Innovations Syst Softw Eng
DOI 10.1007/s11334-009-0100-9

ORIGINAL PAPER

PTSC: probability, time and shared-variable concurrency

Huibiao Zhu · Shengchao Qin · Jifeng He · Jonathan P. Bowen

Received: 21 October 2008 / Accepted: 14 July 2009

Abstract Complex software systems typically involve features like time, concurrency and probability, where probabilistic computations play an increasing role. It is challenging to formalize languages comprising all these features. In this paper, we integrate probability, time and concurrency in one single model, where the concurrency feature is modelled using shared-variable-based communication. The probability feature is represented by a probabilistic nondeterministic choice, probabilistic guarded choice and a probabilistic version of parallel composition. We formalize an operational semantics for such an integration. Based on this model we define a bisimulation relation, from which an observational equivalence between probabilistic programs is investigated and a collection of algebraic laws are explored. We also implement a prototype of the operational semantics to animate the execution of probabilistic programs.

A preliminary version of this paper appeared in SEW-30: 30th IEEE/NASA Software Engineering Workshop [29].

H. Zhu (✉) · J. He
Shanghai Key Laboratory of Trustworthy Computing,
East China Normal University, 3663 Zhongshan Road (North),
Shanghai 200062, China
e-mail: hbzhu@sei.ecnu.edu.cn

J. He
e-mail: jifeng@sei.ecnu.edu.cn

S. Qin
Department of Computer Science, University of Durham,
South Road, Durham DH1 3LE, UK
e-mail: shengchao.qin@durham.ac.uk

J. P. Bowen
Museophile Limited, Oak Barn, Sonning Eye,
Reading RG4 6TN, UK
e-mail: jpbowen@gmail.com
URL: http://www.jpbowen.com

Keywords Probability · Time · Shared-variable concurrency · Operational semantics · Algebraic laws · Bisimulation

1 Introduction

As probabilistic computations play an increasing role in solving various problems [25], various proposals on probabilistic languages have been reported [5,6,8,17,16,18,22–24]. Complex software systems typically involve important features like real-time, probability and shared-variable concurrency. Therefore, system designers would expect a formal model that incorporates all these features to be available for them to use. However, to the best of our knowledge, no one has integrated all these features in one model. In this paper, we tackle this challenging problem by proposing a formal model for a language equipped with probability, time and shared-variable concurrency. Our model is meant to facilitate the specification of complex software systems.

The shared-variable mechanism is typically used for communications among components running in parallel. Although shared-variable concurrency can be seen in many languages (e.g. the Java programming language, the Verilog hardware description language), it proves to be challenging to formalize it [10,27,28], not to mention other orthogonal features like probability and time. In this paper, we successfully tackle this challenge by integrating time, probability and shared-variable concurrency in one model. The probability feature is reflected by the probabilistic nondeterministic choice, probabilistic guarded choice and the probabilistic scheduling of actions from different concurrent components in a program.

As advocated in Hoare and He's unifying theories of programming (*UTP*) [12], three different styles of mathematical representations are normally used: operational, denotational, and algebraic ones, among which the operational style is the most intuitive one. In order to elaborate more on the intuition behind the proposed language and to formally define the behaviour of its programs, we start with the operational semantics in this paper. Upon the operational model, we define a bisimulation relation, from which a collection of algebraic laws are derived.

Much related work has investigated the semantics for probabilistic processes. Morgan and his colleagues explored the abstraction and refinement for probabilistic processes using the weakest precondition (wp) approach [16–18]. Hartog and her colleagues have studied the equivalence between operational and denotational semantics for a variety of probabilistic processes [5–8] using Banach Space approach [4]. Núñez extended Henessey's "testing semantics" for a variety of probabilistic processes [22–24]. As an extension of the guarded command language, a simple probabilistic guarded command language was formalized in [11] under the *UTP* framework. A set of algebraic laws were then explored based on the denotational model.

The *PTSC* model proposed in this paper has recently been used to specify a circuit in the register-transfer level [21]. The circuit takes two integers as the input and sums up them as the output, where the register containing one of the inputs may be faulty. Our algebraic laws proposed for the *PTSC* language have also been employed to verify that an implementation of the circuit with probabilistic behaviour conforms to the probabilistic specification.

The rest of this paper is organized as follows. Section 2 presents our language equipped with probability, time and shared-variable concurrency. Section 3 is devoted to the operational semantics. A bisimulation relation is then defined in Sect. 4 and used for the basis of a set of algebraic laws in Sect. 5. Section 6 gives a prototype animation of the operational semantics, followed by some concluding remarks in Sect. 7.

2 Probabilistic language *PTSC*

In this paper, we propose a probabilistic language *PTSC* (probability, time and shared-variable concurrency), which involves the integration of probability, time and shared-variable concurrency. Apart from the concurrency feature that exists in many conventional languages, probability and time are the other two main features of our language. The synchronization of different parallel components is based on time controls. Overall, our language consists of the following main features:

(1) Probabilistic behaviour: This can be represented by probabilistic nondeterminism, probabilistic guarded choice or probabilistic parallel composition.
(2) Timed behaviour: This can be reflected by event guard $@\ b$ (wait until b is satisfied) and time-delay command.
(3) Shared-variable concurrency: The concurrency model employs a shared-variable-based communication mechanism.

The *PTSC* language has the following syntactical elements:

$P ::=$ **Skip**	(*does nothing*)
$\mid x := e$	(*atomic assignment*)
$\mid$ **if** b **then** P **else** P	(*conditional*)
$\mid$ **while** b **do** P	(*while-loop*)
$\mid @b\ P$	(*event guard*)
$\mid \#n\ P$	(*time delay*)
$\mid P\ ;\ P$	(*sequential*)
$\mid P \sqcap P$	(*non-deterministic choice*)
$\mid P \sqcap_p P$	(*probabilistic choice*)
$\mid P \parallel_p P$	(*prob. parallel composition*)

Note that:

(1) The assignment $x := e$ is an atomic action. The empty statement **Skip** behaves the same as $x := x$.
(2) Regarding $@b\ P$, when the Boolean condition b is satisfied, process P can have the chance to be scheduled. As we consider models for closed systems, the program $@b\ P$ can only let time advance when the Boolean condition b is not met. For $\#n\ P$, after n time units elapse, process P can be scheduled.
(4) The mechanism for parallel composition $P \parallel_p Q$ is a shared-variable interleaving model with probability feature. If process P can perform an atomic action, $P \parallel_p Q$ has conditional probability p to do that atomic action. On the other hand, if process Q can perform an atomic action, $P \parallel_p Q$ has conditional probability $1-p$ to perform that action.
(5) $\sqcap$ stands for the nondeterministic choice, where $\sqcap_p$ stands for the probabilistic nondeterministic choice. $P \sqcap_p Q$ indicates that the probability for $P \sqcap_p Q$ to behave as P is p, where the probability for $P \sqcap_p Q$ to behave as Q is $1-p$.

In order to facilitate algebraic reasoning, we enrich our language with a *guarded choice*. As our parallel composition has probability feature, the guarded choice also shares this feature. A guarded choice construct can be of the following five types:

(1) $[]_{i\in I}\{[p_i]\ choice_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij})\ P_{ij})\}$
(2) $[]_{i\in I}\{@b_i\ P_i\}$
(3) $[]\{\#1\ R\}$

(4) $[]_{i\in I}\{[p_i]\ choice_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij})\ P_{ij})\}$
$[][]_{k\in K}\{@b_k\ Q_k\}$
(5) $[]_{i\in I}\{@b_i\ P_i\}[]\{\#1\ R\}$

Note that for type (1) and (4), the guarded choice construct $[]_{i\in I}\{[p_i]\ choice_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij})\ P_{ij})\}$ should satisfy the following healthiness conditions:

(a) $\forall i \bullet (\bigvee_{j\in J_i} b_{ij} = \mathbf{true})$ and
$(\forall j_1, j_2 \bullet (j_1 \neq j_2) \Rightarrow ((b_{ij_1} \wedge b_{ij_2}) = \mathbf{false}))$
(b) $+_{i\in I}\ p_i = 1$

The first type is composed of a set of assignment-guarded components. The condition (a) indicates that for any $i \in I$, the Boolean conditions b_{ij} from "$choice_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij})\ P_{ij})$" are complete and disjoint. Therefore, there will be exactly one component $b_{ij}\&(x_{ij} := e_{ij})\ P_{ij}$ selected among all $j \in J_i$. Furthermore, for any $i \in I$, the possibility for a component $(x_{ij} := e_{ij})\ P_{ij}$ (where b_{ij} is met) to be scheduled is p_i and it should satisfy the second healthiness condition.

The second type is composed of a set of event-guarded components. If one guard is satisfied, the subsequent behaviour for the whole process will be followed by its subsequent behaviour of the satisfied component.

The third type is composed of one time delay component. Initially, it cannot do anything but let time advance one unit.

The fourth type is the guarded choice composition of the first and second type of guarded choice. If there exists one b_k $(k \in K)$ being satisfied currently, then the event $@\ b_k$ is fired and the subsequent behaviour is Q_k. If there is no satisfied b_k, the behaviour of the fourth type of guarded choice is the same as that of the first type.

The fifth type is the compound of the second and third type of guarded choice. Currently, if there exists i $(i \in I)$ such that b_i is satisfied, then the subsequent behaviour of the whole guarded choice is as P_i. On the other hand, if there is no i $(i \in I)$ such that b_i is satisfied currently, then the whole guarded choice cannot do anything initially but let time advance one unit. The subsequent behaviour is the same as the behaviour of R.

