Denotational Semantics for a Probabilistic Timed Shared-Variable Language

Huibiao Zhu[1], Jeff W. Sanders[2], Jifeng He[1], and Shengchao Qin[3]

[1] Shanghai Key Laboratory of Trustworthy Computing
Software Engineering Institute, East China Normal University
3663 Zhongshan Road (North), Shanghai, China, 200062
{hbzhu,jifeng}@sei.ecnu.edu.cn
[2] African Institute for Mathematical Sciences
6-8 Melrose Road, Muizenberg 7945, South Africa
jsanders@aims.ac.za
[3] School of Computing, University of Teesside
Middlesbrough TS1 3BA, UK
s.qin@tees.ac.uk

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. We have proposed a language, which integrates probability with time and shared-variable concurrency (called *PTSC* [19]). We also explored its operational semantics, where a set of algebraic laws has been investigated via bisimulation.

In this paper we explore the denotational semantics for our probabilistic language. In order to deal with the above three features and the nondeterminism, we introduce a tree structure, called P-tree, to model concurrent probabilistic programs. The denotational semantics of each statement is formalized in the structure of P-tree. Based on the achieved semantics, a set of algebraic laws is explored; i.e., especially those parallel expansion laws. These laws can be proved via our achieved denotational semantics.

1 Introduction

As probabilistic computations play an increasing role in solving various problems [16], various proposals on probabilistic languages have been reported [1–3, 5, 8–10, 13–15]. Complex software systems typically involve important features like real-time [12], probability and shared-variable concurrency. The shared-variable mechanism is typically used for communications among components running in parallel, such as the Java programming language and the Verilog hardware description language. It proves to be challenging to formalize it [6, 17, 18]. Therefore, system designers would expect a formal model that incorporates all these features to be available for them to use.

In [19], we have integrated a formal language model, which equips with probability, time and shared-variable concurrency. Our model is meant to facilitate the specification of complex software systems. The probability feature is reflected by the probabilistic nondeterministic choice, probabilistic guarded choice and the probabilistic scheduling

B. Wolff, M.-C. Gaudel, A. Feliachi (Eds.): UTP 2012, LNCS 7681, pp. 224–247, 2012.
© Springer-Verlag Berlin Heidelberg 2012

of actions from different concurrent components in a program. For this proposed language model, an operational semantics was formalized. On the top of the operational model, an abstract bisimulation relation was defined and several algebraic laws have been derived for program equivalence.

The *PTSC* model proposed in this paper has recently been used to specify a circuit in the register-transfer level [11]. 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 behavior conforms to the probabilistic specification.

As advocated in Hoare and He's Unifying Theories of Programming (*UTP*) [7], three different styles of mathematical representations are normally used: operational, denotational, and algebraic ones. Denotational semantics provides mathematical meanings to programs. Compared with operational semantics, it is more abstract. As *PTSC* integrates probability, time and shared-variable in one single model, it is challenging to formalize its denotational semantics. This paper studies the denotational semantics for *PTSC*. In order to deal with the above three features, together with the feature of nondeterminism, we introduce the concept of P-Tree in our model. The P-tree structure can be considered as the extension of traditional trace structure. Based on the achieved denotational semantics and the exploration of the equivalence of P-trees, a set of algebraic laws is investigated.

For exploring the unifying of the semantics for *PTSC*, we have explored the link between operational semantics and algebraic semantics [20]. Our approach in [20] is to derive operational semantics from algebraic semantics for our proposed probabilistic language. Moreover, we have also explored the animation of the link between the two semantics. Our approach can be considered as the soundness and completeness exploration of operational semantics from algebraic viewpoint, both theoretically and practically.

The remainder of this paper is organized as follows. Section 2 introduces our probabilistic language with time and shared-variable concurrency (i.e., *PTSC*). Section 3 explores the denotational semantic model. In order to deal with the above three features, together with the feature of nondeterminism, we introduce a tree structure in our semantic model, called P-tree. Section 4 investigates the denotational semantics for each statement of *PTSC*. Our P-tree structure is successfully applied in the exploration. For the aim of explore program equivalence, we provide an equivalence relation for P-trees. Based on the achieved semantics and the equivalence of P-tree structure, a set of algebraic laws is explored in section 5. Section 6 concludes the paper.

2 The Language *PTSC*

The *PTSC* language integrates probability and time with shared-variable concurrency. It has been designed to express the scheduling of threads, incorporating with concurrency and nondeterminism as well as probability and time. It is thus well suited to *discrete event simulation* where those features are present. The *PTSC* language has the following syntactical elements:

$$P ::= \mathbf{Skip} \mid x := e \mid \mathbf{Chaos} \mid \mathbf{if}\ b\ \mathbf{then}\ P\ \mathbf{else}\ P \mid \mathbf{while}\ b\ \mathbf{do}\ P$$
$$\mid @b\ P \mid \#n\ P \mid P\ ;\ P$$
$$\mid P \sqcap P \mid P \sqcap_p P \mid P \parallel P \mid P \parallel_p P$$

Note that:

(1) $x := e$ is the atomic assignment. **Skip** behaves the same as $x := x$. **Chaos** stands for the divergent process.
(2) Regarding $@b\ P$, when the Boolean condition b is satisfied, process P can have the chance to be scheduled. The program $@b\ P$ can wait the environment to fire the event if the Boolean condition b is not met currently. For $\#n\ P$, after n time units elapse, process P can be scheduled.
(3) Similar to a conventional programming language, $\mathbf{if}\ b\ \mathbf{then}\ P\ \mathbf{else}\ Q$ stands for the conditional, whereas $\mathbf{while}\ b\ \mathbf{do}\ P$ stands for the iteration.
(4) The mechanism for parallel composition is a shared-variable interleaving model with probability feature. For probabilistic parallel composition $P \parallel_p Q$, 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. On the other hand, $P \parallel Q$ stands for the general parallel composition.
(5) $\sqcap$ stands for the nondeterministic choice, whereas $\sqcap_p$ 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. Guarded choice is classified into five types:

(1) $[\!]_{i\in I}\{[p_i]\ \mathit{choice}_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij})\ P_{ij})\}$
(2) $[\!]_{i\in I}\{@b_i\ P_i\}$
(3) $[\!]\{\#1\ R\}$
(4) $[\!]_{i\in I}\{[p_i]\ \mathit{choice}_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij})\ P_{ij})\}$
$[\!][\!]_{k\in K}\{@b_k\ Q_k\}$
(5) $[\!]_{i\in I}\{@b_i\ P_i\}[\!]\{\#1\ R\}$

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

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

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 "$\mathit{choice}_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij})\ P_{ij})$" 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 firing of these guards is disjoint.

The third type is composed of one time delay component. Initially, it cannot do anything except letting 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 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 except letting 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 behavior, there is no type of guarded choice composing of the first and third type of guarded choice.

3 The Denotational Semantic Model for *PTSC*

In order to deal with shared-variable, probability and time in our model, we introduce the concept of snapshots for our denotational model.

A snapshot is expressed as a triple $(tag,\ p,\ \sigma)$, where:

(1) σ stands for the contributed state. These states are contributed by the program itself or the environment.
(2) tag can be 0, 0^-, 1 and $\surd$. If $tag = 0$, it indicates that the contribution of σ is due to the environment. If $tag = 1$, it indicates the contribution of σ is due to the process itself. On the other hand, if $tag = \surd$, it indicates that time advances one unit and the state σ is the same as the previous one. Flag 0^- is used to model the case that after the environment actions, the subsequent process will be assignment.
(3) For the second element, it is used to express the probability of the contributed state. If $tag = \surd$, it will be $\emptyset$, indicating that we do not need to consider the probability for time delay. On the other hand, if $tag = 0, 0^-, 1$, the second element p will be the probability of the contributing the state σ.

Based on the concept, we are now ready in defining the concept of P^--trees. The introduction of P^--tree can be used in formalizing the denotational semantics for ***PTSC***.

Definition 3.1 (P^--tree)

(1) st is P^--tree, where st stands for the execution state. Here, st can be div, $wait$ or ter.
(2) $\{\!|(tag, p_i, \sigma_i) : U_i \mid i \in I \wedge \Sigma_{i\in I}\, p_i = 1|\!\}$ is P^--tree if each element in every U_i is P^--tree.
(3) $\{\!|(\surd, \emptyset, \sigma) : U|\!\}$ is P^--tree if each element in U is P^--tree.

Note that $st \in U_i$(or U) **iff** $U_i = \{st\}$ (or $U = \{st\}$). □

Here, the notation "$\{\!|\ |\!\}$"stands for a bag. For a P^{-}-tree, st can be used to model the corresponding leaf. It stands for the execution state of the corresponding execution path. During the execution, a program can be in divergent state (i.e., div), waiting state (i.e., $wait$) or terminating state (i.e., ter).

For $\{\!|(tag, p_i, \sigma_i) : U_i \mid i \in I \wedge \Sigma_{i\in I}\, p_i = 1|\!\}$, the property "$\Sigma_{i\in I}\, p_i = 1$" indicates that the summation of all the probabilities for all the corresponding newly updated states is 1. Here, tag can be 0, 1. If $tag = 1$, it indicates that all the newly updated states with the corresponding probabilities are contributed by the program itself. On the other hand, if $tag = 0$, it indicates that all the newly updated states with the corresponding probabilities are contributed by the environment. Moreover, $tag = 0^{-}$ is used to model the case that after the environment's behavior, the subsequent behavior for the process is the assignment action.

For $\{\!|(\surd, \emptyset, \sigma) : U|\!\}$, the snapshot $(\surd, \emptyset, \sigma)$ here is used to model one unit delay behavior. The notation $\emptyset$ in snapshot $(\surd, \emptyset, \sigma)$ indicates that time delay does not concern probability, whereas $\surd$ stands for one unit time advancing and σ stands for the state after the time delay.

