S©MR ISSN 1813-3304

СИБИРСКИЕ ЭЛЕКТРОННЫЕ
МАТЕМАТИЧЕСКИЕ ИЗВЕСТИЯ

Siberian Electronic Mathematical Reports
http://semr.math.nsc.ru

Том 17, стр. 1–47 (2020) УДК 004.423.4, 519.681.2, 519.681.3
DOI 10.33048/semi.2020.17.xxx MSC 18C10, 68Q55, 68Q85

DISCRETE TIME STOCHASTIC AND DETERMINISTIC PETRI BOX CALCULUS DTSDPBC

I.V. TARASYUK

Abstract. We propose dtsdPBC, an extension with deterministically timed multiactions of discrete time stochastic and immediate Petri box calculus (dtsiPBC), previously presented by I.V. Tarasyuk, H. Macià and V. Valero. dtsdPBC enhances the expressiveness of dtsiPBC and extends the application area of the associated specification and analysis techniques. In dtsdPBC, non-negative integers are used to specify fixed (including zero) time delays of deterministic multiactions. The step operational semantics is constructed via labeled probabilistic transition systems. A series of examples that construct the transition systems for the expressions with different types of multiactions and operations demonstrates both the specification capabilities and semantic features of the new calculus.

Keywords: stochastic process algebra, Petri box calculus, discrete time, stochastic multiaction, deterministic multiaction, transition system, operational semantics, semantic example.

1. Introduction

Algebraic process calculi like CSP [21], ACP [4] and CCS [37] are well-known formal models for specification of computing systems and analysis of their behaviour. In such process algebras (PAs), systems and processes are specified by formulas, and verification of their properties is accomplished at a syntactic level via equivalences, axioms and inference rules. In recent decades, stochastic extensions of PAs were proposed, such as MTIPP [18], PEPA [20] and EMPA [6]. Unlike standard PAs, stochastic process algebras (SPAs) do not just specify actions which can occur

Tarasyuk, I.V., Discrete time stochastic and deterministic Petri box calculus dtsdPBC.

Received March, 12, 2020, published Month, xx, 2020.

(qualitative features), but they associate with the actions the distribution parameters of their random time delays (quantitative characteristics).

1.1. **Petri box calculus.** PAs specify concurrent systems in a compositional way via an expressive formal syntax. On the other hand, Petri nets (PNs) provide a graphical representation of such systems and capture explicit asynchrony in their behaviour. To combine the advantages of both models, a semantics of algebraic formulas via PNs was defined.

Petri box calculus (PBC) [7, 9, 8] is a flexible and expressive process algebra developed as a tool for specification of the PNs structure and their interrelations. Its goal was also to propose a compositional semantics for high level constructs of concurrent programming languages in terms of elementary PNs. Formulas of PBC are combined not from single (visible or invisible) actions and variables, like in CCS, but from multisets of elementary actions and their conjugates, called multiactions (*basic formulas*). The empty multiset of actions is interpreted as the silent multiaction specifying some invisible activity. In contrast to CCS, synchronization is separated from parallelism (*concurrent constructs*). Synchronization is a unary multi-way stepwise operation, based on communication of actions and their conjugates. This extends the CCS approach with conjugate matching labels. Synchronization in PBC is asynchronous, unlike that in Synchronous CCS (SCCS) [37]. Other operations are sequence and choice (*sequential constructs*). The calculus includes also restriction and relabeling (*abstraction constructs*). To specify infinite processes, refinement, recursion and iteration operations were added (*hierarchical constructs*). Thus, unlike CCS, PBC has an additional iteration operation to specify infinite behaviour when the semantic interpretation in finite PNs is possible. PBC has a step operational semantics in terms of labeled transition systems, based on the rules of structural operational semantics (SOS). The operational semantics of PBC is of step type, since its SOS rules have transitions with (multi)sets of activities, corresponding to simultaneous executions of activities (steps). A denotational semantics of PBC was proposed via a subclass of PNs equipped with an interface and considered up to isomorphism, called Petri boxes. For more detailed comparison of PBC with other process algebras and the reasoning about importance of non-interleaving semantics see [7, 8].

The extensions of PBC with a deterministic, a nondeterministic or a stochastic model of time were presented.

1.2. **Time extensions of Petri box calculus.** To specify systems with time constraints, deterministic (fixed) or nondeterministic (interval) delays are used.

A time extension of PBC with a nondeterministic time model, called time Petri box calculus (tPBC), was proposed in [23]. In tPBC, timing information is added by associating time intervals (the earliest and the latest firing time) with instantaneous *actions*. tPBC has a step time operational semantics in terms of labeled transition systems. Its denotational semantics was defined in terms of a subclass of labeled time Petri nets (LtPNs), based on tPNs [36] and called time Petri boxes (ct-boxes).

Another time enrichment of PBC, called Timed Petri box calculus (TPBC), was defined in [32, 33], it accommodates a deterministic model of time. In contrast to tPBC, multiactions of TPBC are not instantaneous, but have time durations. Additionally, in TPBC there exist no "illegal" multiaction occurrences, unlike tPBC. The complexity of "illegal" occurrences mechanism was one of the main intentions

to construct TPBC though this calculus appeared to be more complicated than tPBC. TPBC has a step timed operational semantics in terms of labeled transition systems. The denotational semantics of TPBC was defined in terms of a subclass of labeled Timed Petri nets (LTPNs), based on TPNs [42] and called Timed Petri boxes (T-boxes). tPBC and TPBC differ in ways they capture time information, and they are not in competition but complement each other.

The third time extension of PBC, called arc time Petri box calculus (atPBC), was constructed in [40, 41], and it implements a nondeterministic time. In atPBC, multiactions are associated with time delay intervals. atPBC possesses a step time operational semantics in terms of labeled transition systems. Its denotational semantics was defined on a subclass of labeled arc time Petri nets (atPNs), based of those from [10, 17], where time restrictions are associated with the arcs, called arc time Petri boxes (at-boxes).

tPBC, TPBC and atPBC, all adopt the discrete time approach, but TPBC has no immediate (multi)actions.

1.3. **Stochastic extensions of Petri box calculus.** The set of states for the systems with deterministic or nondeterministic delays often differs drastically from that for the timeless systems, hence, the analysis results for untimed systems may be not valid for the time ones. To solve this problem, stochastic delays are considered, which are the random variables with a (discrete or continuous) probability distribution. If the random variables governing delays have an infinite support then the corresponding SPA can exhibit all the same behaviour as its underlying untimed PA.

A stochastic extension of PBC, called stochastic Petri box calculus (sPBC), was proposed in [28, 24]. In sPBC, multiactions have stochastic delays that follow (negative) exponential distribution. Each multiaction is equipped with a rate that is a parameter of the corresponding exponential distribution. The instantaneous execution of a stochastic multiaction is possible only after the corresponding stochastic time delay. The calculus has an interleaving operational semantics defined via transition systems labeled with multiactions and their rates. Its denotational semantics was defined in terms of a subclass of labeled continuous time stochastic PNs, based on CTSPNs [34, 2] and called stochastic Petri boxes (s-boxes). In sPBC, performance of the processes is evaluated by analyzing their underlying continuous time Markov chains (CTMCs). In [25], new equivalence relations were proposed for regular terms of sPBC to choose later a suitable candidate for a congruence.

sPBC was enriched with immediate multiactions having zero delay in [26, 27]. We call such an extension generalized sPBC (gsPBC). An interleaving operational semantics of gsPBC was constructed via transition systems labeled with stochastic or immediate multiactions together with their rates or probabilities. A denotational semantics of gsPBC was defined via a subclass of labeled generalized stochastic PNs, based on GSPNs [34, 2, 3] and called generalized stochastic Petri boxes (gs-boxes). The performance analysis in gsPBC is based on semi-Markov chains (SMCs).

PBC has a step operational semantics, whereas sPBC has an interleaving one. In step semantics, parallel executions of activities (steps) are permitted while in interleaving semantics, we can execute only single activities. Hence, a stochastic extension of PBC with a step semantics was needed to keep the concurrency degree of behavioural analysis at the same level as in PBC. As mentioned in [38, 39], in contrast to continuous time approach (used in sPBC), discrete time approach allows for constructing models of common clock systems and clocked devices. In

such models, multiple transition firings (or executions of multiple activities) at time moments (ticks of the central clock) are possible, resulting in a step semantics. Moreover, employment of discrete stochastic time fills the gap between the models with deterministic (fixed) time delays and those with continuous stochastic time delays. As argued in [1], arbitrary delay distributions are much easier to handle in a discrete time domain. In [30, 31, 29], discrete stochastic time was preferred to enable simultaneous expiration of multiple delays.

In [44, 45, 46, 47], a discrete time stochastic extension dtsPBC of finite PBC was presented. In dtsPBC, the residence time in the process states is geometrically distributed. A step operational semantics of dtsPBC was constructed via labeled probabilistic transition systems. Its denotational semantics was defined in terms of a subclass of labeled discrete time stochastic PNs (LDTSPNs), based on DTSPNs [38, 39] and called discrete time stochastic Petri boxes (dts-boxes). The performance evaluation in dtsPBC is accomplished via the underlying discrete time Markov chains (DTMCs) of the algebraic processes. A variety of stochastic equivalences were proposed to identify stochastic processes with similar behaviour which are differentiated by the semantic equivalence. The interrelations of all the introduced equivalences were studied. Since dtsPBC has a discrete time semantics and geometrically distributed sojourn time in the process states, unlike sPBC with continuous time semantics and exponentially distributed delays, the calculi apply two different approaches to the stochastic extension of PBC, in spite of some similarity of their syntax and semantics inherited from PBC. The main advantage of dtsPBC is that concurrency is treated like in PBC having step semantics, whereas in sPBC parallelism is simulated by interleaving, obliging one to collect the information on causal independence of activities before constructing the semantics.

In [48, 49, 50, 51, 52], we presented an enhanced calculus dtsiPBC, an extension with immediate multiactions of dtsPBC. Immediate multiactions increase the specification capability: they can model logical conditions, probabilistic branching, instantaneous probabilistic choices and activities whose durations are negligible in comparison with those of others. They are also used to specify urgent activities and the ones that are not relevant for performance evaluation. Thus, immediate multiactions can be considered as a kind of instantaneous dynamic state adjustment and, in many cases, they result in a simpler and more clear system representation. The step operational semantics of dtsiPBC was constructed with the use of labeled probabilistic transition systems. Its denotational semantics was defined via a subclass of labeled discrete time stochastic and immediate PNs (LDTSIPNs), based on the extension of DTSPNs [38, 39] with transition labeling and immediate transitions, called dtsi-boxes. The corresponding stochastic process, the underlying SMC, was investigated, with the purpose of performance evaluation. In addition, the alternative solution methods were developed, based on the underlying (reduced) DTMC.

1.4. **Our contributions.** In this paper, we present an extension of dtsiPBC with deterministic multiactions, called *discrete time stochastic and deterministic Petri box calculus* (dtsdPBC), which enhances the expressiveness of dtsiPBC and extends the application area of the associated specification and analysis techniques. In dtsdPBC, besides the probabilities from the real-valued interval $(0; 1)$ that are used to calculate discrete time delays of stochastic multiactions, also non-negative integers are used to specify fixed time delays of deterministic multiactions (including zero delay, which is the case of immediate multiactions). To resolve conflicts among

deterministic multiactions, they are additionally equipped with positive real-valued weights. As argued in [58, 54, 55], a combination of deterministic and stochastic delays fits well to model technical systems with constant (fixed) durations of the regular non-random activities and probabilistically distributed (stochastic) durations of the randomly occurring activities.

It should be stressed that dtsdPBC is rather a qualitative than merely a quantitative extension of dtsiPBC. The main reason is that in the former calculus, the probability of transitions between markings (untimed states, represented by overbars and underbars in the process expressions) generally depends both on the current marking and for how long the deterministic multiactions were enabled. Hence, the marking change probabilities in dtsdPBC may not possess the Markov (memoryless) property. Thus, the timer values should be associated with deterministic multiactions to specify the process states and then obtain the (semi-)Markovian state change probabilities as a result of "unfolding" the discrete residence times at the markings. In other words, the longer that one delays at the markings should be splitted into one time units and be allocated with the consecutive process states, in order to obtain a (semi-)Markovian model.

Another reason is that, unlike dtsiPBC, the activities of different types can be executed from the the same marking in dtsdPBC, depending on the (decreasing) timer values of the enabled deterministic multiactions. In particular, the enabled stochastic multiactions may preempt the enabled waiting (positively delayed deterministic) ones that cannot be executed at the next time moment from a marking. Otherwise, only enabled waiting multiactions are executed from it. Immediate multiactions are always executed first and separately from other types of activities. It is supposed that the activities are ordered according to their priorities as follows: immediate (highest priority), waiting (middle priority) and stochastic (lowest priority) multiactions.

Our novel approach was inspired by some ideas on combining deterministic and stochastic discrete time transition delays in DTSPNs [38, 39], discrete time deterministic and stochastic PNs (DTDSPNs) [58, 54, 55], dts-nets [1], non-Markovian SPNs (NMSPNs) [22] and stochastic preemptive time PNs (spTPNs) [14] (all with parallel step semantics), as well as in defective discrete phase SPNs (DDP-SPNs) [15], discrete deterministic and stochastic PNs (DDSPNs) [56, 57] and DTDSPNs from [60, 61, 59] (all featuring interleaving semantics). The key idea was to interpret the waiting multiactions with the timer values (remaining times to execute) one as the (stochastic) transitions of DTSPNs [38, 39] with the conditional probability 1. Then the waiting multiactions with the timer values greater than one are ignored, i.e. when enabled, they are executed with the probability 0 at the next time moment.

The step operational semantics of dtsdPBC is constructed with the use of labeled probabilistic transition systems. With a number of interesting and non-trivial examples that include the travel system model, we demonstrate how to construct the transition systems with 3 kinds of states (stochastically tangible, waitingly tangible and vanishing) and 4 kinds of transitions (by executing the empty multiset, stochastic, waiting or immediate multiactions) from the expressions with different types of activities and various operations. From stochastically tangible (s-tangible) states, only the empty set or stochastic multiactions can be executed at the next time moment (after one unit delay). From waitingly tangible (w-tangible) states, only waiting multiactions can be executed at the next time moment. From vanishing states,

only immediate multiactions can be executed at the same time moment (after zero delay). The examples also show the specification flexibility and expressive power of the calculus, as well as the most important features and peculiarities of its semantics.

Thus, the main contributions of the paper are the following.