As the first type of guarded choice does not have time advancing behaviour, there is no type of composite guarded choice formed by the first and third type of guarded choice.

3 Operational semantics

The operational semantics of a language models the behaviour of a program in terms of execution steps, which are represented by transition relations. In our operational model, the transitions are expressed in the form of Plotkin's structural operational semantics (SOS) [26]:

$$\langle P,\ \sigma\rangle \xrightarrow{\beta} \langle P',\ \sigma'\rangle$$

where $\langle P, \sigma\rangle$ denotes a runtime configuration, with P standing for the program text that remains to be executed and σ being the current state of the program. We use $\mathbb{C}$ to denote the set of all runtime configurations.
There are four kinds of transitions that a program may perform:

(1) The first kind of transitions models an atomic action with certain probability (p). As mentioned earlier, an assignment is treated as an atomic action.

$$\langle P,\ \sigma\rangle \xrightarrow{c}_p \langle P',\ \sigma'\rangle$$

It says, in state σ, the program P has the probability p to perform the atomic action, leading to the state σ' with the program P' remaining.

(2) The second type models the transition of a time delay. Time advances in discrete unit steps.

$$\langle P,\ \sigma\rangle \xrightarrow{1} \langle P',\ \sigma'\rangle$$

(3) The third type models the selection of the two components for non-deterministic choice. It can be expressed as:

$$\langle P,\ \sigma\rangle \xrightarrow{\tau} \langle P',\ \sigma\rangle$$

Note that the label τ is used to depict a non-deterministic choice action. The program state remains unchanged for this kind of transitions.

(4) The fourth type models the triggered case of event $@\ b$:

$$\langle P,\ \sigma\rangle \xrightarrow{v} \langle P',\ \sigma\rangle$$

Note that the label v is employed to represent such event-triggered transitions. The program state remains unchanged after the transition.

In what follows, we present the operational rules for sequential programs, probabilistic guarded choice, and concurrent programs.

3.1 Sequential process

A sequential program comprising a single assignment performs an atomic action with probability 1. It cannot perform any other types of transitions.

$$\langle x := e,\ \sigma\rangle \xrightarrow{c}_1 \langle \varepsilon,\ \sigma[e/x]\rangle$$

Note that ε stands for the empty process.

For the conditional statement **if** b **then** P **else** Q , the control flow will be passed to P with probability 1 if b is satisfied, otherwise it will be passed to Q with probability 1.

$$\langle \textbf{if } b \textbf{ then } P \textbf{ else } Q\ ,\ \sigma\rangle \xrightarrow{c}_1 \langle P,\ \sigma\rangle, \quad \text{if } b(\sigma)$$
$$\langle \textbf{if } b \textbf{ then } P \textbf{ else } Q\ ,\ \sigma\rangle \xrightarrow{c}_1 \langle Q,\ \sigma\rangle, \quad \text{if } \neg b(\sigma)$$

Here $b(\sigma)$ returns the value of b in the state σ.

The transitions for iteration are similar to conditional.

$$\langle \textbf{while } b \textbf{ do } P,\ \sigma\rangle \xrightarrow{c}_1 \langle P\ ;\ \textbf{while } b \textbf{ do } P,\ \sigma\rangle, \quad \text{if } b(\sigma)$$
$$\langle \textbf{while } b \textbf{ do } P,\ \sigma\rangle \xrightarrow{c}_1 \langle \varepsilon,\ \sigma\rangle, \quad \text{if } \neg b(\sigma)$$

Time delay can advance time in unit steps. It cannot do any other types of transitions.

$$\langle \#n,\ \sigma\rangle \xrightarrow{1} \langle \#(n{-}1),\ \sigma\rangle, \quad \text{where } n > 1$$
$$\langle \#1,\ \sigma\rangle \xrightarrow{1} \langle \varepsilon,\ \sigma\rangle$$

The event $@b$ in $@b\ P$ is satisfied if Boolean condition b is currently satisfied, otherwise it will let time advance one unit. We use "$\xrightarrow{v}$" to model the event-triggered transition instead of "$\xrightarrow{c}$".

$$\langle @b\ P,\ \sigma\rangle \xrightarrow{v} \langle P,\ \sigma\rangle, \quad \text{if } b(\sigma)$$
$$\langle @b\ P,\ \sigma\rangle \xrightarrow{1} \langle @b\ P,\ \sigma\rangle, \quad \text{if } \neg b(\sigma)$$

The selection of process P or Q from $P \sqcap Q$ is nondeterministic.

$$\langle P \sqcap Q,\ \sigma\rangle \xrightarrow{\tau} \langle P,\ \sigma\rangle$$
$$\langle P \sqcap Q,\ \sigma\rangle \xrightarrow{\tau} \langle Q,\ \sigma\rangle$$

For the probabilistic nondeterministic choice $P \sqcap_p Q$, the probability of the selection of P is p and the probability of the selection of Q is $1 - p$.

$$\langle P \sqcap_p Q,\ \sigma\rangle \xrightarrow{c}_p \langle P,\ \sigma\rangle$$
$$\langle P \sqcap_p Q,\ \sigma\rangle \xrightarrow{c}_{1-p} \langle Q,\ \sigma\rangle$$

The process $P\ ;\ Q$ will behave like P initially. After P terminates, Q will be executed.

$$\text{if } \langle P,\ \sigma\rangle \xrightarrow{\beta} \langle \varepsilon,\ \sigma'\rangle, \quad \text{then } \langle P; Q,\ \sigma\rangle \xrightarrow{\beta} \langle Q,\ \sigma'\rangle$$
$$\text{if } \langle P,\ \sigma\rangle \xrightarrow{\beta} \langle P',\ \sigma'\rangle, \quad \text{then } \langle P; Q,\ \sigma\rangle \xrightarrow{\beta} \langle P'; Q,\ \sigma'\rangle$$

where, $P' \neq \varepsilon$, and $\xrightarrow{\beta}$ can be $\xrightarrow{\tau}$, $\xrightarrow{v}$, $\xrightarrow{c}_p$ and $\xrightarrow{1}$.

3.2 Probabilistic guarded choice

In order to investigate algebraic properties for parallel composition, we enrich the language with guarded choice. A guarded choice construct may perform transitions depicted in the following five cases.

(1) Let $P = []_{i\in I}\{[p_i]\ choice_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij})\ P_{ij})\}$.

$$\langle P,\ \sigma\rangle \xrightarrow{c}_{p_i} \langle P_{ij},\ \sigma[e_{ij}/x_{ij}]\rangle, \quad \text{if } b_{ij}(\sigma)$$

For any $i \in I$, there is only one $j \in J_i$ such that $b_{ij}(\sigma) = \textbf{true}$. This indicates that program P can execute assignment $x_{ij} := e_{ij}$ with probability p_i when the corresponding condition $b_{ij}(\sigma) = \textbf{true}$.

(2) Let $P = []_{i\in I}\{@b_i\ P_i\}$.

$$\langle P,\ \sigma\rangle \xrightarrow{v} \langle P_i,\ \sigma\rangle, \quad \text{if } b_i(\sigma)$$
$$\langle P,\ \sigma\rangle \xrightarrow{1} \langle P,\ \sigma\rangle, \quad \text{if } \bigwedge_{i\in I} b_i(\sigma) = \textbf{false}$$

If there exists i $(i \in I)$ such that $b_i(\sigma)$ is satisfied, the event $@b_i$ is enabled, thus P_i is followed, as depicted in the first rule. If $\forall i \bullet b_i(\sigma) = \textbf{false}$, no events are enabled, only time can advance, as indicated in the second rule.

(3) Let $P = []\{\#1\ R\}$.

$$\langle P,\ \sigma\rangle \xrightarrow{1} \langle R,\ \sigma\rangle$$

Process P can only do time advancing transition because it only contains time-delay component. The subsequent behaviour after one time unit elapses is just the behaviour of process R.

(4) Let $P = []_{i\in I}\{[p_i]\ choice_{j\in J}(b_{ij}\&(x_{ij} := e_{ij})\ P_{ij})\}$
$[][]_{k\in K}\{@c_k\ Q_k\}$

$$\langle P,\ \sigma\rangle \xrightarrow{v} \langle Q_k,\ \sigma\rangle, \quad \text{if } c_k(\sigma) = \textbf{true}$$
$$\langle P,\ \sigma\rangle \xrightarrow{c}_{p_i} \langle P_{ij},\ \sigma[e_{ij}/x_{ij}]\rangle,$$
$$\text{if } b_{ij}(\sigma) \wedge (\forall k \bullet c_k(\sigma) = \textbf{false}).$$

For any $k \in K$, if $c_k(\sigma) = \textbf{true}$, then the event "$@c_k$" is fired. The first transition reflects this fact. For any $i \in I$, if $b_{ij}(\sigma) = \textbf{true}$ and $\forall k \bullet c_k(\sigma) = \textbf{false}$, then process P can perform the corresponding assignment "$x_{ij} := e_{ij}$". Now consider the special case "$c_k(\sigma) = $ **true**" and $b_{ij}(\sigma) = \textbf{true}$, we only allow event "$@c_k$" to be fired and do not allow process P to perform assignment $x_{ij} := e_{ij}$. This fact is shown in the first transition and reflected by the additional condition "$\forall k \bullet c_k(\sigma) = $ **false**" in the second transition.