Further, for $(tag, \mu, \sigma) : U$ in a P^{-}-tree, if $st \in U$, it indicates that it cannot contain more than one leaf point. For example, $\{ter, wait\}$ is not allowed. Next we use some examples to illustrate our P^{-}-tree.

Example 3.2. Let $Q_1 =_{df} x := 1\ ;\ \#1\ ;\ x := x + 1 \sqcap_{0.4} x := x + 2$. We find that below is one P^{-}-tree of program Q_1.

$$\{\!|(1, 1, \sigma_1) : \{T_0\}|\!\}$$

where, $T_0 = \{\!|(\surd, \emptyset, \sigma_1) : \{T_1\}|\!\}$ and
$T_1 = \{\!|(1, 0.4, \sigma_1) : \{T_{1,1}\},\ (1, 0.6, \sigma_1) : \{T_{1,2}\}|\!\}$ and
$T_{1,1} = \{\!|(1, 1, \sigma_2) : \{ter\}|\!\},\ T_{1,2} = \{\!|(1, 1, \sigma_3) : \{ter\}|\!\}$
Here, $\sigma_1 = \{x \mapsto 1\}$, $\sigma_2 = \{x \mapsto 2\}$ and $\sigma_3 = \{x \mapsto 3\}$.

For snapshot $(1, 1, \sigma_1)$, it is used to model the contribution of $x := 1$. Snapshot $(\surd, \emptyset, \sigma_1)$ is used to model $\#1$. Tree T_1 models the behavior of $x := x + 1 \sqcap_{0.4} x := x := x + 2$, whereas tree $T_{1,1}$ and $T_{1,2}$ model the behaviour of $x := x+1$ and $x := x+2$ respectively.

Furthermore, let $Q_2 =_{df} x := 1\ ;\ \#1\ ;\ x := x+1 \sqcap x := x+2$, below is the P^{-}-tree of program Q_2.

$$\{\!|(1, 1, \sigma_1) : \{T_2\}|\!\}$$

where, $T_2 = \{\!|(\surd, \emptyset, \sigma_1) : \{T_{3,r} | 0 \le r \le 1\}|\!\}$ and
$T_{3,r} = \{\!|(1, r, \sigma_1) : \{T_{1,1}\},\ (1, 1-r, \sigma_1) : \{T_{1,2}\}|\!\}$
Here, $T_{1,1}$ and $T_{1,2}$ have been defined in the above for modelling $x := x + 1$ and $x := x+2$ respectively. $T_{3,r}$ is used to model the probabilistic choice $x := x+1 \sqcap_r x := x + 2$. Hence, $\{T_{3,r} | 0 \le r \le 1\}$ can model the behaviour of nondeterministic choice $x := x + 1 \sqcap x := x + 2$. □

Next we use an example to illustrate how a process's behaviour and its environment's behaviour cooperate.

Example 3.3. Let $Q_1 = x := 1 \sqcap_{0.4} x := 2$, $Q_2 = y := 1 \sqcap_{0.3} y := 2$, $Q = Q_1 \parallel Q_2$. Consider the P^--trees for process Q_1, Q_2 and Q respectively.

We consider the case that, for process Q, the assignment in Q_1 is scheduled first. In this case, below is one P^--tree for Q_1 at the initial state (tag, μ, σ_0).

$$\begin{aligned}(tag, \mu, \sigma_0) : \{ \{\!| \, & (1, 0.4, \sigma_0) : \{ \{\!|(1, 1, \sigma_1) : \{ter\}|\!\} \}, \\ & (1, 0.6, \sigma_0) : \{ \{\!|(1, 1, \sigma_2) : \{ter\}|\!\} \} \\ & |\!\} \}\end{aligned}$$

where, $\sigma_0 = \{x \mapsto 0,\ y \mapsto 0\}$, $\sigma_1 = \{x \mapsto 1,\ y \mapsto 0\}$, $\sigma_2 = \{x \mapsto 2,\ y \mapsto 0\}$.

As Q_1 is scheduled first, similarly, below is the corresponding P^--tree for Q_2 at the initial state (tag, μ, σ_0).

$$\begin{aligned}(tag, \mu, \sigma_0) : \{ \{\!| \, & (0^-, 0.4, \sigma_0) : \{ \{\!|(0^-, 1, \sigma_1) : \{T_2\}|\!\} \}, \\ & (0^-, 0.6, \sigma_0) : \{ \{\!|(0^-, 1, \sigma_1) : \{T_2'\}|\!\} \} \\ & |\!\} \}\end{aligned}$$

where, $T_2 = \{\!| \, (1, 0.3, \sigma_1) : \{\{\!|(1, 1, \sigma_1') : \{ter\}|\!\}\}, (1, 0.7, \sigma_1) : \{\{\!|(1, 1, \sigma_1'') : \{ter\}|\!\}\} \, |\!\}$

$T_2' = \{\!| \, (1, 0.3, \sigma_2) : \{\{\!|(1, 1, \sigma_2') : \{ter\}|\!\}\}, (1, 0.7, \sigma_2) : \{\{\!|(1, 1, \sigma_2'') : \{ter\}|\!\}\} \, |\!\}$

where, $\sigma_1' = \{x \mapsto 1,\ y \mapsto 1\}$, $\sigma_1'' = \{x \mapsto 1,\ y \mapsto 2\}$

$\sigma_2' = \{x \mapsto 2,\ y \mapsto 1\}$, $\sigma_2'' = \{x \mapsto 2,\ y \mapsto 2\}$

Hence, below is the corresponding P^--tree for process Q (i.e., parallel process $Q_1 \parallel Q_2$), which is the merge of the P^--tree of process Q_1 and the corresponding P^--tree of Q_2.

$$\begin{aligned}(tag, \mu, \sigma_0) : \{ \{\!| \, & (1, 0.4, \sigma_0) : \{ \{\!|(1, 1, \sigma_1) : \{T_2\}|\!\} \}, \\ & (1, 0.6, \sigma_0) : \{ \{\!|(1, 1, \sigma_2) : \{T_2'\}|\!\} \} \\ & |\!\} \}\end{aligned}$$

Similarly, we can also analyze the P^--tree of process Q for the case that the assignment in Q_2 is scheduled first. □

Definition 3.3 (P-tree)

$(tag, \mu, \sigma) : U$ is P-tree if each element in U is P^--tree. □

P-tree is composed of an initial snapshot and a set of P^--trees. The P-tree $(tag, \mu, \sigma) : U$ indicates that each P^--tree in U is initially at the state shown in snapshot (tag, μ, σ). Next we define the sequential composition for P^--trees and P-trees.

Definition 3.4 (Sequential Composition of P^--trees)

(1) $$((tag_0, \mu_0, \sigma_0) : \{st\}) \, ; \, \{(tag, \mu, \sigma) : V \mid (tag, \mu, \sigma) \in \Sigma\}$$

$$=_{df} \begin{cases} (tag_0, \mu_0, \sigma_0) : V, & \text{if } st = ter \\ (tag_0, \mu_0, \sigma_0) : \{st\}, & \text{if } st = wait \vee st = div \end{cases}$$

If $U \neq \{st\}$, then

$$((tag_0, \mu_0, \sigma_0) : U)\ ;\ \{\, (tag, \mu, \sigma) : V \mid (tag, \mu, \sigma) \in \Sigma \,\}$$
$$=_{df}\ (tag_0, \mu_0, \sigma_0) : \{\, T\ ;\ \{\, (tag, \mu, \sigma) : V \mid (tag, \mu, \sigma) \in \Sigma \,\} \mid T \in U \,\}$$

(2) $$\{\!|(tag_i, \mu_i, \sigma_i) : U_i \mid i \in I|\!\}\ ;\ \{(tag, \mu, \sigma) : V \mid (tag, \mu, \sigma) \in \Sigma\}$$
$$=_{df}\ \{\!|\ (tag_i, \mu_i, \sigma_i) : (U_i\ ;\ \{(tag, \mu, \sigma) : V \mid (tag, \mu, \sigma) \in \Sigma\}) \mid i \in I\ |\!\}$$

(3) $$\{(tag, \mu, \sigma) : U \mid (tag, \mu, \sigma) \in \Sigma\}\ ;\ \{(tag_1, \mu_1, \sigma_1) : V \mid (tag_1, \mu_1, \sigma_1) \in \Sigma\}$$
$$=_{df}\ \{\, ((tag, \mu, \sigma) : U)\ ;\ \{(tag_1, \mu_1, \sigma_1) : V \mid (tag_1, \mu_1, \sigma_1) \in \Sigma\})$$
$$\mid (tag, \mu, \sigma) \in \Sigma \,\}$$

□

Here, the notation Σ stands for the set containing all the snapshots. The above definition deals with the sequential composition of P-trees or P^--trees. The first one considers the sequential composition of P-tree and a set of P-trees (at any different initial state). Its definition can be divided into two cases according to the case that the first P-tree is a leaf point or not. The leaf point can be the terminating state, waiting state or divergence state.

The second one considers the sequential composition of a P^--tree and a set of P-trees (at any different initial state). The third one considers the sequential composition of a set of P-trees with another set of P-trees. Both of the two sets of P-trees can be at any different initial state.

Example 3.5. Let $P =_{df} x := 0 \sqcap x := 1$ and $Q =_{df} y := 0 \sqcap_{0.5} y := 1$. Now we want to calculate $Prob(P; Q, x = y)$ and $Prob(Q; P, x = y)$. Assume the initial states of x and y are -1 respectively. Here the notation $Prob(W, c)$ stands for the probability that the final state of program W satisfies condition c.