- New discrete time SPA with stochastic and deterministic activities dtsdPBC.
- Step operational semantics via labeled probabilistic transition systems.
- Transition systems of the expressions with different types of multiactions.

1.5. **Structure of the paper.** The paper is organized as follows. In Section 2, the syntax of algebra dtsdPBC is proposed. In Section 3, we construct the operational semantics of the calculus in terms of labeled probabilistic transition systems and present examples of expressions with their transition systems. Finally, Section 4 summarizes the results obtained and outlines research perspectives in this area.

2. Syntax

In this section, we propose the syntax of dtsdPBC. First, we recall a definition of multiset that is an extension of the set notion by allowing several identical elements.

Definition 1. *Let X be a set. A finite* multiset (bag) *M over X is a mapping $M : X \to \mathbb{N}$ such that $|\{x \in X \mid M(x) > 0\}| < \infty$, i.e. it can contain a finite number of elements only.*

We denote the *set of all finite multisets* over a set X by $\mathbb{N}^X_{fin}$. Let $M, M' \in \mathbb{N}^X_{fin}$. The *cardinality* of M is $|M| = \sum_{x\in X} M(x)$. We write $x \in M$ if $M(x) > 0$ and $M \subseteq M'$ if $\forall x \in X\ M(x) \leq M'(x)$. We define $(M + M')(x) = M(x) + M'(x)$ and $(M - M')(x) = \max\{0, M(x) - M'(x)\}$. When $\forall x \in X,\ M(x) \leq 1,\ M$ can be seen as a proper set $M \subseteq X$. The *set of all subsets (powerset)* of X is denoted by 2^X.

Let $Act = \{a, b, \ldots\}$ be the set of *elementary actions*. Then $\widehat{Act} = \{\hat{a}, \hat{b}, \ldots\}$ is the set of *conjugated actions (conjugates)* such that $\hat{a} \neq a$ and $\hat{\hat{a}} = a$. Let $\mathcal{A} = Act \cup \widehat{Act}$ be the set of *all actions*, and $\mathcal{L} = \mathbb{N}^{\mathcal{A}}_{fin}$ be the set of *all multiactions*. Note that $\emptyset \in \mathcal{L}$, this corresponds to an internal move, i.e. the execution of a multiaction that contains no visible action names. The *alphabet* of $\alpha \in \mathcal{L}$ is defined as $\mathcal{A}(\alpha) = \{x \in \mathcal{A} \mid \alpha(x) > 0\}$.

A *stochastic multiaction* is a pair (α, ρ), where $\alpha \in \mathcal{L}$ and $\rho \in (0; 1)$ is the *probability* of the multiaction α. This probability is interpreted as that of independent execution of the stochastic multiaction at the next discrete time moment. Such probabilities are used to calculate those to execute (possibly empty) sets of stochastic multiactions after one time unit delay. The probabilities of stochastic multiactions are required not to be equal to 1 to avoid extra model complexity, since in this case one should assign with them weights, needed to make a choice when several stochastic multiactions with probability 1 can be executed from a state. The difficulty is that when the stochastic multiactions with probability 1 occur in a step (parallel execution), all other with the less probabilities do not. In this case, the conflicts resolving requires a special attention, as discussed in [38, 39] within SPNs. This decision also allows us to avoid technical difficulties related to conditioning events with probability 0. The probability 1 is left for (implicitly assigned to) waiting multiactions (positively delayed deterministic multiactions, to be defined later), which are delayed for at least one time unit before their execution and have weights to resolve conflicts with other waiting multiactions. On the other hand, there is no

sense to allow probability 0 of stochastic multiactions, since they would never be performed in this case. Let $\mathcal{SL}$ be the set of *all stochastic multiactions*.

A *deterministic multiaction* is a pair $(\alpha, \natural_l^\theta)$, where $\alpha \in \mathcal{L},\ \theta \in \mathbb{N}$ is the non-negative integer-valued *(fixed) delay* and $l \in \mathbb{R}_{>0} = (0; \infty)$ is the positive real-valued *weight* of the multiaction α. This weight is interpreted as a measure of importance (urgency, interest) or a bonus reward associated with execution of the deterministic multiaction at the discrete time moment when the corresponding delay has expired. Such weights are used to calculate the probabilities to execute sets of deterministic multiactions after their time delays. An *immediate multiaction* is a deterministic multiaction with the delay 0 while a *waiting multiaction* is a deterministic multiaction with a positive delay. In case of no conflicts among waiting multiactions, whose remaining times to execute (RTEs, to be explained later in more detail) are equal to one time unit, they are executed with probability 1 at the next time moment. Deterministic multiactions have a priority over stochastic ones, and there is also difference in priorities between immediate and waiting multiactions. One can assume that all immediate multiactions have (the highest) priority 2 and all waiting multiactions have (the medium) priority 1, whereas all stochastic multiactions have (the lowest) priority 0. This means that in a state where all kinds of multiactions can occur, immediate multiactions always occur before waiting ones that, in turn, are always executed before stochastic ones. Different types of multiactions cannot participate together in some step (parallel execution), i.e. just the steps consisting only of immediate multiactions or waiting ones, or those including only stochastic multiactions, are allowed. Let $\mathcal{DL}$ be the set of *all deterministic multiactions*, $\mathcal{IL}$ be the set of *all immediate multiactions* and $\mathcal{WL}$ be the set of *all waiting multiactions*. Obviously, we have $\mathcal{DL} = \mathcal{IL} \cup \mathcal{WL}$.

Let us note that the same multiaction $\alpha \in \mathcal{L}$ may have different probabilities, (fixed) delays and weights in the same specification. An *activity* is a stochastic or a deterministic multiaction. Let $\mathcal{SDL} = \mathcal{SL} \cup \mathcal{DL} = \mathcal{SL} \cup \mathcal{IL} \cup \mathcal{WL}$ be the set of *all activities*. The *alphabet* of an activity $(\alpha, \kappa) \in \mathcal{SDL}$ is defined as $\mathcal{A}(\alpha, \kappa) = \mathcal{A}(\alpha)$. The *alphabet* of a multiset of activities $\Upsilon \in \mathbb{N}_{fin}^{\mathcal{SDL}}$ is defined as $\mathcal{A}(\Upsilon) = \cup_{(\alpha,\kappa)\in\Upsilon}\mathcal{A}(\alpha)$. For an activity $(\alpha, \kappa) \in \mathcal{SDL}$, we define its *multiaction part* as $\mathcal{L}(\alpha, \kappa) = \alpha$ and its *probability* or *weight part* as $\Omega(\alpha, \kappa) = \kappa$ if $\kappa \in (0; 1)$; or $\Omega(\alpha, \kappa) = l$ if $\kappa = \natural_l^\theta,\ \theta \in \mathbb{N},\ l \in \mathbb{R}_{>0}$.

Activities are combined into formulas (process expressions) by the next operations:

;	:	*sequence*;
[]	:	*choice*;
$\|$	:	*parallelism*;
$[f]$	:	*relabeling* of actions;
rs	:	*restriction* over a single action;
sy	:	*synchronization* on an action and its conjugate;
$[**]$	:	*iteration* with three arguments: initialization, body and termination.

Sequence (sequential composition) and choice (composition) have a standard interpretation, like in other process algebras, but parallelism (parallel composition) does not include synchronization, unlike the corresponding operation in CCS [37].

Relabeling functions $f : \mathcal{A} \to \mathcal{A}$ are bijections preserving conjugates, i.e. $\forall x \in \mathcal{A}\ f(\hat{x}) = \widehat{f(x)}$. Relabeling is extended to multiactions in the usual way: for $\alpha \in \mathcal{L}$ we define $f(\alpha) = \sum_{x\in\alpha} f(x)$. Relabeling is extended to activities as follows: for $(\alpha, \kappa) \in \mathcal{SDL}$, we define $f(\alpha, \kappa) = (f(\alpha), \kappa)$. Relabeling is extended to the

multisets of activities as follows: for $\Upsilon \in \mathbb{N}_{fin}^{\mathcal{SDL}}$ we define $f(\Upsilon) = \sum_{(\alpha,\kappa)\in\Upsilon}(f(\alpha),\kappa)$. Remember that sums are considered with the multiplicity when applied to multisets: for example, $f(\alpha) = \sum_{x\in\alpha} f(x) = \sum_{x\in\mathcal{A}} \alpha(x)f(x)$.

Restriction over an elementary action $a \in Act$ means that, for a given expression, any process behaviour containing a or its conjugate $\hat{a}$ is not allowed.

Let $\alpha, \beta \in \mathcal{L}$ be two multiactions such that for some elementary action $a \in Act$ we have $a \in \alpha$ and $\hat{a} \in \beta$, or $\hat{a} \in \alpha$ and $a \in \beta$. Then, synchronization of α and β by a is defined as $(\alpha \oplus_a \beta)(x) = \begin{cases} \alpha(x) + \beta(x) - 1, & \text{if } x = a \text{ or } x = \hat{a}; \\ \alpha(x) + \beta(x), & \text{otherwise.} \end{cases}$
In other words, we require that $\alpha \oplus_a \beta = \alpha + \beta - \{a, \hat{a}\}$, i.e. we remove one exemplar of a and one exemplar of $\hat{a}$ from the multiset sum $\alpha + \beta$, since the synchronization of a and $\hat{a}$ produces $\emptyset$. Activities are synchronized with the use of their multiaction parts, i.e. the synchronization by a of two activities, whose multiaction parts α and β possess the properties mentioned above, results in the activity with the multiaction part $\alpha \oplus_a \beta$. We may synchronize activities of the same type only: either both stochastic multiactions or both deterministic ones *with the same delay*, since stochastic, waiting and immediate multiactions have different priorities, and diverse delays of waiting multiactions contradict their joint timing. Hence, the multiactions of different types cannot be executed together (note also that the execution of immediate multiactions takes no time, unlike that of waiting or stochastic ones). Synchronization by a means that, for a given expression with a process behaviour containing two concurrent activities that can be synchronized by a, there exists also the process behaviour that differs from the former only in that the two activities are replaced by the result of their synchronization.

In the iteration, the initialization subprocess is executed first, then the body is performed zero or more times, and finally, the termination subprocess is executed.

Static expressions specify the structure of processes, i.e. how activities are combined by operations in order to construct the composite process-algebraic formulas. As for the Petri net intuition, static expressions correspond to unmarked PNs. Remember that a marking is the allocation of tokens in the places of a PN and markings are used to describe dynamic behaviour of PNs in terms of transition firings.

We assume that every waiting multiaction has a countdown timer associated, whose value is the discrete time amount left till the moment when the waiting multiaction can be executed. Therefore, besides standard (unstamped) waiting multiactions in the form of $(\alpha, \natural_l^\theta) \in \mathcal{WL}$, a special case of the *stamped* waiting multiactions should be considered in the definition of static expressions. Each (time) stamped waiting multiaction in the form of $(\alpha, \natural_l^\theta)^\delta$ has an extra superscript $\delta \in \{1, \ldots, \theta\}$ assigned that specifies a time stamp indicating the *latest* value of the countdown timer associated with that multiaction. The standard waiting multiactions have no time stamps, to demonstrate irrelevance of the timer values for them (for example, their timers have not yet started or have already finished their operation). The notions of the alphabet, multiaction part, weight part for (the multisets of) stamped waiting multiactions are defined, respectively, like those for (the multisets of) unstamped waiting multiactions.

By reasons of simplicity, we do not assign the timer value superscripts δ to immediate multiactions, which are a special case of deterministic multiactions $(\alpha, \natural_l^\theta)$ with the delay $\theta = 0$ in the form of $(\alpha, \natural_l^0)$, since their timer values can only be equal to 0. Analogously, the superscript δ might be omitted for the waiting multiactions $(\alpha, \natural_l^\theta)$

with the delay $\theta = 1$ in the form of $(\alpha, \natural_l^1)$, since the corresponding timer can only have a single value 1. Nevertheless, to maintain syntactic uniformity among waiting multiactions, we leave the timer value superscripts for those that are 1-delayed.

Definition 2. *Let* $(\alpha, \kappa) \in \mathcal{SDL}$, $(\alpha, \natural_l^\theta) \in \mathcal{WL}$, $\delta \in \{1, \ldots, \theta\}$ *and* $a \in Act$. *A* static expression *of dtsdPBC is defined as*

$$E ::= \ (\alpha, \kappa) \mid (\alpha, \natural_l^\theta)^\delta \mid E; E \mid E[]E \mid E \| E \mid E[f] \mid E \ \mathsf{rs}\ a \mid E \ \mathsf{sy}\ a \mid [E * E * E].$$

Let *StatExpr* denote the set of *all static expressions* of dtsdPBC.

To make the grammar above unambiguous, one can add parentheses in the productions with binary operations: $(E; E)$, $(E[]E)$, $(E \| E)$. However, here and further we prefer the PBC approach and add them to resolve ambiguities only.

To avoid technical difficulties with the iteration operator, we should not allow any concurrency at the highest level of the second argument of iteration. This is not a severe restriction though, since we can always prefix parallel expressions by an activity with the empty multiaction part. Relaxing the restriction can result in PNs which are not safe. Alternatively, we can use a different, safe, version of the iteration operator, but its net translation has six arguments. See also [8] for discussion on this subject. Remember that a PN is *n-bounded* ($n \in \mathbb{N}$) if for all its reachable (from the initial marking by the sequences of transition firings) markings there are at most n tokens in every place, and a PN is *safe* if it is 1-bounded.

Definition 3. *Let* $(\alpha, \kappa) \in \mathcal{SDL}$, $(\alpha, \natural_l^\theta) \in \mathcal{WL}$, $\delta \in \{1, \ldots, \theta\}$ *and* $a \in Act$. *A* regular static expression *of dtsdPBC is defined as*

$$E ::= \ (\alpha, \kappa) \mid (\alpha, \natural_l^\theta)^\delta \mid E; E \mid E[]E \mid E \| E \mid E[f] \mid E \ \mathsf{rs}\ a \mid E \ \mathsf{sy}\ a \mid [E * D * E],$$
$$\textit{where } D ::= \ (\alpha, \kappa) \mid (\alpha, \natural_l^\theta)^\delta \mid D; E \mid D[]D \mid D[f] \mid D \ \mathsf{rs}\ a \mid D \ \mathsf{sy}\ a \mid [D * D * E].$$

Let *RegStatExpr* denote the set of *all regular static expressions* of dtsdPBC.