(5) Let $P = []_{i\in I}\{@b_i\ P_i\}[]\{\#1\ R\}$.

$$\langle P,\ \sigma\rangle \xrightarrow{v} \langle P_i,\ \sigma\rangle \quad \text{if } b_i(\sigma)$$
$$\langle P,\ \sigma\rangle \xrightarrow{1} \langle R,\ \sigma\rangle \quad \text{if } \bigwedge_{i\in I} b_i(\sigma) = \textbf{false}$$

The first transition indicates that event "$@b_i$" is fired if $b_i(\sigma) = \textbf{true}$. However, if all $b_i(\sigma)$ are evaluated to **false**, then only time-advance branch can be selected.

3.3 Parallel process

For brevity of presentation, we first define the following function to represent intermediate processes:

$$\mathbf{par}(P, Q, p_1) =_{df} \begin{cases} P \parallel_{p_1} Q & \text{if } P \neq \varepsilon \text{ and } Q \neq \varepsilon \\ P & \text{if } P \neq \varepsilon \text{ and } Q = \varepsilon \\ Q & \text{if } P = \varepsilon \text{ and } Q \neq \varepsilon \\ \varepsilon & \text{if } P = \varepsilon \text{ and } Q = \varepsilon \end{cases}$$

This intermediate format can reduce the number of transitions for parallel composition by representing several cases in one single rule.

Now we define two functions:

$$\mathit{stable}(\langle P, \sigma\rangle) =_{df} \neg(\ \langle P, \sigma\rangle \xrightarrow{\tau}\ ) \quad \text{and}$$
$$\mathit{stableE}(\langle P, \sigma\rangle) =_{df} \neg(\ \langle P, \sigma\rangle \xrightarrow{v}\ )$$

The notation $\mathit{stable}(\langle P, \sigma\rangle)$ indicates that process P cannot perform the transition representing nondeterministic choice under state σ, while $\mathit{stableE}(\langle P, \sigma\rangle)$ indicates that process P cannot perform event-triggered transitions under state σ.

A probabilistic parallel composition may perform transitions of the following forms:

(1) (a) If $\langle P,\ \sigma\rangle \xrightarrow{\tau} \langle P',\ \sigma\rangle$ and $\mathit{stable}(\langle Q, \sigma\rangle)$,
then $\langle P \parallel_{p_1} Q,\ \sigma\rangle \xrightarrow{\tau} \langle \mathbf{par}(P',\ Q,\ p_1),\ \sigma\rangle$.
$\langle Q \parallel_{p_1} P,\ \sigma\rangle \xrightarrow{\tau} \langle \mathbf{par}(Q,\ P',\ p_1),\ \sigma\rangle$.

(b) If $\langle P,\ \sigma\rangle \xrightarrow{\tau} \langle P',\ \sigma, \rangle$ and
$\langle Q,\ \sigma\rangle \xrightarrow{\tau} \langle Q',\ \sigma\rangle$,
then $\langle P \parallel_{p_1} Q,\ \sigma\rangle \xrightarrow{\tau} \langle \mathbf{par}(P',\ Q',\ p_1),\ \sigma\rangle$

(2) (a) If $\langle P,\ \sigma\rangle \xrightarrow{v} \langle P',\ \sigma\rangle$ and $\mathit{stable}(\langle Q, \sigma\rangle)$ and $\mathit{stableE}(\langle Q, \sigma\rangle)$,
then $\langle P \parallel_{p_1} Q,\ \sigma\rangle \xrightarrow{v} \langle \mathbf{par}(P',\ Q,\ p_1),\ \sigma\rangle$.
$\langle Q \parallel_{p_1} P,\ \sigma\rangle \xrightarrow{v} \langle \mathbf{par}(Q,\ P',\ p_1),\ \sigma\rangle$.

(b) If $\langle P,\ \sigma\rangle \xrightarrow{v} \langle P',\ \sigma\rangle$ and
$\langle Q,\ \sigma\rangle \xrightarrow{v} \langle Q',\ \sigma\rangle$,
then $\langle P \parallel_{p_1} Q,\ \sigma\rangle \xrightarrow{v} \langle \mathbf{par}(P',\ Q',\ p_1),\ \sigma\rangle$

(3) If $\langle P,\ \sigma\rangle \xrightarrow{c}_{p_2} \langle P',\ \sigma'\rangle$ and
$\mathit{stable}(\langle R, \sigma\rangle)$ and $\mathit{stableE}(\langle R, \sigma\rangle)$ $(R \in \{P,\ Q\})$,
then $\langle P \parallel_{p_1} Q,\ \sigma\rangle \xrightarrow{c}_{p_1 \times p_2} \langle \mathbf{par}(P',\ Q,\ p_1),\ \sigma'\rangle$
$\langle Q \parallel_{p_1} P,\ \sigma\rangle \xrightarrow{c}_{(1-p_1)\times p_2} \langle \mathbf{par}(Q,\ P',\ p_1),\ \sigma'\rangle$

(4) If $\langle P,\ \sigma\rangle \xrightarrow{1} \langle P',\ \sigma'\rangle$ and $\langle Q,\ \sigma\rangle \xrightarrow{1} \langle Q',\ \sigma'\rangle$ and
$\mathit{stable}(\langle R, \sigma\rangle)$ and $\mathit{stableE}(\langle R, \sigma\rangle)$ $(R \in \{P,\ Q\})$,
then $\langle P \parallel_{p_1} Q,\ \sigma\rangle \xrightarrow{1} \langle \mathbf{par}(P',\ Q',\ p_1),\ \sigma'\rangle$.

Transition (1)(a) stands for the case that one component makes nondeterministic choice and another component is stable. The whole process also makes a nondeterministic choice under this case. However, if both components make nondeterministic choice, then the whole process can make nondeterministic choice and the subsequent behaviour is the parallel composition of the remaining components. Transition (1)(b) reflects this situation.

The second type stands for the event-fired case. The analysis is similar to the transitions of type (1). Transition (3) covers the case of performing an atomic action. If process P can perform an atomic action with probability p_2, then process $P \parallel_{p_1} Q$ and $Q \parallel_{p_1} P$ can also perform the same atomic action with probability $p_1 \times p_2$ and $(1 - p_1) \times p_2$ respectively.

If both components can perform a time-advancing transition, then the whole parallel process can also let time advance. The aspect is reflected in transition (4).

4 Bisimulation

In operational semantics the behaviour of programs is represented in terms of execution steps. A computation is thus composed of a sequence of transitions. Two syntactically different programs may have the same observational behaviour. This means that we need to define program equivalence (conventionally denoted $\approx$) based on a reasonable abstraction. In considering the equivalence of the programs for our language, bisimulation is a useful approach. It will also be applied to explore algebraic laws of our language in the next section.

In what follows we shall give several auxiliary definitions before we present the definition for bisimulation.

Definition 1 We define the transition relation $\overset{id}{\Longrightarrow}_p$ over $\mathbb{C} \times \mathbb{C}$ as follows:

$$\begin{aligned} \langle P,\ \sigma\rangle \overset{id}{\Longrightarrow}_p \langle P',\ \sigma\rangle =_{df}\ & (P' = P \wedge p = 1)\ \vee \\ & (\exists n, P_1, p_1, \ldots, P_n, p_n \bullet \\ & \quad \langle P,\ \sigma\rangle \xrightarrow{\beta_1}_{p_1} \langle P_1,\ \sigma\rangle \ldots \xrightarrow{\beta_n}_{p_n} \langle P_n,\ \sigma\rangle \\ & \quad \wedge\ P_n = P' \wedge p = p_1 \times \cdots \times p_n) \end{aligned}$$

where $\xrightarrow{\beta_i}_{p_i}$ can be of the forms $\xrightarrow{c}_{p_i}$, $\xrightarrow{\tau}$ and $\xrightarrow{v}$. We assume $p_i = 1$ in the latter two cases.

Note that transition relation "$\overset{id}{\Longrightarrow}_p$" represents a sequence of transitions which keep the program state unchanged.

Definition 2 We define the following two transition relations over $\mathbb{C} \times \mathbb{C}$:

(1) $\langle P,\ \sigma\rangle \overset{c}{\Longrightarrow}_p \langle P',\ \sigma'\rangle$
$=_{df} \exists P_1 \bullet \langle P,\ \sigma\rangle \overset{id}{\Longrightarrow}_{p_1} \langle P_1,\ \sigma\rangle\ \ \wedge$

$$\langle P_1,\ \sigma\rangle \xrightarrow{c}_{p_2} \langle P',\ \sigma'\rangle \ \wedge$$
$$p = p_1 \times p_2$$

(2) $\langle P,\ \sigma\rangle \overset{1}{\Longrightarrow}_p \langle P',\ \sigma\rangle$
$=_{df} \exists P_1 \bullet \langle P,\ \sigma\rangle \overset{id}{\Longrightarrow}_p \langle P_1,\ \sigma\rangle \ \wedge$
$\langle P_1,\ \sigma\rangle \xrightarrow{1} \langle P',\ \sigma\rangle$

Note that the relation $\overset{c}{\Longrightarrow}_p$ is actually a composition of $\overset{id}{\Longrightarrow}_{p_1}$ and $\xrightarrow{c}_{p_2}$ ($p = p_1 \times p_2$), while $\overset{1}{\Longrightarrow}_p$ is a composition of $\overset{id}{\Longrightarrow}_p$ and $\xrightarrow{1}$. These two auxiliary relations will help us present our bisimulation relation in a more abstract manner with respect to atomic actions and time-advancing.

Definition 3 If a transition $\langle P,\ \sigma\rangle \overset{\beta}{\Longrightarrow}_{p_1} \langle P',\ \sigma'\rangle$ is duplicated n times, we denote it as

$\langle P,\ \sigma\rangle \overset{\beta}{\Longrightarrow}_{n,p_1} \langle P',\ \sigma'\rangle$

where $\overset{\beta}{\Longrightarrow}_{p_1}$ can be of the form $\overset{c}{\Longrightarrow}_{p_1}$ or $\overset{1}{\Longrightarrow}_{p_1}$.

