen, Beijing 100700, China
[3] Guangxi University for Nationalities, China Academy of Chinese Medical Sciences,
Nanning, 530004, China
[4] School of Computer and Information Technology, Beijing Jiaotong University,
Beijing 100044, China
[*] Corresponding author: lap64067611@126.com

***Abstract*— Process algebra provides essential tools for studying distributed and concurrent systems. Stochastic process algebra (i.e., $\mathcal{YAWN}$) enhances the process algebra with stochastic extensions which is perfect to analyze phenomena of process with executing durations in the real world. What's more, in system runs, value passing is tightly bounded with their processes. However, stochastic process algebras lack value passing can limit their expressiveness. Based on this, we propose a process algebra of stochastic process algebra with value passing. This new process algebra can specify the behaviors of systems in a more clear and accurate way. In dealing with relationship of bisimulations, we introduce a new policy of weak time comparison between processes in bisimulation which is more convenient and doable in practice.**

***Index Terms*— stochastic process algebra, value passing, equivalence, bisimulation, weak time restriction.**

I. INTRODUCTION

Process algebra is a widely accepted language of specifying distributed and concurrent systems. The fundamental work is done by Milner in CCS [23], Hoare in CSP [13] and Hilston in ACP [2].

Stochastic Process Algebras (SPAs) [3], [6], [7], [16], [18] have been invented in the early 90's, the main idea of stochastic process algebras is to incorporate quantitative information in a qualitative process algebra model. In these approaches proposed so far, the quantitative information is given in terms of distribution functions or random variables. These variables denote the duration of actions, and these durations are specified together with actions.

The basic activity of a process is *action*. In *stochastic* process algebras, actions are equipped with stochastic distribution functions which describes the *execution time* of the actions stochastically. SPAs are suitable to describe functional as well as stochastic behaviors in one single specification.

The actions in SPA give the framework specification of system's behaviors. During the execution of SPAs, value passing occurs intuitively and naturally. Processes cooperate with each other by exchanging messages [10], [11], [27], it can be happened in typical operators like:

- *Sequential composition*, where the prefix action might pass the value to the following action for further execution. This happens commonly in programs;
- *Parallel composition*, where a synchronous communication event can be executed. This is the type of communication in SPAs;
- *Recursive operation*, this operator is useful in dealing with repeated actions with certain rules.

SPAs can be models for describing phenomena of the real world in an abstract level. *Value passing* can enhance SPAs with the abilities to describing phenomena in a more detailed level, more intuitive to understand and more doable to put into use. This can be demonstrated by the above examples. In other systems, (for example, traffic control, weather forecasting, scientific calculation, and stock markets), there are values passing with all processes running all time long. Value passing exists at any moment in system runs. With value passing, we can get a inner sight into the phenomena (e.g., under certain situation, values are final results). So, it is *necessary* and intuitively for us to equip the language of SPAs with value passing. By doing this, we can get a better understanding of the phenomena described in SPAs.

The main idea of SPA with value passing is to enhance actions with notions of *duration* and *value passing* during their executions. Durations are described stochastically by means of distribution functions. Values passing during executions can simulate the key parameters in system runs. In *Markovian* SPA, only *exponential distributions* are considered as delay parameters. As for value passing, we assume that only valid values are permitted during the execution, and the invalid values will trigger an exception.

This work was partially supported by the National Eleventh Five Year Support Project of China (2006BAI04A10), the Innovative Methodology Project supported by MOST of China(2008IM020900, 2009ZX09502-019). National Science Foundation of China (No. 30902003, 30973975, 90709007, and 81072982).

doi:10.4304/jsw.6.5.769-782

An exponential distributions is λ and its mean value is $1/\lambda$.

One of the most attractive features of process algebras is their compositional nature. But it is not the only one, another important aspect of the formalism is the definition of equivalence relations i.e., strong and weak bisimulation [22], [23]. These equivalence relations can be used to compare agents (model verification) and to replace one agent by another which exhibits an equivalent behavior but has a simpler representation (model simplification). Such notions of equivalence are considered part of the semantics of the language, and therefore their definition is an integral part of its development.

One important class of equivalence relations in process algebras are bisimulations. Most SPAs [3], [6], [7], [16], [18] provide bisimulation relations both on action and time. They are characterized by the exponential distributions, i.e., λ, μ and so on. Bisimulations of this kind is an extension of the classic bisimulations (strong and weak). However, in practice, even weak bisimulation is too strict. In real world, under certain situations, we might be more tolerant in the execution time under bisimulation relations. Based on this, we propose a weak time restriction bisimulation called *time restricted bisimulation*. This bisimulation relationship is more tolerant in time restrictions when comparing two processes with criterion of bisimulations. We will prove that this *time restricted* bisimulation relationship can be preserved over all the operators in the language of SPA.

Another use of equivalence relations is over the states within a model. When a set of states are found to have equivalent behaviors, we can analyze them by these relationships to partition the state space and considering only one representative of relation to partition these states. Then, only compare one representative of each partition (model aggregation). This is an important way in state space reduction.

This paper is organized as follows. Section 2 introduces the language of $\mathcal{YAWN}$ with value passing, including syntax and its meanings. Section 3 introduces generalized Markovian transition systems with value passing which can be used as models to express the semantics of the language $\mathcal{YAWN}$ with value passing. Section 4 shows operational semantics of the language $\mathcal{YAWN}$ with value passing. Section 5 introduces the axioms of operators in $\mathcal{YAWN}$ with value passing. Section 6 shows some equivalence relations of the language $\mathcal{YAWN}$ with value passing, including strong bisimulation, weak bisimulation, expansion law and time restricted bisimulations. Section 7 concludes the paper. Section 8 lists the acknowledgements.

II. Language of $\mathcal{YAWN}$ with Value

Now, we define the language in the style of $\mathcal{YAMN}$ with value passing. We first define the set L of all process algebra expressions with value passing. An expression $P \in L$ is said to be closed if and only if every process variable, say X, occurring in P occurs within the scope of the $recX$ operator, and if every process constant is defined by a defining equation.

Definition 2.1 ($\mathcal{L}_{\mathcal{YAMN}}$) Let L be the language with *value passing* defined by the following grammar:

$$\begin{aligned} P :: &= 0 \mid X \mid A \mid a_v.P \mid [\lambda].P \mid \text{if } b \text{ then } P \mid P;P \mid \\ &\quad P+P \mid recX:P \mid P \setminus H \mid P||_S P \\ a_v :: &= \mathbf{i} \mid c?x \mid c!e \end{aligned}$$

We use a_v to stand for the generalized form of actions with value in situation no more specification is needed. $\mathbf{i}$ is un-observable actions which likes the τ in CCS; $c?x$ for input action with value x on channel c; and $c!e$ stands for output action with value e on channel c. When it is necessary, we will use $\mathbf{i}$, $c?x$ and $c!e$ to specify actions under different situations.

0 is an *empty* process which cannot perform any actions. It can also be taken as "STOP" in some literature.

$a_v.P$ is *action prefixing*. After executing value passing action a_v, the process $a_v.P$ will behave as P.

$[\lambda].P$ is *prefix delay*. This term means there is a time delay before the execution of process P. The time is characterized by the stochastic variable λ. λ is an exponential distribution parameter, and the mean time of λ is $1/\lambda$. We use $\mathbf{t} = 1/\lambda$ as the label to stand for the time transition, then we have transition in the form of $[\lambda].P \xrightarrow{\mathbf{i}} P$.

$P;Q$ is the *sequential composition* of two processes P and Q. After the execution of process P, the system $P;Q$ behaves as Q.

$P+Q$ is the *choice composition* of two processes P and Q. If process P is selected for execution, then process Q is dropped and have no chance for further execution.

A is used to express $CONST$. We use $CONST$ to express process constants. A constant $C \in CONST$ is assigned a process with value by means of a defining equation $C \stackrel{def}{=} a_v.C'$. The defining equation $A \stackrel{def}{=} a_v.A$ is an example. Intuitively, A is supposed to be the process that can execute infinite number of action a with value $_v$.

$recX:P$ stands for *recursive expression* of processes. With the sequential and choice operators, only finite behavior can be described. As for some reactive systems that generally never terminate, there should be a way to describe them. $recX:P$ is selected to stands for it. In the above example, $recX:P$ behaves as $P[recX:P/X]$, where $P[recX:P/X]$ is the process term where simultaneously all occurrences of X in P are syntactically replaced by $recX:P$.

$P||_S Q$ is *parallel composition* of process P and process Q. Actions in P or Q which are not in set S can be executed independently at the same time without synchronization. However, actions of P or Q that are in set S can only be executed by synchronization.

Example 2.2 Consider the processes of $P \stackrel{def}{=} a_{va}.b_{vb}.c_{vc}.0$ and $Q \stackrel{def}{=} d_{vd}.b_{vb}.e_{ve}.0$, we know that $R \stackrel{def}{=} P||_b Q$ denotes a process in which both P and Q can perform the actions a_{va} and d_{vd} independently. What's more, action b_{vb} can only be proceed in the synchronize

way. After the synchronization, P and Q can proceed again independently, i.e., they can perform actions of c_{vc} and e_{ve} respectively.

Example 2.3 If we consider the process $(P||_b S)||_b R$, where $S \stackrel{def}{=} R \stackrel{def}{=} Q$($Q$ in Example 2.2), then, all three processes can start independently at the same time. However, P, S, and R can only take part in the synchronization over b_{vb} before they can execute their respective last action.

