A Stochastic Broadcast π-Calculus*

Lei Song
Programming, Logic, and Semantics Group
IT University of Copenhagen, Denmark
leis@itu.dk

Flemming Nielson
Department of Informatics and Mathematical Modeling
Technical University of Denmark
nielson@imm.dtu.dk

Bo Friis Nielsen
bfn@imm.dtu.dk

In this paper we propose a stochastic broadcast π-calculus which can be used to model server-client based systems where synchronization is always governed by only one participant. Therefore, there is no need to determine the joint synchronization rates. We also take immediate transitions into account which is useful to model behaviors with no impact on the temporal properties of a system. Since immediate transitions may introduce non-determinism, we will show how these non-determinism can be resolved, and as result a valid CTMC will be obtained finally. Also some practical examples are given to show the application of this calculus.

1 Introduction

Process algebras such as CCS [19], CSP [17], and ACP [2] have been successfully used to model and analyze concurrent systems. The system behavior of these classical process algebras is usually given by *labeled transition systems* (LTS) which have proved to be a convenient framework for analyzing *qualitative* properties of large complex system. As these models are only concerned about functional aspects of concurrent systems, process algebras have been extended with stochastic variables in order to model performance-oriented systems in recent years. Such examples include TIPP [11], PEPA [15], EMPA [6], stochastic π-calculus [20], IMC [13], StoKlaim [8], and Stochastic Ambient Calculus [21]. The semantics of these models are given by a variant of LTS, *Continuous Time Markov Chain* (CTMC), which can be used to analyze *quantitative* properties directly. Each transition in a CTMC is associated with an exponentially distributed random variable which specifies the duration of this transition. The underlying CTMC captures the necessary information for both functional verification and performance evaluation.

Synchronization in stochastic scenarios have been addressed in [15, 13, 1] using different techniques. In this paper we develop a stochastic broadcast π-calculus aiming at modeling server-client based systems which are used widely in practice. In such systems synchronization are always governed by one participant, so there is no need to determine synchronization rates like others. In our calculus only outputs are associated with rates and their durations are exponentially distributed while inputs are always passive. We all know that the nondeterministic choices among outputs can be resolved by *race conditions* probabilistically. Similarly, to resolve nondeterministic choices among inputs, we let each input be associated with a weight as usual and the probability of an input receiving a message is determined by its weight and the total weight of all current inputs. In addition the communication in our calculus is based on broadcast, that is, when one component outputs a message, it will be received by all the recipients instead of only one of them. Such scenarios can be found in practice very often. For example considering the checking out in a supermarket, the arrivals of customers can be assumed to be exponentially distributed. When a customer comes to the counters, he/she will choose different counters according to

*Supported by the VKR Center of Excellence MT-LAB.

M. Massink and G. Norman (Eds.): 9th Workshop on
Quantitative Aspects of Programming Languages (QAPL 2011)
EPTCS 57, 2011, pp. 74–88, doi:10.4204/EPTCS.57.6

the lengths of their queues, the longer the queue the less likely it will be chosen, meanwhile when the customer is checking out, not only the counter knows it but also other departments will know and react accordingly such as financial, purchasing and so on.

To enhance the expressiveness of our calculus we also take *immediate actions* into account. The immediate actions will happen instantaneously and have been studied in [5, 11, 14]. They are useful to describe certain management and control activities which have no impact on the temporal behavior of a system. Since immediate action takes no time to execute, so race condition does not apply here. Instead we assign each immediate action a weight to resolve nondeterministic choices between immediate transitions which is similar as inputs. For instance activities such as "when the buffer of a server is full, the coming clients will be transferred to another server instantaneously" can only be modeled by using immediate actions. In this paper we give several classical models from performance analysis which can be modeled in a compositional way by making use of immediate transitions. Accordingly, we will call the non-immediate transitions (resp. actions) *Markovian transitions (resp. actions)* in the sequel.

Usually the problem of immediate action is that the existence of an underlying CTMC can no longer be guaranteed. In this paper, we solve this in two ways. As usual immediate transitions take no time and should have priority over Markovian transitions, so when an immediate transition is available, it will block the executions of Markovian transitions. We divide the whole process space into two sets: *Immediate Processes* (IP) and *Markovian Processes* (MP). IP only contains processes where at least one immediate transition is available and MP contains processes where no immediate transition is available. Since immediate transitions can exempt the execution of Markovian transitions, we can say that states in IP can only perform immediate transitions. All states in a CTMC will belong to MP. To calculate the rate from P to P' in a CTMC, we accumulate the rates of all the possible transitions from P to P' where transitions might be via states in IP. Sometimes it is possible for a process reaching a state which and all its derivations belong to IP. In this case, no time is allowed to elapse and the process is said to be *absorbing*. We use a special state Stuck to denote such situation and show how a CTMC can be obtained even with the existence of immediate actions.

Similar with the existing calculi whose semantics are given by LTS, we also give the LTS for our calculus. Differently, each Markovian transition in our LTS is labeled by a rate instead of an action. For example, a typical transition looks like $P \xrightarrow{\lambda} \mathbb{AP}$ where λ denotes that the execution time of this transition is exponentially distributed with rate λ and $\mathbb{AP}$ is a distribution over pairs of action and process (α, Q). Intuitively, if $P \xrightarrow{\lambda} \mathbb{AP}$, that means that P will leave its original state with rate λ (sojourn time of P is exponentially distributed with rate λ) and get to Q via action α with probability p if the probability of (α, Q) in $\mathbb{AP}$ is equal to p. By defining an LTS in this way, the correspondent CTMC can be obtained in a natural way. It is worth mentioning that our framework could also be used as an alternative general way to specify the LTS as rate-base transition systems [9]. Without relying on different techniques, for example multi relations, proved transition systems and unique rate names used in PEPA, stochastic π-calculus, and StoKlaim respectively, we can have a uniform way to define the underlying models for these stochastic calculi.

The paper is organized as follows: the syntax of our calculus is presented in the next section and in Section 3 we give the Labeled Transition System. In Section 4 we illustrate the use of immediate transitions by giving a few examples. We show how to get the underlying CTMC even with existence of immediate transitions in Section 5. Finally, we end by concluding and describing the future work.

2 Syntax

Before introducing our calculus, we first give the following general definition of probability space. A probability space is a triplet $\mathcal{P} = (\Omega, F, \eta)$ where Ω is a set, F is a collection of subsets of Ω that includes Ω and is closed under complement and countable union, and $\eta : F \to [0,1]$ is a probability distribution function such that $\eta(\Omega) = 1$ and for any collection $\{C_i\}_i$ of at most countably many pairwise disjoint elements of F, $\eta(\cup_i C_i) = \sum_i \eta(C_i)$. A probability space (Ω, F, η) is discrete if Ω is countable and $F = 2^\Omega$, and hence abbreviated as (Ω, η). Given probability spaces $\{\mathcal{P}_i = (\Omega_i, \eta_i)\}_{i\in I}$ and weights $w_i > 0$ for each i such that $\sum_{i\in I} w_i = 1$, the *convex combination* $\sum_{i\in I} w_i\mathcal{P}_i$ is defined as the probability space (Ω, η) such that $\Omega = \bigcup_{i\in I}\Omega_i$ and for each set $Y \subseteq \Omega$, $\eta(Y) = \sum_{i\in I} w_i\eta_i(Y \cap \Omega_i)$. Usually, we use $\{\rho_i : P_i\}_{i\in I}$ to denote a probability space $\mathcal{P} = (\{P_i\}_{i\in I}, \eta)$ such that $\eta(\{P_i\}) = \rho_i$, here I is a countable index set. *Dirac* probability space $\{1 : P\}$ will be written as P directly in the sequel. If $\sum_{i\in I}\rho_i \leq 1$ then we call it a *sub probability space*. We also use $\mathcal{P}(P_i) = \rho_i$ to denote the probability of P_i in $\mathcal{P}$. The summation and parallel between two sub probability spaces can be defined in a natural way as follows:

$$\mathcal{P}_1 + \mathcal{P}_2 = \{\rho_1 + \rho_2 : P \mid \mathcal{P}_1(P) = \rho_1 \wedge \mathcal{P}_2(P) = \rho_2 \wedge \mathcal{P}_1(\Omega_1) + \mathcal{P}_2(\Omega_2) \leq 1\},$$

$$\mathcal{P}_1 \parallel \mathcal{P}_2 = \{\rho_1 \times \rho_2 : P_1 \parallel P_2 \mid \mathcal{P}_1(P_1) = \rho_1 \wedge \mathcal{P}_2(P_2) = \rho_2\}.$$

Note in the above $\mathcal{P}_1(\Omega_1) + \mathcal{P}_2(\Omega_2) \leq 1$ is used to guarantee that $\mathcal{P}_1 + \mathcal{P}_2$ is still a valid sub probability space.

We presuppose a countable set $\mathscr{C}$ of constants and a countable set $\mathscr{V}$ of variables ranged over by $a, b, c\ldots$ and $x, y, z\ldots$ respectively such that $\mathscr{C} \cap \mathscr{V} = \emptyset$. $n, m, l\ldots \in \mathscr{C} \cup \mathscr{V}$ are called names. The syntax of processes is given as follows where $\lambda \in \mathbb{R}_{>0}$ is the exponential rate and $w \in \mathbb{R}_{>0}$ is the weight of the input action. When the rate of an output is infinite, it is an *immediate* action which takes no time for it to be performed. We use ∞_w to denote an infinite rate with weight w. In the following $\tilde{\lambda}$ is used to denote either λ or ∞_w and λ_0 ranges over exponential rates as well as 0, that is, $\lambda_0 \in \mathbb{R}_{\geq 0}$. It is obvious that every output action must be prefixed by an exponential rate and every input action has a specified weight, if the rate of an output is infinite then it will be assigned with a weight instead. We assume that there is a countable set of constants, ranged over by A, which are used to denote processes. By giving an equation such that $A \stackrel{def}{=} P$ we say that constant A will behave as P, here A is required to be guarded in P, i.e. every constant appearing in P has to be prefixed by Act. We only consider closed processes here and use $P, Q\cdots$ to range over closed processes $\mathscr{P}$.

$$\mathsf{Act} ::= n(x, w) \mid n\langle m, \tilde{\lambda}\rangle$$

$$P, Q ::= 0 \mid \mathsf{Act}.P \mid \nu aP \mid P + Q \mid [n = m]P, Q \mid P \parallel Q \mid A$$

A substitution $\{a/x\}$ can be applied to a process or process distribution. When applied to a process distribution, it means applying this substitution to each process with probability greater than 0 in it. The set of free names and bound names in P, denoted by $fn(P)$ and $bn(P)$ respectively, are defined as expected and $n(P) = fn(P) \cup bn(P)$ denotes the set of all the names in P. Structural congruence, $\equiv$, is the least equivalence relation and congruence closed by the rules in Table 1 and α-conversion. $\equiv$ is also extended to network distributions as usual.

3 Semantics

The actions of processes $\mathcal{A}$, ranged by $\alpha, \beta\cdots$, are defined by

$$\alpha ::= a(x) \mid \nu\tilde{b}a\langle b\rangle \mid \tau$$

Table 1: Structural Congruence

$$\nu a \nu b P \equiv \nu b \nu a P \quad P + Q \equiv Q + P \quad P \parallel Q \equiv Q \parallel P \quad [a = a]P, Q \equiv P$$
$$[a = b]P, Q \equiv Q \; a \neq b \quad (\nu a P) \parallel Q \equiv \nu a(P \parallel Q) \; a \notin fn(Q)$$

Table 2: Function γ evaluating weight of input on a given channel

$$\begin{array}{lcl}
\gamma(a, 0) & = & 0 \\
\gamma(a, \mathsf{A}ct.P) & = & \begin{cases} w & \mathsf{A}ct = a(x, w) \text{ for some } x \\ 0 & otherwise \end{cases} \\
\gamma(a, \nu b P) & = & \begin{cases} 0 & a = b \\ \gamma(a, P) & otherwise \end{cases} \\
\gamma(a, P + Q) & = & \gamma(a, P) + \gamma(a, Q) \\
\gamma(a, [b = c]P, Q) & = & \begin{cases} \gamma(a, P) & b = c \\ \gamma(a, Q) & b \neq c \end{cases} \\
\gamma(a, P \parallel Q) & = & \gamma(a, P) + \gamma(a, Q) \\
\gamma(a, A) & = & \gamma(a, P) \qquad A \stackrel{def}{=} P
\end{array}$$

Here $\tilde{b}$ is a set of constants, when $b \in \tilde{b}$, b is bounded, otherwise it is free. The functions fn, bn, and n can be lifted from processes to actions as usual.

To evaluate the total weight of inputs on a given channel in a process, we define function $\gamma : \mathscr{C} \times \mathscr{P} \to \mathbb{R}_{>0}$ as Table 2.

Similarly, we also give the function $\mu : \mathscr{C} \times \mathscr{P} \to \mathbb{R}_{>0}$ to evaluate the total rate of outputs on a given channel in a process which is defined in the Table 3. To evaluate the weight of outputs with infinity rates, we define function $\mu_\infty : \mathscr{C} \times \mathscr{P} \to \mathbb{R}_{>0}$ which is the same as μ except that:

$$\mu_\infty(a, b\langle m, \tilde{\lambda}\rangle.P) = \begin{cases} w & \tilde{\lambda} = \infty_w \wedge a = b \\ 0 & \text{otherwise} \end{cases}$$

In addition, $\mu(P) = \sum_{a \in \mathscr{C}} \mu(a, P)$ and $\mu_\infty(P) = \sum_{a \in \mathscr{C}} \mu_\infty(a, P)$ are used to evaluate the total rate and total weight associated with infinite rates of outputs in a process.

We define *process distribution*, ranged over by $\mathbb{P}, \mathbb{Q} \ldots$, as a probability space where $\Omega = \mathscr{P}$. Similarly, *process action distribution* can be defined as a probability space where $\Omega = \mathscr{A} \times \mathscr{P}$. We use $\mathbb{AP}, \mathbb{AQ} \ldots$ to range over process action distributions. The set of all the actions in $\mathbb{AP}$ is defined by $\mathscr{A}(\mathbb{AP}) = \{\alpha \mid \exists P.\mathbb{AP}(\alpha, P) > 0\}$ while the corresponding sub process distribution of α in $\mathbb{AP}$ is denoted by $\mathbb{AP}(\alpha) = \{\rho : P \mid \mathbb{AP}(\alpha, P) = \rho > 0\}$. We will write $\mathbb{AP}$ as $(\alpha, \mathbb{P})$ if $\mathscr{A}(\mathbb{AP}) = \{\alpha\}$ where $\mathbb{P} = \mathbb{AP}(\alpha)$. In addition, we use $\mathbb{AP}(P) = \sum\limits_{\alpha \in \mathscr{A}} \mathbb{AP}(\alpha, P)$ to denote the total probability of P in $\mathbb{AP}$.