First, we consider the P-tree for program $P; Q$, shown below.

$$(tag, \mu, \sigma) : \{\ \{\!|\ (1, r, \sigma) : \{\{\!|(1, 1, \sigma_0) : \{T_1\}|\!\}\},$$
$$(1, 1 - r, \sigma) : \{\{\!|(1, 1, \sigma_1) : \{T_2\}|\!\}\}$$
$$|\!\}\ \mid\ 0 \leq r \leq 1\ \}$$

where, $T_1 = \{\!|\ (1, 0.5, \sigma_0) : \{\{\!|(1, 1, \sigma_0') : \{ter\}|\!\}\},$
$(1, 0.5, \sigma_0) : \{\{\!|(1, 1, \sigma_0'') : \{ter\}|\!\}\}\ |\!\}$

$T_2 = \{\!|\ (1, 0.5, \sigma_1) : \{\{\!|(1, 1, \sigma_1') : \{ter\}|\!\}\},$
$(1, 0.5, \sigma_1) : \{\{\!|(1, 1, \sigma_1'') : \{ter\}|\!\}\}\ |\!\}$

$\sigma = \{x \mapsto -1,\ y \mapsto -1\},$
$\sigma_0 = \{x \mapsto 0,\ y \mapsto -1\},\ \ \sigma_0' = \{x \mapsto 0,\ y \mapsto 0\},\ \ \sigma_0'' = \{x \mapsto 0,\ y \mapsto 1\},$
$\sigma_1 = \{x \mapsto 1,\ y \mapsto -1\},\ \ \sigma_1' = \{x \mapsto 1,\ y \mapsto 0\},\ \ \sigma_1'' = \{x \mapsto 1,\ y \mapsto 1\},$

Based on the P-tree for program $P; Q$, we can have:

$$Prob(P\ ;\ Q,\ x = y)$$
$$= min\{r \times 1 \times 0.5 \times 1 + (1 - r) \times 1 \times 0.5 \times 1 \mid 0 \leq r \leq 1\}$$
$$= min\{0.5 \mid 0 \leq r \leq 1\} = 0.5$$

Next we consider the P-tree for program $Q; P$, shown below.

$$(tag, \mu, \sigma) : \{\ \{\!|\ (1, 0.5, \sigma) : \{\ \{\!|\ (1, 1, \sigma_2) : \{T_{3,r} \mid 0 \le r \le 1\}\ |\!\}\ \},\ (1, 0.5, \sigma) : \{\ \{\!|\ (1, 1, \sigma_3) : \{T_{4,r} \mid 0 \le r \le 1\}\ |\!\}\ \}\ |\!\}\ \}$$

where, $T_{3,r} = \{\!|\ (1, r, \sigma_2) : \{\{\!|(1, 1, \sigma_2') : \{ter\}|\!\}\},\ (1, 1-r, \sigma_2) : \{\{\!|(1, 1, \sigma_2'') : \{ter\}|\!\}\}\ |\!\}$

$T_{4,r} = \{\!|\ (1, r, \sigma_3) : \{\{\!|(1, 1, \sigma_3') : \{ter\}|\!\}\},\ (1, 1-r, \sigma_3) : \{\{\!|(1, 1, \sigma_3'') : \{ter\}|\!\}\}\ |\!\}$

$\sigma_2 = \{x \mapsto -1,\ y \mapsto 0\},\ \ \sigma_2' = \{x \mapsto 0,\ y \mapsto 0\},\ \ \sigma_2'' = \{x \mapsto 1,\ y \mapsto 0\},$
$\sigma_3 = \{x \mapsto -1,\ y \mapsto 1\},\ \ \sigma_3' = \{x \mapsto 0,\ y \mapsto 1\},\ \ \sigma_3'' = \{x \mapsto 1,\ y \mapsto 1\},$

Based on the P-tree for program $Q; P$, we can have:

$$\begin{aligned} & Prob(Q\ ;\ P,\ x = y) \\ = \ & 0.5 \times 1 \times min\{r \times 1 \mid 0 \le r \le 1\} \ +\ 0.5 \times 1 \times min\{r \times 1 \mid 0 \le r \le 1\} \\ = \ & 0 \end{aligned}$$

□

In order to support to later formalization of each statement, we introduce the concept of $idle0(tag, b)$ P^--tree (tag can be 0 or 0^-, and b is a Boolean condition).

Definition 3.6 ($idle0(tag, b)$ P^--tree)

(1) st is $idle0(tag, b)$, where $st = wait$ or ter.

(2) $\{\!|(tag, p_i, \sigma_i) : S_i \mid i \in I|\!\}$ is $idle0(tag, b)$, if for any $i \in I$, $b(\sigma_i)$ and $\forall X \in S_i \bullet$ X is $idle0(tag, b)$.

where, $tag = 0$ or 0^-. □

For the snapshots in an $idle0(0, b)$ P^--tree, the flag parts are all 0. This indicates that all the newly added states (with probabilities) are contributed by the environment and Boolean condition b is satisfied for all these newly added states. Further, there are no $(\surd, \emptyset, \sigma)$ snapshots in an $idle0(0, b)$ P^--tree. This means that all actions reflected in an $idle0(0, b)$ P^--tree is instantaneous. Similarly, the concept of $idle0(0^-, b)$ P^--tree is defined in the above definition.

Now we can also define the concept of $idle(tag, b)$ P^--tree ($tag = 0$ or 0^-, and b is a Boolean condition). An $idle(tag, b)$ tree not only can contain instantaneous action, but also can contain time delay snapshots.

(1') st is $idle(tag, b)$, where $st = wait$ or ter.

(2') $\{\!|(tag, p_i, \sigma_i) : S_i \mid i \in I|\!\}$ is $idle(tag, b)$, if for any $i \in I$, $b_i(\sigma)$ and $\forall X \in S_i \bullet$ X is $idle(tag, b)$.

(3') $\{\!|(\surd, \emptyset, \sigma) : S|\!\}$ is $idle(tag, b)$, if $b(\sigma)$ and $\forall X \in S \bullet X$ is $idle(tag, b)$.

where, $tag = 0$ or 0^-. □

Based on the definitions of P-trees and P^--trees, the denotational semantics for process P can be formalized in the form below.

$$\{\ (tag, \mu, \sigma) : U \mid (tag, \mu, \sigma) \in \Sigma\ \}$$

Here, U contains a set of P^--trees and Σ stands for the set containing all snapshots.

4 Denotational Semantics for *PTSC* Statements

Based on the introduction of P-tree (and P^--tree), this section is to study the denotational semantics for *PTSC*, including sequential constructs, timed constructs, probabilistic choice, nondeterminism, guarded choice and parallel composition.

4.1 Sequential Constructs

Assignment. Assignment is considered as an atomic action. Before the assignment is scheduled, the environment may also have a chance to be scheduled to perform actions.

Firstly, we define function $append(T, x, e)$, which appends the assignment of variable x with e to P^--tree T.

$$\begin{aligned} & append(\{\!|(tag_i, \mu_i, \sigma_i) : U_i \mid i \in I|\!\}, x, e) \\ =_{df}\ & \{\!|(tag_i, \mu_i, \sigma_i) : \{\ attach(\sigma_i, U_i, x, e)\ \} \mid i \in I|\!\} \end{aligned}$$

where,

$$\begin{aligned} & attach(\sigma, U, x, e) \\ =_{df}\ & (\ (1, 1, \sigma[e/x]) : \{ter\}) \lhd U = \{ter\} \rhd \\ & \qquad (\ U \lhd U = \{wait\} \vee U = \{div\} \rhd (\forall T \in U \bullet append(T, x, e))\ ) \end{aligned}$$

Here, $attach(\sigma, U, x, e)$ means adding a new snapshot (the update of variable x with e) to the terminating leaf of all P^--trees in U. Its definition can be defined recursively. When a leaf is encountered, if it is a terminating leaf, the adding will be performed. On the other hand, when divergence leaf or waiting leaf is encountered, the adding will not be performed.

Similarly,

$$\begin{aligned} & append((tag, \mu, \sigma) : U, x, e) \\ =_{df}\ & (tag, \mu, \sigma) : \{\ attach(\sigma, T, x, e) \mid \forall T \in U\ \} \end{aligned}$$

Then, the semantics of $x := e$ can be described as the tree behaviour shown below. Formula $idle(0^-, true)$ here indicates that, before the assignment is scheduled, the environment can have chances to perform instantaneous actions. The symbol 0^- indicates that, after the environment's instantaneous actions, the process itself will perform assignment action.

$$\begin{aligned} & [\![x := e]\!] \\ =_{df}\ & \{\ append((tag, \mu, \sigma) : U,\ x,\ e) \mid (tag, \mu, \sigma) \in \Sigma\ \wedge \\ & \qquad \forall X \in U \bullet X \text{ is } idle0(0^-, true)\ \} \end{aligned}$$

Chaos. For **Chaos** statement, its denotational semantics can be defined as below:

$$[\![\mathbf{Chaos}]\!] =_{df} \{\ (tag, \mu, \sigma) : \{div\} \mid (tag, \mu, \sigma) \in \Sigma\ \}$$

Sequential Composition. $(P\ ;\ Q)$ behaves like P before P terminates, and then behaves like Q afterwards.

$$[\![P\ ;\ Q]\!] =_{df} [\![P]\!]\ ;\ [\![Q]\!]$$

Conditional. The definition of conditional can be defined as below.

$$\begin{aligned}&[\![\mathbf{if}\ b\ \mathbf{then}\ P\ \mathbf{else}\ Q]\!]\\ =_{df}\ &\{\ (tag,\mu,\sigma) : U \lhd b(\sigma) \rhd (tag,\mu,\sigma : V)\\ &\qquad\qquad \mid (tag,\mu,\sigma) \in \Sigma \wedge (tag,\mu,\sigma) : U \in [\![P]\!]\\ &\qquad\qquad\qquad\qquad \wedge (tag,\mu,\sigma) : V \in [\![Q]\!]\ \}\end{aligned}$$

Iteration. In order to define the semantics of iteration, we define the partial order below.

$$\begin{aligned}&\{\!|(tag_i,\mu_i,\sigma_i) : U_i \mid i \in I|\!\} \sqsupseteq \{\!|(tag_j,\mu_j,\sigma_j) : V_j\!\} \mid j \in J|\!\}\\ =_{df}\ &\forall i \in I \bullet \exists j \in J \bullet (tag_i,\mu_i,\sigma_i) : U_i \in \{\!|(tag_i,\mu_i,\sigma_i) : U_i|\!\}\ \wedge\\ &\qquad (tag_j,\mu_j,\sigma_j) : V_j \in \{\!|(tag_j,\mu_j,\sigma_j) : V_j|\!\}\ \wedge\\ &\qquad (tag_i,\mu_i,\sigma_i) = (tag_j,\mu_j,\sigma_j)\ \wedge\\ &\qquad \forall X \in U_i \bullet \exists Y \in V_j \bullet (X \sqsupseteq Y \wedge X \approx Y)\end{aligned}$$

where, the equivalence relation $\approx$ on trees will be defined in section 5.