$P \setminus H$ is *hiding* operator. The purpose of this operator is to mark the scope of actions which should never again take part in synchronization. To do this, a special action is introduced, which is often denoted as τ or **i**: the internal action. Reconsider Example 2.3, we can see that process R could be inhibited from participating in the synchronization over b_{vb} that P and S are already involved in. The effect of the hiding operator is that all actions in H are hidden away: they are no longer visible from outside. Then, a process which is synchronized by P and S can be executed, and R proceeds independently from both can be expressed as $(P||_b S) \setminus \{b\}||_{\{\emptyset\}} R$.

if b then P is the one-armed condition if b then _ in the language. With the help of $+$, the conventional two-armed if b then _ else _ expression can be defined by

$$\text{if } b \text{ then } P \text{ else } Q = \text{if } b \text{ then } P + \text{if } \neg b \text{ then } Q$$

In what follows, we will use the if b then _ else _ construction freely without further comments.

We assume that the operators have the following precedence: prefix $>$ recursion $>$ hiding $>$ choice $>$ parallel composition, i.e., prefix has precedence over recursion, recursion over hiding, etc. Parentheses can be used to circumvent these rules. If we have more than two processes combined (i.e., for example, in $P+Q+R$ or $P||_S Q||_{S'} R$ for $P, Q, R \in \mathcal{L}_{\mathcal{YAWN}}$) then we assume a left-associative evaluation order: $P+Q+R$ and $P||_S Q||_{S'} R$ are assume to be equal to $(P+Q)+R$ and $(P||_S Q)||_{S'} R$ respectively. These rules determine a unique evaluation order, which later will become especially important for the application of SOS rules.

Please note that the $\mathcal{YAMN}$ language comes with bells and whistles: we allow to define recursion by means of process constants, and by $recX$ operators with process variables. The only reason for this is to have a more convenient syntax for $\mathcal{YAWN}$.

Frequently, we have to compare elements of the $\mathcal{YAWN}$ language with value passing syntactically. For two terms $P, Q \in \mathcal{L}_{\mathcal{YAWN}}$, we define $P \equiv Q$ if and only if P and Q are syntactically equal for the value assignment for all executing actions with value passing of the same equivalent class leading to the result also of the same equivalent class.

III. Operational Semantics

In classic process algebras, Labeled Transition System (LTS) is used to demonstrate the operational semantics of the language. In this section, we will give out the definition of transition systems with value passing that will demonstrate the operational semantics for the $\mathcal{L}_{\mathcal{YAWN}}$ with value.

A. Transition Systems with Value

The semantics of $\mathcal{YAMN}$ processes is given in terms of transition systems. So, in order to introduce the operational semantics of the language, we introduce the generalized Markovian transition systems.

Definition 3.1 A generalized Markovian transition system with value ($GMTS_V$) is a tuple $(S, A_V, T, \mathcal{R})$, where

- S is a set of states;
- A_V is a set of labels with value;
- $T \subseteq S \times A_V \times S$ is a set of labeled transitions;
- $\mathcal{R} : T \to \mathbb{R} \cup \{\infty\}$

Typical elements of S are $s, s', s'', s_1, s_2, \cdots$, and typical elements of T are $t, t', t'', t_1, t_2, \cdots$. Transitions labeled with $\mathbf{t}$ are meant to be exponentially distributed time delays. The function $\mathcal{R}$ specifies the rates of the distributions. A GMTS is said to be *properly timed*, if whenever $t \in T$ with $t = (s, a, s')$ and $a \in Act$ (i.e., $\forall a, a \neq \mathbf{t}$), then $\mathcal{R}(t) = \infty$. Hence, all internal or visible actions are considered to have no durations, which is expressed by assigning them infinite rates.

Definition 3.2 We define a GMTS with value passing as $(S, A_V, T, \mathcal{R})$ together with a state $s \in S$ (starting state) a generalized Markovian process (GMP). We denote a GMP by a five-tuple $(S, A_V, T, \mathcal{R}, s)$ where A_V is the label of transition with value.

B. Operational semantics

Based on the language discussed in the previous section and the informal explanation of the syntax, we know that our language can describe the behavior of systems with stochastic actions with value passing. Through the Markovian Transition System (MTS), we know that it is convenient to express the semantics of the behavior of such systems. Now, it is ready for us to give out the formalize rules of the language $\mathcal{L}_{\mathcal{YAWN}}$ with operators described in the previous section in table III-B.

Rule (1) expresses the action prefix. Process term $a_v.P$ executes action a_v first, then behave as P.

Rule (2) expresses the delay prefix. Process term $[\lambda].P$ delays time $\mathbf{t}$ and then executes as P. As we restrict the distribution of λ as exponential distribution, it is clear that the mean time of the delay $\mathbf{t}$ is $1/\lambda$.

Rule (3a) and (3c) express the choice composition between two processes with actions influenced by the environment.

Rule (3b) and (3d) express the choice of two delays. Processes P and Q have delays characterized by μ and ν respectively. We do not compare μ and ν, and we know that only if the delay reaches $1/\mu$, and the process P will continue its execution. Similar, when the delay reaches $1/\nu$, process Q will continue, we will have further explanations later by example.

(1)	$\dfrac{}{a_v.P \xrightarrow{a_v} P}$	(2)	$\dfrac{}{[\lambda].P \xrightarrow{\mathbf{t}} P}\ \mathbf{t}=1/\lambda$
(3a)	$\dfrac{P \xrightarrow{a_v} P'}{P+Q \xrightarrow{a_v} P'}$	(3b)	$\dfrac{P \xrightarrow{[\mu]} P',\ Q \xrightarrow{[\nu]} Q'}{P+Q \xrightarrow{\mathbf{t}} P'}\ \mathbf{t}=\dfrac{1}{\mu}$
(3c)	$\dfrac{Q \xrightarrow{a_v} Q'}{P+Q \xrightarrow{a_v} Q'}$	(3d)	$\dfrac{P \xrightarrow{[\mu]} P',\ Q \xrightarrow{[\nu]} Q'}{P+Q \xrightarrow{\mathbf{t}} Q'}\ \mathbf{t}=\dfrac{1}{\nu}$
(3e)	$\dfrac{P \xrightarrow{a} P',\ Q \xrightarrow{[\lambda]} Q'}{P+Q \xrightarrow{a_v} P'}$	(3f)	$\dfrac{P \xrightarrow{[\lambda]} P',\ Q \xrightarrow{a_v} Q'}{P+Q \xrightarrow{a_v} Q'}$
(4a)	$\dfrac{P \xrightarrow{a_v} P'}{P\|\|_S Q \xrightarrow{a_v} P'\|\|_S Q}\ a_v \notin S$	(4b)	$\dfrac{P \xrightarrow{[\nu]} P',\ Q \xrightarrow{[\mu]} Q'}{P\|\|_S Q \xrightarrow{\mathbf{t}} P'\|\|_S Q}\ \mathbf{t}=\dfrac{1}{\nu}$
(4c)	$\dfrac{Q \xrightarrow{a_v} Q'}{P\|\|_S Q \xrightarrow{a_v} P\|\|_S Q'}\ a_v \notin S$	(4d)	$\dfrac{P \xrightarrow{[\nu]} P',\ Q \xrightarrow{[\mu]} Q'}{P\|\|_S Q \xrightarrow{\mathbf{t}} P\|\|_S Q'}\ \mathbf{t}=\dfrac{1}{\mu}$
(4e)	$\dfrac{P \xrightarrow{a_v} P',\ Q \xrightarrow{a_v} Q'}{P\|\|_S Q \xrightarrow{\mathbf{i}} P'\|\|_S Q'}\ a_v \in S$	(4f)	$\dfrac{P \xrightarrow{[\nu]} P',\ Q \xrightarrow{[\mu]} Q'}{P\|\|_S Q \xrightarrow{\mathbf{t}} P\|\|_S Q'}\ \mathbf{t}=f(\nu,\mu)$
(4g)	$\dfrac{P \xrightarrow{a_v} P',\ Q \xrightarrow{[\lambda]} Q'}{P\|\|_S Q \xrightarrow{a_v} P'\|\|_S Q}\ a_v \notin S$	(4h)	$\dfrac{P \xrightarrow{[\lambda]} P',\ Q \xrightarrow{a_v} Q'}{P\|\|_S Q \xrightarrow{a_v} P\|\|_S Q'}\ a_v \notin S$
(4i)	$\dfrac{P \xrightarrow{a_v} P',\ Q \xrightarrow{[\lambda]} Q'}{P\|\|_S Q \xrightarrow{\mathbf{t}} P\|\|_S Q'}\ \begin{pmatrix} a_v \in S, \\ \mathbf{t}=\frac{1}{\lambda} \end{pmatrix}$	(4j)	$\dfrac{P \xrightarrow{[\lambda]} P',\ Q \xrightarrow{a_v} Q'}{P\|\|_S Q \xrightarrow{\mathbf{t}} P'\|\|_S Q}\ a_v \in S, \mathbf{t}=\dfrac{1}{\lambda}$
(5a)	$\dfrac{P \xrightarrow{a_v} P'}{P \setminus H \xrightarrow{a_v} P' \setminus H}\ a_v \notin H$	(5b)	$\dfrac{P \xrightarrow{[\lambda]} P'}{P \setminus H \xrightarrow{\mathbf{t}} P' \setminus H}\ \mathbf{t}=\dfrac{1}{\lambda}$
(5c)	$\dfrac{P \xrightarrow{a_v} P'}{P \setminus H \xrightarrow{\mathbf{i}} P' \setminus H}\ a \in H$	(5d)	$\dfrac{P \xrightarrow{[\mu]} P',\ P' \xrightarrow{a_v} P'', P'' \xrightarrow{[\nu]} P'''}{P \xrightarrow{\mathbf{t}} P'''}\ \begin{pmatrix} a \in H, \\ \mathbf{t}=\frac{1}{\mu}+\frac{1}{\nu} \end{pmatrix}$
(6a)	$\dfrac{P\{recX : P/X\} \xrightarrow{a_v} P'}{recX : P \xrightarrow{a_v} P'}$	(6b)	$\dfrac{P\{recX : P/X\} \xrightarrow{[\lambda]} P'}{recX : P \xrightarrow{\mathbf{t}} P'}\ \mathbf{t}=\dfrac{1}{\lambda}$
(7a)	$\dfrac{P \xrightarrow{a_v} P'}{A \xrightarrow{a_v} P'}\ A \overset{def}{=} P$	(7b)	$\dfrac{P \xrightarrow{[\lambda]} P'}{A \xrightarrow{\mathbf{t}} P'}\ A \overset{def}{=} P,\ \mathbf{t}=\dfrac{1}{\lambda}$

TABLE I.
OPERATIONAL SEMANTICS OF $\mathcal{YAWN}_V$

Rule (3e) and (3f) express the choice between action and delay proposed by two processes. Under the assumption of *maximal execution*, we propose this rule to execute action and left the delay alone.