We lift new operator to process action distributions in (1). If the channel is restricted, then the broadcast action will change to τ; if the message is restricted, then the broadcast action will be updated accordingly; otherwise the broadcast will stay unchanged while the new operator will be put on the result

Table 3: Function μ evaluating rate on a channel of a process

$$
\begin{array}{lcl}
\mu(a,0) & = & 0 \\
\mu(a,b\langle m,\tilde{\lambda}\rangle.P) & = & \begin{cases}\lambda & \tilde{\lambda}=\lambda \wedge a=b \\ 0 & \text{otherwise}\end{cases} \\
\mu(a,n(x,w).P) & = & 0 \\
\mu(a,\nu bP) & = & \begin{cases}0 & a=b \\ \mu(a,P) & otherwise\end{cases} \\
\mu(a,P+Q) & = & \mu(a,P)+\mu(a,Q) \\
\mu(a,[b=c]P,Q) & = & \begin{cases}\mu(a,P) & b=c \\ \mu(a,Q) & b\neq c\end{cases} \\
\mu(a,P \parallel Q) & = & \mu(a,P)+\mu(a,Q) \\
\mu(a,A) & = & \mu(a,P) \qquad A \stackrel{def}{=} P
\end{array}
$$

process.

$$
\begin{aligned}
\nu a\mathbb{AP} =& \{\rho : (\tau,\nu aP) \mid \mathbb{AP}(\nu\tilde{b}a\langle b\rangle,P)=\rho\} \\
&\cup \{\rho : (\nu ab\langle a\rangle,P) \mid \mathbb{AP}(b\langle a\rangle,P)=\rho \wedge a\neq b\} \\
&\cup \{\rho : (\alpha,\nu aP) \mid \mathbb{AP}(\alpha,P)=\rho \wedge a\notin fn(\alpha)\}
\end{aligned} \tag{1}
$$

The semantics of our calculus is shown in Table 4 where I and J are finite index sets. We use $\rightsquigarrow$ to denote $\longrightarrow$ or $- - \succ$, where $\xrightarrow{\lambda}$ is the *Markovian Transition* with rate λ, and $-\stackrel{w}{-}\succ$ is the *Immediate Transition* with rate infinity and weight w. A transition with rate 0 $\xrightarrow{0}$ is called a *passive transition* [15]. All the transitions have the form $P \overset{\lambda_0}{\rightsquigarrow} \mathbb{AP}$ which means that P will evolve into process Q by performing action α with probability ρ if $\mathbb{AP}(\alpha,Q)=\rho$. In addition, when it is a Markovian transition with rate λ, it means that P will leave to other states with rate λ or the duration of the transition is exponentially distributed with rate λ. It is not hard to see from the semantics that for a Markovian transition, all the actions in the resulting distribution $\mathcal{A}(\mathbb{AP})$ are either outputs or τ actions, while for the passive transitions, $\mathcal{A}(\mathbb{AP})$ only contains an input action, therefore can be written as $(a(x),\mathbb{P})$ where $\mathbb{P}=\mathbb{AP}(a(x))$. Rule (REC) means that process $a(x,w).P$ can receive a message on channel a and then evolve into P with probability 1. Similarly, in rule (mBRD) $a\langle b,\lambda\rangle.P$ will leave to other states with rate λ and evolve into P by broadcasting the message b on channel a with probability 1, this is a Markovian transition. If the rate of output is infinite, it should be performed instantly. This is called immediate transition which is shown by (iBRD). The weight associated with the infinite rate is used to resolve nondeterministic choices as in input actions. Rule (RES) only applies to Markovian and immediate transitions, since $\lambda>0$ can guarantee that the transition is not passive. The new operator on process action distribution is defined by (1). By definition of γ in Table 2, if $\gamma(a,P)=0$ which means P is not ready to receive messages on channel a, in this case P will ignore all the messages broadcasted on channel a. This results in rule (LOS). Every input action is associated with a weight which can be used to resolve nondeterministic choices among different input actions probabilistically. For example after receiving a message b on channel a, $a(x,2).P_1+a(x,1).P_2$ will evolve into $P_1\{b/x\}$ with probability $\frac{2}{2+1}$ and $P_2\{b/x\}$ with probability $\frac{1}{2+1}$. This is shown in (SUM1). (PAR1) is straightforward since our calculus is based on broadcast. Two parallelized processes will evolve together after receiving a message on a certain channel. Intuitively, when we put processes P_1 and P_2 together, the compositional process

P will leave to other states with rate $\lambda_1+\lambda_2$ if the rates of P_1 and P_2 for leaving their original states are λ_1 and λ_2 respectively. Whether P_1 or P_2 will be executed first depends on the race condition, that is, P_1 will be executed before P_2 with probability $\frac{\lambda_1}{\lambda_1+\lambda_2}$ and the probability for the other case is $\frac{\lambda_2}{\lambda_1+\lambda_2}$. This is captured by rules (SUM2) and (PAR2) when $\rightsquigarrow = \longrightarrow$. In (PAR2) when $\rightsquigarrow = \longrightarrow$ we also need to consider all the possible synchronization between P and Q. For example, if P can evolve into a sub process distribution $\mathbb{AP}(\nu\tilde{b_i}a_i\langle b_i\rangle)$ after action $\nu\tilde{b_i}a_i\langle b_i\rangle$ and Q will evolve into $\mathbb{Q}_i\{b_i/x\}$ after receiving b_i on channel a_i, then $P \parallel Q$ will evolve into sub process distribution $\mathbb{AP}(\nu\tilde{b_i}a_i\langle b_i\rangle) \parallel \mathbb{Q}_i\{b_i/x\}$ by performing action $\nu\tilde{b_i}a_i\langle b_i\rangle$ after leaving from the original state ($\nu\tilde{b_i} \cap fn(Q) = \emptyset$). Since P and Q may have several outputs available at the same time, we need to list all the possible synchronization and then add all the resulting sub process action distributions to form the final result. The following example is to show how (PAR2) works.