Definition 4 A symmetric relation $R \subseteq \mathbb{C} \times \mathbb{C}$ is a bisimulation if and only if for all $\langle P, \sigma\rangle R \langle Q, \sigma\rangle$

(1) If $\langle P,\ \sigma\rangle \xrightarrow{x} \langle P',\ \sigma\rangle$,
then $\exists Q' \bullet \langle Q,\ \sigma\rangle(\xrightarrow{\tau} \ \vee \ \xrightarrow{v})^*\langle Q',\ \sigma\rangle \ \wedge$
$\langle P', \sigma\rangle R \langle Q', \sigma\rangle$.

where, $\xrightarrow{x}$ can be of the transition type $\xrightarrow{\tau}$ or $\xrightarrow{v}$.

(2) If $\langle P,\ \sigma\rangle \overset{c}{\Longrightarrow}_{n_1,p_1} \langle P',\ \sigma'\rangle$,

(2-1) if $\sigma \neq \sigma'$, then $\exists Q', n_2, p_2 \bullet$
$\langle Q,\ \sigma\rangle \overset{c}{\Longrightarrow}_{n_2,p_2} \langle Q',\ \sigma'\rangle \ \wedge$
$\langle P', \sigma'\rangle R \langle Q', \sigma'\rangle \ \wedge$
$n_1 \times p_1 = n_2 \times p_2$.

(2-2) if $\sigma = \sigma'$, then
$\langle P', \sigma'\rangle R \langle Q, \sigma'\rangle \ \wedge n_1 \times p_1 = 1$
$\vee \ \exists Q', n_2, p_2 \bullet$
$\langle Q,\ \sigma\rangle \overset{c}{\Longrightarrow}_{n_2,p_2} \langle Q',\ \sigma'\rangle \ \wedge$
$\langle P', \sigma'\rangle R \langle Q', \sigma'\rangle \ \wedge$
$n_1 \times p_1 = n_2 \times p_2$.

(3) If $\langle P,\ \sigma\rangle \overset{1}{\Longrightarrow}_{n_1,p_1} \langle P',\ \sigma\rangle$,
then $\exists Q', n_2, p_2 \bullet$
$\langle Q,\ \sigma\rangle \overset{1}{\Longrightarrow}_{n_2,p_2} \langle Q',\ \sigma\rangle \ \wedge$
$\langle P', \sigma\rangle R \langle Q', \sigma\rangle \ \wedge$
$n_1 \times p_1 = n_2 \times p_2$.

Two configurations should have the same interface when studying their equivalence; i.e., their state parts should be the same. Our bisimulation relation is based on three different kinds of observations: a τ transition or a v-transition (triggered by an event), atomic action, and time advancing.

Note that (2-2) in the above definition models the case that if a process performs an atomic action without any contribution to the program state change, then its bisimilar process may or may not perform an atomic action with similar effect. This partly reflects the concept of *weak bisimulation* [19,20].

Lemma 1 *If S_1 and S_2 are bisimulations (over $\mathbb{C} \times \mathbb{C}$), then the following relations are also bisimulations (over $\mathbb{C} \times \mathbb{C}$):*

(1) Id (2) $S_1 \circ S_2$ (3) $S_1 \cup S_2$

where, Id is the identity relation and $S_1 \circ S_2$ stands for the relational composition of S_1 and S_2.

Definition 5 (1) Two configurations $\langle P_1, \sigma\rangle$ and $\langle P_2, \sigma\rangle$ are bisimilar, written as $\langle P_1, \sigma\rangle \approx \langle P_2, \sigma\rangle$, if there exists a bisimulation relation R such that $\langle P_1, \sigma\rangle R \langle P_2, \sigma\rangle$.
(2) Two processes P and Q are bisimilar, denoted as $P \approx Q$, if for any state σ

$\langle P,\ \sigma\rangle \approx \langle Q,\ \sigma\rangle$

Lemma 2 *$\approx$ is an equivalence relation over $\mathbb{C} \times \mathbb{C}$.*

Theorem 3 *$\approx$ is a congruence.*

Proof We can proceed the proof by structural induction. The detailed proof is left in Appendix A. □

This theorem indicates that bisimilar relation $\approx$ is preserved by all operators.

5 Algebraic laws

Operational semantics can be used to deduce interesting properties of programs. In this section, we investigate the algebraic laws of our timed language with probability and shared-variable concurrency.

For assignment, conditional, iteration, nondeterministic choice and sequential composition, our language enjoys similar algebraic properties as those reported in [9,12]. In what follows, we shall only focus on novel algebraic properties with respect to time, probabilistic nondeterministic choice and parallel composition.

Two consecutive time delays can be combined into a single one, where the length of the delay is the sum of the original two lengths.

(delay-1) $\#n; \#m = \#(n+m)$

Probabilistic nondeterministic choice is idempotent.

(prob-1) $P \sqcap_p P = P$

However, it is not purely symmetric and associative. Its symmetry and associativity rely on the change of the associated probabilities:

(prob-2) $P \sqcap_{p_1} Q = Q \sqcap_{1-p_1} P$

(prob-3) $P \sqcap_p (Q \sqcap_q R) = (P \sqcap_x Q) \sqcap_y R$
where $x = p/(p + q - p \times q)$ and $y = p + q - p \times q$

The proof for law (prob-3) is given in Appendix B. Other proofs are omitted (except for the proof of law (par-3-1)).

Sequential composition also distributes through probabilistic nondeterministic choice.

(prob-4) $P; (Q \sqcap_{p_1} R) = (P; Q) \sqcap_{p_1} (P; R)$

(prob-5) $(P \sqcap_{p_1} Q); R = (P; R) \sqcap_{p_1} (Q; R)$

Probabilistic parallel composition is also not purely symmetric and associative. Its symmetry and associativity rely on the change of the associated probabilities as well.

(par-1) $P \parallel_p Q = Q \parallel_{1-p} P$
(par-2) $P \parallel_p (Q \parallel_q R) = (P \parallel_x Q) \parallel_y R$
where, $x = p/(p + q - p \times q)$ and $y = p + q - p \times q$

In what follows we give a collection of parallel expansion laws, which enable us to expand a probabilistic parallel composition to a guarded choice construct. As mentioned earlier, there exist five types of guarded choice. To take into account a parallel composition of two arbitrary guarded choices, we end up with fifteen different expansion laws.

The first five laws we shall discuss deal with parallel compositions where the first component is an assignment-guarded choice.

If the second component is also an assignment-guarded choice, the scheduling rule is that any assignment could be scheduled with the associated probability provided that its Boolean condition is satisfied. Suppose the assignment guard from the first component is scheduled, the subsequent behaviour is the parallel composition of the remaining process of the first component with the whole second component. Law (par-3-1) below depicts this case.

(par-3-1) Let

$P = []_{i \in I}\{[p_i]\, choice_{j \in J_i}(b_{ij} \& (x_{ij} := e_{ij})\, P_{ij})\}$ and
$Q = []_{k \in K}\{[q_k]\, choice_{l \in L_k}(b_{kl} \& (x_{kl} := e_{kl})\, Q_{kl})\}$

Then

$$\begin{aligned} P \parallel_r Q \\ = []_{i \in I}\{[r \times p_i]\, choice_{j \in J_i}(b_{ij} \& (x_{ij} := e_{ij}) \\ \mathbf{par}(P_{ij}, Q, r)\} \\ [][]_{k \in K}\{[(1 - r) \times q_k]\, choice_{l \in L_k}(b_{kl} \& (x_{kl} := e_{kl}) \\ \mathbf{par}(P, Q_{kl}, r)\} \end{aligned}$$

Proof The transitions of P can be described as below:

$\langle P, \sigma \rangle \xrightarrow{c}_{p_i} \langle P_{ij}, \sigma[e_{ij}/x_{ij}] \rangle$, if $b_{ij}(\sigma)$
$\langle Q, \sigma \rangle \xrightarrow{c}_{q_k} \langle Q_{kl}, \sigma[e_{kl}/x_{kl}] \rangle$, if $b_{kl}(\sigma)$

For the above algebraic laws, based on the transition rules for parallel composition, we can have the transition for $P \parallel_r Q$ (LHS).

$\langle LHS, \sigma \rangle \xrightarrow{c}_{r \times p_i} \langle P_{ij}, \sigma[e_{ij}/x_{ij}] \rangle$, if $b_{ij}(\sigma)$
$\langle LHS, \sigma \rangle \xrightarrow{c}_{(1-r) \times q_k} \langle Q_{kl}, \sigma[e_{kl}/x_{kl}] \rangle$, if $b_{kl}(\sigma)$

For the guarded choice expansion of the algebraic laws (RHS), its transition rules can be described as:

$\langle RHS, \sigma \rangle \xrightarrow{c}_{r \times p_i} \langle P_{ij}, \sigma[e_{ij}/x_{ij}] \rangle$, if $b_{ij}(\sigma)$
$\langle RHS, \sigma \rangle \xrightarrow{c}_{(1-r) \times q_k} \langle Q_{kl}, \sigma[e_{kl}/x_{kl}] \rangle$, if $b_{kl}(\sigma)$

Because the two processes (LHS and RHS) have the same transitions, we can say that their behaviour are equal (also bisimilar). □

If the second component is an event-guarded choice, the behaviour of the parallel composition can be described as the guarded choice of a set of assignment-guarded components and a set of event-guarded components. If an assignment guard (from the first component) is scheduled first, the subsequent behaviour is the parallel composition of the remaining part of the first component (P_{ij}) with the second component (Q); if an event guard is triggered, the subsequent behaviour is the parallel composition of the first component (P) with the remaining part of the second component (Q_k). This is presented in law (par-3-2).