Rule (4a) and (4c) express the execution of parallel composition of processes where the executing action is not within the scope of synchronization. Under this situation, the executing process just continues its execution and the other processes just waiting for their turns.

Rule (4b) and (4d) express how the paralleled processes trait their delays: each process waits for its time to end the delay and continue its further executions.

Rule (4g) and (4h) express how the paralleled processes with action (no synchronization with others) and delay trait their behavior. The system executes the action, while waiting for the delay at the same time.

Rule (4i) and (4j) express how the paralleled processes with action (synchronize with other process) and delay trait their behavior. The system can not execute the synchronization, it just wait for the end of delay if no synchronization available. Then continue its further executions.

Rule (5a) express how a process with hiding actions executes un-hidden actions, which is intuitive and do not need further explanation.

Rule (5b) express how a process with hiding actions traits delay: it just waits to the end of the delay and then continues its executions.

Rule (5c) express how a process dealing with hiding actions. It just executes the action, however, the execution

cannot be observed from outside. So, according to the definition, we name the action as $\mathbf{i}$.

Rule (5d) tells us the delay of execution of a hiding action. There can be a sum of two delays of the hiding action: *before* μ and *after* ν. So, we have the result $\mathbf{t} = \frac{1}{\mu} + \frac{1}{\nu}$.

Rule (6a) and (6b) express how the recursive terms traits their actions and delays. It is rather intuitive base on explanations of the rules above.

Rule (7a) and (7b) express how a process term assigns to a constant A. They are also intuitive and easy to understand.

Some literatures trait action in SPAs with duration in the form of $([\lambda].a_v)$, which is rather intuitive in the understanding of the execution. We separate them in our language of $\mathcal{YAWN}_V$ as action a_v which do not have durations, and delay $[\lambda]$ which characterize the time between two actions. There is no difference in the essence of the two kinds of expressions. The latter form is more flexible and compact, so we adapt it here.

IV. Axioms

In this section, we propose the axioms of operators in the SPA language with value passing. It is based on the study of operational semantics and the equivalences relations as strong bisimulation and weak bisimulation.

A. Value

As data play an important role in the language of $\mathcal{YAWN}$ with value passing, axiomatization for such process operators must involve dealing with data domain. However, it turns out that, we can factor out data reasoning from process reasoning by employing *conditional equations* [10] of the form

$$b \rhd P = Q$$

where P and Q are process terms and b is a boolean expression representing the condition on the data domain under which P and Q are equal. An example of a proof rule is:

$$\frac{b' \wedge b \rhd P = Q, \quad b' \wedge \neg b \rhd 0 = Q}{b' \rhd \ \mathsf{if}\ b\ \mathsf{then}\ P = Q}$$

It captures the intuitive meaning of the conditional construct: $\mathsf{if}\ b\ \mathsf{then}\ P$ behaves like P when b is true, and like 0 otherwise. In this rule, all we need to know about is construct $\mathsf{if}\ \_\ \mathsf{then}\ \_$ when manipulating syntactical terms. From a "goal-directed" point of view, it moves the parts involving data (b) from the process term $(\mathsf{if}\ b\ \mathsf{then}\ P)$ to the conditional guard part. Such conditions can be used to discharge constructs involving data when some other inference rules are applied.

Reasoning about $\mathcal{YAWN}$ with value passing will inevitably involve the reasoning about data. However, instead of inventing rules for all possible data domains, we would like to factor out reasoning about data from reasoning about processes as much as possible. Therefore our proof system will be parameterized over data reasoning of the form $b \models b'$, with the intuitive meaning that whenever b is true then so is b'.

Now, we present the axioms of data in the language of $\mathcal{L}_{\mathcal{YAWN}}$ in Table II:

This set of inference rules we put forward in Table II can be taken as a natural generalization of pure equational reasoning. For each construct in our language, there is a corresponding introduction rule with a set of axioms.

In this paper, we introduce value passing into the language of $\mathcal{L}_{\mathcal{YAWN}}$. We try to focus on the core meaning of the value passing and not of the kind and quantity of the value.

Example 4.1 There are two testing systems guarded by scores. When the score is greater than 60, the system $s1$ would respond message "PASS". For system $s2$ with the same value, it shows color "GREEN" as a respond. Both the systems obey the same rules, and output results with different kinds of values, and this is very popular in scoring systems in the real world. We treat them as equal in our language for they obey the same rules which can be described as

$$b \models rule_{s1} = rule_{s2} = \mathsf{if}\ score \geq 60\ \mathsf{then}\ true$$

and the result is also of the same equal class that can be described as

$$b' \models \begin{cases} true_{s1} = & \mathsf{PASS} \\ true_{s2} = & \mathsf{GREEN} \end{cases}$$

We omit the input action in the design of the systems designed above. System $s1$ and $s2$ are simple, they can deal with value satisfying condition $\mathsf{if}\ score \geq 60\ \mathsf{then}\ true$. These systems ignore other $score$ and respond nothing according to the condition b.

B. Sequential Composition

Essentially, axioms of prefix and sequential composition are of the same class. They are all sequential operators, and they obey rules (1) and (2).

$(S1)$ $\quad P.0 = P$

$(S2)$ $\quad P.0.Q = P$

$(S3)$ $\quad (P.Q).R = P.(Q.R)$

Axioms of Sequential Composition:A_S

A process will STOP when it encounters with 0. So, the execution of process will stop at the point of 0, and left the other actions aside. Thus, we know that $S1$ and $S2$ are right. As to $S3$, it is rather intuitive to understand, for the parenthesis do not influence the execution sequences of $P.Q.R$.

C. Choice Composition

In this section, we present the axioms of choice composition. They are based on the rules of $(3a)$, $(3b)$, $(3c)$,

(D1)	$\alpha-$ Conversion	$\frac{}{c?x.P = c?y.Q[y/x]} \quad y \notin fv(t)$
(D2)	Premise	$\frac{}{true \rhd P=Q} \quad P = Q$
(D3)	Input	$\frac{b \rhd P = Q}{b \rhd c?x.P = c?x.Q}$
(D4)	Output	$\frac{b \models e = e', \quad b \rhd P = Q}{b \rhd c!e.P = c!e'.Q}$
(D5)	Choice	$\frac{b \rhd P = Q}{b \rhd P + R = Q + R}$
(D6)	Partition	$\frac{b \models b_1 \vee b_2, \quad b_1 \rhd P = Q, \quad b_2 \rhd P = Q}{b \rhd P = Q}$
(D7)	Condition	$\frac{b' \wedge b \rhd P = Q, \quad b' \wedge \neg b \rhd 0 = Q}{b' \rhd \text{ if } b \text{ then } P = Q}$
(D7)	Parallel	$\frac{b \rhd P = Q}{b \rhd P \|\|_S R = Q \|\|_S R}$
(D8)	Hiding	$\frac{b \rhd P = Q}{b \rhd P \setminus H = Q \setminus H}$

TABLE II.
AXIOMS OF VALUE UNDER CONDITION A_D

and $(3d)$.

$$
\begin{array}{ll}
(C1) & \alpha.P + \beta.Q = \beta.Q + \alpha.P \\
(C2) & \alpha.P + [\lambda].Q = \alpha.P \\
(C3) & [\mu].P + [\nu].P = 1/(\mu + \nu).P \\
(C4) & P + (Q + R) = (P + Q) + R \\
(C5) & P + 0 = P \\
 & \text{Axioms of Choice Composition:} A_C
\end{array}
$$

The axioms of choice composition deal with actions and delays separately. $C1$ shows that the exchange of position in choice composition does not affect the execution. $C1$ left the choice for the outside environment. $C2$ shows that the execution policy of *maximal* processes during the execution of choice composition: the system does not wait if there is an action ready for execution. $C3$ shows that when there is a choice between two identical processes with different (exponential) delays, the system would delay as the sum of the two stochastic variables. $C4$ shows that the choice composition among processes with parenthesis does not affect the executing policy of choices. $C5$ shows that the system would select P under the situation of $P + 0$, which means the system can do nothing but P. This is intuitive, for 0 means STOP of the execution. If $P + 0 = 0$ means the system is out of control, and STOPs at wrong point.

D. Internal Action $\mathbf{i}$

We present axioms of *internal* action (i.e., τ in classic process algebra). They are based on the rules of $(5c)$ and $(5d)$.

$$
\begin{array}{ll}
(I1) & \alpha.\mathbf{i}.P = \alpha.P \\
(I2) & P + \mathbf{i}.P = \mathbf{i}.P \\
(I3) & \alpha.(P + \mathbf{i}.Q) + \mathbf{i}.Q = \alpha.(P + \mathbf{i}.Q) \\
 & \text{Axioms of Internal Action:} A_{\mathbf{i}}
\end{array}
$$

The axioms of internal actions are designed for the observable equivalences. When the system is executing an internal action, the action being executed cannot be observed from outside. This is what $I1$ means. $I2$ and $I3$ are rather intuitive: as we cannot tell if there are internal actions being executed, we assume there are *internal* executions in system runs.