Example 1 *Suppose we have two processes:* $P = n\langle y,3\rangle \parallel (m(x,2).P_1 + m(x,4).P_2)$ *and* $Q = m\langle z,2\rangle \parallel n(x,1).Q_1$. *By the semantics,* $P \xrightarrow{3} \{1 : (n\langle y\rangle, m(x,2).P_1 + m(x,4).P_2)\} \equiv (n\langle y\rangle, \mathbb{P})$ *and* $Q \xrightarrow{2} \{1 : (m\langle z\rangle, m(x,1).Q_1)\} \equiv (m\langle z\rangle, \mathbb{Q})$. *When we put the two processes in parallel, we have to consider all possible synchronization between them. P can broadcast y on channel n and Q can broadcast z on channel m, in the meanwhile P can receive a message on channel m and Q can receive a message on channel n, formally,*

$$P \xrightarrow{0} \left\{\begin{array}{ll} \frac{1}{3}: & (m(x), n\langle y,3\rangle \parallel P_1) \\ \frac{2}{3}: & (m(x), n\langle y,3\rangle \parallel P_2) \end{array}\right\} \equiv (m(x), \mathbb{P}'),$$

$$Q \xrightarrow{0} \{1 : (n(x), m\langle z,2\rangle \parallel Q_1)\} \equiv (n(x), \mathbb{Q}').$$

In $P \parallel Q$, either P or Q will broadcast a message first, and the non-determinism is resolved probabilistically by race condition, i.e. $P \parallel Q$ will perform $n\langle y\rangle$ first with probability $\frac{3}{5}$ and the probability of $m\langle z\rangle$ being executed first is $\frac{2}{5}$. When $n\langle y\rangle$ is performed, Q will receive it and evolve into $\mathbb{Q}'\{y/x\}$. Similarly, when $m\langle z\rangle$ is executed, P will evolve into $\mathbb{P}'\{z/x\}$ accordingly. So

$$P \parallel Q \xrightarrow{5} \frac{3}{5} \times (n\langle y\rangle, \mathbb{P} \parallel \mathbb{Q}'\{y/x\}) + \frac{2}{5} \times (m\langle z\rangle, \mathbb{Q} \parallel \mathbb{P}'\{z/x\}).$$

Rules (SUM3) and (PAR3) are similar with rules (SUM2) and (PAR2), but they only apply to processes where Q can only have a passive transition (with label 0), this is guaranteed by $\mu(Q) = 0$. (SUM3) and (PAR3) cannot be omitted since in (SUM2) and (PAR2) both P and Q are required to have non-passive transition, while in (SUM3) and (PAR3) only one of them has non-passive transition. The arguments for these rules when $\rightsquigarrow = - - \succ$ are similar. Rules (CON) and (STR) are standard and need no more comments.

From the syntax and semantics we know that the nondeterministic choices among Markovian transitions can be resolved by a race condition while both the nondeterministic choices among immediate outputs and inputs can be resolved based on their weights. But still there might be nondeterministic choices during the evolution of a process, such as nondeterminism between passive transitions and Markovian transitions and nondeterminism between immediate transitions and Markovian transitions. These nondeterminism can be resolved easily since we assume that immediate transitions can preempt other transitions while passive transitions should not be considered when talking about the underlying CTMC of a process. We will talk about this in details in Section 5.

In this section we will not discuss immediate transitions, we leave it to the next section. The following simple example is to show how to get a CTMC from a process without immediate actions and we often omit the tail process 0.

Table 4: Inference Rules($\rightsquigarrow$ denotes either $\rightarrow$ or $\dashrightarrow$)

$$\frac{}{a(x,w).P \xrightarrow{0} \{1 : (a(x), P)\}}\ (\text{REC}) \qquad \frac{}{a\langle b,\lambda\rangle.P \xrightarrow{\lambda} \{1 : (a\langle b\rangle, P)\}}\ (\text{mBRD})$$

$$\frac{}{a\langle b,\infty_w\rangle.P \xrightarrow{w} \{1 : (a\langle b\rangle, P)\}}\ (\text{iBRD}) \qquad \frac{P \overset{\lambda}{\rightsquigarrow} \mathbb{AP}}{\nu a P \overset{\lambda}{\rightsquigarrow} \nu a \mathbb{AP}}\ (\text{RES})$$

$$\frac{}{P \xrightarrow{0} \{1 : (a(x), P)\}}\ , x \notin fn(P) \text{ and } \gamma(a,P) = 0\ (\text{LOS})$$

$$\frac{P \xrightarrow{0} (a(x), \mathbb{P}) \quad Q \xrightarrow{0} (a(x), \mathbb{Q})}{P + Q \xrightarrow{0} (a(x), \frac{\gamma(a,P)}{\gamma(a,P+Q)}\mathbb{P} + \frac{\gamma(a,Q)}{\gamma(a,P+Q)}\mathbb{Q})}\ , \gamma(a, P+Q) \neq 0\ (\text{SUM1})$$

$$\frac{P \overset{\lambda_1}{\rightsquigarrow} \mathbb{AP} \quad Q \overset{\lambda_2}{\rightsquigarrow} \mathbb{AQ}}{P + Q \overset{\lambda_1+\lambda_2}{\rightsquigarrow} \frac{\lambda_1}{\lambda_1+\lambda_2}\mathbb{AP} + \frac{\lambda_2}{\lambda_1+\lambda_2}\mathbb{AQ}}\ (\text{SUM2})$$

$$\frac{P \overset{\lambda}{\rightsquigarrow} \mathbb{AP} \quad (\rightsquigarrow = \longrightarrow \wedge \mu(Q) = 0) \vee (\rightsquigarrow = \dashrightarrow \wedge \mu_\infty(Q) = 0)}{P + Q \overset{\lambda}{\rightsquigarrow} \mathbb{AP}}\ (\text{SUM3})$$

$$\frac{P \xrightarrow{0} (a(x), \mathbb{P}) \quad Q \xrightarrow{0} (a(x), \mathbb{Q})}{P \parallel Q \xrightarrow{0} (a(x), \mathbb{P} \parallel \mathbb{Q})}\ (\text{PAR1})$$

$$\frac{\left(\begin{array}{l} P \overset{\lambda_1}{\rightsquigarrow} \mathbb{AP} \quad \bigcup_{j\in J} \tilde{b_j} \cap fn(P) = \emptyset \quad \forall_{j\in J}\, \nu\tilde{b_j} a_j\langle b_j\rangle \in \mathcal{A}(\mathbb{AQ}).P \xrightarrow{0} (a_j(x), \mathbb{P}_j) \\ Q \overset{\lambda_2}{\rightsquigarrow} \mathbb{AQ} \quad \bigcup_{i\in I} \tilde{b_i} \cap fn(Q) = \emptyset \quad \forall_{i\in I}\, \nu\tilde{b_i} a_i\langle b_i\rangle \in \mathcal{A}(\mathbb{AP}).Q \xrightarrow{0} (a_i(x), \mathbb{Q}_i) \end{array}\right)}{P \parallel Q \overset{\lambda_1+\lambda_2}{\rightsquigarrow} \left(\begin{array}{l} \frac{\lambda_1}{\lambda_1+\lambda_2}(+_{i\in I}(\nu\tilde{b_i} a_i\langle b_i\rangle, \mathbb{AP}(\nu\tilde{b_i} a_i\langle b_i\rangle) \parallel \mathbb{Q}_i\{b_i/x\}) + (\tau, \mathbb{AP}(\tau) \parallel Q)) \\ +\frac{\lambda_2}{\lambda_1+\lambda_2}(+_{j\in J}(\nu\tilde{b_j} a_j\langle b_j\rangle, \mathbb{P}_j\{b_j/x\} \parallel \mathbb{AQ}(\nu\tilde{b_j} a_j\langle b_j\rangle)) + (\tau, P \parallel \mathbb{AQ}(\tau))) \end{array}\right)}\ (\text{PAR2})$$

$$\frac{\left(\begin{array}{l} P \overset{\lambda}{\rightsquigarrow} \mathbb{AP} \quad \forall_{i\in I}\, \nu\tilde{b_i} a_i\langle b_i\rangle \in \mathcal{A}(\mathbb{AP}).Q \xrightarrow{0} (a_i(x), \mathbb{Q}_i) \quad \bigcup_{i\in I} \tilde{b_i} \cap fn(Q) = \emptyset \\ (\rightsquigarrow = \longrightarrow \wedge \mu(Q) = 0) \vee (\rightsquigarrow = \dashrightarrow \wedge \mu_\infty(Q) = 0) \end{array}\right)}{P \parallel Q \overset{\lambda}{\rightsquigarrow} +_{i\in I}(\nu\tilde{b_i} a_i\langle b_i\rangle, \mathbb{AP}(\nu\tilde{b_i} a_i\langle b_i\rangle) \parallel \mathbb{Q}_i\{b_i/x\}) + (\tau, \mathbb{AP}(\tau) \parallel Q)}\ (\text{PAR3})$$

$$\frac{P \overset{\lambda_0}{\rightsquigarrow} \mathbb{AP}}{A \overset{\lambda_0}{\rightsquigarrow} \mathbb{AP}}\ , A \overset{def}{=} P\ (\text{CON}) \qquad \frac{P \equiv Q \overset{\lambda_0}{\rightsquigarrow} \mathbb{AQ} \equiv \mathbb{AP}}{P \overset{\lambda_0}{\rightsquigarrow} \mathbb{AP}}\ (\text{STR})$$