Let E be a regular static expression. The *underlying timer-free regular static expression* $\lfloor E$ of E is obtained by removing from it all timer value superscripts.

The set of *all stochastic multiactions (from the syntax) of* E is $\mathcal{SL}(E) = \{(\alpha, \rho) \mid (\alpha, \rho)$ is a subexpression of $E\}$. The set of *all immediate multiactions (from the syntax) of* E is $\mathcal{IL}(E) = \{(\alpha, \natural_l^0) \mid (\alpha, \natural_l^0)$ is a subexpression of $E\}$. The set of *all waiting multiactions (from the syntax) of* E is $\mathcal{WL}(E) = \{(\alpha, \natural_l^\theta) \mid (\alpha, \natural_l^\theta)$ or $(\alpha, \natural_l^\theta)^\delta$ is a subexpression of E for $\delta \in \{1, \ldots, \theta\}\}$. Thus, the set of *all deterministic multiactions (from the syntax) of* E is $\mathcal{DL}(E) = \mathcal{IL}(E) \cup \mathcal{WL}(E)$ and the set of *all activities (from the syntax) of* E is $\mathcal{SDL}(E) = \mathcal{SL}(E) \cup \mathcal{DL}(E) = \mathcal{SL}(E) \cup \mathcal{IL}(E) \cup \mathcal{WL}(E)$.

Dynamic expressions specify the states of processes, i.e. some particular stages of the process behaviour. As for the Petri net intuition, dynamic expressions correspond to marked PNs. Dynamic expressions are obtained from static ones, by annotating them with upper or lower bars which specify the active components of the system at the current moment of time. The dynamic expression with upper bar (the overlined one) $\overline{E}$ denotes the *initial*, and that with lower bar (the underlined one) $\underline{E}$ denotes the *final* state of the process specified by a static expression E.

For every overlined stamped waiting multiaction in the form of $\overline{(\alpha, \natural_l^\theta)^\delta}$, the superscript $\delta \in \{1, \ldots, \theta\}$ specifies the *current* value of the *running* countdown timer associated with the waiting multiaction. That decreasing discrete timer is started with the *initial* value θ (equal to the delay of the waiting multiaction) at

the moment when the waiting multiaction becomes overlined. Then such a newly overlined stamped waiting multiaction $\overline{(\alpha, \natural_l^\theta)^\theta}$ may be seen similar to the freshly overlined unstamped waiting multiaction $\overline{(\alpha, \natural_l^\theta)}$. Such similarity will be captured by the structural equivalence, to be defined later.

While the stamped waiting multiaction stays overlined with the process execution, the timer decrements by one discrete time unit with each global time tick until the timer value becomes 1. This means that one unit of time remains till execution of that multiaction (the remaining time to execute, RTE, equals one) that should follow in the next moment with probability 1, in case the stamped waiting multiaction is still overlined, there are no conflicting with it waiting multiactions, whose RTEs equal to one, and it is not affected by restriction. An activity is affected by restriction, if it is within the scope of a restriction operation with the argument action, such that it or its conjugate is contained in the multiaction part of that activity.

Definition 4. *Let $E \in StatExpr$ and $a \in Act$. A* dynamic expression *of dtsdPBC is defined as*

$$G ::= \overline{E} \mid \underline{E} \mid G; E \mid E; G \mid G[]E \mid E[]G \mid G\|G \mid G[f] \mid G \text{ rs } a \mid G \text{ sy } a \mid [G * E * E] \mid [E * G * E] \mid [E * E * G].$$

Let $DynExpr$ denote the set of *all dynamic expressions* of dtsdPBC.

Let G be a dynamic expression. The *underlying static (line-free) expression* $\lfloor G \rfloor$ of G is obtained by removing from it all upper and lower bars. Note that if the underlying static expression of a dynamic one is not regular, the corresponding PN can be non-safe (though, it is 2-bounded in the worst case [8]).

Definition 5. *A dynamic expression G is* regular *if its underlying static expression $\lfloor G \rfloor$ is regular.*

Let $RegDynExpr$ denote the set of *all regular dynamic expressions* of dtsdPBC.

Let G be a regular dynamic expression. The *underlying timer-free regular dynamic expression* $\lfloor\!\lfloor G$ of G is obtained by removing from it all timer value superscripts.

The set of *all stochastic (immediate or waiting, respectively) multiactions (from the syntax) of* G is defined as $\mathcal{SL}(G) = \mathcal{SL}(\lfloor G \rfloor)$ ($\mathcal{IL}(G) = \mathcal{IL}(\lfloor G \rfloor)$ or $\mathcal{WL}(G) = \mathcal{WL}(\lfloor G \rfloor)$, respectively). Thus, the set of *all deterministic multiactions (from the syntax) of* G is $\mathcal{DL}(G) = \mathcal{IL}(G) \cup \mathcal{WL}(G)$ and the set of *all activities (from the syntax) of* G is $\mathcal{SDL}(G) = \mathcal{SL}(G) \cup \mathcal{DL}(G) = \mathcal{SL}(G) \cup \mathcal{IL}(G) \cup \mathcal{WL}(G)$.

3. Operational semantics

In this section, we define the operational semantics via labeled transition systems.

3.1. **Inaction rules.** The inaction rules for dynamic expressions describe their structural transformations in the form of $G \Rightarrow \widetilde{G}$ which do not change the states of the specified processes. The goal of those syntactic transformations is to obtain the well-structured resulting expressions called operative ones to which no inaction rules can be further applied. The application of an inaction rule to a dynamic expression does not lead to any discrete time tick or any transition firing in the corresponding PN, hence, its current marking stays unchanged.

Thus, an application of every inaction rule does not require any delay, i.e. the dynamic expression transformation described by the rule is accomplished instantly.

In Table 1, we define inaction rules for regular dynamic expressions being overlined and underlined static ones. In this table, $(\alpha, \natural_l^\theta) \in \mathcal{WL}$, $\delta \in \{1, \ldots, \theta\}$, $E, F, K \in RegStatExpr$ and $a \in Act$. The first inaction rule suggests that the timer value of each newly overlined waiting multiaction is set to the delay of it.

TABLE 1. Inaction rules for overlined and underlined regular static expressions

$\overline{(\alpha, \natural_l^\theta)} \Rightarrow \overline{(\alpha, \natural_l^\theta)^\theta}$	$\overline{E; F} \Rightarrow \overline{E}; F$	$\underline{E}; F \Rightarrow E; \overline{F}$
$E; \underline{F} \Rightarrow \underline{E; F}$	$\overline{E[]F} \Rightarrow \overline{E}[]F$	$\overline{E[]F} \Rightarrow E[]\overline{F}$
$\underline{E}[]F \Rightarrow \underline{E[]F}$	$E[]\underline{F} \Rightarrow \underline{E[]F}$	$\overline{E\|F} \Rightarrow \overline{E}\|\overline{F}$
$\underline{E}\|\underline{F} \Rightarrow \underline{E\|F}$	$\overline{E[f]} \Rightarrow \overline{E}[f]$	$\underline{E}[f] \Rightarrow \underline{E[f]}$
$\overline{E \text{ rs } a} \Rightarrow \overline{E} \text{ rs } a$	$\underline{E} \text{ rs } a \Rightarrow \underline{E \text{ rs } a}$	$\overline{E \text{ sy } a} \Rightarrow \overline{E} \text{ sy } a$
$\underline{E} \text{ sy } a \Rightarrow \underline{E \text{ sy } a}$	$\overline{[E * F * K]} \Rightarrow [\overline{E} * F * K]$	$[\underline{E} * F * K] \Rightarrow [E * \overline{F} * K]$
$[E * \underline{F} * K] \Rightarrow [E * \overline{F} * K]$	$[E * \underline{F} * K] \Rightarrow [E * F * \overline{K}]$	$[E * F * \underline{K}] \Rightarrow \underline{[E * F * K]}$

In Table 2, we introduce inaction rules for regular dynamic expressions in the arbitrary form. In this table, $E, F \in RegStatExpr$, $G, H, \widetilde{G}, \widetilde{H} \in RegDynExpr$ and $a \in Act$. By reason of brevity, two distinct inaction rules with the same premises are collated in some cases, resulting in the inaction rules with double conclusion.

TABLE 2. Inaction rules for arbitrary regular dynamic expressions

$$\frac{G \Rightarrow \widetilde{G},\ \circ \in \{;, []\}}{G \circ E \Rightarrow \widetilde{G} \circ E,\ E \circ G \Rightarrow E \circ \widetilde{G}} \qquad \frac{G \Rightarrow \widetilde{G}}{G\|H \Rightarrow \widetilde{G}\|H,\ H\|G \Rightarrow H\|\widetilde{G}}$$

$$\frac{G \Rightarrow \widetilde{G}}{G[f] \Rightarrow \widetilde{G}[f]} \qquad \frac{G \Rightarrow \widetilde{G},\ \circ \in \{\text{rs}, \text{sy}\}}{G \circ a \Rightarrow \widetilde{G} \circ a} \qquad \frac{G \Rightarrow \widetilde{G}}{[G * E * F] \Rightarrow [\widetilde{G} * E * F]}$$

$$\frac{G \Rightarrow \widetilde{G}}{[E * G * F] \Rightarrow [E * \widetilde{G} * F]} \qquad \frac{G \Rightarrow \widetilde{G}}{[E * F * G] \Rightarrow [E * F * \widetilde{G}]}$$

Example 1. *Let* $E = (\{a\}, \natural_1^3)[](\{b\}, \frac{1}{3})$. *The following inferences by the inaction rules are possible from* $\overline{E}$*:*

$$\overline{(\{a\}, \natural_1^3)[](\{b\}, \tfrac{1}{3})} \Rightarrow \overline{(\{a\}, \natural_1^3)}[](\{b\}, \tfrac{1}{3}) \Rightarrow \overline{(\{a\}, \natural_1^3)^3}[](\{b\}, \tfrac{1}{3}),$$
$$\overline{(\{a\}, \natural_1^3)[](\{b\}, \tfrac{1}{3})} \Rightarrow (\{a\}, \natural_1^3)[]\overline{(\{b\}, \tfrac{1}{3})}.$$

Definition 6. *A regular dynamic expression* G *is* operative *if no inaction rule can be applied to it.*

Let $OpRegDynExpr$ denote the set of *all operative regular dynamic expressions* of dtsdPBC. Note that any dynamic expression can be always transformed into a (not necessarily unique) operative one by using the inaction rules.

In the following, we consider regular expressions only and omit the word "regular".

Definition 7. *The relation* $\approx = (\Rightarrow \cup \Leftarrow)^*$ *is a* structural equivalence *of dynamic expressions in dtsdPBC, where* * *is the reflexive and transitive closure operation. Thus, two dynamic expressions* G *and* G' *are* structurally equivalent*, denoted by* $G \approx G'$*, if they can be reached from each other by applying the inaction rules in a forward or a backward direction.*

Let X be some set. We denote the Cartesian product $X \times X$ by X^2. Let $\mathcal{E} \subseteq X^2$ be an equivalence relation on X. Then the *equivalence class* (with respect to $\mathcal{E}$) of an element $x \in X$ is defined by $[x]_{\mathcal{E}} = \{y \in X \mid (x,y) \in \mathcal{E}\}$. The equivalence $\mathcal{E}$ partitions X into the *set of equivalence classes* $X/_{\mathcal{E}} = \{[x]_{\mathcal{E}} \mid x \in X\}$.

Let G be a dynamic expression. Then $[G]_{\approx} = \{H \mid G \approx H\}$ is the equivalence class of G with respect to the structural equivalence, called the (corresponding) *state*. Next, G is an *initial* dynamic expression, denoted by $init(G)$, if $\exists E \in RegStatExpr\ G \in [\overline{E}]_{\approx}$. Further, G is a *final* dynamic expression, denoted by $final(G)$, if $\exists E \in RegStatExpr\ G \in [\underline{E}]_{\approx}$.

Example 2. *Let E be from Example 1. We have $init(\overline{E})$ and* $[\overline{E}]_{\approx} = \{\overline{(\{a\},\natural_1^3)[](\{b\},\frac{1}{3})}, \overline{(\{a\},\natural_1^3)[](\{b\},\frac{1}{3})}(\{b\},\frac{1}{3}), (\{a\},\natural_1^3)[]\overline{(\{b\},\frac{1}{3})}, \overline{(\{a\},\natural_1^3)^3[](\{b\},\frac{1}{3})}, (\{a\},\natural_1^3)^3[]\overline{(\{b\},\frac{1}{3})}, \{\overline{(\{a\},\natural_1^3)^3[](\{b\},\frac{1}{3})}\}$. *Then* $[\overline{E}]_{\approx} \cap OpRegDynExpr = \{(\{a\},\natural_1^3)[]\overline{(\{b\},\frac{1}{3})}, \overline{(\{a\},\natural_1^3)^3}[](\{b\},\frac{1}{3}), (\{a\},\natural_1^3)^3[]\overline{(\{b\},\frac{1}{3})}\}$.

Let G be a dynamic expression and $s = [G]_{\approx}$. The set of *all enabled stochastic multiactions of s* is $EnaSto(s) = \{(\alpha,\rho) \in \mathcal{SL} \mid \exists H \in s \cap OpRegDynExpr\ \overline{(\alpha,\rho)}$ is a subexpression of $H\}$, i.e. it consists of all stochastic multiactions that, being overlined, are the subexpressions of some operative dynamic expression from the state s. Analogously, the set of *all enabled immediate multiactions of s* is $EnaImm(s) = \{(\alpha,\natural_l^0) \in \mathcal{IL} \mid \exists H \in s \cap OpRegDynExpr\ \overline{(\alpha,\natural_l^0)}$ is a subexpression of $H\}$. The set of *all enabled waiting multiactions of s* is $EnaWait(s) = \{(\alpha,\natural_l^\theta) \in \mathcal{WL} \mid \exists H \in s \cap OpRegDynExpr\ \overline{(\alpha,\natural_l^\theta)^\delta},\ \delta \in \{1,\ldots,\theta\},$ is a subexpression of $H\}$, i.e. it consists of all waiting multiactions that, being superscribed with the values of their timers and overlined, are the subexpressions of some operative dynamic expression from the state s. The set of *all newly enabled waiting multiactions of s* is $EnaWaitNew(s) = \{(\alpha,\natural_l^\theta) \in \mathcal{WL} \mid \exists H \in s \cap OpRegDynExpr\ \overline{(\alpha,\natural_l^\theta)^\theta}$ is a subexpression of $H\}$, i.e. it consists of all waiting multiactions that, being superscribed with the initial values of their timers (delays of those waiting multiactions) and overlined, are the subexpressions of some operative dynamic expression from the state s.