E. Parallel Composition

We present axioms of parallel composition here. They are based on the rules of $(4a)$, $(4b)$, $(4c)$, $(4d)$, $(4e)$, and $(4f)$.

$$
\begin{array}{ll}
(P1) & P\|\|_S 0 = P \\
(P2) & P\|\|_S Q = Q\|\|_S P \\
(P3) & (P\|\|_S Q)\|\|_T R = P\|\|_S (Q\|\|_T R) \\
 & \text{Axioms of Parallel Composition:} A_P
\end{array}
$$

Axiom $P1$ means the same as $C5$ ($P+0 = P$): when a process P is paralleled with an empty process, it just executes as P. $P2$ shows that the execution of paralleled processes do not care about the position under parallel composition. That is, communication under parallel composition is preserved. $P3$ shows that the parenthesis of paralleled processes do not affect the execution when it is paralleled with other processes.

F. Hiding Operation

We present axioms of hiding operation here. They are based on the rules of $(5a)$, $(5b)$, $(5c)$, and $(5d)$.

$$
\begin{array}{ll}
(H1) & P \setminus L = P \ \text{ if } Act(P) \notin L \\
(H2) & P \setminus K \setminus L = P \setminus (K \cup L) \\
(H3) & (P||_S Q) \setminus L = P||_{S \cup L} Q \\
(H4) & (P + Q) \setminus L = P \setminus L + Q \setminus L \\
 & \text{Axioms of Hiding Operation:} A_H
\end{array}
$$

Hiding operator is useful in the software engineering. We can take single programs as processes, and the composition of all associated programs so as to form a system which can complete designed functions. The program/process with certain sub-functions usually with input, output, and some other kind of control. When they are compiled into one system (sometime one executive file), most of the programs' input, output, and control are transformed into internal communications which cannot be observed outside.

$H1$ means process P with hiding action set L which contains no action during the execution of P, so the hiding operator cannot affects the P. $H2$ means the function of composition of more than one sets of hiding actions into one hiding action set. $H3$ means that hiding set in parallel composition can be added to the synchronization set. In this rule, the hiding action set is the set that process P and Q will synchronized. $H4$ means that the hiding operator can distribute through the choice composition of processes.

G. Recursive Operation

We present axioms of recursive operator here, which are based on the rules of $(6a)$, $(6b)$, $(7a)$, and $(7b)$.

$$
\begin{array}{ll}
(R1) & rec(X : P) = P\{recX : P/X\} \\
(R2) & If\ P = E\{P/X\}\ then\ P = recX : E/X \\
(R3) & recX : (X = X + P) = recX : P \\
(R4) & recX : (X = \mathbf{i}.X + P) = recX : (\mathbf{i}.P) \\
(R5) & recX : (X = \mathbf{i}.(X + P) + Q = \\
 & recX : (\mathbf{i}.X + P + Q) \\
 & \text{Axioms of Recursive operator:} A_R
\end{array}
$$

$R1$ is rather intuitive, the unwind of $rec(X : P)$ is the substitution of variable X with P such form the recursive expression. $R2$ shows how to define the recursive expression. $R3$, $R4$, and $R5$ are rather intuitive based on the understanding of the rules.

V. Equivalent Relations

From the very beginning, an essential part of process algebra theory has been devoted to the development of equivalence notions. The starting point of all process algebraic equivalences is the observation that different processes may exhibit the same behavior. R.J. van Glabbeek has extensively studies different notions of an experiment that interacts with an interactive process in order to determine its behavior [8], [9]. We consider so called "strong" equivalence, where internal and external actions are treated in the same way. Afterward, we discuss "weak" equivalence, which aims to abstract away internal state/action as much as possible.

In this section, we define $\mathcal{YAMN}$ processes to be equivalent (and substitutive) and their requirements. Since GMP are very similar to IMC transition systems, we adopt the definitions from [14].

The congruences we are going to define are strong Markovian bisimulation and weak Markovian congruence.

A. Strong Markovian Bisimulation

To define strong Markovian bisimulation, we first need a function γ_M that sums up all rates from transitions that start in a single state s and end in some state in a set C.

Definition 5.1 (γ_M) Let $(S, A_V, T, \mathcal{R})$ be a GMTS and for $s \in S$ and $C \subseteq S$, let

$$T_C^s = \{t | t \in \{s\} \times \{t\} \times C\}.$$

Then the function γ_M is defined as

$$\gamma_M : \begin{cases} S \times 2^S & \to \mathbb{R} \\ (s, C) & \mapsto \sum_{t \in T_C^s} \mathcal{R}(t) \end{cases}$$

Example 5.2 Consider a GMTS with states s, s_1, s_2, s_3, s_4, s_5 and states sets of $C_1 = \{s_1, s_2, s_4\}$ and $C_2 = \{s_3, s_5\}$. In Fig.1 (a), we see transitions going from s to s_i for $i = 1, \cdots, 5$. Then, after the cumulative rate of (a), we get

$$\gamma_M(s, C_1) = 2\lambda + \nu \text{ and } \gamma_M(s, C_2) = \mu + \kappa.$$

which is (b).

Definition 5.3 An equivalence relation $R \subseteq \mathcal{L}_{\mathcal{YAWN}} \times \mathcal{L}_{\mathcal{YAWN}}$ is a *strong Markovian bisimulation*. It is a family of symmetric relations $\mathbf{R} = \{R^b \mid b \in BExp\}$ which satisfies: if and only if PR^bQ implies for all $a \in Act_{\mathbf{t}}$ and all equivalence classes C of R^b:

1) If $P \xrightarrow{b_1, a_v} P'$ with $bv(a) \cap fv(b, P, Q) = \emptyset$, then there is a $b \wedge b_1$-partition B with $fv(B) \subseteq fv(b, P, Q)$ such that for each $b' \in B$ there exist b_2, a' and u' with $b' \models b_2$, $a =^{b'} a'$, $Q \xrightarrow{b_2, a'} Q''$ and $P' R^b Q'$;

2) If $P \not\xrightarrow{\mathbf{i}}$ then $\gamma_M(P, C) = \gamma_M(Q, C)$.

$bv(a)$ is the variable through which the value can be carried for execution by action a, i.e., $bv(c?x) = \{x\}$ and $bv(\mathbf{i}) = bv(c!e) = \emptyset$. $fv(a)$ is the value which can be used by action a during its execution.

Two processes P and Q are strongly bisimilar ($P \sim Q$) if they are contained in *strong bisimulation*.

This definition amalgamates strong bisimilarity for stochastic processes with value passing during their executions. In order to compare the stochastic timing behavior, the cumulative rate function γ_M is used. What's

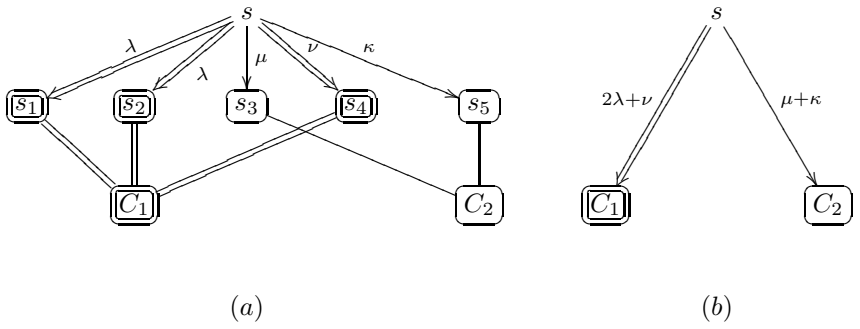

Figure 1. Illustration of Example 5.2

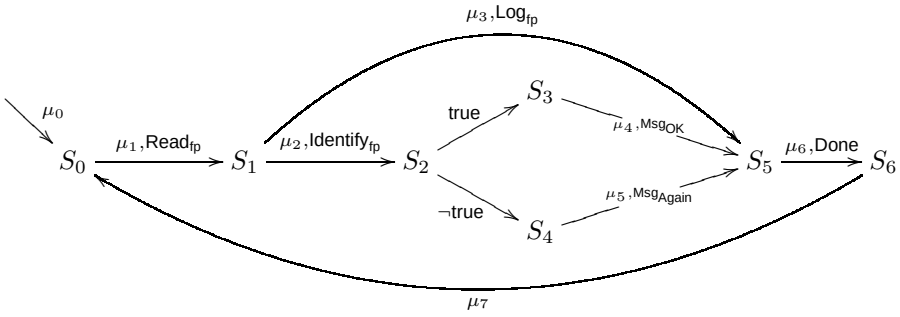

Figure 2. Illustration of Sys_{fp}

more, maximal progress is realized because the stochastic timing behavior is irrelevant for unstable expressions.

Example 5.4 We assume that under certain situation, people have to register themselves either with a card or with their fingerprint to identify their identity so as to get their permissions. There are two register systems available, one system is equipped with a fingerprint reader (short for Sys_{fp}), and the other is equipped with a card reader (short for Sys_{crd}). The systems Sys_{fp} can be specified as:

And the formalized description of the Sys_{fp} is:

$$\begin{aligned}
[\mu_0].Sys_{fp} &:= ([\mu_1].Read_{fp}).Sys'_{fp} \\
Sys'_{fp} &:= (([\mu_2].Identify_{fp})||([\mu_3].Log_{fp})).Sys''_{fp} \\
Sys''_{fp} &:= ((\mathsf{true}).([\mu_4].\mathsf{Msg}_{OK}) + \\
&\quad (\neg\mathsf{true}).([\mu_5].\mathsf{Msg}_{Again})).Sys'''_{fp} \\
Sys'''_{fp} &:= ([\mu_6].\mathsf{Done}).[\mu_7].Sys_{fp}
\end{aligned}$$

The systems Sys_{crd} can be specified in Figure 1.