Then we can use this order to give the order $\sqsupseteq$ for programs. Based on this, we can give the definition for iteration. The iteration construct is defined in the same way as its counterpart in conventional programming language.

$$[\![\mathbf{while}\ b\ \mathbf{do}\ P]\!] =_{df} \mu X \bullet [\![\mathbf{if}\ b\ \mathbf{then}\ (P; X)\ \mathbf{else}\ I\!I]\!]$$

where:

(1) $[\![I\!I]\!] =_{df} \{\ (tag,\mu,\sigma) : \{ter\} \mid (tag,\mu,\sigma) \in \Sigma\ \}$

(2) The notation $\mu X \bullet F(X)$ denotes the weakest fixed point of the monotonic function F.

4.2 Timed Constructs

Time Delay. Firstly we consider the time delay statement. The definition is based on two $tick$ and $tick'$ functions.

$$\begin{aligned}&tick'((tag,\mu,\sigma) : \{st\})\\ =_{df}\ &\begin{cases}(tag,\mu,\sigma) : \{st\} & \text{if } st = wait \text{ or } div\\ (tag,\mu,\sigma) : \{(\surd,\emptyset,\sigma) : \{ter\}\} & \text{if } st = ter\end{cases}\end{aligned}$$

and

$$tick'((tag,\mu,\sigma) : U) =_{df} (tag,\mu,\sigma) : \{tick(X) \mid X \in U\}$$

Then, a further function $tick$ can be defined as:

$$tick(\{\!|(tag_i,\mu_i,\sigma_i) : U_i \mid i \in I|\!\})$$

$=_{df} \{\!|\ tick'((tag_i, \mu_i, \sigma_i) : U_i) \mid i \in I\ |\!\}$

Based on the above definitions, we can have the semantics of $\#1$ via the tree behaviour.

$$[\![\#1]\!] =_{df} \{\ (tag, \mu, \sigma) : \{tick(X) \mid X \text{ is } idle0(0, true)\} \mid (tag, \mu, \sigma) \in \Sigma\ \}$$

$$[\![\#n]\!] =_{df} [\![\#1]\!]\ ;\ [\![\#(n-1)]\!]$$

For $\#1$, before time advancing, the environment may perform assignments with certain probabilities at the current time point. This behaviour can be expressed as an $idle0(0, true)$ tree. For the behaviour of $\#n$, it can be defined recursively.

Event Guard. Now we are ready to consider the event triggering behaviour. Firstly we define the concept of $trig(b, f)$ ($f = 0$ or 1) as below.

$\{\!|\ (0, p_i, \sigma_i) : U_i \mid i \in I\ |\!\}$ is $trig(b, f)$,

if it satisfies the following conditions

$$\begin{pmatrix} \exists i \in I \bullet b(\sigma_i) \wedge \\ \forall i \in I \bullet b(\sigma_i) \Rightarrow U_i = \{ter\} \wedge \\ \forall i \in I \bullet \neg b(\sigma_i) \Rightarrow \forall X \in U_i \bullet X \text{ is } leaffired(b, f) \end{pmatrix}$$

For the concept of $leaffired(b, f)$ P^--tree ($f = 0$ or 1), it can be defined as below.

(1) $\{\!|\ (0, p_i, \sigma_i) : U_i \mid i \in I\ |\!\}$ is $leaffired(b, f)$ ($f = 0$ or 1) if it satisfies the following condition

$$\forall i \in I \bullet \begin{pmatrix} b(\sigma_i) \Rightarrow U_i = \{ter\} \wedge \\ \neg b(\sigma_i) \Rightarrow \forall X \in U_i \bullet X \text{ is } leaffired(b, f) \end{pmatrix}$$

(2) $\{\!|\ (\surd, \emptyset, \sigma) : U\ |\!\}$ is $leaffired(b, 1)$ if it satisfies the following condition

$$\begin{pmatrix} b(\sigma) \Rightarrow U = \{ter\} \wedge \\ \neg b(\sigma) \Rightarrow \forall X \in U \bullet X \text{ is } (leaffired(b, 0) \vee leaffired(b, 1)) \end{pmatrix}$$

For $@b$, there are two firing cases. The first case is that event $@b$ is fired at the initial state, which is denoted as formula $Immefired(b)$. The second case is that it waits for the environment to fire it. This case can be described using two formulae $Await(b, 1)$ and $Trig(b, 1)$. Formula $Await(b, 1)$ indicates that all the environment behaviour cannot fire $@b$. $Trig(b, 1)$ indicates that $@b$ is fired finally. Then the semantics of $@\,b$ can be defined as:

$$[\![@\,b]\!] =_{df} Immefired(b)\ \cup\ (Await(b, 1)\ ;\ Trig(b, 1))$$

where,

$$\begin{aligned} & Immefired(b) \\ =_{df}\ & \{\ (tag, \mu, \sigma) : \{ter\} \mid (tag, \mu, \sigma) \in \Sigma \wedge b(\sigma)\ \} \end{aligned}$$

$$\begin{aligned} & Await(b, f) \\ =_{df}\ & \{\ (tag, \mu, \sigma) : U \mid (tag, \mu, \sigma) \in \Sigma \wedge \neg b(\sigma) \\ & \qquad \wedge \forall X \in U \bullet (f = 0 \wedge X \text{ is } idle0(0, \neg b) \vee f = 1 \wedge X \text{ is } idle(0, \neg b))\ \} \end{aligned}$$

$$
\begin{aligned}
&Trig(b,f)\\
=_{df}\ &\{\ (tag,\mu,\sigma):U \mid (tag,\mu,\sigma)\in\Sigma\wedge\neg b(\sigma)\wedge\forall X\in U\bullet X \text{ is } trig(b,f)\ \}
\end{aligned}
$$

Here, $f=0,\ 1$.

4.3 Probabilistic Nondeterminism

Firstly we consider the definition for probabilistic nondeterminism $P\sqcap_r Q$.

$$
\begin{aligned}
&[\![P\sqcap_r Q]\!]\\
=_{df}\ &\{\ (tag,\mu,\sigma):U \mid (tag,\mu,\sigma)\in\Sigma\ \wedge\ \forall X\in U\ \bullet\ X \text{ is } idle0(0^-,true)\ \}\ ;\\
&\{\ (tag,\mu,\sigma):\{T(r)\} \mid (tag,\mu,\sigma)\in\Sigma\ \}
\end{aligned}
$$

where,

$$T(r)=_{df}\ \{\!|\ (1,r,\sigma):U,\ (1,1-r,\sigma):V\ |\!\}$$

and, $(tag,\mu,\sigma):U\in[\![P]\!]$, $(tag,\mu,\sigma):V\in[\![Q]\!]$

Moreover, we can give the definition for $P\sqcap Q$.

$$
\begin{aligned}
&[\![P\sqcap Q]\!]\\
=_{df}\ &\{\ (tag,\mu,\sigma):U \mid (tag,\mu,\sigma)\in\Sigma\ \wedge\ \forall X\in U\ \bullet\ X \text{ is } idle0(0^-,true)\ \}\ ;\\
&\{\ (tag,\mu,\sigma):\{T(r)\mid 0\le r\le 1\} \mid (tag,\mu,\sigma)\in\Sigma\ \}
\end{aligned}
$$

4.4 Guarded Choice

As mentioned earlier, there are five types of guarded choice. Now we give the denotational semantics for these five types of guarded choice.

Assignment Guarded Choice. Firstly, we consider the assignment guarded choice, which is composed of a set of assignment guarded components.

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

Then,

$$
\begin{aligned}
&[\![P]\!]\\
=_{df}\ &\{((tag,\mu,\sigma):idle0(0^-,true))\ ;\ ImmeAssi(\{P\}) \mid (tag,\mu,\sigma)\in\Sigma\}
\end{aligned}
$$

where,

$$
\begin{aligned}
&ImmeAssi(S)=_{df}\ \{(tag,\mu,\sigma):\{T(P)\mid P\in S\} \mid (tag,\mu,\sigma)\in\Sigma\}\\
&T(P)=_{df}\ \{\!|\ \forall j\in J_i\bullet \text{if } b_{ij} \text{ then } (1,p_i,\sigma_{ij}):V_{ij}\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mid i\in I\wedge(1,p_i,\sigma_{ij}):V_{ij}\in[\![P_{ij}]\!]\ |\!\}
\end{aligned}
$$

The formula $idle0(0^-,true)$ here indicates that, before any assignment is scheduled, the environment will have chances to perform the instantaneous actions. The execution

of assignment guarded components is expressed by formulae $ImmeAssi(\{P\})$ and $T(P)$.

Event Guarded Choice. Now we consider the denotational semantics for the second type of guarded choice, which is composed of a set of event guarded components.

Let $P = [\!]_{i\in I}\{@b_i\, P_i\}$.