Thus, the set of *all enabled deterministic multiactions of s* is $EnaDet(s) = EnaImm(s) \cup EnaWait(s)$ and the set of *all enabled activities of s* is $Ena(s) = EnaSto(s) \cup EnaDet(s) = EnaSto(s) \cup EnaImm(s) \cup EnaWait(s)$. Then $Ena(s) = Ena([G]_{\approx})$ is an algebraic analogue of the set of all transitions enabled at the initial marking of the PN corresponding to G. Note that the activities, resulted from synchronization, are not present explicitly in the syntax of the dynamic expressions. Nevertheless, their enabledness status can be recovered by observing that of the pair of synchronized activities from the syntax (they both should be enabled for enabling their synchronous product), even if they are affected by restriction after the synchronization.

Example 3. *Let E be from Example 1. Then we have $EnaSto([\overline{E}]_{\approx}) = \{(\{b\},\frac{1}{3})\}$, $EnaImm([\overline{E}]_{\approx}) = \emptyset$ and $EnaWait([\overline{E}]_{\approx}) = EnaWaitNew([\overline{E}]_{\approx}) = \{(\{a\},\natural_1^3)\}$, hence, $Ena([\overline{E}]_{\approx}) = \{(\{a\},\natural_1^3),(\{b\},\frac{1}{3})\}$.*

Definition 8. *An operative dynamic expression G is* saturated (with the values of timers)*, if each enabled waiting multiaction of $[G]_{\approx}$, being (certainly) superscribed with the value of its timer and possibly overlined, is the subexpression of G.*

Let $SaOpRegDynExpr$ denote the set of *all saturated operative dynamic expressions* of dtsdPBC.

Proposition 1. *Any operative dynamic expression can be always transformed into the saturated one by applying the inaction rules in a forward or a backward direction.*

Proof. Let G be a dynamic expression, $(\alpha, \natural_l^\theta) \in EnaWait([G]_\approx)$ and there exists $H \in [G]_\approx \cap OpRegDynExpr$ that contains a subexpression $\overline{(\alpha, \natural_l^\theta)^\delta}$, $\delta \in \{1, \ldots, \theta - 1\}$. Then all operative dynamic expressions from $[G]_\approx \cap OpRegDynExpr$ contain a subexpression $\overline{(\alpha, \natural_l^\theta)^\delta}$ or $(\alpha, \natural_l^\theta)^\delta$, i.e. the (possibly overlined) enabled waiting multiaction $(\alpha, \natural_l^\theta)$ with the (non-initial) timer value superscript $\delta \leq \theta - 1$. Note that the timer value superscript δ is the same for all such structurally equivalent operative dynamic expressions. Indeed, all inaction rules, besides the first one, do not change the values of timers, but those rules just modify the overlines and underlines of dynamic expressions. The first inaction rule just sets up the timer of each overlined waiting multiaction $\overline{(\alpha, \natural_l^\theta)}$ with the initial value $\delta = \theta$, equal to the delay of that waiting multiaction, as follows: $\overline{(\alpha, \natural_l^\theta)^\theta}$. Then the remaining inaction rules can shift out the overline of that enabled waiting multiaction before setting up its timer, which results in a non-overlined enabled waiting multiaction without timer value superscript $(\alpha, \natural_l^\theta)$. Thus, for $(\alpha, \natural_l^\theta) \in EnaWait([G]_\approx)$, it may happen that $\overline{(\alpha, \natural_l^\theta)^\theta}$ a subexpression of some $H \in [G]_\approx \cap OpRegDynExpr$ and $(\alpha, \natural_l^\theta)$ is a subexpression of a different $H' \in [G]_\approx \cap OpRegDynExpr$.

Let now G be an operative dynamic expression that is not saturated. By the arguments above, the saturation can be violated only if G contains as a subexpression at least one newly enabled waiting multiaction $(\alpha, \natural_l^\theta)$ of $[G]_\approx$ that is not superscribed with the timer value. By the definition of the new-enabling, there exists $H \in [G]_\approx \cap OpRegDynExpr$ such that $\overline{(\alpha, \natural_l^\theta)^\theta}$ is a subexpression of H. Since $G \approx H$, there is a sequence of the inaction rules applications (in a forward or a backward direction) that transforms G into H. Then the reverse sequence transforms H into G. Let us remove from that reverse sequence the following backward application of the first inaction rule: $\overline{(\alpha, \natural_l^\theta)} \Leftarrow \overline{(\alpha, \natural_l^\theta)^\theta}$. Then such a reduced reverse sequence will turn H into $G_1 \in [G]_\approx \cap OpRegDynExpr$, by replacing $(\alpha, \natural_l^\theta)$ in G with $(\alpha, \natural_l^\theta)^\theta$.

Let us start from G_1 and apply the above procedure to the remaining not superscribed with the timer values newly enabled waiting multiactions of $[G]_\approx$ (which are also those of such kind of $[G_1]_\approx$). After repeated application of the mentioned procedure for all $n \geq 1$ non-superscribed newly enabled waiting multiactions of G, we shall get from it the saturated operative dynamic expression $G_n = \widetilde{G} \in [G]_\approx \cap OpRegDynExpr$. Note that the presented transformation of G into $\widetilde{G}$ does not change the enabling, since it does not change any overlines or underlines in the syntax of the traversed operative dynamic expressions, but only iteratively assigns the timer value superscripts to all newly enabled waiting multiactions of G. Hence, $EnaWait([G]_\approx) = EnaWait([G_1]_\approx) = \cdots = EnaWait([G_n]_\approx) = EnaWait([\widetilde{G}]_\approx)$. □

Thus, any dynamic expression can be always transformed into a (not necessarily unique) saturated operative one by (possibly reverse) applying the inaction rules.

Example 4. *Let E be from Example 1. We have* $[\overline{E}]_\approx \cap SaOpRegDynExpr = \{\overline{(\{a\}, \natural_1^3)^3}[](\{b\}, \frac{1}{3}),\ (\{a\}, \natural_1^3)^3[]\overline{(\{b\}, \frac{1}{3})}\}$. *Consider the sequence of inaction rules,*

applied (in a forward or a backward direction) in the following transformation of a non-saturated $G \in [\overline{E}]_{\approx} \cap OpRegDynExpr$ *with the non-superscribed with the timer value (unstamped) enabled waiting multiaction* $(\{a\}, \natural_1^3)$ *into (a saturated)* $H \in [\overline{E}]_{\approx} \cap OpRegDynExpr$*, in which* $(\{a\}, \natural_1^3)$ *is stamped:*

$$G = (\{a\}, \natural_1^3)[]\overline{(\{b\}, \tfrac{1}{3})} \approx \overline{(\{a\}, \natural_1^3)[](\{b\}, \tfrac{1}{3})} \approx \overline{(\{a\}, \natural_1^3)}[](\{b\}, \tfrac{1}{3}) \approx \overline{(\{a\}, \natural_1^3)^3}[](\{b\}, \tfrac{1}{3}) = H.$$

The reduced reverse sequence of inaction rules induces the following transformations of H *that result in a saturated* $G_1 = \widetilde{G} \in [\overline{E}]_{\approx} \cap OpRegDynExpr$*, in which* $(\{a\}, \natural_1^3)$ *is stamped:*

$$H = \overline{(\{a\}, \natural_1^3)^3}[](\{b\}, \tfrac{1}{3}) \approx \overline{(\{a\}, \natural_1^3)^3[](\{b\}, \tfrac{1}{3})} \approx (\{a\}, \natural_1^3)^3[]\overline{(\{b\}, \tfrac{1}{3})} = G_1 = \widetilde{G}.$$

Let G be a saturated operative dynamic expression. Then $\circlearrowleft G$ is written for the *timer decrement* operator $\circlearrowleft$, applied to G. It denotes a saturated operative dynamic expression, obtained from G via decrementing by one time unit all greater than 1 values of the timers associated with all (if any) stamped waiting multiactions from the syntax of G. Thus, each such stamped waiting multiaction changes its timer value from δ in G to $\max\{1, \delta - 1\}$ in $\circlearrowleft G$, where $\delta \in \mathbb{N}_{\geq 1}$. More formally, the timer decrement operator affects the (possibly overlined) stamped waiting multiactions being the subexpressions of G as follows. The overlined stamped waiting multiaction $\overline{(\alpha, \natural_l^\theta)^\delta}$ is replaced with $\overline{(\alpha, \natural_l^\theta)^{\max\{1,\delta-1\}}}$ while the stamped waiting multiaction without overline or underline $(\alpha, \natural_l^\theta)^\delta$ is replaced with $(\alpha, \natural_l^\theta)^{\max\{1,\delta-1\}}$.

Note that when $\delta = 1$, we have $\max\{1, \delta - 1\} = \max\{1, 0\} = 1$, hence, the timer value $\delta = 1$ may remain unchanged for a stamped waiting multiaction that is not executed by some reason at the next time moment, but stays stamped. For example, that stamped waiting multiaction may be affected by restriction. If the timer values cannot be decremented with a time tick for all stamped waiting multiactions (if any) from G then $\circlearrowleft G = G$ and we obtain so-called *empty loop* transition, defined later.

Observe that the timer decrement operator keeps stamping of the waiting multiactions, since it does not change any overlines or underlines, but it may only decrease their timer values, so that the stamped waiting multiactions stay stamped (with their timer values, possibly decremented by one).

Example 5. *Let* E *be from Example 1. We have* $Ena([\overline{E}]_{\approx}) = \{(\{a\}, \natural_1^3), (\{b\}, \frac{1}{3})\}$ *and* $Ena([\overline{E}]_{\approx}) \cap \mathcal{WL} = \{(\{a\}, \natural_1^3)\}$*. The following one time unit timer decrements are possible from the saturated operative dynamic expressions belonging to* $[\overline{E}]_{\approx}$*:*

$$\circlearrowleft(\overline{(\{a\}, \natural_1^3)^3}[](\{b\}, \tfrac{1}{3})) = \overline{(\{a\}, \natural_1^3)^2}[](\{b\}, \tfrac{1}{3}),$$
$$\circlearrowleft((\{a\}, \natural_1^3)^3[]\overline{(\{b\}, \tfrac{1}{3})}) = (\{a\}, \natural_1^3)^2[]\overline{(\{b\}, \tfrac{1}{3})}.$$

3.2. **Action and empty move rules.** The action rules are applied when some activities are executed. With these rules we capture the prioritization among different types of multiactions. We also have the empty move rule, used to capture a delay of one discrete time unit when no immediate or waiting multiactions are executable. In this case, the empty multiset of activities is executed. The action and empty move rules will be used later to determine all multisets of activities which can be executed from the structural equivalence class of every dynamic expression (i.e.

from the state of the corresponding process). This information together with that about probabilities or delays and weights of the activities to be executed from the current process state will be used to calculate the probabilities of such executions.

The action rules with stochastic (immediate or waiting, respectively) multiactions describe dynamic expression transformations in the form of $G \stackrel{\Gamma}{\to} \widetilde{G}$ ($G \stackrel{I}{\to} \widetilde{G}$ or $G \stackrel{W}{\to} \widetilde{G}$, respectively) due to execution of non-empty multisets Γ of stochastic (I of immediate or W of waiting, respectively) multiactions. The rules represent possible state changes of the specified processes when some non-empty multisets of stochastic (immediate or waiting, respectively) multiactions are executed. The application of an action rule with stochastic (immediate or waiting, respectively) multiactions to a dynamic expression leads in the corresponding PN to a discrete time tick at which some stochastic or waiting transitions fire (or to the instantaneous firing of some immediate transitions) and possible change of the current marking. The current marking stays unchanged only if there is a self-loop produced by the iterative execution of a non-empty multiset, which must be one-element, i.e. a single stochastic (immediate or waiting, respectively) multiaction. The reason is the regularity requirement that allows no concurrency at the highest level of the second argument of iteration.

The empty move rule (applicable only when no immediate or waiting multiactions can be executed from the current state) describes dynamic expression transformations in the form of $G \stackrel{\emptyset}{\to} \circlearrowleft G$, called the *empty moves*, due to execution of the empty multiset of activities at a discrete time tick. When no timer values are decremented within G with the empty multiset execution at the next moment (for example, if G contains no stamped waiting multiactions), we have $\circlearrowleft G = G$. In such a case, the empty move from G is in the form of $G \stackrel{\emptyset}{\to} G$, called the *empty loop*. The application of the empty move rule to a dynamic expression leads to a discrete time tick in the corresponding PN at which no transitions fire and the current marking is not changed, but the timer values of the waiting transitions enabled at the marking (if any) are decremented by one. This is a new rule that has no prototype among inaction rules of PBC, since it represents a time delay, but no notion of time exists in PBC. The PBC rule $G \stackrel{\emptyset}{\to} G$ from [9, 8] in our setting would correspond to the rule $G \Rightarrow G$ that describes staying in the current state when no time elapses. Since we do not need the latter rule to transform dynamic expressions into operative ones and it can destroy the definition of operative expressions, we do not have it in dtsdPBC.

Thus, an application of every action rule with stochastic or waiting multiactions or the empty move rule requires one discrete time unit delay, i.e. the execution of a (possibly empty) multiset of stochastic or (non-empty) multiset of waiting multiactions leading to the dynamic expression transformation described by the rule is accomplished instantly after one time unit. An application of every action rule with immediate multiactions does not take any time, i.e. the execution of a (non-empty) multiset of immediate multiactions is accomplished instantly at the current moment.

Note that expressions of dtsdPBC can contain identical activities. To avoid technical difficulties, such as the proper calculation of the state change probabilities for multiple transitions, we can always enumerate coinciding activities from left to right in the syntax of expressions. The new activities, resulted from synchronization will be annotated with concatenation of numberings of the activities they come

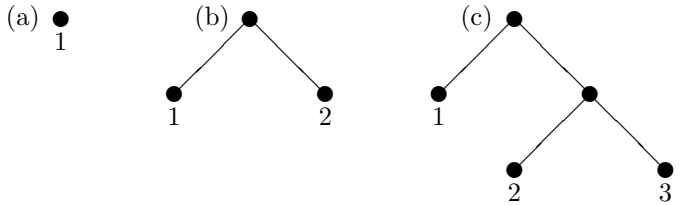


FIG. 1. The binary trees encoded with the numberings 1, (1)(2) and (1)((2)(3))

from, hence, the numbering should have a tree structure to reflect the effect of multiple synchronizations. We now define the numbering which encodes a binary tree with the leaves labeled by natural numbers.