And the formalized description of the Sys_{crd} is:

$$\begin{aligned}
[\nu_0].Sys_{crd} &:= ([\nu_1].Read_{crd}).Sys'_{crd} \\
Sys'_{crd} &:= (([\nu_2].Identify_{crd})||([\nu_3].Log_{crd})).Sys''_{crd} \\
Sys''_{crd} &:= ((\mathsf{true}).([\nu_4].\mathsf{Msg}_{PASS}) + \\
&\quad (\neg\mathsf{true}).([\nu_5].\mathsf{Msg}_{WrongCard})).Sys'''_{crd} \\
Sys'''_{crd} &:= ([\nu_6].\mathsf{Done}).[\nu_7].Sys_{crd}
\end{aligned}$$

From the Fig.2 and Fig.3, it is intuitive that $Sys_{crd} \sim Sys_{crd}$ during their execution when $\nu_i = \nu_i$ for $i = 0, 1, ..., 7$.

This example also show us that the data of the same action can be different in systems which in the equivalence relation of strong bisimulation, i.e., $Read_{crd}$ can take value passing of *card* while the $Read_{fp}$ can take value passing of *fingerprint*. They belong to different kinds of data, however, they identify the same person and achieve the same goal as well.

Check executing actions, it is clear that $Sys_{fp} \sim Sys_{crd}$. What's more, from the point of value passing under condition of $value(Read_{fp})/b_{fp}$ and $value(Read_{crd})/b_{crd}$, we know that the core "value" of them are equal: $value(Read_{fp})/b_{fp} = value(Read_{crd})/b_{crd}$.

B. Expansion Law

The following law expresses the most basic principle of the operational semantics of process algebras. It states that for each parallel composition of "sums" of processes P (where the choice operator takes the role of the sum here) there exists a process P' such that $P \sim P'$ and P' is the "sum" of parallel compositions. This means that parallelism is not represented explicitly, but encoded by the choice operator.

Definition 5.5 (Expansion Law) Let

$$P = \sum_I [\lambda_i].P_i + \sum_J (b_j, p_j).P_j$$

and

$$Q = \sum_K [\mu_k].Q_k + \sum_L (b_l, q_l).Q_l$$

where i, j, k, l range over the respective index sets I, J, K, L, b_j and b_l stands for the condition of value under which the action p_j and q_l can perform their

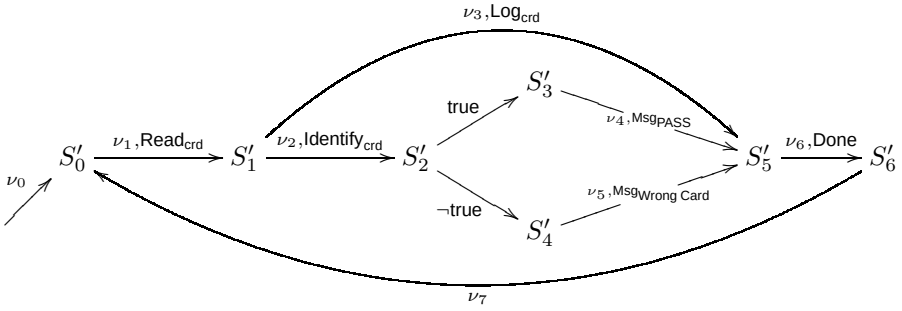

Figure 3. Illustration of Sys_{crd}

executions. Let $S \subseteq Act$. Then

$$\begin{aligned} P||_S Q \quad \sim \quad & \sum_I [\lambda_i].P_i + \sum_J (b_j, p_j).P_j + \\ & \sum_K [\mu_k].Q_k + \sum_L (b_l, q_l).Q_l + \\ & \sum_{\substack{A \cap B \cap S \\ (b_j,p_j)=(b_l,q_l)}} (b_j, p_j).(P_j||_S Q_l) \end{aligned}$$

where $A = \{(b_j, p_j)|j \in J\},\ B = \{(b_l, q_l)|l \in L\}$

Example 5.6 Revisit Example 5.3, we assume that

$$\begin{aligned} P \quad = \quad & (\mathsf{true}).([\nu_4].\mathsf{Msg}_{PASS}) + \\ & (\neg\mathsf{true}).([\nu_5].\mathsf{Msg}_{WrongCard})) \end{aligned}$$

and

$$\begin{aligned} Q \quad = \quad & (\mathsf{true}).([\nu_4].\mathsf{Msg}_{OK}) + \\ & (\neg\mathsf{true}).([\nu_5].\mathsf{Msg}_{Again})) \end{aligned}$$

as time delays characterized by μ_4, μ_5, ν_4 and ν_5 have no influences with boolean expression true and ¬true. We can exchange the position between *delays* and *boolean expressions* by shorten true to b_p and b_q, Msg_{PASS} to M_P, $\mathsf{Msg}_{WrongCard}$ to M_W, Msg_{OK} to M_O and Msg_{Again} to M_A. Then, we have

$$P = [\nu_4].(b_p, M_P) + [\nu_5].(\neg b_p, M_W)$$

and

$$Q = [\mu_4].(b_q, M_O) + [\mu_5].(\neg b_q, M_A)$$

as there is no action for P and Q to synchronize, then it can be expanded as

$$\begin{aligned} P||_\emptyset Q \quad = \quad & ([\nu_4].(b_p, M_P) + [\nu_5].(\neg b_p, M_W))||_\emptyset \\ & ([\mu_4].(b_q, M_O) + [\mu_5].(\neg b_q, M_A)) \\ \sim \quad & [\nu_4].(b_p, M_P) + [\nu_5].(\neg b_p, M_W) + \\ & [\mu_4].(b_q, M_O) + [\mu_5].(\neg b_q, M_A) \end{aligned}$$

We know that ν_4, ν_5, μ_4 and μ_5 are variables of exponential distributions, by the "memoryless" property of Markovian process, at any time point, the time passed cannot influence them. So it is reasonable for $[\nu_4].(b_p, M_P) + [\nu_5].(\neg b_p, M_W) + [\mu_4].(b_q, M_O) + [\mu_5].(\neg b_q, M_A)$ to simulate the execution of $P||_\emptyset Q$.

C. Weak Markovian Congruence

The weak Markovian congruence abstracts away internal actions. To treat internal transitions properly, we need the following definition.

Definition 5.7 (Weak Markovian Bisimulation) An equivalence relation R with $R = \mathcal{L}_{\mathcal{YAWN}} \times \mathcal{L}_{\mathcal{YAWN}}$ is called weak Markovian bisimulation is a family of symmetric relations $\mathbf{R} = \{R^b \mid b \in BExp\}$, and satisfies: iff PR^bQ implies for all $a \in Act$ and all equivalence classes C of R^b:

1) If $P \xrightarrow{b_1, a_v} P'$ with $bv(a) \cap fv(b, P, Q) = \emptyset$, then there is a $b \wedge b_1$-partition B with $fv(B) \subseteq fv(b, P, Q)$ such that for each $b' \in B$ there exist b_2, a' and u' with $b' \models b_2$, $a =^{b'} a'$, $Q \xrightarrow{b_2, \hat{a}'} Q''$ and such that
   - If $a \equiv c?x$ then there is a b'-partition B' such that for each $b'' \in B$ there are b'_2 and Q'' with $b'' \models b'_2$, $Q' \xrightarrow{b'_2, \mathbf{i}} Q''$ and $P' R^{b''} Q''$;
   - otherwise $P' R^{b'} Q'$.
2) $P \xrightarrow{\mathbf{i}} P'$ and $P' \not\xrightarrow{\mathbf{i}}$ imply $\gamma_M(p, C^{\mathbf{i}}) = \gamma_M(q, C^{\mathbf{i}})$

P and Q are called *weakly Markovian bisimulation* equivalent ($P \approx Q$) if there is a weak Markovian bisimulation R such that PRQ.

Example 5.8 Revisit Example 5.3, we can abstract away some internal actions to form a system as which can be observed by the outside observers. First we simplify Sys_{fp} in Figure 4.

Based on Definition 5.7, we know that system Sys_{fp} and Sys_{crd} in the relation of weakly Markovian bisimulation in Fig.4 iff they satisfy: $\mu_i = \nu_i$ for $i = 1, 2, 3, 4$.

In [14], Hermanns proved that $\approx$ is a congruence for all IMC operators (and hence also for $\mathcal{YAMN}$) except choice operator. The reasons for this are well known due to Milner [23], and the deficiency is fixed with the follow definition:

Definition 5.9 (Weak Markovian Congruence) P and Q are said to be *weakly Markovian congruent* ($P \simeq Q$) is a family of symmetric relations $\mathbf{R} = \{R^b \mid b \in BExp\}$, if and only if for all $a \in Act$, all $C \in \mathcal{YAMN}/\approx$:

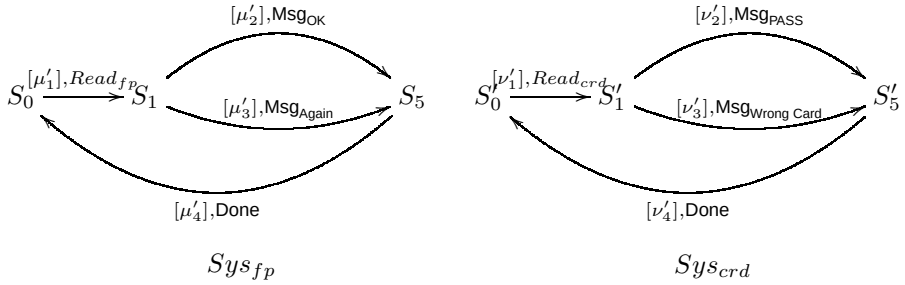

Figure 4. Illustration of simplified Sys_{fp} and Sys_{crd}

1) If $P \xrightarrow{b_1,a_v} P'$ with $bv(a) \cap fv(b,P,Q) = \emptyset$, then there is a $b \wedge b_1$-partition B with $fv(B) \subseteq fv(b,P,Q)$ such that for each $b' \in B$ there exist b_2, a' and u' with $b' \models b_2$, $a =^{b'} a'$, $Q \xrightarrow{b_2,\hat{a'}} Q''$ and such that
   - If $a \equiv c?x$ then there is a b'-partition B' such that for each $b'' \in B$ there are b_2' and Q'' with $b'' \models b_2'$, $Q' \xrightarrow{b_2',\mathbf{i}} Q''$ and $P'R^{b''}Q''$;
   - otherwise $P'R^{b'}Q'$.
2) P stable $\Rightarrow \gamma_M(P,C) = \gamma_M(Q,C)$;
3) P stable $\Leftrightarrow Q$ stable.