Then,

$$\begin{aligned} & [\![P]\!] \\ =_{df}\ & Immefired([\!]_{i\in I}\{@b_i\, P_i\}) \\ & \cup\ (Await(b,1)\ ;\ Trig(b,1)\ ;\ Imme([\!]_{i\in I}\{@b_i\, P_i\})) \end{aligned}$$

where,

$$\begin{aligned} & b = \vee_{i\in I}\ b_i \\ & Immefired([\!]_{i\in I}\{@b_i\, P_i\}) \\ =_{df}\ & \{(tag,\mu,\sigma) : U \mid (tag,\mu,\sigma) \in \Sigma \wedge \\ & \qquad\qquad \exists i \in I \bullet (b_i(\sigma) \wedge (tag,\mu,\sigma) : U \in [\![P_i]\!])\} \end{aligned}$$

Time Delay Guarded Choice. For the third type of guarded choice, it is composed of only one time delay component.

$$[\![[\!]\{\#1\ P\}]\!] =_{df} [\![\#1]\!]\ ;\ [\![P]\!]$$

Guarded Choice Composing of Assignment Guarded Choice and Event Guarded Choice. For the fourth type of guarded choice, it is composed of a set of assignment guarded components and a set of event guarded components.

Let $P = [\!]_{i\in I}\{[p_i]\ \mathit{choice}_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij})\, P_{ij})\}[\!][\!]_{k\in K}\{@c_k\, Q_k\}$ and
$P1 = [\!]_{i\in I}\{[p_i]\ \mathit{choice}_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij})\, P_{ij})\}$.

Then,

$$\begin{aligned} & [\![P]\!] \\ =_{df}\ & Immefied([\!]_{k\in K}\{@c_k\, Q_k\}) \\ & \cup\ (\ Await(c,0)\ ;\ ImmAssi(P1)\ ) \\ & \cup\ (\ Await(c,0)\ ;\ Trig(c,0)\ ;\ Imme([\!]_{k\in K}\{@c_k\, Q_k\})\ ) \end{aligned}$$

where, $c = \vee_{k\in K}\ c_k$

As the fourth type of guarded choice contains assignment guarded components, the waiting period of waiting the event guards to be fired is at the current time point. The formulae $Await(c,0)$ and $Trig(c,0)$ are applied.

Guarded Choice Composing of Event Guarded Choice and Time Delay Guarded Choice. Now we consider the denotational semantics for the fifth type of guarded choice, which is composed of event guarded choice and time delay guarded choice.

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

Then,

$$\begin{aligned}&[\![P]\!]\\ =_{df}\quad &Immefired([\!]_{i\in I}\{@b_i\, P_i\})\\ &\cup\ (\ Await(b,0)\ ;\ Trig(b,0)\ ;\ Imme([\!]_{i\in I}\{@b_i\, P_i\})\ )\\ &\quad (\ Await(b,0)\ ;\ \mathbf{phase1}\ ;\ [\![R]\!]\ )\end{aligned}$$

where, $b\ =\ \vee_{i\in I}\, b_i$

$\mathbf{phase1} =_{df} \{\ (tag,\mu,\sigma) : \{\{\!|(\surd,\emptyset,\sigma) : \{ter\}|\!\}\} \mid (tag,\mu,\sigma)\in\Sigma\ \}$

For the fifth type of guarded choice, the event guards can be fired immediately, or waiting for the environment to fire them. The waiting period should be at the current time point. If at the current time point, all the events are not fired, then time will advance one unit. This can be expressed by formula $\mathbf{phase1}$.

Or Construct for Guarded Choice. In order to support the expansion laws of general parallel composition, we introduce the concept of the **or** Construct for Guarded Choice. Below are the two cases for the **or** Construct.

(1) Let $P\ =\ [\!]_{i\in I}\{[p_i]\ \mathit{choice}_{j\in J_i}(b_{ij}\&(x_{ij}:=e_{ij})\, P_{ij})\}$ and
$Q\ =\ [\!]_{k\in K}\{[q_k]\ \mathit{choice}_{l\in L_k}(c_{kl}\&(y_{kl}:=f_{kl})\, Q_{kl})\}$.

Then,

$$\begin{aligned}&[\![P\ \mathbf{or}\ Q]\!]\\ =_{df}\ &\{((tag,\mu,\sigma) : idle0(0^-,true))\ ;\ ImmeAssi(\{P,Q\}) \mid (tag,\mu,\sigma)\in\Sigma\}\end{aligned}$$

(2) Let $P =\ [\!]_{i\in I}\{[p_i]\ \mathit{choice}_{j\in J_i}(b_{ij}\&(x_{ij}:=e_{ij})\, P_{ij})\}$
$\quad [\!][\!]_{m\in M}\{@c_m\ R_m\}$
$P1 =\ [\!]_{i\in I}\{[p_i]\ \mathit{choice}_{j\in J_i}(b_{ij}\&(x_{ij}:=e_{ij})\, P_{ij})\}$
$Q =\ [\!]_{k\in K}\{[q_k]\ \mathit{choice}_{l\in L_k}(b_{kl}\&(y_{kl}:=f_{kl})\, Q_{kl})\}$
$\quad [\!][\!]_{m\in M}\{@c_m\ R_m\}$
$Q1 =\ [\!]_{k\in K}\{[q_k]\ \mathit{choice}_{l\in L_k}(b_{kl}\&(y_{kl}:=f_{kl})\, Q_{kl})\}$

Then,

$$\begin{aligned}&[\![P\ \mathbf{or}\ Q]\!]\\ =_{df}\quad &Immefired([\!]_{m\in M}\{@c_m\, R_m\})\\ &\cup\ (\ Await(c,0)\ ;\ ImmAssi(\{P1,Q1\})\ )\\ &\cup\ (\ Await(c,0)\ ;\ Trig(c,0)\ ;\ Imme([\!]_{m\in M}\{@c_m\, R_m\})\ )\end{aligned}$$

where, $c\ =\ \vee_{m\in M}\, c_m$

4.5 Probabilistic Parallel Composition

Now we consider the probabilistic parallel composition. In order to deal with the definition for probability parallel composition, we first define the probabilistic merge operator $\otimes_r$. This can be done by the case analysis.

Firstly we consider the case that P can perform probabilistic atomic actions initially. If Q can also perform probabilistic atomic actions initially, $P \otimes_r Q$ can also perform probabilistic atomic actions initially from both P and Q. And the probability of these assignments needs to be updated with the probability parameter r (or $1 - r$).

If Q can observe the environment's behaviour and its subsequent behaviour is assignment, we regard $P \otimes_r Q$ as undefined. This consideration is reflected in (1.b) and it can support the understanding of (1.a).

If Q can observe the environment's behaviour and its subsequent behaviour is not assignment, their merge $P \otimes_r Q$ can also perform these assignments from P without any change of the probabilities. This understanding supports the definition of event firing behaviour.

On the other hand, if Q can do time delay, their merge $P \otimes_r Q$ is undefined. This is because assignment is the instantaneous behavior and the two process cannot be compared.

Below are the detailed definition for the above four cases when P can perform probabilistic atomic actions initially.

(1) If $P = \{\!|(1, p_i, \sigma_i) : U_i \mid i \in I|\!\}$ and

(1.a) if $Q = \{\!|(1, q_k, \sigma_k) : V_k \mid k \in K|\!\}$, then

$$\begin{aligned} & P \otimes_r Q \\ =_{df} & \{\!| (1, r \times p_i, \sigma_i) : \{X \otimes_r Q \mid X \in U_i\}, \\ & \quad (1, (1-r) \times p_k, \sigma_k) : \{P \otimes_r Y \mid Y \in V_k\} \mid i \in I \wedge k \in K |\!\} \end{aligned}$$

(1.b) if $Q = \{\!|(0^-, q_k, \sigma_k) : V_k \mid k \in K|\!\}$, then $P \otimes_r Q =_{df} undefined$

(1.c) if $Q = \{\!|(0, q_k, \sigma_k) : V_k \mid k \in K|\!\}$, then there exists a permutation j_1, $j_2, \cdots, j_{|I|}$ of K such that $\forall i \in \bullet p_i = q_{j_i}$ and $\sigma_i = \sigma_{j_i}$, and there exists a bijection $f_i : U_i \to V_{j_i}$ such that $\forall X \in U_i \bullet X \otimes_r f_{j_i}(X)$ is well-defined

$$P \otimes_r Q =_{df} \{\!|(1, p_i, \sigma_i) : \{X \otimes_r f_{j_i}(X) \mid X \in U_i\} \mid i \in U|\!\}$$

(1.d) if $Q = \{\!|(\surd, \emptyset, \sigma) : V|\!\}$, then $P \otimes_r Q =_{df} undefined$

Secondly, we consider the case when P is in observing the environment's behaviour and its subsequent behaviour is probabilistic assignment. Item (2.a) is similar to (1.b). The analysis for (2.b) is also similar to (1.b). Furthermore, If Q is in observing the environment's behaviour and its subsequent behaviour is not probabilistic assignment, we regard their merge $(P \otimes_r Q)$ is still in observing the environment's behaviour and its subsequent behaviour is still assignment action. On the other hand, item (2.d) considers the case that Q can let time advance. As P is currently in observing the environment's instantaneous action, we regard $P \otimes_r Q$ as undefined in this case.