Definition 9. *The* numbering *of expressions is* $\iota ::= \ n \mid (\iota)(\iota)$*, where* $n \in \mathbb{N}$*.*

Let Num denote the set of *all numberings* of expressions.

Example 6. *The numbering* 1 *encodes the binary tree in Figure 1(a) with the root labeled by* 1*. The numbering* (1)(2) *corresponds to the binary tree in Figure 1(b) without internal nodes and with two leaves labeled by* 1 *and* 2*. The numbering* (1)((2)(3)) *represents the binary tree Figure 1(c) with one internal node, which is the root for the subtree* (2)(3)*, and three leaves labeled by* $1, 2$ *and* 3*.*

The new activities resulting from synchronizations in different orders should be considered up to permutation of their numbering. In this way, we shall recognize different instances of the same activity. If we compare the contents of different numberings, i.e. the sets of natural numbers in them, we shall identify the mentioned instances. The *content* of a numbering $\iota \in Num$ is

$$Cont(\iota) = \begin{cases} \{\iota\}, & \iota \in \mathbb{N}; \\ Cont(\iota_1) \cup Cont(\iota_2), & \iota = (\iota_1)(\iota_2). \end{cases}$$

After the enumeration, the multisets of activities from the expressions become the proper sets. In the following, we suppose that the identical activities are enumerated when needed to avoid ambiguity. This enumeration is considered to be implicit.

Definition 10. *Let* $G \in OpRegDynExpr$*. We define the* set of all non-empty multisets of activities which can be potentially executed from G*, denoted by* $Can(G)$*. Let* $(\alpha, \kappa) \in \mathcal{SDL},\ E, F \in RegStatExpr,\ H \in OpRegDynExpr$ *and* $a \in Act$*.*

(1) *If* $final(G)$ *then* $Can(G) = \emptyset$*.*
(2) *If* $G = \overline{(\alpha, \kappa)^{\delta}}$ *and* $\kappa = \natural_l^{\theta},\ \theta \in \mathbb{N}_{\geq 2},\ l \in \mathbb{R}_{>0},\ \delta \in \{2, \ldots, \theta\}$*, then* $Can(G) = \emptyset$*.*
(3) *If* $G = \overline{(\alpha, \kappa)}$ *and* $\kappa \in (0; 1)$ *or* $\kappa = \natural_l^0,\ l \in \mathbb{R}_{>0}$*, then* $Can(G) = \{\{(\alpha, \kappa)\}\}$*.*
(4) *If* $G = \overline{(\alpha, \kappa)^1}$ *and* $\kappa = \natural_l^{\theta},\ \theta \in \mathbb{N}_{\geq 1},\ l \in \mathbb{R}_{>0}$*, then* $Can(G) = \{\{(\alpha, \kappa)\}\}$*.*
(5) *If* $\Upsilon \in Can(G)$ *then* $\Upsilon \in Can(G \circ E),\ \Upsilon \in Can(E \circ G)\ (\circ \in \{;, []\})$, $\Upsilon \in Can(G \| H),\ \Upsilon \in Can(H \| G),\ f(\Upsilon) \in Can(G[f]),\ \Upsilon \in Can(G\ \mathsf{rs}\ a)$ *(when* $a, \hat{a} \notin \mathcal{A}(\Upsilon)$*),* $\Upsilon \in Can(G\ \mathsf{sy}\ a),\ \Upsilon \in Can([G * E * F])$, $\Upsilon \in Can([E * G * F]),\ \Upsilon \in Can([E * F * G])$.
(6) *If* $\Upsilon \in Can(G)$ *and* $\Xi \in Can(H)$ *then* $\Upsilon + \Xi \in Can(G \| H)$*.*
(7) *If* $\Upsilon \in Can(G\ \mathsf{sy}\ a)$ *and* $(\alpha, \kappa), (\beta, \lambda) \in \Upsilon$ *are different,* $a \in \alpha,\ \hat{a} \in \beta$*, then*
   (a) $(\Upsilon + \{(\alpha \oplus_a \beta, \kappa \cdot \lambda)\} - \{(\alpha, \kappa), (\beta, \lambda)\}) \in Can(G\ \mathsf{sy}\ a)$ *if* $\kappa, \lambda \in (0; 1)$*;*
   (b) $(\Upsilon + \{(\alpha \oplus_a \beta, \natural_{l+m}^{\theta})\} - \{(\alpha, \kappa), (\beta, \lambda)\}) \in Can(G\ \mathsf{sy}\ a)$ *if* $\kappa = \natural_l^{\theta}$, $\lambda = \natural_m^{\theta},\ \theta \in \mathbb{N},\ l, m \in \mathbb{R}_{>0}$*.*

When we synchronize the same multiset of activities in different orders, we obtain several activities with the same multiaction and probability or delay and weight parts, but with different numberings having the same content. Then we only consider a single one of the resulting activities to avoid introducing redundant ones.
The synchronization of stochastic multiactions $(\alpha,\rho)_1$ *and* $(\beta,\chi)_2$ *in different orders generates the activities* $(\alpha\oplus_a\beta,\rho\cdot\chi)_{(1)(2)}$ *and* $(\beta\oplus_a\alpha,\chi\cdot\rho)_{(2)(1)}$. *The synchronization of deterministic multiactions* $(\alpha,\natural_l^\theta)_1$ *and* $(\beta,\natural_m^\theta)_2$ *in different orders generates the activities* $(\alpha\oplus_a\beta,\natural_{l+m}^\theta)_{(1)(2)}$ *and* $(\beta\oplus_a\alpha,\natural_{m+l}^\theta)_{(2)(1)}$. *Since* $Cont((1)(2))=\{1,2\}=Cont((2)(1))$, *in both cases, only the first activity (symmetrically, the second one) resulting from synchronization appears in a multiset from* $Can(G\ \mathsf{sy}\ a)$.

If $\Upsilon\in Can(G)$ then by definition of $Can(G)$, $\forall\Xi\subseteq\Upsilon$, $\Xi\neq\emptyset$, we have $\Xi\in Can(G)$.

Let $G\in OpRegDynExpr$ and $Can(G)\neq\emptyset$. Obviously, if there are only stochastic (immediate or waiting, respectively) multiactions in the multisets from $Can(G)$ then these stochastic (immediate or waiting, respectively) multiactions can be executed from G. Otherwise, besides stochastic ones, there are also deterministic (immediate and/or waiting) multiactions in the multisets from $Can(G)$. By the note above, there are non-empty multisets of deterministic multiactions in $Can(G)$ as well, i.e. $\exists\Upsilon\in Can(G)\ \Upsilon\in\mathbb{N}_{fin}^{\mathcal{DL}}\setminus\{\emptyset\}$. In this case, no stochastic multiactions can be executed from G, even if $Can(G)$ contains non-empty multisets of stochastic multiactions, since deterministic multiactions have a priority over stochastic ones, and should be executed first. Further, if there are no stochastic, but both waiting and immediate multiactions in the multisets from $Can(G)$, then, analogously, no waiting multiactions can be executed from G, since immediate multiactions have a priority over waiting ones (besides that over stochastic ones).

When there are only waiting and, possibly, stochastic multiactions in the multisets from $Can(G)$ then, from above, only waiting ones can be executed from G. Then just *maximal* non-empty multisets of waiting multiactions can be executed from G, since all non-conflicting waiting multiactions cannot wait anymore and they should occur at the next time moment with probability 1. The next definition formalizes these requirements.

Definition 11. *Let* $G\in OpRegDynExpr$. *The* set of all non-empty multisets of activities which can be executed from G *is*

$$Now(G)=\begin{cases} Can(G)\cap\mathbb{N}_{fin}^{\mathcal{IL}}, & Can(G)\cap\mathbb{N}_{fin}^{\mathcal{IL}}\neq\emptyset;\\ \{W\in Can(G)\cap\mathbb{N}_{fin}^{\mathcal{WL}}\mid & (Can(G)\cap\mathbb{N}_{fin}^{\mathcal{IL}}=\emptyset)\wedge\\ \forall V\in Can(G)\cap\mathbb{N}_{fin}^{\mathcal{WL}}\ W\subseteq V\ \Rightarrow\ V=W\}, & (Can(G)\cap\mathbb{N}_{fin}^{\mathcal{WL}}\neq\emptyset);\\ Can(G), & otherwise. \end{cases}$$

Consider an operative dynamic expression $G\in OpRegDynExpr$. The expression G is *s-tangible (stochastically tangible)*, denoted by $stang(G)$, if $Now(G)\subseteq\mathbb{N}_{fin}^{\mathcal{SL}}\setminus\{\emptyset\}$. In particular, we have $stang(G)$, if $Now(G)=\emptyset$. The expression G is *w-tangible (waitingly tangible)*, denoted by $wtang(G)$, if $\emptyset\neq Now(G)\subseteq\mathbb{N}_{fin}^{\mathcal{WL}}\setminus\{\emptyset\}$. The expression G is *tangible*, denoted by $tang(G)$, if $stang(G)$ or $wtang(G)$, i.e. $Now(G)\subseteq(\mathbb{N}_{fin}^{\mathcal{SL}}\cup\mathbb{N}_{fin}^{\mathcal{WL}})\setminus\{\emptyset\}$. Again, we particularly have $tang(G)$, if $Now(G)=\emptyset$. Otherwise, the expression G is *vanishing*, denoted by $vanish(G)$, and in this case

$\emptyset \neq Now(G) \subseteq \mathbb{N}^{\mathcal{IL}}_{fin} \setminus \{\emptyset\}$. Note that the operative dynamic expressions from $[G]_{\approx}$ may have different types in general. The next example demonstrates two operative dynamic expressions H and H' with $H \approx H'$, such that $vanish(H)$, but $stang(H')$.

Example 7. *Let* $G = (\overline{(\{a\}, \natural^0_1)}[](\{b\}, \natural^0_2))\|\overline{(\{c\}, \frac{1}{2})}$ *and* $G' = ((\{a\}, \natural^0_1)[]\overline{(\{b\}, \natural^0_2)})\|\overline{(\{c\}, \frac{1}{2})}$. *Then* $G \approx G'$, *since* $G \Leftarrow G'' \Rightarrow G'$ *for* $G'' = \overline{((\{a\}, \natural^0_1)[](\{b\}, \natural^0_2))}\|(\{c\}, \frac{1}{2})$, *but* $Can(G) = \{\{(\{a\}, \natural^0_1)\}, \{(\{c\}, \frac{1}{2})\}, \{(\{a\}, \natural^0_1), (\{c\}, \frac{1}{2})\}\}$, $Can(G') = \{\{(\{b\}, \natural^0_2)\}, \{(\{c\}, \frac{1}{2})\}, \{(\{b\}, \natural^0_2), (\{c\}, \frac{1}{2})\}\}$, $Now(G) = \{\{(\{a\}, \natural^0_1)\}\}$, $Now(G') = \{\{(\{b\}, \natural^0_2)\}\}$. *Clearly, we have* $vanish(G)$ *and* $vanish(G')$. *The executions like that of* $\{(\{c\}, \frac{1}{2})\}$ *(and all multisets including it) from* G *and* G' *must be disabled using preconditions in the action rules, since immediate multiactions have a priority over stochastic ones, hence, the former are always executed first.*

Let $H = \overline{(\{a\}, \natural^0_1)}[](\{b\}, \frac{1}{2})$ *and* $H' = (\{a\}, \natural^0_1)[]\overline{(\{b\}, \frac{1}{2})}$. *Then* $H \approx H'$, *since* $H \Leftarrow H'' \Rightarrow H'$ *for* $H'' = \overline{(\{a\}, \natural^0_1)[](\{b\}, \frac{1}{2})}$, *but* $Can(H) = Now(H) = \{\{(\{a\}, \natural^0_1)\}\}$ *and* $Can(H') = Now(H') = \{\{(\{b\}, \frac{1}{2})\}\}$. *We have* $vanish(H)$, *but* $stang(H')$. *To make the action rules correct under structural equivalence, the executions like that of* $\{(\{b\}, \frac{1}{2})\}$ *from* H' *must be disabled using preconditions in the action rules, since immediate multiactions have a priority over stochastic ones, hence, the choices between them are always resolved in favour of the former.*

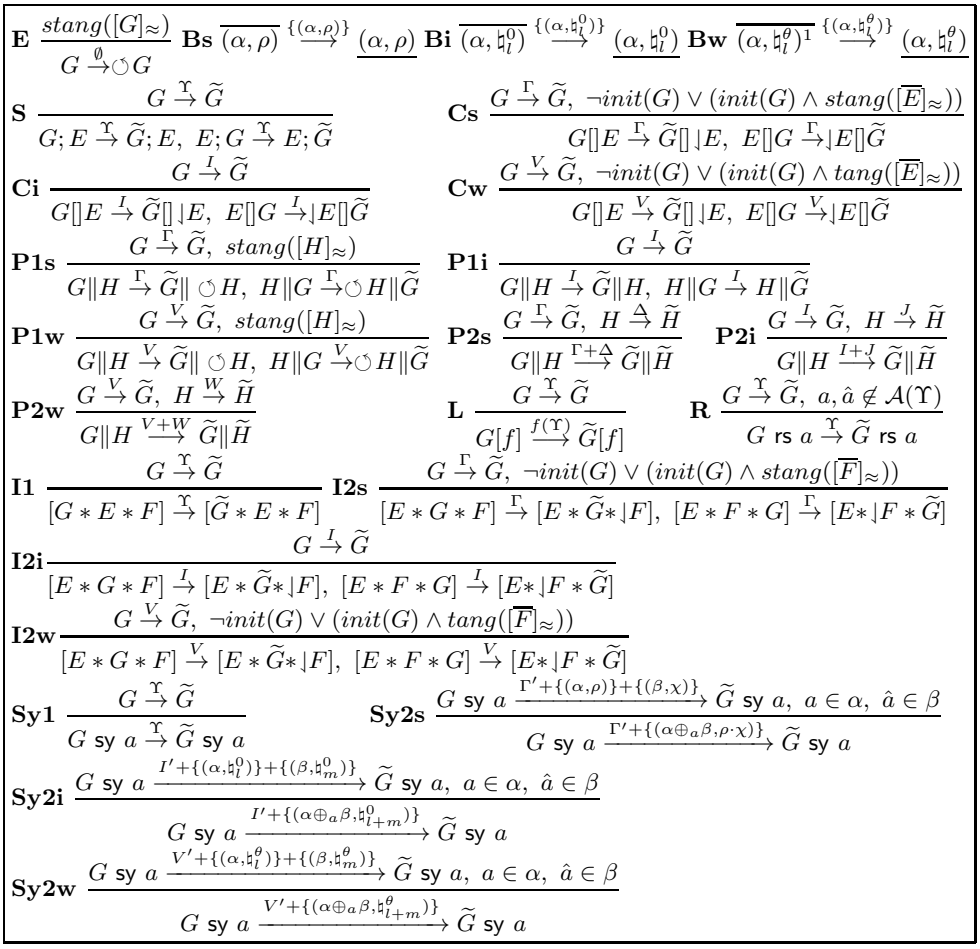
Let $G \in RegDynExpr$. We write $stang([G]_{\approx})$, if $\forall H \in [G]_{\approx} \cap OpRegDynExpr$ $stang(H)$. We write $wtang([G]_{\approx})$, if $\exists H \in [G]_{\approx} \cap OpRegDynExpr$ $wtang(H)$ and $\forall H' \in [G]_{\approx} \cap OpRegDynExpr$ $tang(H')$. We write $tang([G]_{\approx})$, if $stang([G]_{\approx})$ or $wtang([G]_{\approx})$. Otherwise, we write $vanish([G]_{\approx})$, and in this case $\exists H \in [G]_{\approx} \cap OpRegDynExpr$ $vanish(H)$.