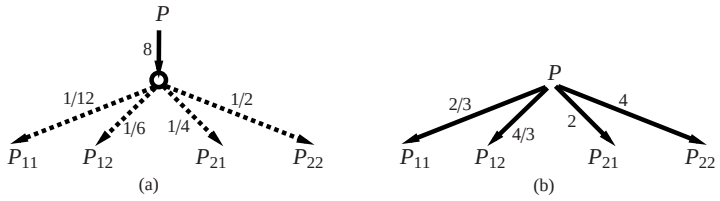

Figure 1: A Simple CTMC

Example 2 *Given a process $P \equiv a\langle b_1, 2\rangle \parallel a\langle b_2, 6\rangle \parallel (a(x,1).P_1 + a(x,2).P_2)$, then P will broadcast a message (b_1 or b_2) on channel a with an exponential delay 8 and then evolve into a process action distribution $\mathbb{AP}$. By semantics in Table 4, we know*

$$\mathbb{AP} = \{\frac{1}{12} : (a\langle b_1\rangle, P_{11}), \frac{1}{6} : (a\langle b_1\rangle, P_{12}), \frac{1}{4} : (a\langle b_2\rangle, P_{21}), \frac{1}{2} : (a\langle b_2\rangle, P_{22})\}$$

where

$$P_{11} = a\langle b_2, 6\rangle \parallel P_1\{b_1/x\} \quad P_{12} = a\langle b_2, 6\rangle \parallel P_2\{b_1/x\}$$
$$P_{21} = a\langle b_1, 2\rangle \parallel P_1\{b_2/x\} \quad P_{22} = a\langle b_1, 2\rangle \parallel P_2\{b_2/x\}$$

This is displayed in Fig. 1(a) and the correspondent CTMC is shown in Fig. 1(b), here we use dot lines to denote probabilistic choices and omit actions of the transitions.

In the above example we briefly illustrated how to get a CTMC from a process. Now we are going to give the general construction by which we can get the correspondent CTMC from a process. Use $\mathcal{M}(P,Q)$ to denote the rate from P to Q in a CTMC, and define $\mathsf{Der}(P)$ as the smallest set of processes satisfying: i) $P \in \mathsf{Der}(P)$; ii) $P_2 \in \mathsf{Der}(P)$ iff there exists $P_1 \in \mathsf{Der}(P)$ such that $P_1 \xrightarrow{\lambda} \mathbb{AP}$ with $\mathbb{AP}(P_2) > 0$. So $\mathsf{Der}(P)$ is the set of all the processes which are reachable from P with positive probability via arbitrary steps. For each two processes $P_1, P_2 \in \mathsf{Der}(P)$, the rate from P_1 to P_2 is equal to $\lambda \times \mathbb{AP}(P_2)$, that is $\mathcal{M}(P_1,P_2) = \lambda \times \mathbb{AP}(P_2)$ such that $P_1 \xrightarrow{\lambda} \mathbb{AP}$, otherwise $\mathcal{M}(P_1,P_2) = 0$.

4 Immediate Transitions

In this section, we will give a few examples and show how can we benefit from immediate transitions.

First we consider a model called *Closed Queueing Networks* (CQN) [7] from performance analysis. A queueing network is a collection of servers. Customers must proceed from one server to another in order to satisfy their service requirements. The queueing network is closed if neither arrivals nor departures of customers are permitted; instead the number of customers in the network are fixed at all times. We use M to denote the number of servers in the network and N for the number of customers circulating around. The service time for a customer at server i is exponentially distributed with rate λ_i and the probability a customer will proceed to the j-th server after completing a service request at server i is equal to p_{ij} for $i, j = 1, 2, \ldots, M$.

Example 3 *Suppose we are given a CQN with 5 servers and 15 customers shown in Fig. 2 where the numbers in the rectangles denote the length of the queue of each server as well as their indexes, the*

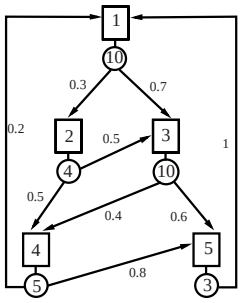

Figure 2: A CQN with 5 Servers

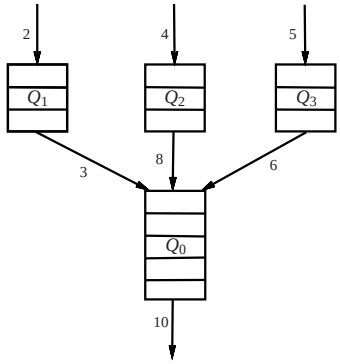

Figure 3: Open Queue Network with Blocking

numbers on the edges denote the transition probabilities and the numbers in the circles denote the rate of each service time.

The communication in the CQN is "point-to-point" in the sense that each leaving customer will arrive at only one server, while in our calculus the communication is based on broadcast. But with immediate transitions we can model such "point-to-point" communication as follows. Here we assume that the weight of each input is 1 by default, that is, $c_i(x)$ is equal to $c_i(x,1)$. $SQ_i(w_i)$ denotes server i with w_i customers waiting for service in its queue. In Table 5, $CQN(1,2,3,4,5)$ denotes the system where the i-th parameter is the length of the i-th queue.

Table 5: Model of Closed Queueing Network

$$\mathsf{Rev}(i,j,w_j) = c_i(x).[x=w](c\langle r,\infty_{p_{ij}}\rangle.(\mathsf{SQ}_j(w_j+1) \parallel \mathsf{SQ}_i(w-1)) + c(x).\mathsf{SQ}_j(w_j)) \quad w \in [1,15]$$

$$\mathsf{SQ}_j(w_j) = \sum_{1\le i\le 5, p_{ij}>0} \mathsf{Rev}(i,j,w_j) + ([w_j=0]0, c_j\langle w_j,\lambda_j\rangle)$$

$$\mathsf{CQN}(1,2,3,4,5) = \nu c(\|_{i=1}^{5}\ \mathsf{SQ}_i(i))$$

Each $SQ_j(w_j)$ contains two parts: receivers denoted by $\sum_{1\le i\le 5, p_{ij}>0} Rev(i,j,w_j)$ listen on the channels of their predecessors, and after being notified that a customer is coming, it will try to broadcast on channel c immediately with a specific weight $\infty_{p_{ij}}$. The one which succeeds to do so will be the destination of the customer and all the others will be informed by receiving a message on c. The other part $[w_j=0]0, c_j\langle w_j,\lambda_j\rangle$ takes care of the requests of the customers in its queue if it is not empty, the rate of the customer leaving depends on the service rate. By putting these five servers in parallel, we get the whole system $CQN(1,2,3,4,5)$. Our semantics guarantees that each leaving customer will finally arrive at one and only one server. Actually, this can be seen as a way to model "point-to-point" communication with immediate transitions. For example if a customer is leaving server 3, then both server 4 and 5 will be informed by broadcasting a message on channel c_3. But after that server 4 and 5 will try to broadcast an acknowledge r on channel c, with probability 0.4 and 0.6 respectively. The one which succeeds to send r will be the real destination of the customer. The other server will know this fact by listening on channel c and roll back to its original state at the same time. The model in Table 5 is quite flexible, for example, we can add self loops easily, that is, the destination of a leaving customer can be the same

server as its departure server, or instead of having fixed transfer probabilities, we can make them change based on the current lengths of queues.