(2) If $P = \{\!|(0^-, p_i, \sigma_i) : U_i\} \mid i \in I|\!\}$ and

(2.a) if $Q = \{\!|(1, q_k, \sigma_k) : V_k\} \mid k \in K|\!\}$, then $P \otimes_r Q =_{df} undefined$

(2.b) if $Q = \{\!|(0^-, q_k, \sigma_k) : V_k\}| \, k \in K|\!\}$, then $P \otimes_r Q =_{df} undefined$

(2.c) if $Q = \{\!|(0, q_k, \sigma_k) : V_k\}| \ k \in K|\!\}$, then there exists a permutation j_1, $j_2, \cdots, j_{|I|}$ of K such that $\forall i \in \bullet p_i = q_{j_i}$ and $\sigma_i = \sigma_{j_i}$, and there exists a bijection $f_i : U_i \to V_{j_i}$ such that $\forall X \in U_i \bullet X \otimes_r f_{j_i}(X)$ is well-defined

$$P \otimes_r Q =_{df} \{\!|(0^-, p_i, \sigma_i) : \{X \otimes_r f_{j_i}(X) \mid X \in U_i\} \mid i \in I|\!\}$$

(2.d) if $Q = \{\!|(\surd, \emptyset, \sigma) : V\}|\!\}$, then $P \otimes_r Q =_{df} undefined$

Thirdly, we consider the case that P is in observing the environment's behaviour and its subsequent behaviour is not probabilistic assignment. Item (3.a) and (3.b) are similar to (1.c) and (2.c) respectively. On the other hand, if Q is also in observing the environment's behaviour and its subsequent behaviour is not probabilistic assignment, their merge ($P \otimes_r Q$) belongs to the same execution type. The consideration for (3.d) is similar to (2.d).

(3) If $P = \{\!|(0, p_i, \sigma_i) : U_i \mid i \in I|\!\}$ and

(3.a) if $Q = \{\!|(1, p_k, \sigma_k) : V_k\} \mid k \in K|\!\}$, then $P \otimes_r Q$ is the same as (1.c)

(3.b) if $Q = \{\!|(0^-, p_k, \sigma_k) : V_k\} \mid k \in K|\!\}$, then $P \otimes_r Q$ is the same as (2.c)

(3.c) if $Q = \{\!|(0, p_k, \sigma_k) : V_k\} \mid k \in K|\!\}$, then there exists a permutation j_1, $j_2, \cdots, j_{|I|}$ of K such that $\forall i \in \bullet p_i = q_{j_i}$ and $\sigma_i = \sigma_{j_i}$, and there exists a bijection $f_i : U_i \to V_{j_i}$ such that $\forall X \in U_i \bullet X \otimes_r f_{j_i}(X)$ is well-defined

$$P \otimes_r Q =_{df} \{\!|(0, p_i, \sigma_i) : \{X \otimes_r f_{j_i}(X) \mid X \in U_i\} \mid i \in I|\!\}$$

(3.d) if $Q = \{\!|(\surd, \emptyset, \sigma) : V|\!\}$, then $P \otimes_r Q =_{df} undefined$

Lastly, we consider that the case that P can perform time delay initially. Item (4.a), (4.b) and (4.c) are all about performing or observing instantaneous actions initially. Hence, their merges ($P \otimes_r Q$) are considered as undefined. Item (4.d) indicates that Q can also let time advance. Therefore, their merge ($P \otimes_r Q$) can also do time advancing initially.

(4) If $P = \{\!|(\surd, \emptyset, \sigma) : U\}|\!\}$ and

(4.a) if $Q = \{\!|(1, p_k, \sigma_k) : V_k\} \mid k \in K|\!\}$, then $P \otimes_r Q =_{df} undefined$

(4.b) if $Q = \{\!|(0^-, p_k, \sigma_k) : V_k\} \mid k \in K|\!\}$, then $P \otimes_r Q =_{df} undefined$

(4.c) if $Q = \{\!|(0, p_k, \sigma_k) : V_k \mid k \in K|\!\}$, then $P \otimes_r Q =_{df} undefined$

(4.d) if $Q = \{\!|(\surd, \emptyset, \sigma) : V|\!\}$, then there exists a bijection $f : U \to V$ such that $\forall X \in U \bullet X \otimes_r f(X)$ is well-defined

$$P \otimes_r Q =_{df} \{\!|(\surd, \emptyset, \sigma) : \{X \otimes_r f(X) \mid X \in U\}|\!\}$$

Further, we also need the table below to complete the definition of $\otimes_r$.

$div \otimes_r div =_{df} div$	$div \otimes_r wait =_{df} div$	$div \otimes_r ter =_{df} div$
$wait \otimes_r div =_{df} div$	$wait \otimes_r wait =_{df} wait$	$wait \otimes_r ter =_{df} wait$
$ter \otimes_r div =_{df} div$,	$ter \otimes_r wait =_{df} wait$	$ter \otimes_r ter =_{df} ter$

We also need the following definition for further support. Here we assume that T is not empty (i.e., $T \neq st$).

$div \otimes_r T =_{df} div$	$T \otimes_r div =_{df} div$
$ter \otimes_r T =_{df} T$	$T \otimes_r ter =_{df} T$

Further, if $T \neq div$, then $wait \otimes_r T = wait$ and $T \otimes_r wait =_{df} wait$

Based on the definition of $\otimes_r$, now we can give the semantics for probabilistic parallel programs.

$$\begin{aligned}&[\![P \parallel_r Q]\!]\\ =_{df}\ &\{\ (tag, \mu, \sigma) : \{X \otimes_r Y \mid X \in U\ \wedge\ Y \in V\ \wedge\ X \otimes_r Y \text{is well-defined}\}\\ &\qquad\qquad \mid (tag, \mu, \sigma) \in \Sigma\ \wedge\ (tag, \mu, \sigma) : U \in [\![P]\!] \wedge (tag, \mu, \sigma) : V \in [\![Q]\!]\ \}\end{aligned}$$

4.6 General Parallel Composition

Now we start to define the semantics for general parallel composition $\|$. We first give the definition for the merge operator $\otimes$, which is symmetric.

Firstly we consider the case that P can perform probabilistic atomic actions initially.

(1) If $P = \{\!|(1, p_i, \sigma_i) : U_i \mid i \in I|\!\}$ and

(1.a) if $Q = \{\!|(1, q_k, \sigma_k) : V_k\} \mid k \in K|\!\}$, then $P \otimes Q =_{df} undefined$

(1.b) if $Q = \{\!|(tag, q_k, \sigma_k) : V_k\} \mid k \in K|\!\}$ ($tag = 0^-$ or 0), then there exists a permutation $j_1, j_2, \cdots, j_{|I|}$ of K such that $\forall i \in \bullet p_i = q_{j_i}$ and $\sigma_i = \sigma_{j_i}$, and there exists a bijection $f_i : U_i \to V_{j_i}$ such that $\forall X \in U_i \bullet X \otimes f_{j_i}(X)$ is well-defined

$$P \otimes Q =_{df} \{\!|(1, p_i, \sigma_i) : \{X \otimes f_{j_i}(X) \mid X \in U_i\} \mid i \in I|\!\}$$

(1.d) if $Q = \{\!|(\surd, \emptyset, \sigma) : V|\!\}$, then $P \otimes Q =_{df} undefined$

Secondly, we consider the case that P is in the observing the environment's behaviour and its subsequent behaviour is the probabilistic assignment.

(2) If $P = \{\!|(0^-, p_i, \sigma_i) : U_i \mid I \in I|\!\}$ and

(2.b) if $Q = \{\!|(tag, q_k, \sigma_k) : V_k \mid k \in K|\!\}$ ($tag = 0^-$ or 0), then there exists a permutation $j_1, j_2, \cdots, j_{|I|}$ of K such that $\forall i \in \bullet p_i = q_{j_i}$ and $\sigma_i = \sigma_{j_i}$, and there exists a bijection $f_i : U_i \to V_{j_i}$ such that $\forall X \in U_i \bullet X \otimes f_{j_i}(X)$ is well-defined

$$P \otimes Q =_{df} \{\!|(0^-, p_i, \sigma_i) : \{X \otimes f_{J_i}(X) \mid X \in U_i\} \mid i \in I|\!\}$$

(2.d) if $Q = \{\!|(\surd, \emptyset, \sigma) : V|\!\}$, then $P \otimes Q =_{df} undefined$

Thirdly, we consider the case that P is in the observing the environment's behaviour and its subsequent behaviour is not the probabilistic assignment.

(3) If $P = \{\!|(0, p_i, \sigma_i) : U_i \mid i \in I|\!\}$ and

(3.c) if $Q = \{\!|(0, q_k, \sigma_k) : V_k \mid k \in K|\!\}$, then there exists a permutation j_1, $j_2, \cdots, j_{|I|}$ of K such that $\forall i \in \bullet p_i = q_{j_i}$ and $\sigma_i = \sigma_{j_i}$, and there exists a bijection $f_i : U_i \to V_{j_i}$ such that $\forall X \in U_i \bullet X \otimes f_{j_i}(X)$ is well-defined

$$P \otimes Q =_{df} \{\!|(0, p_i, \sigma_i) : \{X \otimes f_{j_i}(X) \mid X \in U_i\} \mid i \in I|\!\}$$

(3.d) if $Q = \{\!|(\surd, \emptyset, \sigma) : V|\!\}$, then $P \otimes Q =_{df} undefined$

Fourthly, we consider the case that P can perform time delay initially.

(4) If $P = \{\!|(\surd, \emptyset, \sigma) : U|\!\}$ and

(4.d) if $Q = \{\!|(\surd, \emptyset, \sigma) : V|\!\}$, then there exists a bijection $f : U \to V$ such that $\forall X \in U \bullet X \otimes f(X)$ is well-defined

$$P \otimes_r Q =_{df} \{\!|(\surd, \emptyset, \sigma) : \{X \otimes f(X) \mid X \in U\}|\!\}$$

For the definition of $st \otimes st'$ and $st \otimes T$ (st can be ter, $wait$ or div), they are similar to those in the definition of $\otimes_r$.

Based on the above defined merge operator $\otimes$, now we are ready to give the semantics for general parallel composition.

$$\begin{aligned}
&[\![P \parallel Q]\!] \\
=_{df}\ &\{\ (tag, \mu, \sigma) : \{X \otimes Y \mid X \in U\ \wedge\ Y \in V\ \wedge\ X \otimes Y \text{is well-defined}\} \\
&\qquad\qquad \mid (tag, \mu, \sigma) \in \Sigma\ \wedge\ (tag, \mu, \sigma) : U \in [\![P]\!] \wedge (tag, \mu, \sigma) : V \in [\![Q]\!]\ \}
\end{aligned}$$

5 Program Equivalence

In this section we are exploring some *flattening* relations between P-trees (or P^--trees). It will be used in giving the definitions for the concept of tree equivalence. Based on the concept of the equivalence of P-trees (or P^--trees), we can define the equivalence between *PTSC* programs.