In Table 3, we define the action and empty move rules. In the table, $(\alpha, \rho), (\beta, \chi) \in \mathcal{SL}$, $(\alpha, \natural^0_l), (\beta, \natural^0_m) \in \mathcal{IL}$ and $(\alpha, \natural^\theta_l), (\beta, \natural^\theta_m) \in \mathcal{WL}$. Further, $E, F \in RegStatExpr$, $G, H \in SatOpRegDynExpr$, $\widetilde{G}, \widetilde{H} \in RegDynExpr, G[]E, E[]G, [E * G * F], [E * F * G] \in SatOpRegDynExpr$ and $a \in Act$. Next, $\Gamma, \Delta \in \mathbb{N}^{\mathcal{SL}}_{fin} \setminus \{\emptyset\}$, $\Gamma' \in \mathbb{N}^{\mathcal{SL}}_{fin}$, $I, J \in \mathbb{N}^{\mathcal{IL}}_{fin} \setminus \{\emptyset\}$, $I' \in \mathbb{N}^{\mathcal{IL}}_{fin}$, $V, W \in \mathbb{N}^{\mathcal{WL}}_{fin} \setminus \{\emptyset\}$, $V' \in \mathbb{N}^{\mathcal{WL}}_{fin}$ and $\Upsilon \in \mathbb{N}^{\mathcal{SDL}}_{fin} \setminus \{\emptyset\}$.

We use the following abbreviations in the names of the rules from the table: "**E**" for "**E**mpty move", "**B**" for "**B**asis case", "**S**" for "**S**equence", "**C**" for "**C**hoice", "**P**" for "**P**arallel", "**L**" for "re**L**abeling", "**R**" for "**R**estriction", "**I**" for "**I**teraton" and "**Sy**" for "**Sy**nchronization". The first rule in the table is the empty move rule **E**. The other rules are the action rules, describing transformations of dynamic expressions, which are built using particular algebraic operations. If we cannot merge the rules with stochastic, immediate ans waiting multiactions in one rule for some operation then we get the coupled action rules. In such cases, the names of the action rules with stochastic multiactions have a suffix '**s**', those with immediate multiactions have a suffix '**i**', and those with waiting multiactions have a suffix '**w**'. To make presentation more compact, the action rules with double conclusion are combined from two distinct action rules with the same premises.

Almost all the rules in Table 3 (excepting **E**, **Bw**, **P2s**, **P2i**, **P2w**, **Sy2s**, **Sy2i** and **Sy2w**) resemble those of gsPBC, but the former correspond to execution of multisets of activities, not of single activities, as in the latter, and our rules have simpler preconditions (if any), since all immediate multiactions in dtsdPBC have the same priority level, unlike those of gsPBC.

TABLE 3. Action and empty move rules



The preconditions in rules **E**, **Cs**, **P1s**, and **I2s** are needed to ensure that (possibly empty) multisets of stochastic multiactions are executed only from *s-tangible* saturated operative dynamic expressions, such that all dynamic expressions structurally equivalent to them are s-tangible as well. For example, assuming that $stang([G]_{\approx})$ in rule **Cs**, if $init(G)$ then $G \approx \overline{F}$ for a static expression F and $G[]E \approx \overline{F}[]E \approx \overline{F[]E} \approx F[]\overline{E}$. Hence, it should be guaranteed $stang([F[]\overline{E}]_{\approx})$, which holds iff $stang([\overline{E}]_{\approx})$. The case $E[]G$ is treated similarly. Assuming that $stang([G]_{\approx})$ in rule **P1s**, it should be guaranteed $stang([G\|H]_{\approx})$ and $stang([H\|G]_{\approx})$, which holds iff $stang([H]_{\approx})$. The precondition in rule **I2s** is analogous to that in rule **Cs**.

Analogously, the preconditions in rules **Cw**, and **I2w** are needed to ensure that non-empty multisets of waiting multiactions are executed only from *w-tangible* saturated operative dynamic expressions, such that all dynamic expressions structurally equivalent to them are tangible. This requirement (about tangible expressions) means that only (possibly empty) multisets of stochastic multiactions or non-empty multisets of waiting multiactions, and no immediate multiactions, can be executed

from the subprocess that is composed *alternatively (in choice)* with the subprocess G. Hence, the multiset W of waiting multiactions, executed from G, can also be executed from the composition of G and that alternative subprocess, since immediate multiactions cannot occur from the latter. Otherwise, it would prevent the execution of W from G in the composite process, by disregarding the alternative choice of the branch specified by G, due to the zero delays and priority (captured by all action rules) of immediate multiactions over all other multiaction types.

The precondition in rule **P1w** is an exception from the above. It also ensures that non-empty multisets of waiting multiactions are executed only from *w-tangible* saturated operative dynamic expressions, such that all dynamic expressions structurally equivalent to them are tangible, but all the expressions structurally equivalent to H specifying parallel with G subprocess should be s-tangible. This stricter requirement (about s-tangible, instead of just tangible, expressions) means that only (possibly empty) multisets of stochastic multiactions, and no immediate or waiting multiactions, can be executed from the subprocess H that is composed *concurrently (in parallel)* with the subprocess G. Hence, the multiset W of waiting multiactions, executed from G, is also a maximal (by the inclusion relation) multiset that can be executed from the parallel composition of G and H. The reason is that only the timers decrement by one time unit (by applying rule **E**) is actually possible in H while executing W from G, due to priority (captured by all action rules) of waiting multiactions over stochastic ones. Thus, taking the rule precondition $stang([H]_{\approx})$ instead of $tang([H]_{\approx})$ preserves maximality of the steps of waiting multiactions while applying parallel composition.

In rules **P1s** and **P1w**, the timer value decrementing by one $\circlearrowleft H$, applied to the s-tangible saturated operative dynamic expression H that is composed in parallel with G, from which stochastic multiactions are executed at the next time tick, is used to maintain the time progress uniformity in the composite expression. Although rules **P1s** and **P1w** can be merged, we have not done it, aiming to emphasize the exceptional precondition in rule **P1w**.

In rules **Cs**, **Ci** and **Cw**, the timer values discarding $\lfloor E$, applied to the static expression E that is composed in choice with G, from which activities are executed, signifies that the timer values of the non-chosen subexpression (branch) become irrelevant in the composite expression and thus may be removed. Analogously, in rules **I2s**, **I2i** and **I2w**, the timer values discarding $\lfloor F$ is applied to the static expression F that is an alternative to G, from which activities are executed, since the choice is always made between the body and termination subexpressions of the composite iteration expression (between the second and third iteration arguments).

Rule **E** corresponds to one discrete time unit delay (passage of one unit of time) while executing no activities and therefore it has no analogues among the rules of gsPBC with the continuous time model. Rule **E** is a *global* one, i.e. it is applied only to the whole (topmost level of) expressions, rather than to their parts. The reason is that all other action rules describe dynamic expressions transformations due to execution of *non-empty* multisets of activities. Hence, the actionless time move described by rule **E** cannot "penetrate" with action rules through the expressions structure. This guarantees that time progresses uniformly in all their subexpressions.

Rule **Bw** differs from the more standard ones **Bs** and **Bi** that both resemble rule **B** in gsPBC. The reason is that in **Bw**, the overlined waiting multiaction

has an extra superscript '1', indicating that one time unit is remained until the multiaction's execution (RTE equals one) that should follow in the next moment.

Rules **P2s**, **P2i** and **P2w** have no similar rules in gsPBC, since interleaving semantics of the algebra allows no simultaneous execution of activities. On the other hand, **P2s**, **P2i** and **P2w** have in PBC the analogous rule **PAR** that is used to construct step semantics of the calculus, but the former two rules correspond to execution of multisets of activities, unlike that of multisets of multiactions in the latter rule. Rules **P2s**, **P2i** and **P2w** cannot be merged, since otherwise simultaneous execution of different types of multiactions would be allowed.

Rules **Sy2s**, **Sy2i** and **Sy2w** differ from the corresponding synchronization rules in gsPBC, since the probability or the weight of synchronization in the former rules and the rate or the weight of synchronization in the latter rules are calculated in two distinct ways. Rules **Sy2i** and **Sy2w** cannot be merged, since otherwise synchronous execution of immediate and waiting multiactions would be allowed.

Rule **Sy2s** establishes that the synchronization of two stochastic multiactions is made by taking the product of their probabilities, since we are considering that both must occur for the synchronization to happen, so this corresponds, in some sense, to the probability of the independent event intersection, but the real situation is more complex, since these stochastic multiactions can also be executed in parallel. Nevertheless, when scoping (the combined operation consisting of synchronization followed by restriction over the same action [8]) is applied over a parallel execution, we get as final result just the simple product of the probabilities, since no normalization is needed there. Multiplication is an associative and commutative binary operation that is distributive over addition, i.e. it fulfills all practical conditions imposed on the synchronization operator in [19]. Further, if both arguments of multiplication are from $(0; 1)$ then the result belongs to the same interval, hence, multiplication naturally maintains probabilistic compositionality in our model. Our approach is similar to the multiplication of rates of the synchronized actions in MTIPP [18] in the case when the rates are less than 1. Moreover, for the probabilities ρ and χ of two stochastic multiactions to be synchronized we have $\rho \cdot \chi < \min\{\rho, \chi\}$, i.e. multiplication meets the performance requirement stating that the probability of the resulting synchronized stochastic multiaction should be less than the probabilities of the two ones to be synchronized. While performance evaluation, it is usually supposed that the execution of two components together require more system resources and time than the execution of each single one. This resembles the *bounded capacity* assumption from [19]. Thus, multiplication is easy to handle with and it satisfies the algebraic, probabilistic, time and performance requirements. Therefore, we have chosen the product of the probabilities for the synchronization. See also [13, 12] for a discussion about binary operations producing the rates of synchronization in the continuous time setting.

In rules **Sy2i** and **Sy2w**, we sum the weights of two synchronized immediate (waiting, respectively) multiactions, since the weights can be interpreted as the rewards [43], thus, we collect the rewards. Moreover, we express that the synchronized execution of immediate (waiting) multiactions has more importance than that of every single one. The weights of immediate and waiting (i.e. deterministic) multiactions can also be seen as bonus rewards associated with transitions [5]. The rewards are summed during synchronized execution of immediate (waiting)

multiactions, since in that case all the synchronized activities can be seen as participated in the execution. We prefer to collect more rewards, thus, the transitions providing greater rewards will have a preference and they will be executed with a greater probability. In particular, since execution of immediate multiactions takes no time, we prefer to collect in a step (parallel execution) as many synchronized immediate multiactions as possible to get more significant progress in behaviour. Under behavioural progress we understand an advance in executing activities, which does not always imply a progress in time, as in the case when the activities are immediate multiactions. This aspect will be used later, while evaluating performance via analysis of the embedded discrete time Markov chains (EDTMCs) of expressions. Since every state change in EDTMC takes one unit of (its local) time, greater advance in operation of the EDTMC allows one to calculate quicker many performance indices. As for waiting multiactions, only the maximal multisets of them, executable from a state, occur with a time tick. The reason is that each waiting multiaction has a probability 1 to occur in the next moment, when the remaining time of its timer (RTE) equals one and there exist no conflicting waiting multiactions. Hence, all waiting multiactions with the RTE being one that are executable together from a state must participate in a step from that state. Since there may exist different such maximal multisets of waiting multiactions, a probabilistic choice among all possible steps is made, imposed by the weights of those multiactions. Thus, the steps of waiting multiactions always produce maximal overall weights, but they are mainly used to calculate the probabilities of alternative maximal steps rather than the cumulative bonus rewards.

We do not have self-synchronization, i.e. synchronization of an activity with itself, since all the (enumerated) activities executed together are considered to be different. This allows us to avoid rather cumbersome and unexpected behaviour, as well as many technical difficulties [8].

Notice that the timers of all waiting multiactions that lose their enabledness when a state change occurs become inactive (turned off) and their values become irrelevant while the timers of all those preserving their enabledness continue running with their stored values. Hence, we adopt the *enabling memory* memory policy [35, 1, 2, 3] when the process states are changed and the enabledness of deterministic multiactions is possibly modified (remember that immediate multiactions may be seen as those with the timers displaying a single value 0, so we do not need to store their values). Then the timer values of waiting multiactions are taken as the enabling memory variables.

Similar in [23], we are mainly interested in the dynamic expressions, inferred by applying the inaction rules (also in the reverse direction) and action rules from the overlined static expressions, such that no stamped (i.e. superscribed with the timer values) waiting multiaction is a subexpression of them. The reason is to ensure that time proceeds uniformly and only enabled waiting multiactions are stamped. We call such dynamic expressions reachable, by analogy with the reachable states of PNs. Formally, a dynamic expression G is *reachable*, if there exists a static expression E without timer value superscripts, such that $\overline{E} \approx G$ or $\overline{E} \approx G_0 \stackrel{\Upsilon_1}{\to} H_1 \approx G_1 \stackrel{\Upsilon_2}{\to} \ldots \stackrel{\Upsilon_n}{\to} H_n \approx G$ for some $\Upsilon_1, \ldots, \Upsilon_n \in \mathbb{N}_{fin}^{\mathcal{SDL}}$.