In Example 3 we have shown a typical closed queueing network where we assume that every server has a queue with infinite capacity. But in practice the capacity of queues is often limited. In [10] a variant of closed queueing network is proposed where the queue of each server i only has finite capacity B_i for $1 \leq i \leq M$. A customer which requests service at server i while the queue of server i is full will instantly be routed to another server j with probability p_{ij} as if it is served by server i at an infinite speed. For this kind of model, it is hard (if not impossible) to model it in a compositional way without using immediate actions. With immediate actions, the model is easy to obtain without changing the model in Table 5 a lot.

Example 4 *Suppose the queueing network we are about to model is the same as the queueing network in Example 3 except that the capacity of every queue is maximum 10. If one server is full, it will just transfer the coming customers to its next servers. The model can be obtained by simply replacing the* $\mathsf{SQ}_j(w_j+1)$ *in* $\mathsf{Rev}(i,j,w_j)$ *in Table 5 with* $[w_i = 10]c_i\langle 11,\infty_1\rangle, \mathsf{SQ}_j(w_j+1)$. *This means instead of accepting any coming customers, we require an extra checking on the current queue. If a customer arrives at a server whose queue is not full, the customer will be accepted, otherwise, the customer will also be accepted but will be transferred to the next server via action* $c_i\langle 11,\infty_1\rangle$ *just like it is served with infinite rate, that gives excuse of parameter 11. The same process will continue until the customer arrives at some server which has free space for it.*

From the semantics in Table 4, we know that all the outputs are non-blocking, i.e., for one message to be broadcasted, it is not necessary to have recipients. But sometimes we may have models where components are not completely independent and one component can do something only after some other components finish, that is, some behaviors are blocking. Blocking here means that an output action cannot happen spontaneously but has to wait until certain conditions are fulfilled. For instance in Example 4, every queue has finite capacity. When a customer arrives at a server without free space, it will simply be transferred to other servers. What if the customer cannot be transferred but can only wait until the server has free spaces? Refer to the following example from [18] which is also a variant of queueing network called *open queueing network with blocking*.

The network consists of N parallel servers called *merging* queues; there is a queue receiving the outputs of these merging queues and is called *merged* queue (or queue 0). The service time at queue i is exponentially distributed with rate λ_i. The queue network is open since the number of the customers circulating in the network is not fixed and some external customers may arrive from the outside. Arrivals to queue i are independent *Poisson Processes* with rate μ_i. There is no external arrival to the merged queue. The length of the i-th merging queue is B_i. The capacity of the queue 0 is B_0, and λ_0 is its service rate at queue 0. If a customer arrives at a merging queue when it is full, the customer will be lost. When a customer completes service at server i, it will be transferred to queue 0 only if it is not full; otherwise, the customer waits in the i-th server until it can enter queue 0. During this time the i-th server cannot serve other customers that might wait in its queue. In this case, the queue is said to be *blocked* and queue 0 is *blocking*. Since there are M servers in parallel, there might be more than one queue blocked at the same time. When more than one queues are blocked, it is assumed that they will enter queue 0 on a "First-Blocked-First-Enter" basis.

From the description of open queue network with blocking, we know there are two kinds of actions involving in this model: one is blocking and the other one is non-blocking. For instance, when queue 0 is full, any other arrivals have to wait until queue 0 has free space, this is blocking action. On the other hand, when the merging queues are full, instead of blocking the external arrivals it will just discard them,

Table 6: Model of Open Queueing Network with Blocking

$$A_i = a_i\langle arrival, \mu_i\rangle.A_i$$
$$S_i(w_i) = [w_i = 0,1,2]a_i(x,1).S_i(w_i+1) + [w_i = 1,2,3]s_i\langle leave_i, \lambda_i\rangle.b_i(x,1).S_i(w_i-1)$$
$$S_0(w,\tilde{S}) = [w = 1,2,3,4,5]s_0\langle leave_0, \lambda_0\rangle.S_0(w-1,\tilde{S}) + [w=5]\sum_i s_i(x,1).S_0(w,\tilde{S}\,i)$$
$$+ [w = 0,1,2,3,4 \wedge H(\tilde{S}) = i]b_i\langle unblocking, \infty_1\rangle.S_0(w+1,T(\tilde{S}))$$
$$+ [w = 0,1,2,3,4 \wedge H(\tilde{S}) = \perp]\sum_i s_i(x,1).b_i\langle unblocking, \infty_1\rangle.S_0(w+1,\tilde{S})$$

so the external arrivals are non-blocking actions. In the following we will show how to model both kinds of actions in our calculus.

Example 5 *Fig. 3 gives a concrete example with 3 servers marked as Q_1, Q_2, and Q_3 respectively. The length of each queue is 3 and the numbers on the in-edges and out-edges are used to denote arrival rates and service rates, that is, $\lambda_1 = 3$, $\lambda_2 = 8$, $\lambda_3 = 6$, $\mu_1 = 2$, $\mu_2 = 4$, and $\mu_3 = 5$, and $\lambda_0 = 10$. The Q_0 at the bottom is the queue 0 with service rate 10, and the capacity of its queue is 5. Initially, every queue is empty.*

Here is the model of the queueing network in Fig. 3 where A_i denotes the arrival process of queue i, $Q_i(w)$ denotes queue i with w customers in the queue for $1 \le i \le 3$, and $Q_0(w,\tilde{S})$ denotes queue 0 with w customers. The $\tilde{S}$ is a sequence of queues which are waiting for queue 0 when it is full, it is an element of SQ which is defined by enumerating all the possible sequences of queues waiting for queue 0. The symbol $\perp$ is used to denote empty sequence.

$$SQ = \{\perp, 1, 2, 3, 12, 21, 13, 31, 23, 32, 123, 132, 213, 231, 312, 321\}$$

For simplicity, we define two functions on this set: $H(\tilde{S}) : SQ \to \{\perp, 1, 2, 3\}$ and $T(\tilde{S}) : SQ \to SQ$ which return the head of sequence and the left sequence by deleting the first element respectively. For example, $H(123) = 1$ and $T(123) = 23$; $H(\perp) = \perp$ and $T(\perp) = \perp$. We use $\tilde{S}\,i$ to denote a new sequence by attaching i to the end of $\tilde{S}$. Note here that these functions are just used to give a compact model, they can be replaced by the standard operators by enumerating all the possible cases. It is similar for conditions like $[w = 0,1,2 \wedge H(\tilde{S}) = i]$. The model of each component in Fig. 3 is shown in Table 6. The whole system can be denoted as $P = \|_i A_i \|_i S_i(0) \| S_0(0,\perp)$ with $1 \le i \le 3$.