(par-3-2) Let

$P = []_{i \in I}\{[p_i]\, choice_{j \in J_i}(b_{ij} \& (x_{ij} := e_{ij})\, P_{ij})\}$

and

$Q = []_{k \in K}\{@c_k\, Q_k\}$

Then

$$\begin{aligned} P \parallel_r Q \\ = []_{i \in I}\{[p_i]\, choice_{j \in J_i}(b_{ij} \& (x_{ij} := e_{ij})\, \mathbf{par}(P_{ij}, Q, r)\} \\ [][]_{k \in K}\{@c_k\, \mathbf{par}(P, Q_k, r)\} \end{aligned}$$

If the second component is a time delay construct, then only assignment guards can be scheduled initially. For the whole parallel composition, the subsequent behaviour following the scheduled assignment guard is the parallel composition of the remaining part of the first component (P_{ij}) with the time delay component. The whole process does not have time delay component. This case is expressed in law (par-3-3).

(par-3-3) Let

$P = []_{i \in I}\{[p_i]\, choice_{j \in J_i}(b_{ij} \& (x_{ij} := e_{ij})\, P_{ij})\}$ and
$Q = []\{\#1\ R\}$

Then

$$\begin{aligned} P \parallel_r Q \\ = []_{i \in I}\{[p_i]\, choice_{j \in J_i}(b_{ij} \& (x_{ij} := e_{ij})\, \mathbf{par}(P_{ij}, Q, r))\} \end{aligned}$$

If the second component is a guarded choice comprising a set of assignment components and a set of event guarded components, the guarded choice for the parallel process also contains assignment components. The probability of the assignment should be updated based on the associated probability for the assignments from each parallel branch and the probability of the parallel composition. Further, it also contains the event components. Law (par-3-4) expresses this case.

(par-3-4) Let

$$P = []_{i\in I}\{[p_i]\ choice_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij})\ P_{ij})\}\text{ and}$$
$$Q = []_{k\in K}\{[q_k]\ choice_{l\in L_k}(b_{kl}\&(x_{kl} := e_{kl})\ Q_{kl})\}\ [][]_{m\in M}\{@c_m\ R_m\}$$

Then

$$\begin{aligned}
&P \parallel_r Q\\
&= []_{i\in I}\{[r\times p_i]\ choice_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij})\ \mathbf{par}(P_{ij}, Q, r))\}\\
&\quad [][]_{k\in K}\{[(1-r)\times q_k]\ choice_{l\in L_k}(b_{kl}\&(x_{kl} := e_{kl})\ \mathbf{par}(P, Q_{kl}, r)\}\\
&\quad [][]_{m\in M}\{@c_k\ \mathbf{par}(P, R_m, r)\}
\end{aligned}$$

If the second component is composed of a set of event guard component and the time delay component, the parallel process only has assignment components and event components. It does not have time delay component due to the assignment components in the first parallel branch.

(par-3-5) Let

$$P = []_{i\in I}\{[p_i]\ choice_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij})\ P_{ij})\}\text{ and}$$
$$Q = []_{l\in L}\{@c_l\ Q_l\}[]\{\#1\ R\}$$

Then

$$\begin{aligned}
&P \parallel_r Q\\
&= []_{i\in I}\{[p_i]\ choice_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij})\ \mathbf{par}(P_{ij}, Q, r))\}\\
&\quad [][]_{l\in L}\{@c_l\ \mathbf{par}(P, Q_l, r)\}
\end{aligned}$$

In what follows, we consider parallel compositions where the first component is an event-guarded choice, while the second component can be of any form.

If the second component is also an event-guarded choice, there are several scenarios. If one guard from the first component is triggered but no guards from the second component are triggered, the subsequent behaviour is the parallel composition of the remaining part of the first component with the second component. If two guards from both components are triggered simultaneously, the subsequent behaviour is the parallel composition of the remaining processes of both sides. This is illustrated in law (par-3-6).

(par-3-6) Let $P = []_{i\in I}\{@b_i\ P_i\}$ and $Q = []_{j\in J}\{@c_j\ Q_j\}$

Then

$$\begin{aligned}
&P \parallel_r Q\\
&= []_{i\in I}\{@(b_i\wedge\neg c)\ \mathbf{par}(P_i, Q, r)\}\\
&\quad [][]_{j\in J}\{@(c_j\wedge\neg b)\ \mathbf{par}(P, Q_j, r)\}\\
&\quad [][]_{i\in I\wedge j\in J}\{@(b_i\wedge c_j)\ \mathbf{par}(P_i, Q_j, r)\}
\end{aligned}$$

where, $b = \vee_{i\in I}\ b_i$ and $c = \vee_{j\in J}\ c_j$

If the second component is a time delay construct, the whole process will wait for some events to be triggered. The whole process can also let time advance. Law (par-3-7) expresses this case.

(par-3-7) Let $P = []_{i\in I}\{@b_i\ P_i\}$ and $Q = []\{\#1\ R\}$

Then

$$P \parallel_r Q = []_{i\in I}\{@b_i\ \mathbf{par}(P_i, Q, r)\}[]\{\#1\ \mathbf{par}(P, R, r)\}$$

If the second component is the guarded choice of a set of assignment-guarded components and a set of event-guarded components, any assignment guard can be scheduled. As both components have event-guard components, there are also three possibilities for the guards to be triggered, as discussed in (par-3-6). This case is described in law (par-3-8).

(par-3-8) Let $P = []_{i\in I}\{@b_i\ P_i\}$ and

$$Q = []_{j\in J}\{[q_j]\ choice_{k\in K_j}(b_{jk}\&(x_{jk} := e_{jk})\ Q_{jk})\}\ [][]_{l\in L}\{@c_l\ R_l\}$$

Then

$$\begin{aligned}
&P \parallel_r Q\\
&= []_{j\in J}\{[q_j]\ choice_{k\in K_j}(b_{ij}\&(x_{jk} := e_{jk})\ \mathbf{par}(P_{jk}, Q, r))\}\\
&\quad [][]_{i\in I}\{@(b_i\wedge\neg c)\ \mathbf{par}(P_i, Q, r)\}\\
&\quad [][]_{l\in L}\{@(c_l\wedge\neg b)\ \mathbf{par}(P, R_l, r)\}\\
&\quad [][]_{i\in I\wedge l\in L}\{@(b_i\wedge c_l)\ \mathbf{par}(P_i, Q_l, r)\}
\end{aligned}$$

where, $b, = \vee_{i\in I}\ b_i$ and $c = \vee_{l\in L}\ c_l$

If the second component is composed of a set of event components and time components, the whole parallel process also has event components and their structure is similar to the one in law (par-3-8). Further, the parallel process also has time delay component. Law (par-3-9) describes this case.

(par-3-9) Let $P = []_{i\in I}\{@b_i\ P_i\}$ and $Q = []_{j\in J}\{@c_j\ Q_j\}[]\{\#1\ R\}$

Then

$$\begin{aligned} & P \parallel_r Q \\ & \quad = [\,]_{i\in I}\{@(b_i \wedge \neg c)\ \mathbf{par}(P_i, Q, r)\} \\ & \quad\quad [\,][\,]_{j\in J}\{@(c_j \wedge \neg b)\ \mathbf{par}(P, Q_j, r)\} \\ & \quad\quad [\,][\,]_{i\in I \wedge j\in J}\{@(b_i \wedge c_j)\ \mathbf{par}(P_i, Q_j, r)\} \\ & \quad\quad [\,]\{\#1\ \mathbf{par}(P, R, r)\} \end{aligned}$$

where, $b = \vee_{i\in I}\, b_i$ and $c = \vee_{j\in J}\, c_j$

We shall next consider the parallel composition where the first component is a time-delay guarded construct.

The following law captures the case where the second component is also a time-delay guarded construct. The whole process performs a time delay and then behaves as the parallel composition of the remaining parts from both sides:

(par-3-10) Let $P = [\,]\{\#1\ R\}$ and $Q = [\,]\{\#1\ T\}\}$

Then

$$P \parallel_r Q = [\,]\{\#1\ \mathbf{par}(R, T, r)\}$$

If the second component is composed of a set of assignment components and a set of event components, the whole parallel process also has these types of components. It does not have time delay components due to the assignment components in the second parallel branch.

(par-3-11) Let

$$\begin{aligned} & P = [\,]\{\#1\ T\} \ \text{ and} \\ & Q = [\,]_{i\in I}\{[q_i]\ choice_{j\in J_i}(b_{jk}\&(x_{ij} := e_{ij})\ Q_{ij})\} \\ & \quad\quad [\,][\,]_{k\in K}\{@c_k\ R_k\} \end{aligned}$$

Then

$$\begin{aligned} & P \parallel_r Q \\ & \quad = [\,]_{i\in I}\{[q_i]\ choice_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij})\ \mathbf{par}(P, Q_{ij}, r))\} \\ & \quad\quad [\,][\,]_{k\in K}\{@c_k\ \mathbf{par}(P, R_k, r)\} \end{aligned}$$

If the second component is composed of a set of event guard components and time delay component, the whole process also has this structure. Law (par-3-12) shows this case.