Below we define a set of flattening relations R_i and R'_i ($i = 0, 1, 2, 3$). Based on these flattening relations, we define $R =_{df} Id \cup \bigcup_{i \in \{0,1,2,3\}} (R_i \cup R_i^{-1})$ and $R' =_{df} Id \cup \bigcup_{i \in \{0,1,2,3\}} (R'_i \cup R_i'^{-1})$.

For $\{\!|(tag, 1, \sigma) : U,\ (tag, 0, \sigma) : V\}|\!\}$, due to the 0 probability, the branch $(tag, 0, \sigma) : V$ can be eliminated. This means that the two P^--trees $\{\!|(tag, 1, \sigma) : U,\ (tag, 0, \sigma) : V\}|\!\}$ and $\{\!|(tag, 1, \sigma) : U|\!\}$ should be equivalent. Therefore, we give the definition of flattening relation R_0.

Definition 5.1 (*Flattening Relation* R_0)

(1) $\{\!|(tag, 1, \sigma) : U,\ (tag, 0, \sigma) : V\}|\!\}\ R_0\ \{\!|(tag, 1, \sigma) : W|\!\}$, where $U\, R\, W$[1].

(2) If $\forall X \in U \bullet \exists Y \in Y \bullet (X, Y) \in R_0$ and $\forall Y \in V \bullet \exists X \in U \bullet (X, Y) \in R_0$, then $(tag, \mu, \sigma) : U\ R'_0\ (tag, \mu, \sigma) : V$. □

[1] Let U and V be two sets of P^--trees and S be a relation between P^--trees. The notation $U\, S\, V$ means that $\forall X \in U \bullet \exists Y \in Y \bullet (X, Y) \in S$ and $\forall Y \in V \bullet \exists X \in U \bullet (X, Y) \in S$

For $\{\{\!|(1,1,\sigma) : U|\!\}\}$ at the initial state (tag,μ,σ), the contributed snapshot $(1,1,\sigma)$ indicates that the process performs assignment-like action with probability 1 and the data state remains the unchanged. This means that the contributed new snapshot can be eliminated, indicating that $(tag,\mu,\sigma) : \{\{\!|(1,1,\sigma) : U|\!\}\}$ is the same as $(tag,\mu,\sigma) : U$. We give the definition of flattening relation R_1 for aiming this.

Definition 5.2 (*Flattening Relation* R_1)

(1) $(tag,\mu,\sigma) : \{\{\!|(1,1,\sigma) : U|\!\}\}\ R_1'\ (tag,\mu,\sigma) : V,$ where $U\ R\ V$.

(2) If $(tag_i,\mu_i,\sigma_i) : U_i\ R_1'(tag_i,\mu_i,\sigma_i) : V_i,\quad i \in I$

then $\{\!|(tag_i,\mu_i,\sigma_i) : U_i \mid i \in I\}\!|\ R_1\{\!|(tag_i,\mu_i,\sigma_i) : V_i \mid i \in I\}\!|$ □

For P^--tree $\{\!|(1,r,\sigma) : U,\ (1,1-r,\sigma) : U|\!\}$ at the initial state (tag,μ,σ), we find that the two probabilistic branches of the P^--tree enter into the same process. The two data sates are the same as the σ, the sum of the two corresponding probabilities are 1, and the two subsequent behaviours are the same. Therefore, we can say the behaviour of the P^--tree $\{\!|(1,r,\sigma) : U,\ (1,1-r,\sigma) : U|\!\}$ at the initial state (tag,μ,σ) should be the same as the behaviour $(tag,\mu,\sigma) : U$. Our definition for flattening relation R_2 is for achieving this kind of equivalence.

Definition 5.3 (*Flattening Relation* R_2)

(1) $(tag,\mu,\sigma) : \{\ \{\!|(1,r,\sigma) : U,\ (1,1-r,\sigma) : U|\!\}\ \}\ R_2'\ (tag,\mu,\sigma) : V,$
where $U\ R\ V$.

(2) $(tag,\mu,\sigma) : \{\{\!|(1,r,\sigma) : U,\ (1,1-r,\sigma) : U|\!\} \mid r \in A\}\ R_2'\ (tag,\mu,\sigma) : V,$
where $U\ R\ V$

(3) If $(tag_i,\mu_i,\sigma_i) : U_i\ R_2'\ (tag_i,\mu_i,\sigma_i) : V_i,\quad i \in I$

then $\{\!|(tag_i,\mu_i,\sigma_i) : U_i\} \mid i \in I|\!\}\ R_2\ \{\!|(tag_i,\mu_i,\sigma_i) : V_i \mid i \in I\}\!|\}$ □

Definition 5.4 (*Flattening Relation* R_3)

(1) Define $P\ R_3'\ Q,$ where:

$P = (tag,\mu,\sigma) : \{\{\!|(1,p,\sigma) : U, (1,1-p,\sigma) : \{\{\!|(1,q,\sigma) : V, (1,1-q,\sigma) : W|\!\}\}|\!\}\}$

$Q = (tag,\mu,\sigma) : \{\{\!|(1,y,\sigma) : \{\{\!|(1,x,\sigma) : U', (1,1-x,\sigma) : V'\}\!\}\}, (1,1-y,\sigma) : W'\}\!\}\}$

where, $U\ R\ U',\quad V\ R\ V',\quad W\ R\ W',$

and, $x = p/(p+q-p\times q),\quad y = p+q-p\times q$

(2) If $(tag_i,\mu_i,\sigma_i) : U_i\ R_3'\ (tag_i,\mu_i,\sigma_i) : V_i,\quad i \in I$

then $\{\!|(tag_i,\mu_i,\sigma_i) : U_i \mid i \in I|\!\}\ R_3\ \{\!|(tag_i,\mu_i,\sigma_i) : V_i\} \mid i \in I|\!\}$ □

In the above definition for flattening relation R_3, for P and Q, both of them have three execution branches (i.e., U, V and W). The three new data states for U, V and W (i.e., U', V' and W') are the same as the initial data state of P and Q. Further, the probabilities to reach to U (i.e., U') for P and Q are both p, whereas the probabilities to reach to V (i.e., V') for P and Q are both $(1-p)\times q$. And the probability to reach to W (i.e., W') for both P and Q are $(1-p)\times(1-q)$. This indicates that P and Q should have the same behaviour.

Based on the above definitions of flattening relations and R (and R'), we know that R (and R') is an equivalence relation. Let $\approx$ and $\approx'$ stand for the largest relations satisfying R and R' respectively.

Now we start to consider program equivalence. Its definition can be based on the equivalence $\approx$ and $\approx'$ (for P^--trees and P-trees), shown below.

Definition 5.5 (Program Equivalence)

$$P \approx Q$$
$$=_{df} \forall (tag, \mu, \sigma) : U \in [\![P]\!], (tag, \mu, \sigma) : V \in [\![Q]\!] \bullet$$
$$(tag, \mu, \sigma) : U \approx' (tag, \mu, \sigma) : V \qquad \square$$

6 Algebraic Laws

In this section we explore the algebraic laws for *PTSC* programs based on the defined program equivalence. For assignment, conditional, iteration, nondeterministic choice and sequential composition, our language enjoys similar algebraic properties as those reported in [4, 7]. In what follows, we shall only focus on novel algebraic properties with respect to time, probabilistic nondeterministic choice and parallel composition.

6.1 Sequential Constructs

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) $\#n; \#m \ \approx \ \#(n+m)$

Probabilistic nondeterministic choice is idempotent.

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

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

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

(prob-3) $P \sqcap_p (Q \sqcap_q R) \ \approx \ (P \sqcap_x Q) \sqcap_y R$

where $x = p/(p+q-p\times q)$ and $y = p+q-p\times q$

6.2 Parallel Construct

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) $P \parallel_p Q \ \approx \ Q \parallel_{1-p} P$

(par-2) $P \parallel_p (Q \parallel_q R) \ \approx \ (P \parallel_x Q) \parallel_y R$

where, $x = p/(p+q-p\times q)$ and $y = p+q-p\times q$

For general parallel composition, it is purely symmetric and associative.

(par-1') $P \parallel Q \approx Q \parallel P$

(par-2') $P \parallel (Q \parallel R) \approx (P \parallel Q) \parallel R$

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 probabilistic parallel composition of two arbitrary guarded choices, we end up with fifteen different expansion laws. Similarly, for general parallel composition, we also have fifteen different expansion laws. We select some parallel expansion cases for both probabilistic and general parallel compositions.

Firstly, we consider the case that both of the two parallel components are with the form of assignment-guarded choice. The expansion law is shown is (par-3-1).

(par-3-1) Let

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

Then

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

and

$$\begin{aligned} & P \parallel Q \\ \approx\ & \mathbb{I}_{i\in I}\{[p_i]\ \mathit{choice}_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij})\ P_{ij} \parallel Q\} \\ & \textbf{or}\ \mathbb{I}_{k\in K}\{[q_k]\ \mathit{choice}_{l\in L_k}(b_{kl}\&(x_{kl} := e_{kl})\ P \parallel Q_{kl}\} \end{aligned}$$

Next we consider the case that one parallel component is with the form of assignment guarded choice and another component is with the form of event guarded choice. Law (par-3-2) below shows the expansion law for probabilistic and general parallel composition. The probability factor for the initial step expansion does not affect for the two parallel compositions.