As we said before, broadcasts such that $s_i\langle leave_i, \lambda_i\rangle$ are blocking, so when there is no input $s_i(x,w)$ available, that is, the queue 0 is full, server i has to wait before it can perform other actions. To do so, we let the server i wait for the message unblocking on channel b_i after one customer leaving from it to server 0. If queue 0 has free spaces, it will perform action $b_i\langle unblocking, \infty_1\rangle$ right after it receives the request from server i. Otherwise if the queue 0 is full, it will attach the request to the end of its waiting list. When server 0 is ready to handle the request after several steps, it will send the message unblocking to server i instantly via immediate action $b_i\langle unblocking, \infty_1\rangle$. Server i will receive it at the same time and then unblock itself.

Fig. 4 shows a fragment of the execution of P where $\longrightarrow$, $- - \succ$, and $\cdots\succ$ denote Markovian, immediate and probabilistic transition respectively. In additional $A = \|_i A_i$ and $S(q_1,q_2,q_3) = \|_i S_i(q_i)$, when q_i is barred, it means that server i is blocked. For example, $S(q_1,\overline{q_2},q_3) = S_1(q_1) \| b_2(x,1).S_2(q_2) \| S_3(q_3)$. When in state $A \| S(3,2,\overline{1}) \| S_0(4,3)$, it means that there is a free space in queue 0 while the

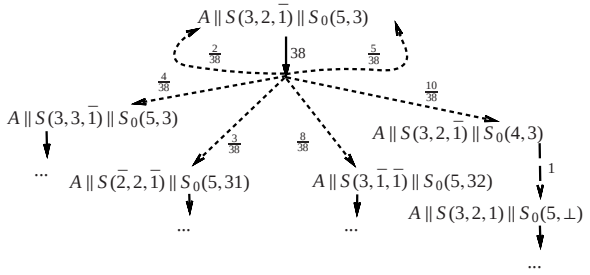

Figure 4: Execution Fragment of P

server 3 is waiting for service. In this case, server 0 should response to it and transfer to the state $A \parallel S(3,2,1) \parallel S_0(5,\perp)$ *instantly where the request is removed from the waiting list to the queue 0. The self loop with probability* $\frac{2}{38}$ *denotes that the arrivals of external customers to server 1 while its queue is full. In this case, the arriving customers will be discarded without causing any effects. The other self loop with probability* $\frac{5}{38}$ *is similar except that the arriving customer is discarded because server 3 is blocked.*

From Example 5, we can see that blocking actions can be represented easily by using immediate transitions. In general when broadcast $a\langle b,\lambda\rangle$ is blocking, it should be prefixed with an input such as $c(x,w).a\langle b,\lambda\rangle$. When certain conditions are fulfilled, the process should trigger $a\langle b,\lambda\rangle$ by sending a message on channel c instantly, that is, by action like $c\langle unblocking,\infty_w\rangle$.

5 The Underlying CTMC

In Examples 3, 4, and 5, we see that immediate transition is indeed powerful to model some systems. But the main disadvantage is that the underlying CTMC of a process is not so obvious anymore. In this section we will show how to define the underlying CTMC in case of immediate transitions. Different from [3] where the eliminations of immediate transitions are based on the weak behavioral equivalence, we solve this by dealing with a set of equations as follows.

In this calculus choices between Markovian actions are probabilistic depending on their rates while choices between immediate actions are also probabilistic depending on their associated weights. In addition, if both types of actions are involved in a choice, the immediate action should be prioritized, since they take no time, so the probability of the Markovian action being executed before the immediate one is zero [14]. Since the priority of immediate actions are not shown in Table 4, a CTMC cannot be obtained directly from a process based on the semantics. In the following we distinguish between MP and IP. MP only contains processes which do not have immediate transitions while IP only contains processes which have immediate transitions available. Formally, $\mathsf{IP} = \{P \in \mathscr{P} \mid \exists \mathbb{AP}.P - \frac{\lambda}{-} \succ \mathbb{AP}\}$ and $\mathsf{MP} = \{P \in \mathscr{P} \mid \nexists \mathbb{AP}.P - \frac{\lambda}{-} \succ \mathbb{AP}\} = \mathscr{P} \setminus \mathsf{IP}$. It is not hard to see that every state in a CTMC should be seen as a Markovian process in this calculus, so instead of considering all the processes in $\mathsf{Der}(P)$ as in Section 3, we only need to consider set $\mathsf{Der}(P) \cap \mathsf{MP}$ when talking about the corresponding CTMC of P. The question now is how to give the value of $\mathcal{M}(P,Q)$ for any $P,Q \in \mathsf{Der}(P) \cap \mathsf{MP}$. Due to immediate actions, it is not enough to just consider one step Markovian transition as before since P might have to go through several immediate processes before reaching Q.

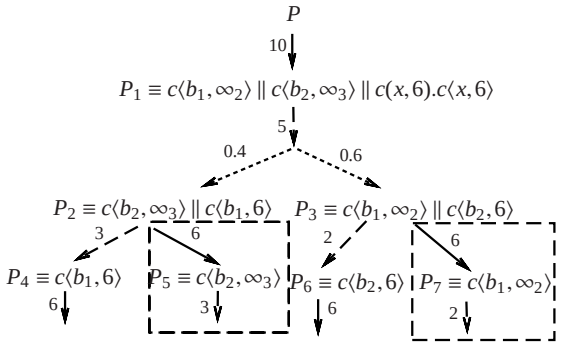

Figure 5: Example with Immediate Transitions

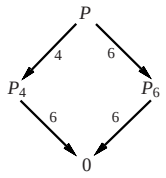

Figure 6: CTMC corresponding to P

We use $\mathsf{Pr}^i(R,Q)$ to denote the probability from R to Q via all possible immediate transitions where $R \in \mathscr{P}$ and $Q \in \mathsf{MP}$. The value of $\mathsf{Pr}^i(R,Q)$ is given by the smallest solution defined by the following set of equations:

$$\mathsf{Pr}^i(R,Q) = \begin{cases} 1 & R = Q \\ \sum_{\mathbb{AR}(P)>0} \mathbb{AR}(P) \times \mathsf{Pr}^i(P,Q) & R -\!\!\!\xrightarrow{\lambda}\!\!\!\succ \mathbb{AR} \wedge \mathsf{Der}(R) \cap \mathsf{MP} \neq \emptyset \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Then for any $P, Q \in \mathsf{MP}$, $\mathcal{M}(P,Q)$ can be defined as follows:

$$\mathcal{M}(P,Q) = \lambda \times \sum_{\mathbb{AP}(R)>0} \mathbb{AP}(R) \times \mathsf{Pr}^i(R,Q) \quad P \xrightarrow{\lambda} \mathbb{AP}$$

Example 6 *Suppose* $P \equiv a\langle b, 10\rangle \parallel a(x,1).c\langle b_1, \infty_2\rangle \parallel a(x,1).c\langle b_2, \infty_3\rangle \parallel c(x,1).c\langle x, 6\rangle$*, by the semantics in Table 4 we can draw a derivation tree as Fig. 6. We omit passive transitions like* $\xrightarrow{0}$ *here.*