We use $P \cong Q$ to stands for the weak Markovian congruence.

Lemma 5.10 If P, Q, and R are processes, and $P \cong Q$, then:

1) $a.P \cong a.Q$, and $[\lambda].P \cong [\lambda].Q$;
2) $P + R \cong Q + R$, and $R + P \cong R + Q$;
3) $P||_S R \cong Q||_S R$, and $R||_S P \cong R||_S Q$;
4) $recX : P \cong recX : Q$.

Proof: All the proofs are alike, and we prove the parallel composition as representation of them.

We all know that the weak bisimulation of CCS [10], [22], which only restrict the bisimulation with pure action and states during the execution. In the definition above, we add another restriction based on the exponential distribution. The restricted exponential distribution can be used to calculate the mean time of the delay or duration of executions.

$\Rightarrow$: From $P \cong Q$ to $P||_S R \cong Q||_S R$, there are several action types available, and we will discuss them one by one:

- For action $a \in S$, then $a \in Act(P)$ and $a \in Act(Q)$ are changed into internal action **i**, and we know that $(P||_S R \cong Q||_S R) \setminus a$ is still $P||_S R \cong Q||_S R$ for $a \in S$;
- For $a \in Act(R)$, $R \cong R$ and $R \xrightarrow{b \triangleright a_v} R'$, it is easy to know that $P||_S R' \cong Q||_S R'$;
- For action $a \notin S \cup Act(R)$:
  - For input action, as $a?x.P' \cong \mathbf{i}.a?x.\mathbf{i}.Q'$ and $P' \cong Q'$, we know that $\gamma(P,P') = \gamma(Q,Q')$, though there are internal action **i**s duration the execution of Q, there is no difference between the execution of $a?x$ both in P and Q, then, we know that $P||_S R \cong Q||_S R$;
  - For output action, as $c!e.P' \cong \mathbf{i}.c!e'.\mathbf{i}.Q'$ and $P' \cong Q'$, we know that $\gamma(P,P') = \gamma(Q,Q')$, though there are internal action **i**s duration the execution of Q, there is no difference between the execution of $c!e$ in P and $c!e'$ in Q, then, we know that $P||_S R \cong Q||_S R$;
  - For internal action **i**, it could not be observed from outside, and there is no change in $P||_S R \cong Q||_S R$, and of course it equals with itself.

$\Leftarrow$: We know that $P||_S R \cong Q||_S R$, and there are several kinds of actions during their execution. We also figure them out one by one:

- For action $a \in Act(R)$ and $R \xrightarrow{b \triangleright a_v} R'$, the situation have nothing to do with $P \cong Q$;
- For action $a \in S$, it is an internal action **i**, and $P||_S R \cong Q||_S R$ evolves into $P'||_S R' \cong Q'||_S R'$, as we know that $(P,P') \in C$ and $(Q,Q') \in C$, and there is no way to calculate the delay or duration, so, it is still unable to know that $P \cong Q$ on action **i**;
- For action $a \notin S \cup Act(R)$
  - For input action, as $a?x.P'||_S R \cong \mathbf{i}.a?x.\mathbf{i}.Q'||_S R$, we know that $\gamma(P||_S R, P'||_S R) = \gamma(Q||_S R, Q'||_S R)$, though there are internal action **i**s duration the execution of Q, there is no difference between the execution of $a?x$ both in $P||_S R$ and $Q||_S R$, then, get ride of R, we get that $P \cong Q$;
  - For output action, as $c!e.P'||_S R \cong \mathbf{i}.c!e'.\mathbf{i}.Q'||_S R$, we know that $\gamma(P||_S R, P'||_S R) = \gamma(Q||_S R, Q'||_S R)$, though there are internal action **i**s duration the execution of Q, there is no difference between the execution of $c!e$ in P and $c!e'$ in Q, then, we know that $P'||_S R \cong Q'||_S R$, since there is no change on R, we have $P \cong Q$;
  - For internal action **i**, it could not be observed from outside, and there is no change in $P||_S R \cong Q||_S R$, and of course it equals with itself.

Based on the analyze above, we complete the proof. □

Example 5.11 Revisit Example 5.8. We know that under the condition that $\mu_i = \nu_i < \infty$ for $i = 1,2,3,4$, system Sys_{fp} is in weak congruence with Sys_{crd} according to the definition 5.7.

D. Time Restricted Markovian Bisimulation

For bisimulation relationships, we think that Examples of 5.3 and 5.8 are perfect for it. Even though, it is hard to keep the system in bisimulation relations in real world systems. We have reason to assume that the subprocess $Identify_{fp}$ in Sys_{fp} does not have the same time as $Identify_{crd}$ in Sys_{crd}, because they use different kinds of devices to get their information. So, it is reasonable for us to assume that μ_2 in Figure 2 does not equal with ν_2 in Figure 3. From this point of view, it is hard to build even a weak bisimulation over Sys_{fp} and Sys_{crd}.

In real world systems, we do not distinguish the two systems if they function well. Then, what makes the distance between bisimulation relations from the real world? The key reason is *time*, which appears in the definition of bisimulations both strong and weak in Markovian relations.

Here, we give out another definition of bisimulation. It is built on the opinion of *time restriction* tr. tr means there is a time restriction/limitation for the execution of certain kind of actions. Usually, we use $tr(P,Q)$ to show the time restriction for process P to involve into Q duration a serial action(s). Use formula $\frac{1}{\gamma_M(P,Q)}$ to describe the mean time of process P involves into Q during a serial action(s). If the execution time is within the restriction, we call it *normal*. Otherwise, we call it *abnormal*. If two systems constructed with the same description of actions, but with different execution durations, we call them bisimulation if their execution times are all within the restriction. In other words, both of the systems can satisfy the requirements on both actions and on *time restrictions*.

According to the bisimulations equivalences of strong and weak, we give out the definitions of strong and weak bisimulation equivalences with time restrictions.

Definition 5.12 (Time Restricted Strong Bisimulation with Value) An equivalence relation $R \subseteq \mathcal{L}_{\mathcal{YAWN}} \times \mathcal{L}_{\mathcal{YAWN}}$ is a *time restricted strong Markovian bisimulation*, is a family of symmetric relations $\mathbf{R} = \{R^b \mid b \in BExp\}$, and satisfies: iff PR^bQ implies for all $a \in Act_{\mathbf{t}}$, time restriction tr and all equivalence classes C of R^b:

1) If $P \xrightarrow{b_1,a_v} P'$ with $bv(a) \cap fv(b,P,Q) = \emptyset$, then there is a $b \wedge b_1$-partition B with $fv(B) \subseteq fv(b,P,Q)$ such that for each $b' \in B$ there exist b_2, a' and u' with $b' \models b_2$, $a =^{b'} a'$, $Q \xrightarrow{b_2,a'} Q''$ and $P'R^bQ'$;
2) If $P \overset{\mathbf{i}}{\not\rightarrow}$ then $\frac{1}{\gamma_M(P,C)} \leq tr(P,C)$ and $\frac{1}{\gamma_M(Q,C)} \leq tr(Q,C)$.

Two process P and Q are strongly bisimilar ($P \sim_{tr} Q$) if they are contained in some *strong bisimulation*.

Example 5.13 Revisit Example 5.3. We know that for $i = 0,1,...,7$, strong bisimulation $Sys_{fp} \sim Sys_{crd}$ means $\mu_i = \nu_i$. From the point of practice, we know that this condition is too hard to satisfy. However, to full fill the design requirements of these two systems, we can restrict the total responding time to no more than TR. That is, systems can accomplish its requirements. Thus, we might specify the systems step by step to split TR into tr_i ($i \in \mathbb{N}$). Then, we might design Sys_{fp} (Sys_{crd}) under condition that $\mu_i \leq tr_i$ ($\nu_i \leq tr_i$) for $i = 0,1,...,7$. The time restriction of the two systems can be illustrated by Fig.5 where tr_i for $i = 0,1,...,7$ are time restrictions for the action above the arrow.

Under the time restrictions specified by Fig.5, we know that $Sys_{fp} \sim_{tr} Sys_{crd}$ iff $\mu_i \leq tr_i$ and $\nu_i \leq tr_i$ for $i = 0,1,...,7$ (which means that $(\mu_i,\nu_i) \in C^i$ where C^i is a serious of equivalent class). This might loosen the definition of strong Markovian bisimulation 5.1 as $\mu_i = \nu_i$ for $i = 0,1,...,7$. However, this change can meet the needs in practice, and it is more practical and easy to control.

This definition can also be explained as follows: all the systems satisfying their design requirements can be taken as equal. When system Sys_{fp} and sys_{crd} working independently, all of them can function well according to the design requirements. That is, one system can take the place of another in practice according to the design requirements, this can also be taken as a kind of equivalence in both algebra and practice.