(par-3-2) Let

$$P = \mathbb{I}_{i\in I}\{[p_i]\ \mathit{choice}_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij})\ P_{ij})\} \quad \text{and} \quad Q = \mathbb{I}_{k\in K}\{@c_k\ Q_k\}$$

Then

$$\begin{aligned} & P \odot Q \\ \approx\ & \mathbb{I}_{i\in I}\{[p_i]\ \mathit{choice}_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij})\ P_{ij} \odot Q\} \\ & \mathbb{I}\mathbb{I}_{k\in K}\{@c_k\ P \odot Q_k\} \end{aligned}$$

where, $\odot \in \{\parallel_r, \parallel\}$.

Now we consider the case that one parallel component is with the form of assignment guarded choice and another component is with the form of time delay component. Only assignment guards can be scheduled initially. This case is expressed in law (par-3-3).

(par-3-3) Let

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

Then

$$\begin{aligned} & P \odot Q \\ \approx\ & [\!]_{i\in I}\{[p_i]\ \mathit{choice}_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij})\ P_{ij} \odot Q\} \end{aligned}$$

where, $\odot \in \{\ \|_r,\ \|\ \}$.

If one parallel component is with the form of assignment guarded choice and another parallel component is with the form of the guarded choice composing of assignment guarded components and event guarded components. Law (par-3-4) shows the expansion for the probabilistic parallel composition and general parallel composition for this case.

(par-3-4) Let

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

Then

$$\begin{aligned} & P \|_r Q \\ \approx\ & [\!]_{i\in I}\{[r \times p_i]\ \mathit{choice}_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij})\ P_{ij} \|_r Q)\} \\ & [\!][\!]_{k\in K}\{[(1-r) \times q_k]\ \mathit{choice}_{l\in L_k}(b_{kl}\&(x_{kl} := e_{kl})\ P \|_r Q_{kl}\} \\ & [\!][\!]_{m\in M}\{@c_k\ P \|_r R_m\} \end{aligned}$$

$$\begin{aligned} & P \| Q \\ \approx\ & [\!]_{i\in I}\{[p_i]\ \mathit{choice}_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij})\ P_{ij} \| Q)\} \\ & [\!][\!]_{m\in M}\{@c_k\ P \| R_m\} \\ & \textbf{or} \\ & [\!]_{k\in K}\{[q_k]\ \mathit{choice}_{l\in L_k}(b_{kl}\&(x_{kl} := e_{kl})\ P \| Q_{kl}\} \\ & [\!][\!]_{m\in M}\{@c_k\ P \| R_m\} \end{aligned}$$

Now we consider the case that both of the two parallel components are of the form of event guarded choice. For probabilistic parallel composition and general parallel composition, there are three event triggered cases. Also the probability factor does not have effects in the initial step expansion. 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

$$\begin{aligned} & P \odot Q \\ \approx\ & [\!]_{i\in I}\{@(b_i \wedge \neg c)\ P_i \odot Q\}[\!][\!]_{j\in J}\{@(c_j \wedge \neg b)\ P \odot Q_j\} \\ & [\!][\!]_{i\in I\wedge j\in J}\{@(b_i \wedge c_j)\ P_i \odot Q_j\} \end{aligned}$$

where, $\odot \in \{\ \|_r,\ \|\ \}$

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

We now move to the case that both of the two parallel components are of the form comprising assignment-guarded components and event-guarded components. Law

(par-3-13) stands for the expansion for probabilistic parallel composition and general parallel composition.

(par-3-13) Let

$$P = \|_{i\in I}\{[p_i]\ \mathit{choice}_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij})\ P_{ij})\}[\!]_{k\in K}\{@b_k\ R_k\}$$

and $$Q = \|_{l\in L}\{[q_l]\ \mathit{choice}_{m\in M_l}(c_{lm}\&(x_{lm} := e_{lm})\ P_{lm})\}[\!]_{n\in N}\{@c_n\ T_n\}$$

Then

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

$$\begin{aligned}
&P \| Q\\
\approx\ &\|_{i\in I}\{[p_i]\ \mathit{choice}_{j\in J_i}(b_{ij}\&(x_{ij} := e_{ij})\ P_{ij}\ \|\ Q)\}\\
&[\!]\|_{k\in K}\{@(b_k\wedge\neg c)\ R_k\ \|\ Q\}[\!]\|_{n\in N}\{@(c_n\wedge\neg b)\ R_k\ \|\ Q\}\\
&[\!]\|_{k\in K\wedge n\in N}\{@(b_k\wedge c_n)\ R_k\ \|\ Q_n\}\\
&\textbf{or}\\
&\|_{l\in L}\{[q_l]\ \mathit{choice}_{m\in M_l}(c_{lm}\&(x_{lm} := e_{lm})\ P, \|\ Q_{lm})\}\\
&[\!]\|_{k\in K}\{@(b_k\wedge\neg c)\ R_k\ \|\ Q\}[\!]\|_{n\in N}\{@(c_n\wedge\neg b)\ R_k\ \|\ Q\}\\
&[\!]\|_{k\in K\wedge n\in N}\{@(b_k\wedge c_n)\ R_k\ \|\ Q_n\}\\
&\text{where,}\quad b = \vee_{k\in K}\ b_k\quad\text{and}\quad c = \vee_{n\in N}\ c_n
\end{aligned}$$

7 Conclusion

Recently we have proposed the language *PTSC* [19], which integrates probability, time and shared-variable concurrency. In this paper, we studied the denotational semantics for *PTSC*. For dealing with the above three features, as well as the nondeterminism, we introduced the concept of P-tree in our denotational semantics. Based on the P-tree, we defined the denotational semantics for each *PTSC* statement. In order to deal with program equivalence based on the achieved denotational semantics, we defined a set of flattening relations. We have explored a set of algebraic laws for *PTSC*, especially a set of parallel expansion laws. The correctness of these laws is based on the concept of program equivalence.

For the future, we would like to link the denotational semantics with operational semantics and algebraic semantics respectively. Moreover, the deduction approach for *PTSC* is also challenging to work on.

Acknowledgement. This work is supported by National Basic Research Program of China (No. 2011CB302904), National High Technology Research and Development Program of China (No. 2011AA010101 and No. 2012AA011205), National Natural Science Foundation of China (No. 61061130541 and No. 61021004), and Shanghai Leading Academic Discipline Project (No. B412).

References

1. den Hartog, J.: Probabilistic Extensions of Semantic Models. PhD thesis, Vrije University, The Netherlands (2002)
2. den Hartog, J., de Vink, E.: Mixing up nondeteminism and probability: A premliminary report. Electronic Notes in Theoretical Computer Science 22 (1999)
3. den Hartog, J., de Vink, E., de Bakker, J.: Metric semantics and full abstractness for action refinement and probabilistic choice. Electronic Notes in Theoretical Computer Science 40 (2001)
4. He, J.: Provably Correct Systems: Modelling of Communication Languages and Design of Optimized Compilers. The McGraw-Hill International Series in Software Engineering (1994)
5. He, J., Seidel, K., McIver, A.: Probabilistic models for the guarded command language. Science of Computer Programming 28(2-3), 171–192 (1997)
6. He, J., Zhu, H.: Formalising Verilog. In: Proc. ICECS 2000: IEEE International Conference on Electronics, Circuits and Systems, pp. 412–415. IEEE Computer Society Press (December 2000)
7. Hoare, C.A.R., He, J.: Unifying Theories of Programming. Prentice Hall International Series in Computer Science (1998)
8. McIver, A., Morgan, C.: Partial correctness for probabilistic demonic programs. Theoretical Computer Science 266(1-2), 513–541 (2001)
9. McIver, A., Morgan, C.: Abstraction, Refinement and Proof of Probability Systems. Monographs in Computer Science. Springer (October 2004)
10. McIver, A., Morgan, C., Seidel, K.: Probabilistic predicate transformers. ACM Transactions on Programming Languages and Systems 18(3), 325–353 (1996)
11. Ndukwu, U., Sanders, J.W.: Reason about a distributed probabilistic system. Technical Report 401, UNU/IIST, P.O. Box 3058, Macau SAR, China (August. 2008)
12. Nissanke, N.: Realtime Systems. Prentice Hall International Series in Computer Science (1997)
13. Núñez, M.: Algebraic theory of probabilistic processes. The Journal of Logic and Algebraic Programming 56, 117–177 (2003)
14. Núñez, M., de Frutos-Escrig, D.: Testing semantics for probabilistic LOTOS. In: Proc FORTE 1995: IFIP TC6 Eighth International Conference on Formal Description Techniques, Montreal, Canada. IFIP Conference Proceedings, vol. 43, pp. 367–382. Chapman and Hall (1996)
15. Núñez, M., de Frutos-Escrig, D., Díaz, L.F.L.: Acceptance Trees for Probabilistic Processes. In: Lee, I., Smolka, S.A. (eds.) CONCUR 1995. LNCS, vol. 962, pp. 249–263. Springer, Heidelberg (1995)
16. Park, S., Pfenning, F., Thrun, S.: A probabilistic language based upon sampling functions. In: Proc. POPL 2005: 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 171–182. ACM (January 2005)
17. Zhu, H.: Linking the Semantics of a Multithreaded Discrete Event Simulation Language. PhD thesis, London South Bank University (February 2005)
18. Zhu, H., He, J., Bowen, J.P.: From algebraic semantics to denotational semantics for verilog. Innovations in Systems and Software Engineering: A NASA Journal 4(4), 341–360 (2008)
19. Zhu, H., Qin, S., He, J., Bowen, J.P.: PTSC: probability, time and shared-variable concurrency. Innovations in Systems and Software Engineering: A NASA Journal 5(4), 271–284 (2009)
20. Zhu, H., Yang, F., He, J., Bowen, J.P., Sanders, J.W., Qin, S.: Linking operational semantics and algebraic semantics for a probabilistic timed shared-variable language. J. Log. Algebr. Program. 81(1), 2–25 (2012)