Therefore, we consider a dynamic expression $G = \overline{(\{a\}, \natural_1^2)^1}[](\{b\}, \natural_2^3)^1$ as "illegal" and that $H = \overline{(\{a\}, \natural_1^2)^1}[](\{b\}, \natural_2^3)^2$ as "legal", since the latter is obtained from the

overlined static expression without timer value superscripts $\overline{E} = \overline{(\{a\}, \natural_1^2)[](\{b\}, \natural_2^3)}$ after one time tick. On the other hand, G is "illegal" only when it is intended to specify a complete process, but it may become "legal" as a part of some complete specification, like G rs a, since after two time ticks from $\overline{E \text{ rs } a}$, the timer values cannot be decreased further when the value 1 is approached. Thus, we should allow the dynamic expressions like G, by assuming that they are incomplete specifications, to be further composed. Further, a dynamic expression $G = \overline{(\{a\}, \frac{1}{2})}; (\{b\}, \natural_1^2)^1$ is "illegal", since the waiting multiaction $(\{b\}, \natural_1^2)$ is not enabled in $[G]_{\approx}$ and its timer cannot start before the stochastic multiaction $(\{a\}, \frac{1}{2})$ is executed. Enabledness of the stamped waiting multiactions is considered in the next proposition.

Proposition 2. *Let G be a reachable dynamic expression. Then only waiting multiactions from $EnaWait([G]_{\approx})$ are stamped in G.*

Proof. By the definition of reachability, there exists $E \in StatExpr$ without stamped waiting multiactions, such that G is derived from $\overline{E}$ by applying the inaction rules (also those reversed) and action rules.

In that derivation, only the first *inaction* rule can add timer value superscripts to the waiting multiactions from $\mathcal{WL}(G) = \mathcal{WL}(E)$ that are overlined. The other inaction rules (also reversed) can just "shift" the upper bars from / to those stamped waiting multiactions while preserving the enabledness of all waiting multiactions from $\mathcal{WL}(G)$. Thus, just the waiting multiactions from $EnaWait([G]_{\approx})$ *become* stamped in the subexpressions of G, such as $\overline{(\alpha, \natural_l^\theta)^\theta}$ or $(\alpha, \natural_l^\theta)^\theta$.

Further, in the derivation, the *action* rules cannot add timer value superscripts to the waiting multiactions from $\mathcal{WL}(G)$, but the action rules can make such waiting multiactions non-enabled (disabled), i.e. belonging to $\mathcal{WL}(G) \setminus EnaWait([G]_{\approx})$. Such "disabling" action rules correspond either to the executing an overlined stamped (with the value 1) waiting multiaction (rule **Bw**) or to the choice of some alternative process branch (rules **Cs**, **Ci**, **Cw**, **I2s**, **I2i**, **I2w**). In the both cases, all the disabled waiting multiactions loose their timer value superscripts. Thus, only the waiting multiactions from $EnaWait([G]_{\approx})$ *remain* stamped in G.

Hence, $\overline{E}$ does not contain stamped waiting multiactions and in the derivation of G from it, only the waiting multiactions from $EnaWait([G]_{\approx})$ *become* and *remain* stamped in G. Therefore, only waiting multiactions from $EnaWait([G]_{\approx})$ are stamped in G. □

In Table 4, inaction rules, action rules (with stochastic or immediate, or waiting multiactions) and empty move rule are compared according to the three questions about their application: whether it changes the current state, whether it leads to a time progress, and whether it results in execution of some activities. Positive answers to the questions are denoted by the plus sign while negative ones are specified by the minus sign. If both positive and negative answers can be given to some of the questions in different cases then the plus-minus sign is written. Notice that the process states are considered up to structural equivalence of the corresponding expressions, and time progress is not regarded as a state change.

3.3. **Transition systems.** We now construct labeled probabilistic transition systems associated with dynamic expressions. The transition systems are used to define the operational semantics of dynamic expressions.

TABLE 4. Comparison of inaction, action and empty move rules

Rules	State change	Time progress	Activities execution
Inaction rules	$-$	$-$	$-$
Action rules with stochastic/waiting multiactions	$\pm$	$+$	$+$
Action rules with immediate multiactions	$\pm$	$-$	$+$
Empty move rule	$-$	$+$	$-$

Let G be a dynamic expression and $s = [G]_{\approx}$. The set of *all multisets of activities executable in* s is defined as $Exec(s) = \{\Upsilon \mid \exists H \in s\ \exists \widetilde{H}\ H \xrightarrow{\Upsilon} \widetilde{H}\}$. Here $H \xrightarrow{\Upsilon} \widetilde{H}$ is an inference by the rules from Table 3. It can be proved by induction on the structure of expressions that $\Upsilon \in Exec(s) \setminus \{\emptyset\}$ implies $\exists H \in s\ \Upsilon \in Now(H)$. The reverse statement does not hold in general, since the preconditions in the action rules disable executions of the activities with the lower-priority types from every $H \in s$, as the next example shows.

Example 8. *Let H, H' be from Example 7 and $s = [H]_{\approx} = [H']_{\approx}$. We have $Now(H) = \{\{(\{a\}, \natural_1^0)\}\}$ and $Now(H') = \{\{(\{b\}, \frac{1}{2})\}\}$. Since only rules* **Ci** *and* **Bi** *can be applied to H while no action rule can be applied to H', we get $Exec(s) = \{\{(\{a\}, \natural_1^0)\}\}$. Then, for $H' \in s$ and $\Upsilon = \{(\{b\}, \frac{1}{2})\} \in Now(H')$, we get $\Upsilon \notin Exec(s)$.*

The state s is *s-tangible (stochastically tangible)*, denoted by $stang(s)$, if $Exec(s) \subseteq \mathbb{N}_{fin}^{\mathcal{SL}}$. For an s-tangible state s we always have $\emptyset \in Exec(s)$ by rule **E**, hence, we may have $Exec(s) = \{\emptyset\}$. The state s is *w-tangible (waitingly tangible)*, denoted by $wtang(s)$, if $Exec(s) \subseteq \mathbb{N}_{fin}^{\mathcal{WL}} \setminus \{\emptyset\}$. The state s is *tangible*, denoted by $tang(s)$, if $stang(s)$ or $wtang(s)$, i.e. $Exec(s) \subseteq \mathbb{N}_{fin}^{\mathcal{SL}} \cup \mathbb{N}_{fin}^{\mathcal{WL}}$. Again, for a tangible state s we may have $\emptyset \in Exec(s)$ and $Exec(s) = \{\emptyset\}$. Otherwise, the state s is *vanishing*, denoted by $vanish(s)$, and in this case $Exec(s) \subseteq \mathbb{N}_{fin}^{\mathcal{IL}} \setminus \{\emptyset\}$.

Since for every $H \in s$, $Now(H)$ containing the multisets of activities with the lower-priority types is not included into $Exec(s)$, and the types of states are determined from the highest-priority types of the executable activities, the state type definitions based on $Now(H)$, $H \in s$, and on $Exec(s)$ are consistent.

If $\Upsilon \in Exec(s)$ and $\Upsilon \in \mathbb{N}_{fin}^{\mathcal{SL}} \cup \mathbb{N}_{fin}^{\mathcal{IL}}$ then by rules **P2s**, **P2i**, **Sy2s**, **Sy2i** and definition of $Exec(s)$ $\forall \Xi \subseteq \Upsilon$, $\Xi \neq \emptyset$, we have $\Xi \in Exec(s)$, i.e. $2^{\Upsilon} \setminus \{\emptyset\} \subseteq Exec(s)$.

Since the inaction rules only distribute and move upper and lower bars along the syntax of dynamic expressions, all $H \in s$ have the same underlying static expression F. Process expressions always have a finite length, hence, the number of all (enumerated) activities and the number of all operations in the syntax of F are finite as well. The action rules **Sy2s**, **Sy2i** and **Sy2w** are the only ones that generate new activities. They result from the handshake synchronization of actions and their conjugates belonging to the multiaction parts of the first and second constituent activity, respectively. Since we have a finite number of operators sy in F and all the multiaction parts of the activities are finite multisets, the number of the new synchronized activities is also finite. The action rules contribute to $Exec(s)$ (in addition to the empty set, if rule **E** is applicable) only the sets consisting both of activities from F and the new activities, produced by **Sy2s**, **Sy2i** and **Sy2w**. Since we have a finite number n of all such activities, we get $|Exec(s)| \leq 2^n < \infty$.

Thus, summation and multiplication by elements from the finite set $Exec(s)$ are well-defined. Similar reasoning can be used to demonstrate that for all dynamic expressions H (not just for those from s), $Now(H)$ is a finite set.

Definition 12. *The* derivation set *of a dynamic expression G, denoted by $DR(G)$, is the minimal set such that*

- $[G]_{\approx} \in DR(G)$*;*
- *if* $[H]_{\approx} \in DR(G)$ *and* $\exists \Upsilon\ H \stackrel{\Upsilon}{\to} \widetilde{H}$ *then* $[\widetilde{H}]_{\approx} \in DR(G)$*.*

The set of *all s-tangible states from* $DR(G)$ is denoted by $DR_{ST}(G)$, and the set of *all w-tangible states from* $DR(G)$ is denoted by $DR_{WT}(G)$. The set of *all tangible states from* $DR(G)$ is denoted by $DR_T(G) = DR_{ST}(G) \cup DR_{WT}(G)$. The set of *all vanishing states from* $DR(G)$ is denoted by $DR_V(G)$. Then $DR(G) = DR_T(G) \uplus DR_V(G) = DR_{ST}(G) \uplus DR_{WT}(G) \uplus DR_V(G)$ ($\uplus$ denotes disjoint union).

Let now G be a dynamic expression and $s, \tilde{s} \in DR(G)$.

Let $\Upsilon \in Exec(s) \setminus \{\emptyset\}$. The *probability that the multiset of stochastic multiactions* Υ *is ready for execution in* s or the *weight of the multiset of deterministic multiactions* Υ *which is ready for execution in* s is

$$PF(\Upsilon, s) = \begin{cases} \prod\limits_{(\alpha,\rho)\in\Upsilon} \rho \cdot \prod\limits_{\{\{(\beta,\chi)\}\in Exec(s) \mid (\beta,\chi)\notin\Upsilon\}} (1-\chi), & s \in DR_{ST}(G); \\ \sum\limits_{(\alpha,\natural_l^\theta)\in\Upsilon} l, & s \in DR_{WT}(G) \cup DR_V(G). \end{cases}$$

In the case $\Upsilon = \emptyset$ and $s \in DR_{ST}(G)$ we define

$$PF(\emptyset, s) = \begin{cases} \prod\limits_{\{(\beta,\chi)\}\in Exec(s)} (1-\chi), & Exec(s) \neq \{\emptyset\}; \\ 1, & Exec(s) = \{\emptyset\}. \end{cases}$$

If $s \in DR_{ST}(G)$ and $Exec(s) \neq \{\emptyset\}$ then $PF(\Upsilon, s)$ can be interpreted as a *joint* probability of independent events (in a probability sense, i.e. the probability of intersection of these events is equal to the product of their probabilities). Each such an event consists in the positive or the negative decision to be executed of a particular stochastic multiaction. Every executable stochastic multiaction decides probabilistically (using its probabilistic part) and independently (from others), if it wants to be executed in s. If Υ is a multiset of all executable stochastic multiactions which have decided to be executed in s and $\Upsilon \in Exec(s)$ then Υ is ready for execution in s. The multiplication in the definition is used because it reflects the probability of the independent event intersection. Alternatively, when $\Upsilon \neq \emptyset$, $PF(\Upsilon, s)$ can be interpreted as the probability to execute *exclusively* the multiset of stochastic multiactions Υ in s, i.e. the probability of *intersection* of two events calculated using the conditional probability formula in the form of $\mathsf{P}(X \cap Y) = \mathsf{P}(X|Y)\mathsf{P}(Y)$. The event X consists in the execution of Υ in s. The event Y consists in the non-execution in s of all the executable stochastic multiactions not belonging to Υ. Since the mentioned non-executions are obviously independent events, the probability of Y is a product of the probabilities of the non-executions: $\mathsf{P}(Y) = \prod_{\{\{(\beta,\chi)\}\in Exec(s) \mid (\beta,\chi)\notin\Upsilon\}} (1-\chi)$. The conditioning of X by Y makes the executions of the stochastic multiactions from Υ independent, since all of them can be executed in parallel in s by definition of $Exec(s)$. Hence, the

probability to execute Υ *under condition* that no executable stochastic multiactions not belonging to Υ are executed in s is a product of probabilities of these stochastic multiactions: $\mathsf{P}(X|Y) = \prod_{(\alpha,\rho)\in\Upsilon}\rho$. Thus, the probability that Υ is executed *and* no executable stochastic multiactions not belonging to Υ are executed in s is the probability of X conditioned by Y multiplied by the probability of Y: $\mathsf{P}(X\cap Y) = \mathsf{P}(X|Y)\mathsf{P}(Y) = \prod_{(\alpha,\rho)\in\Upsilon}\rho\cdot\prod_{\{\{(\beta,\chi)\}\in Exec(s)|(\beta,\chi)\notin\Upsilon\}}(1-\chi)$. When $\Upsilon = \emptyset$, $PF(\Upsilon, s)$ can be interpreted as the probability not to execute in s any executable stochastic multiactions, thus, $PF(\emptyset, s) = \prod_{\{(\beta,\chi)\}\in Exec(s)}(1-\chi)$. When only the empty multiset of activities can be executed in s, i.e. $Exec(s) = \{\emptyset\}$, we take $PF(\emptyset, s) = 1$, since nothing more can be executed in s in this case. Since the probabilities of all stochastic multiactions are strictly less than 1, for $s \in DR_{ST}(G)$ we have $PF(\emptyset, s) \in (0; 1]$. Hence, we always execute the empty multiset of activities in s at the next moment with a certain positive probability.

If $s \in DR_{WT}(G) \cup DR_V(G)$ then $PF(\Upsilon, s)$ could be interpreted as the *overall (cumulative)* weight of the deterministic multiactions from Υ, i.e. the sum of all their weights. The summation here is used since the weights can be seen as the rewards which are collected [43]. This means that concurrent execution of the deterministic multiactions has more importance than that of every single one. The weights of deterministic multiactions can also be interpreted as bonus rewards of transitions [5]. The rewards are summed when deterministic multiactions are executed in parallel, because all of them participated in the execution. In particular, since execution of immediate multiactions takes no time, we prefer to collect in a step (parallel execution of activities) as many parallel immediate multiactions as possible to get more progress in behaviour. This aspect will be used later, while while evaluating performance on the basis of the EDTMCs of expressions. Concerning waiting multiactions, only the maximal multisets of them executable from a state occur in the next moment. Therefore, the steps of waiting multiactions produce maximal overall weights, which are used to calculate probabilities of alternative maximal steps rather than the cumulative bonuses. Note that this reasoning is the same as that used to define the weight of synchronized immediate (waiting, respectively) multiactions in the rules **Sy2i** and **Sy2w**.