Definition 5.14 (Time Restricted Weak Markovian Bisimulation) An equivalence relation R with $R \subseteq \mathcal{L}_{\mathcal{YAWN}} \times \mathcal{L}_{\mathcal{YAWN}}$ is called *time restricted weak Markovian bisimulation*, is a family of symmetric relations $\mathbf{R} = \{R^b \mid b \in BExp\}$. It satisfies: iff PR^bQ implies for all $a \in Act$, time restriction tr and all equivalence classes C of R^b

1) If $P \xrightarrow{b_1,a_v} P'$ with $bv(a) \cap fv(b,P,Q) = \emptyset$, then there is a $b \wedge b_1$-partition B with $fv(B) \subseteq fv(b,P,Q)$ such that for each $b' \in B$ there exist b_2, a' and u' with $b' \models b_2$, $a =^{b'} a'$, $Q \xrightarrow{b_2,\hat{a'}} Q''$ and such that
   - If $a \equiv c?x$ then there is a b'-partition B' such that for each $b'' \in B$ there are b_2' and Q'' with $b'' \models b_2'$, $Q' \xrightarrow{b_2',\mathbf{i}} Q''$ and $P'R^{b''}Q''$;
   - otherwise $P'R^{b'}Q'$.
2) $P \xrightarrow{\mathbf{i}} P'$ and $P' \overset{\mathbf{i}}{\not\rightarrow}$ imply $\frac{1}{\gamma_M(p,C^{\mathbf{i}})} \leq tr(p,C^{\mathbf{i}})$ and $\frac{1}{\gamma_M(q,C^{\mathbf{i}})} \leq tr(p,C^{\mathbf{i}})$.

P and Q are called time restricted weak Markovian bisimulation equivalent ($P \approx_{tr} Q$) if there is a weak Markovian bisimulation R such that PRQ.

Example 5.15 When we make clear of the Example 5.13, it is easy to understand this one. Time restricted weak Markovian bisimulation take *input* action $Read_{fp}$ and $Read_{crd}$ as a special case. It is easy to understand in Fig.4 of Example 5.8. *Fingerprint* might take different steps in number to get its information as *card reader*. It is reasonable to assume that *card reader* takes less steps to get its value than *fingerprint*. Because it might take more steps to calculate the value of fingerprint, thus *fingerprint* has more internal action $\mathbf{i}$ than *card reader*. As we know

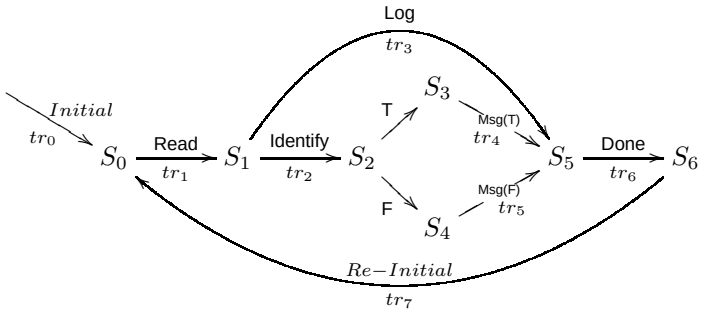

Figure 5. Illustration of Sys_{fp}

that either $Read_{fp}$ in Sys_{fp} or $Read_{crd}$ in Sys_{crd} is restricted by executing time to less than tr_1 (Fig.5). This satisfies condition 2 in definition 5.14.

Checking the execution of Sys_{fp} and Sys_{crd} step by step, we know that they belong to the time restricted weak Markovian bisimulation.

For any different abstract levels of the description of systems, there are different atomic actions. The higher the abstract level, the more abstract atomic actions are required for that level. Thus, atomic actions of higher abstract level contain more internal actions. Another way to turn normal action (observable) into internal action is the composition of compositions into a larger system. The *output* of one component is the *input* of another, thus, at least two normal actions are abstracted away by composition.

Lemma 5.16 Time restricted congruence is a congruence with respect to all operators of $\mathcal{L}_{\mathcal{YAWN}}$. If P, Q and R are expressions of $\mathcal{L}_{\mathcal{YAWN}}$ and $a \in Act$, $\lambda \in \mathbb{R}$ and $X \in Var$, then

- $P \approx_{tr} Q$ implies $a.P \approx_{tr} a.Q$;
- $P \approx_{tr} Q$ implies $[\lambda].P \approx_{tr} [\lambda].Q$;
- $P \approx_{tr} Q$ implies $P + R \approx_{tr} Q + R$ and $R + P \approx_{tr} R + Q$;
- $P \approx_{tr} Q$ implies $P||_S R \approx_{tr} Q||_S R$ and $R||_S P \approx_{tr} R||_S Q$;
- $P \approx_{tr} Q$ implies $recX : P \approx_{tr} recX : Q$.

Proof: All the proofs are alike, and we prove the choice composition as representation of them.

$\Rightarrow$: We suggest that the executing time of P as $tr_P \leq tr_P$, Q as $tr_Q \leq tr_Q$, and R as tr_R as we know that $P \approx_{tr} Q$. So, we get $tr_P = tr_Q$, according to the definition 5.14. We have the executing time of $P + Q$ is $max(tr_P, tr_Q)$, the executing time of $R + Q$ is $max(tr_R, tr_Q)$. Then, we get $max(tr_R, tr_P) = max(tr_R, tr_Q)$. As to the pure actions, it is easy to know that $P = Q$. Then we have $P + R = Q + R$. As the choice composition does not distinguish the position under the summation, then we have $R + P = R + Q$. Put the executing time of actions and pure action together, we have $P + R \approx_{tr} Q + R$.

$\Leftarrow$: We suggest that the executing time of $P + Q$ is $max(tr_P, tr_Q)$. The executing time of $R + Q$ is $max(tr_R, tr_Q)$. As we know that $P + R \approx_{tr} Q + R$, then we have $max(tr_R, tr_P) = max(tr_R, tr_Q)$. Now, there are four situations:

1) $max(tr_P, tr_R) = tr_P$ and $max(tr_Q, tr_R) = tr_Q$, then we have $tr_P = tr_Q$, i.e., we have $P \approx_{tr} Q$ as needed;
2) $max(tr_P, tr_R) = tr_P$ and $max(tr_Q, tr_R) = tr_R$, then we have $tr_P \geq tr_R$ and $tr_R \geq tr_Q$ which is conflict with $max(tr_R, tr_P) = max(tr_R, tr_Q)$. So, this condition is impossible;
3) $max(tr_P, tr_R) = tr_R$ and $max(tr_Q, tr_R) = tr_Q$, then we have $tr_R \geq tr_P$ and $tr_Q \geq tr_R$, which is conflict with $max(tr_R, tr_P) = max(tr_R, tr_Q)$. So this condition is impossible;
4) $max(tr_P, tr_R) = tr_R$ and $max(tr_Q, tr_R) = tr_R$. We know that $max(tr_P, tr_Q) \leq tr_R$, this means that the executing time of P and Q are less than tr_R. As we know that tr_R is assumed to be under the time restriction of the requirements, so we have $P \approx_{tr} Q$.

Based on the above analyze, we get the proof done. □

Based on the assumption that $\frac{1}{\mu_i} \leq tr_i$ ($\frac{1}{\nu_i} \leq tr_i$) for $i = 0, 1, ..., 7$. We know that either the *time delay* before next action or the *duration* of an action is no more than the design requirements which is a serious of real numbers. That is $\frac{1}{\mu_i} \leq tr_i < \infty$ ($\frac{1}{\nu_i} \leq tr_i < \infty$) for $i = 0, 1, ..., 7$. In other words, the variables characterizing executing time in processes cannot be infinite. From the point of Markovian chains, all the chains of this kind is stable. This is a bridge to fill the gap between time restricted weak Markovian bisimulation and time restricted weak Markovian congruence which will be defined in the following.

Definition 5.17 (Time Restricted Weak Markovian Congruence) P and Q are said to be *weakly Markovian congruent* ($P \simeq_{tr} Q$) is a family of symmetric relations $\mathbf{R} = \{R^b \mid b \in BExp\}$. If and only if $\forall a \in Act$, time restriction tr and all $C \in \mathcal{YAMN}/\approx_{tr}$:

1) If $P \xrightarrow{b_1, a_v} P'$ with $bv(a) \cap fv(b, P, Q) = \emptyset$, then there is a $b \wedge b_1$-partition B with $fv(B) \subseteq fv(b, P, Q)$ such that for each $b' \in B$ there exist b_2, a' and u' with $b' \models b_2$, $a =^{b'} a'$, $Q \xrightarrow{b_2, \hat{a}'} Q''$ and such that
   - If $a \equiv c?x$ then there is a b'-partition B' such

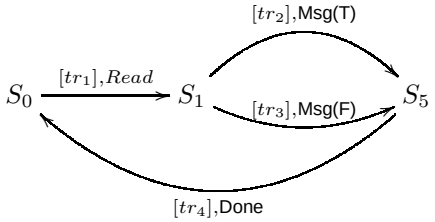

Figure 6. Illustration of simplified Sys_{tr}

that for each $b'' \in B$ there are b_2' and Q'' with $b'' \models b_2'$, $Q' \xrightarrow{b_2',\mathbf{i}} Q''$ and $P'R^{b''}Q''$;
- otherwise $P'R^{b'}Q'$.

2) $\frac{1}{\gamma_M(P,C)} \leq tr(P,C)$, and P is stable;

3) $\frac{1}{\gamma_M(Q,C)} \leq tr(P,C)$, and Q is stable.

Example 5.18 Revisit Example 5.8 and 5.11, we know that system Sys_{fp} and Sys_{crd} obey the same time restriction which can be illustrated in Fig.6.

Fig.6 illustrates the observable actions and durations of the execution of system Sys_{fp} and Sys_{crd}. We know that system $Sys_{fp} \approx Sys_{crd}$ in Example 5.8. $Sys_{fp} \approx_{tr} Sys_{crd}$ satisfies the condition $max(\frac{1}{\mu_i}, \frac{1}{\nu_i}) \leq tr_i$ for $i = 1, 2, 3, 4$.

Example 5.19 If we take the Example 5.8, 5.11, 5.15 and 5.18 as the abstract levels of observations. Take Example 5.3 as a refined level of observation, we can build a time restriction on execution between these two different levels.