In Fig. 5 nondeterministic choices emerge, such as process $c\langle b_2, \infty_3\rangle \parallel c\langle b_1, 6\rangle$ *can choose either immediate transition* $-\!\!\!\xrightarrow{3}\!\!\!\succ$ *or Markovian transition* $\xrightarrow{6}$*. But since the probability of the Markovian transition being executed before the immediate one is zero, so the transitions inside the dashed rectangle is impossible and should be ignored.*

In this example, $\mathsf{Der}(P) = \{P, P_1, P_2, P_3, P_4, P_5, P_7, 0\}$*,* $\mathsf{Der}(P) \cap \mathsf{IP} = \{P_1, P_2, P_3, P_5, P_7\}$*, and* $\mathsf{Der}(P) \cap \mathsf{MP} = \{P, P_4, P_6, 0\}$*. To define the CTMC of P, we only need to consider the processes in* $\mathsf{Der}(P) \cap \mathsf{MP}$ *and the corresponding CTMC of P is shown in Fig. 6 which is quite simple compared to the derivation tree in Fig. 5.*

In the second case of Equation (2), we require that $\mathsf{Der}(R) \cap \mathsf{MP} \neq \emptyset$, that is, there exists at least a Markovian process which is reachable from R. But sometimes it is also possible for one process reaching an immediate state from which no Markovian process can be reached, that is, we have immediate loop. We call states in $\mathsf{SP} = \{P \mid \mathsf{Der}(P) \subseteq \mathsf{IP}\}$ absorbing states and use a special process Stuck to denote them. Accordingly, the set of states of the CTMC should be $(\mathsf{Der}(P) \cap \mathsf{MP}) \cup \{\mathsf{Stuck}\}$ and $\mathcal{M}(P, \mathsf{Stuck})$ where $P \in \mathsf{MP}$ can be defined as follows:

$$\mathcal{M}(P, \mathsf{Stuck}) = \lambda - \sum_{Q \in \mathsf{Der}(P) \cap \mathsf{MP}} \mathcal{M}(P,Q) \quad P \xrightarrow{\lambda} \mathbb{AP}.$$

6 Conclusions and Future Works

In this paper we give a stochastic broadcast π calculus which is useful to model some server-client based systems where synchronization is governed by only one participant. Both Markovian transitions and immediate transitions are taken into account. A few examples are given to show the expressivity of the calculus which is enhanced a lot with immediate transitions. The semantics is given by Labeled Transition System without relying on techniques such as multi relations, proved transition systems, unique rate names, and so on. Each transition is labeled with rate or weight instead of action and the resulting distribution is over pairs of actions and processes instead of only processes. In this way, the underlying CTMC can be obtained naturally even with existence of immediate transitions.

A number of further developments are possible. In the future we would like to provide semantics to some of the most representative stochastic process languages as [9, 4] and compare these different ways. Another possible extension is to support parameters, that is, we do not need to know the value of each parameter at beginning. We can also reuse the model by assigning parameters with different values and so on. Sometimes, some CTMCs have special form, that is, product form which can be solved efficiently [16, 12]. We try to answer whether the underlying CTMC of a process is in product form or not by syntax-checking.

References

[1] A. Aldini, M. Bernardo & F. Corradini (2009): *A process algebraic approach to software architecture design*. Springer-Verlag New York Inc.

[2] J.A. Bergstra & J.W. Klop (1984): *Process algebra for synchronous communication*. *Information and Control* 60(1-3), pp. 109–137, doi:10.1016/S0019-9958(84)80025-X.

[3] M. Bernardo & A. Aldini (2007): *Weak Markovian bisimilarity: abstracting from prioritized/weighted internal immediate actions*. In: *Theoretical Computer Science: Proceedings of the 10th Italian Conference on ICTCS'07*. World Scientific Pub Co Inc, p. 39, doi:10.1142/9789812770998(0)0008.

[4] M. Bernardo, R. De Nicola & M. Loreti (2010): *Uniform Labeled Transition Systems for Nondeterministic, Probabilistic, and Stochastic Processes*. *Trustworthly Global Computing* , pp. 35–56doi:10.1007/978-3-642-15640-3(0)3.

[5] M. Bernardo, L. Donatiello & R. Gorrieri (1994): *MPA: a stochastic process algebra*. *University of Bologna* .

[6] M. Bernardo & R. Gorrieri (1998): *A tutorial on EMPA: A theory of concurrent processes with nondeterminism, priorities, probabilities and time*. *Theoretical Computer Science* 202(1-2), pp. 1–54, doi:10.1016/S0304-3975(97)00127-8.

[7] J.P. Buzen (1973): *Computational Algorithms for Closed Queueing Networks with Exponential Servers*. *Communications of the ACM* 16(9), pp. 527–531, doi:10.1145/362342.362345.

[8] R. De Nicola, J.P. Katoen, D. Latella, M. Loreti & M. Massink (2007): *Model checking mobile stochastic logic*. *Theoretical Computer Science* 382(1), pp. 42–70, doi:10.1145/362342.362345.

[9] R. De Nicola, D. Latella, M. Loreti & M. Massink (2009): *Rate-based transition systems for stochastic process calculi*. *Automata, Languages and Programming* , pp. 435–446doi:10.1007/978-3-642-02930-1(0)36.

[10] N.M. van Dijk (1988): *On Jackson's product form with jump-over blocking*. *Operations Research Letters* 7(5), pp. 233–235, doi:10.1016/0167-6377(88)90037-5.

[11] N. Götz, U. Herzog & M. Rettelbach (1993): *Multiprocessor and distributed system design: The integration of functional specification and performance analysis using stochastic process algebras*. *Performance evaluation of computer and communication systems* , pp. 121–146doi:10.1007/BFb0013851.

[12] P.G. Harrison (2004): *Reversed processes, product forms and a non-product form*. *Linear Algebra and Its Applications* 386, pp. 359–381, doi:10.1016/j.laa.2004.02.020.

[13] H. Hermanns (2002): *Interactive markov chains*. Springer, doi:10.1007/3-540-45804-2(0)5.

[14] H. Hermanns, M. Rettelbach & T. Weiss (1995): *Formal characterisation of immediate actions in SPA with nondeterministic branching*. *The Computer Journal* 38(7), p. 530, doi:10.1093/comjnl/38.7.530.

[15] J. Hillston (1996): *A compositional approach to performance modelling*. Cambridge University Press.

[16] J. Hillston & N. Thomas (1999): *Product form solution for a class of PEPA models*. *Performance Evaluation* 35(3-4), pp. 171–192, doi:10.1016/S0166-5316(99)00005-X.

[17] C.A.R. Hoare (1978): *Communicating sequential processes*. *Communications of the ACM* 21(8), p. 677, doi:10.1145/357980.358021.

[18] H.S. Lee & S.M. Pollock (1989): *Approximate analysis for the merge configuration of an open queueing network with blocking*. *IIE transactions* 21(2), pp. 122–129, doi:10.1080/07408178908966215.

[19] R. Milner (1989): *Communication and concurrency*. Prentice Hall International Series in Computer Science.

[20] C. Priami (1995): *Stochastic π-calculus*. *The Computer Journal* 38(7), p. 578.

[21] M.G. Vigliotti & P.G. Harrison (2006): *Stochastic ambient calculus*. *Electronic Notes in Theoretical Computer Science* 164(3), pp. 169–186, doi:10.1016/j.entcs.2006.07.018.