(par-3-12) Let $P = [\,]\{\#1\ T\}$ and
$Q = [\,]_{i\in I}\{@b_i\ Q_i\}[\,]\{\#1\ R\}$

Then

$$P \parallel_r Q = [\,]_{i\in I}\{@b_i\ \mathbf{par}(P, Q_i, r)\}[\,]\{\#1\ \mathbf{par}(T, R, r)\}$$

We now move to the parallel composition where the first component comprises both assignment-guarded components and event-guarded components. Law (par-3-13) stands for the case that the second component also has this type of structures. For the whole parallel process, assignment can be from any of the two parallel branches with the probability updated.

(par-3-13) Let

$$\begin{aligned} & P = [\,]_{i\in I}\{[p_i]\ choice_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij})\ P_{ij})\} \\ & \quad\quad [\,][\,]_{k\in K}\{@b_k\ R_k\} \end{aligned}$$

and

$$\begin{aligned} & Q = [\,]_{l\in L}\{[q_l]\ choice_{m\in M_l}(c_{lm}\&(x_{lm} := e_{lm})\ P_{lm})\} \\ & \quad\quad [\,][\,]_{n\in N}\{@c_n\ T_n\} \end{aligned}$$

Then

$$\begin{aligned} & P \parallel_r Q \\ & \quad = [\,]_{i\in I}\{[r \times p_i]\ choice_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij})\ \mathbf{par}(P_{ij}, Q, r))\} \\ & \quad\quad [\,][\,]_{l\in L}\{[(1-r) \times q_l]\ choice_{m\in M_l}(c_{lm}\&(x_{lm} := e_{lm}) \\ & \quad\quad\quad\quad \mathbf{par}(P, Q_{lm}, r))\} \\ & \quad\quad [\,][\,]_{k\in K}\{@(b_k \wedge \neg c)\ \mathbf{par}(R_k, Q, r)\} \\ & \quad\quad [\,][\,]_{n\in N}\{@(c_n \wedge \neg b)\ \mathbf{par}(R_k, Q, r)\} \\ & \quad\quad [\,][\,]_{k\in K \wedge n\in N}\{@(b_k \wedge c_n)\ \mathbf{par}(R_k, Q_n, r)\} \end{aligned}$$

where, $b = \vee_{k\in K}\, b_k$ and $c = \vee_{n\in N}\, c_n$

The following law (par-3-14) captures the scenario where the second component consists of both event-guarded choices and time-delays:

(par-3-14) Let

$$\begin{aligned} & P = [\,]_{i\in I}\{[p_i]\ choice_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij})\ P_{ij})\} \\ & \quad\quad [\,][\,]_{k\in K}\{@b_k\ R_k\} \end{aligned}$$

and $Q = [\,]_{l\in L}\{@c_l\ Q_l\}[\,]\{\#1\ T\}$

Then

$$\begin{aligned} & P \parallel_r Q \\ & \quad = [\,]_{i\in I}\{[p_i]\ choice_{j\in J}(b_{ij}\&(x_{ij} := e_{ij})\ \mathbf{par}(P_{ij}, Q, r))\} \\ & \quad\quad [\,][\,]_{k\in K}\{@(b_k \wedge \neg c)\ \mathbf{par}(R_k, Q, r)\} \\ & \quad\quad [\,][\,]_{l\in L}\{@(c_l \wedge b)\ \mathbf{par}(P, Q_l, r)\} \\ & \quad\quad [\,][\,]_{k\in K \wedge l\in L}\{@(b_k \wedge c_l)\ \mathbf{par}(R_k, Q_l, r)\} \end{aligned}$$

where, $b = \vee_{k\in K}\, b_k$ and $c = \vee_{l\in L}\, c_l$

This law is similar to the law (par-3-13) given above.

The following law (par-3-15) is about the parallel composition of two guarded choices composing of both event-guarded components and time delay components.

(par-3-15) Let $P = [\,]_{i\in I}\{@b_i\ P_i\}[\,]\{\#1\ R\}$ and
$Q = [\,]_{j\in J}\{@c_j\ Q_j\}[\,]\{\#1\ T\}$

Then

$$
\begin{aligned}
&P \parallel_r Q\\
&\quad = []_{k\in K}\{@(b_i \wedge \neg c)\ \mathbf{par}(P_i, Q, r)\}\\
&\qquad [][]_{j\in J}\{@(c_j \wedge \neg b)\ \mathbf{par}(P, Q_j, r)\}\\
&\qquad [][]_{j\in J}\{@(b_i \wedge c_j)\ \mathbf{par}(P_i, Q_j, r)\}\\
&\qquad []\{\#1\ \mathbf{par}(R, T, r)\}
\end{aligned}
$$

where, $b = \vee_{i\in I}\, b_i$ and $c = \vee_{j\in J}\, c_j$

6 Animation of operational semantics

Operational semantics provides a set of transition rules that models how a program performs step by step. If we can have an executed version of operational semantics, the correctness of operational semantics can be checked from various test results. This means that a simulator for the operational semantics of our proposed language is highly desirable.

Transition rules for the operational semantics can be translated into Prolog logic programming clauses [3]. Prolog has been successfully applied in rapid-prototyping, including [1,2]. Building on this, for the development of the simulator of *PTSC*, we have selected Prolog as our programming language.

The configuration in a transition can be expressed as a list (indicated by square brackets) in Prolog:

$[\,P,\ Sigma\,]$

where state $Sigma$ can also be implemented as a list that contains values for program variables.

The transitions for the operational semantics can be directly translated into Prolog clauses. For some individual specific transitions, there may be several transitions for that kind of transition expressed in Prolog because it covers several different cases. For example, for the transition (3) of parallel composition (see p. 5):

$$
\begin{aligned}
&\text{If}\ \ \langle P,\ \sigma\rangle \xrightarrow{c}_{p_2} \langle P',\ \sigma'\rangle\ \text{ and}\\
&\qquad stable(\langle R,\sigma\rangle)\ \text{ and }\ stableE(\langle R,\sigma\rangle)(R\in\{P,\ Q\}),\\
&\text{then}\ \ \langle P \parallel_{p_1} Q,\ \sigma\rangle \xrightarrow{c}_{p_1\times p_2} \langle \mathbf{par}(P',\ Q,\ p_1),\ \sigma'\rangle.\\
&\qquad \langle Q \parallel_{p_1} P,\ \sigma\rangle \xrightarrow{c}_{(1-p_1)\times p_2} \langle \mathbf{par}(Q,\ P',\ p_1),\ \sigma'\rangle.
\end{aligned}
$$

This transition can be translated into Prolog clauses shown in Fig. 1. Every clause can represent the specific individual case.

Next we use the example below to demonstrate how the simulator works. Consider the program $(x := x+1\ ;\ x := x+2) \parallel_{0.4} (x := 2\ ;\ x := 4)$. The execution obtained from the simulator is shown in Fig. 2. From the execution based on the simulator, we know there are six execution sequences leading the program to the terminating state, where the transitions contain their own specific probability.

In total, there are six execution sequences leading the original program to the terminating state, where each transition step contains probability either 0.4, 0.6 or 1. These six execution sequences are:

(1)(2)(3)(4)(5)	(1)(2)(6)(7)(8)
(1)(2)(6)(9)(10)	(1)(11)(12)(13)(14)
(1)(11)(15)(16)(17)	(1)(11)(15)(18)(19)

where (i) stands for row i (given at the end of row i).

In summary, the simulator can display the execution sequences of programs based on the operational semantics. It provides an animation tool for our operational semantics. From various test examples including those above, the results displayed give us additional confidence concerning the validity of the operational semantics.

Fig. 1 Transition rules in Prolog: an example

$$\frac{[\,S1,\ Sigma\,] \text{—}['c', Y] \longrightarrow [\,epsilon,\ Sigma1\,] \wedge stableB(S1, S2, Sigma)}{[S1 \mid X \mid S2,\ Sigma] \text{—}['c', X*Y] \longrightarrow [S2,\ Sigma1\,]}$$

$$\frac{[\,S1,\ Sigma\,] \text{—}['c', Y] \longrightarrow [\,epsilon,\ Sigma1\,] \wedge stableB(S1, S2, Sigma)}{[S2 \mid X \mid S1,\ Sigma] \text{—}['c', (1-X)*Y] \longrightarrow [S2,\ Sigma1\,]}$$

$$\frac{[\,S1,\ Sigma\,] \text{—}['c', Y] \longrightarrow [\,S11,\ Sigma1\,]\ \wedge\ S11 \sim= epsilon \wedge stableB(S1, S2, Sigma)}{[\,S1|X|S2,\ Sigma\,]\ \text{—}['c', X*Y] \longrightarrow [\,S11|X|S2,\ Sigma1\,]}$$

$$\frac{[\,S1,\ Sigma\,]\ \text{—}['c', Y] \longrightarrow [\,S11,\ Sigma1\,]\ \wedge\ S11 \sim= epsilon \wedge stableB(S1, S2, Sigma)}{[\,S2|X|S1,\ Sigma\,]\ \text{—}['c', (1-X)*Y] \longrightarrow [\,S2|X|S11,\ Sigma1\,]}$$

Note that, $stableB(P,Q,\sigma) =_{df} stable(\langle P,\sigma\rangle) \wedge stableE(\langle P,\sigma\rangle) \wedge stable(\langle Q,\sigma\rangle) \wedge stableE(\langle Q,\sigma\rangle)$. We use "$S1\,|X|\,S2$" to represent "$S1 \parallel_X S2$" and "$\text{—}['c', X]\ \longrightarrow$" to represent "$\xrightarrow{c}_X$" in the Prolog clauses. Meanwhile, "$epsilon$" stands for the empty process ε and $\sim=$ stands for $\neq$.