If we want Fig. 1 in Example 5.3 to obey the time restriction in Fig. 6, it is intuitive for us to know that for Sys_{fp} as Fig.2 with time restriction as Fig.6 satisfy the following conditions:

$$max(\frac{1}{\mu_0}, \frac{1}{\mu_6} + \frac{1}{\mu_7}, \nu_0, \frac{1}{\nu_6} + \frac{1}{\nu_7}) \leq tr_4;$$
$$max(\mu_1, \nu_1) \leq tr_1;$$
$$max(\frac{1}{\mu_3}, \frac{1}{\nu_2} + \frac{1}{\nu_4}) \leq tr_2;$$
$$max(\frac{1}{\mu_3}, \frac{1}{\nu_2} + \frac{1}{\nu_5}) \leq tr_3.$$

and for Sys_{crd} as Fig.3 with time restriction in Fig.6 satisfies the following conditions:

$$max(\frac{1}{\nu_0}, \frac{1}{\nu_6} + \frac{1}{\nu_7}, \nu_0, \frac{1}{\nu_6} + \frac{1}{\nu_7}) \leq tr_4;$$
$$max(\nu_1, \nu_1) \leq tr_1;$$
$$max(\frac{1}{\nu_3}, \frac{1}{\nu_2} + \frac{1}{\nu_4}) \leq tr_2;$$
$$max(\frac{1}{\nu_3}, \frac{1}{\nu_2} + \frac{1}{\nu_5}) \leq tr_3.$$

Based on the above conditions, we know that $Sys_{fp} \cong_{tr} Sys_{crd}$ and $Sys_{fp} \approx_{tr} Sys_{crd}$. It is intuitive that $\cong_{tr}$ and $\approx_{tr}$ under the time restricted weak Markovian bisimulation and congruence are of the same equivalence class $(\cong_{tr}, \approx_{tr}) \in \equiv$.

Theorem 5.20 $P \approx_{tr} Q \Leftrightarrow P \cong_{tr} Q$.

Proof sketch: The only difference between $P \approx_{tr} Q$ and $P \cong_{tr} Q$ lies in the definitions of 5.14 and 5.17. It is based on whether the Markovian chain is stable or not. Based on the assumption that the mean time of *delays* and executing *durations* are no more than time restriction tr and $tr < \infty$, we know that all actions in Markovian chains can full fill their time restrictions. They terminate within time limitations (i.e., no more than tr). Both *time restricted Markovian bisimulation* and *time restricted Markovian congruence* are based on the assumption that executing time is no more than time restriction.

□

Lemma 5.21 Time restricted congruence is a congruence with respect to all operators of $\mathcal{L}_{\mathcal{YAWN}}$. If P, Q and R are expressions of $\mathcal{L}_{\mathcal{YAWN}}$ and $a \in Act$, $\lambda \in \mathbb{R}$ and $X \in Var$, then

- $P \cong_{tr} Q$ implies $a.P \cong_{tr} a.Q$;
- $P \cong_{tr} Q$ implies $[\lambda].P \cong_{tr} [\lambda].Q$;
- $P \cong_{tr} Q$ implies $P + R \cong_{tr} Q + R$ and $R + P \cong_{tr} R + Q$;
- $P \cong_{tr} Q$ implies $P||_S R \cong_{tr} Q||_S R$ and $R||_S P \cong_{tr} R||_S Q$;
- $P \cong_{tr} Q$ implies $rexX : P \cong_{tr} rexX : Q$.

Proof: Similar with the proof of Lemma 5.16.

□

VI. CONCLUSION

In this paper, we introduced the language of $\mathcal{YAWN}$ with value passing which is perfect to describe the stochastic phenomena in real world. The supporting model of $\mathcal{YAWN}$ is continuous time Markovian chains, which are frequently used in modeling manufacturing system, computer networks, communication systems and so on.

In analyzing the behaviors of complex systems, i.e., computer networks and operating systems, it is inevitable to deal with value passing (i.e., data of all kinds and forms and control based on values). During the analyzing of actions in processes, we introduced value passing into the language of $\mathcal{YAWN}$ with value passing. Thus, there are several kinds of actions with value passing including input, output, internal action, and generalized form.

After we had a general view of the language dealing with stochastic processes, we gave out the syntax of $\mathcal{YAWN}$ with its informal descriptions. In order to define the semantics more clearly, we gave out the formal meaning of the language $\mathcal{YAWN}$. Then, we introduced generalized Markovian transition system (short for GMTS) as the model of $\mathcal{L}_{\mathcal{YAWN}}$. Based on GMTS, we gave out the operational semantics of the language $\mathcal{YAWN}$ with value passing. Based on the theory of classic process algebras, we showed axioms of the operators in the language of $\mathcal{YAWN}$.

The axioms gave out the basic equivalence relations of the basic operators in $\mathcal{YAWN}$ with value passing. One important task for process algebras is to build the equivalent relationships between processes. We treated them in two different ways: one way is to treat both action

and time (delay or duration) strictly. This policy introduce the equivalent relations as strong Markovian bisimulation, weak Markovian congruence and expansion law.

Another way is to treat action and time with different policy: treating action strictly while treating time loosely with a duration (i.e., within time limitations). This means that executing time within time limitation can be considered as equal in the comparison of two processes. This policy produces time restricted strong bisimulation, time restricted weak Markovian bisimulation, and time restricted weak Markovian congruence.

When all the bisimulation relations are defined, we proved that they can be applied to all the operators inside the language of $\mathcal{YAMN}$ with value passing.

REFERENCES

[1] Luca de Alfaro. Stochastic transition systems. *In David Sangiorgi and Robert de Simone, editos, CONCUR '98: Concurrency Theory (Proceedings), volume 1466 of Lecture Notes in Computer Science.*, Springer Verlag, September 2001.

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

[3] M. Bernardo and R. Gorrieri. Extended Markovian Process Algebra. In Ugo Montanari and Vladimiro Sassone, editors, *CONCUR'96: Concurrency Theory (7th International Conference, Pisa, Italy, August 1996)*, volume 1119 of Lecture Notes in Computer Science. Springer, 1996.

[4] M. Bravetti, M. Bernardo, R. Gorrieri, *Generalized Semi Markovian Process Algebra*, Technical Report UBLCS-97-9, University of Bologna (Italy), October 1997

[5] Ed Brinksma Holger Hermanns, Process algebra and Markov chains. In Ed Brinksma, Holger Hermanns, and Joost-Pieter Katoen, editors, Lectures on Formal Methods and Performance Analysis, volume 2090 of Lecture Notes in Computer Science, page 183-231, Springer Verlag, 2001

[6] P. Buchholz, Markovian Process Algebra: Composition and Equivalence. In U. Herzog and M. Rettelbach, editors, *Pro. of the 2nd Workshop on Process Algebras and Performance Modelling* , Erlangen-Regensberg, July 1994. IMMD, Universität Erlangen-Nürnberg.

[7] Norbert Götz, Stochastische Prozeßalgebren – Integration von funktionalem Entwurf and Leistungsbewertung Verteilter Systeme. PhD thesis. *Univerität Erlangen-Nürnberg, Germany*, 1994.

[8] R.J. van Glabbeek, The linear time - branching time spectrum (extended abstract). *CWI Amsterdam Report* CS-R9029.

[9] R.J. van Glabbeek, The linear time - branching time spectrum II: The semantics of sequential systems with silent moves (Extended Abstract) In Eike Best, editor, Fourth International Conference on Concurrency Theory (CONCUR '93, Hildesheim, Germany), volume 715 of LNCS, pp. 66-81, Springer, 1993.

[10] A. Ingólfsdóttir and H. Lin. *A symbolic Approach to value passing Processes*. In J.A. Bergstra, A. Ponse, and S.A. Smolka, eds., *Handbook of Process Algebra*, 427-478, Elsevier, 2001.

[11] M. Hennessy, H. Lin (1996) Proof systems for message-passing process algebras *Formal Aspects of Computing*, 379-407, Volume 8, 1996, Springer London

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

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

[14] A Holger Hermanns. *Interactive Markov Chains*. PhD thesis, Universität Erlangen- Nürnberg, Germany, 1998.

[15] Holger Hermanns and Michael Rettelbach, Towards a superset of LOTOS for performance prediction, In Ribaudo [25], page 77-94, 1996.

[16] H. Hermanns, M. Rettelbach, *Syntax, Semantics, Equivalences, and Axioms for MTIPP*, in Proc. of PAPM '94, pp. 71-87, Erlangen, 1994

[17] H. Hermanns, U. Herzog, and J.-P. Katoen. *Process algebra for performance evaluation*. Theoretical Computer Science, 274(1-2):43–87, 2002.

[18] Jane Hillston, A Compositional Approach to Performance Modelling, *PhD thesis, University of Edinburgh*, 1994.

[19] Ronald A. Howard. Dynamic Probabilistic Systems. *volume 2: Semimarkov and Decision Processes*, John Wiley & Sons, 1971.

[20] J.P. Katoen. Concepts, Algorithms and Tools for Model Checking. Erlangen: Institut f. Mathematische Maschinen und Datenverarbeitung, 1999. 188-255.

[21] J-P. Katoen. Quantitative and qualitative extensions of event structures, *PhD thesis*, University of Twente, 1996.

[22] Robin Milner, A Calculus of Communicating Systems, volume 92 of Lecture Notes in Computer Science. *Prentice Hall International*, 1989.

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

[24] Martin L. Puterman. *Markovian Decision Processes*. John Wiley & Sons, 1994.

[25] Marina Ribaudo, editor. Proceedings of the fourth workshop on process algebras and performance modelling. Edizione C.L.U.T. Torino, 1996.

[26] Moshe Y. Vardi. *Automatic verification of probabilistic concurrent finite-state systems*. In *26th Annual Symposium on Foundations of Computer Science (FOCS '85)*, pages 327-338. IEEE Computer Society Press, October 1985.

[27] Guang Zheng, Shaorong Li, Jinzhao Wu, and Lian Li. A Non-interleaving Denotational Semantics of Value Passing CCS with Action Refinement. *FAW 2007, LNCS 4613, pp. 178-190, 2007.*, Springer-Verlag Berlin Heidelberg 2007.