Fig. 2 The simulation: an example

```
? − track    [(x := x + 1; x := x + 2) |0.4| (x := 2; x := 4), [x = 0]].   (1)
1 —[0, 0.4] ⟶ [x = x + 2 |0.4| x = 2; x = 4, [x = 1]]                    (2)
2 —[0, 0.4] ⟶ [x = 2; x = 4, [x = 3]]                                    (3)
3 —[0, 1]   ⟶ [x = 4, [x = 2]]                                           (4)
4 —[0, 1]   ⟶ [epsilon, [x = 4]]                                         (5)
2 —[0, 0.6] ⟶ [x = x + 2 |0.4| x = 4, [x = 2]]                           (6)
3 —[0, 0.4] ⟶ [x = 4, [x = 4]]                                           (7)
4 —[0, 1]   ⟶ [epsilon, [x = 4]]                                         (8)
3 —[0, 0.6] ⟶ [x = x + 2, [x = 4]]                                       (9)
4 —[0, 1]   ⟶ [epsilon, [x = 6]]                                         (10)
1 —[0, 0.6] ⟶ [x = x + 1; x = x + 2 |0.4| x = 4, [x = 2]]                (11)
2 —[0, 0.6] ⟶ [x = x + 1; x = x + 2, [x = 4]]                            (12)
3 —[0, 1]   ⟶ [x = x + 2, [x = 5]]                                       (13)
4 —[0, 1]   ⟶ [epsilon, [x = 7]]                                         (14)
2 —[0, 0.4] ⟶ [x = x + 2 |0.4| x = 4, [x = 3]]                           (15)
3 —[0, 0.4] ⟶ [x = 4, [x = 5]]                                           (16)
4 —[0, 1]   ⟶ [epsilon, [x = 4]]                                         (17)
3 —[0, 0.6] ⟶ [x = x + 2, [x = 4]]                                       (18)
4 —[0, 1]   ⟶ [epsilon, [x = 6]]                                         (19)
```

Note that the notation $[X, Y]$ in the displayed execution sequence indicates that X is the duration of the transition and Y is the probability of the transition. Meanwhile, "*track*" is the command in the simulator to display the execution sequence.

7 Conclusion

In this paper, we integrated probability with a timed concurrent language. The probability feature was added into non-deterministic choice, parallel composition and guarded choice. Parallel processes communicate with each other via shared variables. We formalized a SOS for the language which incorporates time, concurrency and probability. Transitions are classified into four types: probabilistic atomic action transition, time delay transition, non-deterministic selection transition and event triggered transition. On top of the operational model, an abstract bisimulation relation was defined and a rich set of algebraic laws have been derived for program equivalence. A prototype was also built for the animation of the execution of probabilistic programs.

As an immediate future work, it would be interesting to work out a denotational model and investigate the consistency between operational and denotational models. We would like to explore an observation-oriented model as advocated in *UTP* [12], which, we believe, would make the linking theory easier to build. The probabilistic trace structure could be applied in achieving the *UTP*-based denotational model.

It is also worth mentioning that the approach we integrate probability with other language features is rather general. It can be adapted when adding the probability feature into complex computational models catered for various application domains. Internet-based computing can be such an application domain. With the development of Internet technology, web services and transactions have been becoming more and more important in practice. It would be interesting to see if probability can be added into the recently proposed web services and web transaction models [13,14,30,31]. It will also be interesting to see if probability can be taken into account when reasoning about business processes [15].

Acknowledgments This work is supported in part by National Basic Research Program of China (No. 2005CB321904), National High Technology Research and Development Program of China (No. 2007AA01 0302), National Natural Science Foundation of China (No. 90718004), Macau Science and Technology Development PEARL project (No. 041/2007/A3) and Shanghai Leading Academic Discipline Project (No. B412). Shengchao Qin is supported in part by the UK EPSRC funded project EP/E021948/1. Jonathan P. Bowen is a Visiting Professor at King's College London and an Emeritus Professor at London South Bank University.

Appendix

A Proof of Theorem 3: $\approx$ is a congruence

Assume $P \approx Q$. We know for any state σ, $\langle P, \sigma\rangle \approx \langle Q, \sigma\rangle$. This means there exists a bisimulation S_σ such that $\langle P, \sigma\rangle \approx \langle Q, \sigma\rangle$. Let $S = \cup_\sigma S_\sigma$. We know S is also a bisimulation.

(1) For the proof of $P\ ;\ R \approx Q\ ;\ R$, let

$$S_{1,1} =_{df} Id \cup \{ (\langle P;\ R,\ \sigma\rangle,\ \langle Q;\ R,\ \sigma\rangle) \mid \langle P,\ \sigma\rangle\ S\ \langle Q,\ \sigma\rangle \}$$

For the proof of $R\ ;\ P \approx R\ ;\ Q$, let

$$S_{1,2} =_{df} S \cup \{ (\langle R;\ P,\ \sigma\rangle,\ \langle R;\ Q,\ \sigma\rangle) \mid \langle P,\ \sigma\rangle\ S\ \langle Q,\ \sigma\rangle \}$$

(2) For the proof of

if b **then** P **else** $R \approx$ **if** b **then** P **else** R,

let

$$S_{2,1} =_{df} Id \cup S \cup \{ (\langle \textbf{if } b \textbf{ then } P \textbf{ else } R, \sigma\rangle, \langle \textbf{if } b \textbf{ then } Q \textbf{ else } R, \sigma\rangle \mid \langle P, \sigma\rangle \, S \, \langle Q, \sigma\rangle \},$$

For the proof of

if b **then** R **else** $P \approx$ **if** b **then** R **else** Q,

let

$$S_{2,2} =_{df} Id \cup S \cup \{ (\langle \textbf{if } b \textbf{ then } R \textbf{ else } P, \sigma\rangle, \langle \textbf{if } b \textbf{ then } R \textbf{ else } Q, \sigma\rangle) \mid \langle P, \sigma\rangle \, S \, \langle Q, \sigma\rangle \}$$

(3) For the proof of **while** b **do** $P \approx$ **while** b **do** Q, let

$$S_3 =_{df} Id \cup \{ (\langle \textbf{while } b \textbf{ do } P, \sigma\rangle, \langle \textbf{while } b \textbf{ do } Q, \sigma\rangle) \mid \langle P, \sigma\rangle \, S \, \langle Q, \sigma\rangle \} \cup \{ (\langle U; \textbf{while } b \textbf{ do } P, \sigma\rangle, \langle V; \textbf{while } b \textbf{ do } Q, \sigma\rangle) \mid \langle P, \sigma\rangle \, S \, \langle Q, \sigma\rangle \wedge \langle U, \sigma\rangle \, S \, \langle V, \sigma\rangle \}$$

(4) For the proof of $P \sqcap R \approx Q \sqcap R$, let

$$S_{4,1} =_{df} Id \cup S \cup \{(\langle P \sqcap R, \sigma\rangle, \langle Q \sqcap R, \sigma\rangle) \mid \langle P, \sigma\rangle \, S \, \langle Q, \sigma\rangle\}$$

For the proof of $R \sqcap P \approx R \sqcap Q$, let

$$S_{4,2} =_{df} Id \cup S \cup \{(\langle R \sqcap P, \sigma\rangle, \langle R \sqcap Q, \sigma\rangle) \mid \langle P, \sigma\rangle \, S \, \langle Q, \sigma\rangle\}$$

(5) For the proof of $P \sqcap_p R \approx Q \sqcap_p R$, let

$$S_{5,1} =_{df} Id \cup S \cup \{(\langle P \sqcap_p R, \sigma\rangle, \langle Q \sqcap_p R, \sigma\rangle) \mid \langle P, \sigma\rangle S \langle Q, \sigma\rangle\}$$

For the proof of $R \sqcap_p P \approx R \sqcap_p Q$, let

$$S_{5,2} =_{df} Id \cup S \cup \{(\langle R \sqcap_p P, \sigma\rangle, \langle R \sqcap_p Q, \sigma\rangle) \mid \langle P, \sigma\rangle S \langle Q, \sigma\rangle\}$$

(6) For the proof of the probabilistic guarded choice, without loss of generality, we only consider the first type of guarded choice here. Let

$$T_1 = [] \{ [p] \, choice(b\&(x := e) \, P, \; G1), \; G2\} \quad \text{and}$$
$$T_2 = [] \{ [p] \, choice(b\&(x := e) \, Q, \; G1), \; G2\}$$

In order to consider the proof of $T_1 \approx T_2$, let

$$S_6 =_{df} Id \cup S \cup \{(\langle T_1, \sigma\rangle, \langle T_2, \sigma\rangle) \mid \langle P, \sigma\rangle, \, S \, \langle Q, \sigma\rangle\}$$

(7) For the proof of $P \parallel_p R \approx Q \parallel_p R$, let

$$S_{7,1} =_{df} Id \cup S \cup \{(\langle P \parallel_p R, \sigma\rangle, \langle Q \parallel_p R, \sigma\rangle) \mid \langle P, \sigma\rangle S \langle Q, \sigma\rangle\}$$

For the proof of $R \parallel_p P \approx R \parallel_p Q$, let

$$S_{7,2} =_{df} Id \cup S \cup \{(\langle R \parallel_p P, \sigma\rangle, \langle R \parallel_p P, \sigma\rangle) \mid \langle P, \sigma\rangle S \langle Q, \sigma\rangle\}$$

We can show that each $S_{i,j}$ (or S_i)is a bisimulation. □

B Proof of Law *(prob-3)*

Now we give the proof for the algebraic law (prob-3) (see p. 7).
Let

$$S =_{df} \{ (\langle P \sqcap_{p_1} (Q \sqcap_{p_2} R), \sigma\rangle, \langle (P \sqcap_x Q) \sqcap_y R), \sigma\rangle) \mid P, Q, R \text{ are programs } \wedge \sigma \in \Sigma \wedge p1, p2 \in (0, 1) \}$$

where,

(1) $x = p_1/(p_1 + p_2 - p_1 \times p_2)$,
$y = p_1 + p_2 - p_1 \times p_2$
(2) Σ denotes the set containing all the states.

Further, let $T =_{df} Id \cup S \cup S^{-1}$.
Now we need to prove that T is a bisimulation relation.

(1) From the transitions of $\sqcap_p$, we know that both $\langle P \sqcap_{p_1} (Q \sqcap_{p_2} R), \sigma\rangle$ and $\langle (P \sqcap_x Q) \sqcap_y R), \sigma\rangle$ cannot do transition of type $\xrightarrow{\tau}$ and $\xrightarrow{v}$. This indicates that we don't need to check the first item of bisimulation definition.
(2) Now we need to prove the item (2) of bisimulation relation for T.

   (a) If $\langle P, \sigma\rangle \overset{c}{\Longrightarrow}_{n'_1, p'_1} \langle P', \sigma'\rangle$,
   then $\langle P \sqcap_{p_1} (Q \sqcap_{p_2} R), \sigma\rangle \overset{c}{\Longrightarrow}_{n'_1, p_1 \times p'_1} \langle P', \sigma'\rangle$
   and

$$\langle (P \sqcap_x Q) \sqcap_y R, \sigma \rangle \overset{c}{\Longrightarrow}_{n'_1, u_1} \langle P', \sigma' \rangle$$

where, $u_1 = y \times x \times p'_1$.
From x and y, we know $u_1 = p_1 \times p'_1$.

(b) If $\langle Q, \sigma \rangle \overset{c}{\Longrightarrow}_{n'_2, p'_2} \langle Q', \sigma' \rangle$,
then $\langle P \sqcap_{p_1} (Q \sqcap_{p_2} R), \sigma \rangle \overset{c}{\Longrightarrow}_{n'_2, (1-p_1)\times p_2 \times p'_2} \langle P', \sigma' \rangle$
and

$$\langle (P \sqcap_x Q) \sqcap_y R, \sigma \rangle \overset{c}{\Longrightarrow}_{n'_2, u_2} \langle P', \sigma' \rangle$$

where, $u_2 = y \times (1 - x) \times p'_2$.
From x and y, we know that $u_2 = (1 - p_1) \times p_2 \times p'_2$

(c) If $\langle R, \sigma \rangle \overset{c}{\Longrightarrow}_{n'_3, p'_3} \langle R', \sigma' \rangle$,
then
$\langle P \sqcap_{p_1} (Q \sqcap_{p_2} R), \sigma \rangle \overset{c}{\Longrightarrow}_{n'_1, (1-p_1)\times (1-p_2) \times p'_3} \langle P', \sigma' \rangle$
and

$$\langle (P \sqcap_x Q) \sqcap_y R, \sigma \rangle \overset{c}{\Longrightarrow}_{n'_3, u_3} \langle P', \sigma' \rangle$$

where, $u_3 = (1 - y) \times p'_3$.
From x and y, we know $u_3 = (1 - p_1) \times (1 - p_2) \times p'_3$.
The above analysis leads to the satisfactory of the item (2) of bisimulation definition for the pair of configurations ($\langle P \sqcap_{p_1} (Q \sqcap_{p_2} R), \sigma \rangle$, $\langle (P \sqcap_x Q) \sqcap_y R), \sigma \rangle$).

(3) The proof of item (3) of bisimulation relation for T is similar to the above proof of item (2). □

References

1. Bowen JP (2000) Combining operational semantics, logic programming and literate programming in the specification and animation of the verilog hardware description language. In: Proceedings of IFM 2000: 2nd international conference on integrated formal methods. Lecture notes in computer science. vol 1945. Springer, Heidelberg, pp 277–296
2. Bowen JP, He J, Xu Q (2000) An animatable operational semantics of the verilog hardware description language. In: Proceedings of ICFEM 2000: 3rd IEEE international conference on formal engineering methods. IEEE Computer Society Press, pp 199–207
3. Clocksin WF, Mellish CS (2003) Programming in prolog, 5th edn. Springer, Heidelberg
4. de Bakker J, de Vink E (1996) Control flow semantics. MIT, Cambridge
5. den Hartog J (2002) Probabilistic extensions of semantic models. PhD thesis, Vrije University, The Netherlands
6. den Hartog J, de Vink E (1999) Mixing up nondeteminism and probability: a premliminary report. Electronic Notes in Theoretical Computer Science 22
7. den Hartog J, de Vink E (2002) Verifying probabilistic programs using a Hoare like logic. Int J Found Comput Sci 40(3):315–340
8. den Hartog J, de Vink E, de Bakker J (2001) Matrix semantics and full abstractness for action refinement and probabilistic choice. Electronic Notes in Theoretical Computer Science 40
9. He J (1994) Provably correct systems: modelling of communication languages and design of optimized compilers. International Series in Software Engineering. McGraw-Hill, NY
10. He J, Zhu H (2000) Formalising Verilog. In: Proceedings of ICECS 2000: IEEE international conference on electronics, circuits and systems. IEEE Computer Society Press, Nanjing, pp 412–415
11. He J, Seidel K, McIver A (1997) Probabilistic models for the guarded command language. Sci Comput Program 28(2–3):171–192
12. Hoare CAR, He J (1998) Unifying theories of programming. International series in computer science. Prentice Hall, Englewood Cliffs
13. Li J, Zhu H, Pu G, JH (2007) A formal model for compensable transactions. In: Proceedings of ICECCS 2007: 12th IEEE international conference on engineering of complex computer systems. IEEE Computer Society Press, Nanjing, pp 64–73
14. Li J, Zhu H, He J (2008) An observational model for transactional calculus of services orchestration. In: Proceedings of ICTAC 2008: 5th international colloquium on theoretical aspects of computing, Istanbul, Turkey, 1–3 September, 2008. Lecture notes in computer science, vol 5048. Springer, Heidelberg, pp 149–168
15. Luo C, Qin S, Qiu Z (2008) Verifying bpel-like programs with hoare logic. In: Proc. TASE 2008: 2nd IEEE international symposium on theoretical aspects of software engineering. IEEE Computer Society, Nanjing, China, pp 151–158
16. McIver A, Morgan C (2001) Partial correctness for probabilistic demonic programs. Theor Comput Sci 266(1-2):513–541
17. McIver A, Morgan C (2004) Abstraction, Refinement and Proof of Probability Systems. Monographs in Computer Science. Springer, Heidelberg
18. McIver A, Morgan C, Seidel K (1996) Probabilistic predicate transformers. ACM Trans Program Lang Syst 18(3):325–353
19. Milner R (1990) Communication and Concurrency. International Series in Computer Science. Prentice Hall, Englewood Cliffs
20. Milner R (1999) Communication and mobile system: π-calculus. Cambridge University Press, London
21. Ndukwu U, Sanders JW (2008) Reason about a distributed probabilistic system. Tech. Rep. 401, UNU/IIST, P.O. Box 3058, Macau SAR, China
22. Núñez M (2003) Algebraic theory of probabilistic processes. J Logic Algebraic Program 56:117–177
23. Núñez M, de Frutos-Escrig D (1996) Testing semantics for probabilistic LOTOS. In: Proceedings of FORTE'95: IFIP TC6 eighth international conference on formal description techniques, Montreal, Canada, October 1995. IFIP Conference Proceedings, vol 43. Chapman & Hall, London, pp 367–382
24. Núñez M, de Frutos-Escrig D, Díaz LFL (1995) Acceptance trees for probabilistic processes. In: Proceedings of CONCUR'95: 6th international conference on concurrency, Philadelphia, PA, USA, August, 1995. Lecture Notes in Computer Science, vol 962. Springer, Heidelberg
25. Park S, Pfenning F, Thrun S (2005) A probabilistic language based upon sampling functions. In: Proceedings of POPL 2005: 32nd ACM SIGPLAN-SIGACT symposium on principles of programming languages. ACM, New York, pp 171–182
26. Plotkin G (1981) A structural approach to operational semantics. Tech. Rep. 19, University of Aahus. Also published in J Logic Algebraic Program. 60–61:17–139, 2004
27. Zhu H (2005) Linking the semantics of a multithreaded discrete event simulation language. PhD thesis, South Bank University, London
28. Zhu H, He J (2000) A semantics of Verilog using duration calculus. In: Proceedings of international conference on software: theory and practice. pp 421–432
29. Zhu H, Qin S, He J, Bowen JP (2006) Integrating probability with time and shared-variable concurrency. In: Proceedings of SEW-30:

30th NASA/IEEE software engineering workshop. IEEE Computer Society, Nanjing, pp 179–189
30. Zhu H, He J, Li J (2007a) Unifying denotational semantics with operational semantics for web services. In: Proceedings of ICDCIT 2007: 4th international conference on distributed computing and internet technology, Bangalore, India, 17–20 December 2007. Lecture notes in computer science, vol 4882. Springer, Heidelberg, pp 225–239
31. Zhu H, He J, Li J, Bowen JP (2007b) Algebraic approach to linking the semantics of web services. In: Proceedings of SEFM 2007: 5th IEEE international conference on software engineering and formal methods. IEEE Computer Society Press, pp 315–326