Note that the definition of $PF(\Upsilon, s)$ (and those of other probability functions we shall present) is based on the enumeration of activities which is considered implicit.

Let $\Upsilon \in Exec(s)$. Besides Υ, some other multisets of activities may be ready for execution in s, hence, a conditioning or normalization is needed to calculate the execution probability. The *probability to execute the multiset of activities* Υ *in* s is

$$PT(\Upsilon, s) = \frac{PF(\Upsilon, s)}{\sum_{\Xi\in Exec(s)} PF(\Xi, s)}.$$

If $s \in DR_{ST}(G)$ then $PT(\Upsilon, s)$ can be interpreted as the *conditional* probability to execute Υ in s calculated using the conditional probability formula in the form of $\mathsf{P}(Z|W) = \frac{\mathsf{P}(Z\cap W)}{\mathsf{P}(W)}$. The event Z consists in the exclusive execution of Υ in s, hence, $\mathsf{P}(Z) = PF(\Upsilon, s)$. The event W consists in the exclusive execution of any set (including the empty one) $\Xi \in Exec(s)$ in s. Thus, $W = \cup_j Z_j$, where $\forall j,\ Z_j$ are mutually exclusive events (in a probability sense, i.e. intersection of these events is the empty event) and $\exists i,\ Z = Z_i$. We have $\mathsf{P}(W) = \sum_j \mathsf{P}(Z_j) = \sum_{\Xi\in Exec(s)} PF(\Xi, s)$, because summation reflects the probability of the mutually exclusive event union. Since $Z \cap W = Z_i \cap (\cup_j Z_j) = Z_i = Z$, we have $\mathsf{P}(Z|W) =$

$\frac{\mathsf{P}(Z)}{\mathsf{P}(W)} = \frac{PF(\Upsilon,s)}{\sum_{\Xi\in Exec(s)} PF(\Xi,s)}$. Note that $PF(\Upsilon,s)$ can also be seen as the *potential* probability to execute Υ in s, since we have $PF(\Upsilon,s) = PT(\Upsilon,s)$ only when *all* sets (including the empty one) consisting of the executable stochastic multiactions can be executed in s. In this case, all the mentioned stochastic multiactions can be executed in parallel in s and we have $\sum_{\Xi\in Exec(s)} PF(\Xi,s) = 1$, since this sum collects the products of *all* combinations of the probability parts of the stochastic multiactions and the negations of these parts. But in general, for example, for two stochastic multiactions (α,ρ) and (β,χ) executable in s, it may happen that they cannot be executed in s together, i.e. $\emptyset, \{(\alpha,\rho)\}, \{(\beta,\chi)\} \in Exec(s)$, but $\{(\alpha,\rho), (\beta,\chi)\} \notin Exec(s)$. For $s \in DR_{ST}(G)$ we have $PT(\emptyset,s) \in (0;1]$, i.e. there is a non-zero probability to execute the empty multiset of activities in s at the next moment.

If $s \in DR_{WT}(G) \cup DR_V(G)$ then $PT(\Upsilon,s)$ can be interpreted as the weight of the set of deterministic multiactions Υ which is ready for execution in s *normalized* by the weights of *all* the sets executable in s. This approach is analogous to that used in the EMPA definition of the probabilities of immediate actions executable from the same process state [6] (inspired by way in which the probabilities of conflicting immediate transitions in GSPNs are calculated [3]). The only difference is that we have a step semantics and, for every set of deterministic multiactions executed in parallel, we should use its cumulative weight. For the analogy with the interleaving semantics of EMPA, we should interpret the weights of immediate actions of EMPA as the cumulative weights of the sets of deterministic multiactions of dtsdPBC.

The advantage of our two-stage approach to definition of the probability to execute a set of activities is that the resulting probability formula $PT(\Upsilon,s)$ is valid both for (sets of) stochastic and deterministic multiactions. It allows one to unify the notation used later while constructing the operational semantics.

Note that the sum of outgoing probabilities for the expressions belonging to the derivations of G is equal to 1. More formally, $\forall s \in DR(G)\ \sum_{\Upsilon\in Exec(s)} PT(\Upsilon,s) = 1$. This, obviously, follows from the definition of $PT(\Upsilon,s)$, and guarantees that it defines a probability distribution.

The *probability to move from s to $\tilde{s}$ by executing any multiset of activities* is

$$PM(s,\tilde{s}) = \sum_{\{\Upsilon \mid \exists H\in s\ \exists \widetilde{H}\in\tilde{s}\ H\stackrel{\Upsilon}{\to}\widetilde{H}\}} PT(\Upsilon,s).$$

The summation in the definition above reflects the probability of the mutually exclusive event union, since $\sum_{\{\Upsilon \mid \exists H\in s,\ \exists \widetilde{H}\in\tilde{s},\ H\stackrel{\Upsilon}{\to}\widetilde{H}\}} PT(\Upsilon,s) = \frac{1}{\sum_{\Xi\in Exec(s)} PF(\Xi,s)} \cdot \sum_{\{\Upsilon \mid \exists H\in s,\ \exists \widetilde{H}\in\tilde{s},\ H\stackrel{\Upsilon}{\to}\widetilde{H}\}} PF(\Upsilon,s)$, where for each Υ, $PF(\Upsilon,s)$ is the probability of the exclusive execution of Υ in s. Note that $\forall s \in DR(G)$
$\sum_{\{\tilde{s} \mid \exists H\in s\ \exists \widetilde{H}\in\tilde{s}\ \exists\Upsilon\ H\stackrel{\Upsilon}{\to}\widetilde{H}\}} PM(s,\tilde{s}) = \sum_{\{\tilde{s} \mid \exists H\in s\ \exists \widetilde{H}\in\tilde{s}\ \exists\Upsilon\ H\stackrel{\Upsilon}{\to}\widetilde{H}\}}$
$\sum_{\{\Upsilon \mid \exists H\in s\ \exists \widetilde{H}\in\tilde{s}\ H\stackrel{\Upsilon}{\to}\widetilde{H}\}} PT(\Upsilon,s) = \sum_{\Upsilon\in Exec(s)} PT(\Upsilon,s) = 1.$

Example 9. *Let $E = (\{a\},\rho)[](\{a\},\chi)$, where $\rho,\chi \in (0;1)$. $DR(\overline{E})$ consists of the equivalence classes $s_1 = [\overline{E}]_{\approx}$ and $s_2 = [\underline{E}]_{\approx}$. We have $DR_T(\overline{E}) = \{s_1,s_2\}$. The execution probabilities are calculated as follows. Since $Exec(s_1) = \{\emptyset, \{(\{a\},\rho)\}, \{(\{a\},\chi)\}\}$, we get $PF(\{(\{a\},\rho)\},s_1) = \rho(1-\chi)$, $PF(\{(\{a\},\chi)\},s_1) = \chi(1-\rho)$ and $PF(\emptyset,s_1) = (1-\rho)(1-\chi)$. Then $\sum_{\Xi\in Exec(s_1)} PF(\Xi,s_1) = \rho(1-\chi)+\chi(1-\rho)+(1-\rho)(1-\chi) = 1-\rho\chi$. Thus, $PT(\{(\{a\},\rho)\},s_1) = \frac{\rho(1-\chi)}{1-\rho\chi}$, $PT(\{(\{a\},\chi)\},s_1) =$*

TABLE 5. Calculation of the probability functions PF, PT, PM for $s_1 \in DR(\overline{E})$ and $E = (\{a\}, \rho)[](\{a\}, \chi)$

$s_1 \backslash \Upsilon$	$\emptyset$	$\{(\{a\}, \rho)\}$	$\{(\{a\}, \chi)\}$	Σ
PF	$(1-\rho)(1-\chi)$	$\rho(1-\chi)$	$\chi(1-\rho)$	$1-\rho\chi$
PT	$\frac{(1-\rho)(1-\chi)}{1-\rho\chi}$	$\frac{\rho(1-\chi)}{1-\rho\chi}$	$\frac{\chi(1-\rho)}{1-\rho\chi}$	1
PM	$\frac{(1-\rho)(1-\chi)}{1-\rho\chi}$ (s_1)	$\frac{\rho+\chi-2\rho\chi}{1-\rho\chi}$ (s_2)		1

TABLE 6. Calculation of the probability functions PF, PT, PM for $s'_1 \in DR(\overline{E'})$ and $E' = (\{a\}, \natural_l^0)[](\{a\}, \natural_m^0)$

$s'_1 \backslash \Upsilon$	$\{(\{a\}, \natural_l^0)\}$	$\{(\{a\}, \natural_m^0)\}$	Σ
PF	l	m	$l+m$
PT	$\frac{l}{l+m}$	$\frac{m}{l+m}$	1
PM	1 (s'_2)		1

$\frac{\chi(1-\rho)}{1-\rho\chi}$ *and* $PT(\emptyset, s_1) = PM(s_1, s_1) = \frac{(1-\rho)(1-\chi)}{1-\rho\chi}$. *Further,* $Exec(s_2) = \{\emptyset\}$, *hence,* $\sum_{\Xi \in Exec(s_2)} PF(\Xi, s_2) = PF(\emptyset, s_2) = 1$ *and* $PT(\emptyset, s_2) = PM(s_2, s_2) = \frac{1}{1} = 1$. *Finally,* $PM(s_1, s_2) = PT(\{(\{a\}, \rho)\}, s_1) + PT(\{(\{a\}, \chi)\}, s_1) = \frac{\rho(1-\chi)}{1-\rho\chi} + \frac{\chi(1-\rho)}{1-\rho\chi} = \frac{\rho+\chi-2\rho\chi}{1-\rho\chi}$. *In Table 5, the calculation of the probability functions* $PF(\Upsilon, s_1)$, $PT(\Upsilon, s_1)$, $PM(s_1, s)$ *is explained, where* $\Upsilon \in Exec(s_1)$, $s \in \{s_1, s_2\}$ *(the value of* s *is depicted in the parentheses near the value of* $PM(s_1, s)$*) and* $\Sigma = \sum_{\Xi \in Exec(s_1)} PX(\Xi, s_1)$, $PX \in \{PF, PT, PM\}$.

Let $E' = (\{a\}, \natural_l^0)[](\{a\}, \natural_m^0)$, *where* $l, m \in \mathbb{R}_{>0}$. $DR(\overline{E'})$ *consists of the equivalence classes* $s'_1 = [\overline{E'}]_\approx$ *and* $s'_2 = [\underline{E'}]_\approx$. *We have* $DR_T(\overline{E'}) = \{s'_2\}$ *and* $DR_V(\overline{E'}) = \{s'_1\}$. *The execution probabilities are calculated as follows. Since* $Exec(s'_1) = \{\{(\{a\}, \natural_l^0)\}, \{(\{a\}, \natural_m^0)\}\}$, *we get* $PF(\{(\{a\}, \natural_l^0)\}, s'_1) = l$ *and* $PF(\{(\{a\}, \natural_m^0)\}, s'_1) = m$. *Then* $\sum_{\Xi \in Exec(s'_1)} PF(\Xi, s'_1) = l + m$. *Thus,* $PT(\{(\{a\}, \natural_l^0)\}, s'_1) = \frac{l}{l+m}$ *and* $PT(\{(\{a\}, \natural_m^0)\}, s'_1) = \frac{m}{l+m}$. *Next,* $Exec(s'_2) = \{\emptyset\}$, *hence,* $\sum_{\Xi \in Exec(s'_2)} PF(\Xi, s'_2) = PF(\emptyset, s'_2) = 1$ *and* $PT(\emptyset, s'_2) = PM(s'_2, s'_2) = \frac{1}{1} = 1$. *Finally,* $PM(s'_1, s'_2) = PT(\{(\{a\}, \natural_l^0)\}, s'_1) + PT(\{(\{a\}, \natural_m^0)\}, s'_1) = \frac{l}{l+m} + \frac{m}{l+m} = 1$. *In Table 6, the calculation of the probability functions* $PF(\Upsilon, s'_1)$, $PT(\Upsilon, s'_1)$, $PM(s'_1, s')$ *is explained, where* $\Upsilon \in Exec(s'_1)$, $s' \in \{s'_2\}$ *(the value of* s' *is depicted in the parentheses near the value of* $PM(s'_1, s')$*) and* $\Sigma = \sum_{\Xi \in Exec(s'_1)} PX(\Xi, s'_1)$, $PX \in \{PF, PT, PM\}$.

Definition 13. *Let* G *be a dynamic expression. The* (labeled probabilistic) transition system *of* G *is a quadruple* $TS(G) = (S_G, L_G, \mathcal{T}_G, s_G)$, *where*

- *the set of* states *is* $S_G = DR(G)$;
- *the set of* labels *is* $L_G = \mathbb{N}_{fin}^{\mathcal{SDL}} \times (0; 1]$;
- *the set of* transitions *is* $\mathcal{T}_G = \{(s, (\Upsilon, PT(\Upsilon, s)), \tilde{s}) \mid s, \tilde{s} \in DR(G),\ \exists H \in s\ \exists \widetilde{H} \in \tilde{s}\ H \xrightarrow{\Upsilon} \widetilde{H}\}$;
- *the* initial state *is* $s_G = [G]_\approx$.

Example 10. *Let* E *be from Example 1. The next inferences by rule* **E** *are possible from the elements of* $[\overline{E}]_\approx$*:*

$$\overline{(\{a\},\natural_1^3)[](\{b\},\tfrac{1}{3})} \approx \overline{(\{a\},\natural_1^3)^3}[](\{b\},\tfrac{1}{3}) \stackrel{\emptyset}{\to} \overline{(\{a\},\natural_1^3)^2}[](\{b\},\tfrac{1}{3}),$$
$$\overline{(\{a\},\natural_1^3)[](\{b\},\tfrac{1}{3})} \approx (\{a\},\natural_1^3)^3[]\overline{(\{b\},\tfrac{1}{3})} \stackrel{\emptyset}{\to} (\{a\},\natural_1^3)^2[]\overline{(\{b\},\tfrac{1}{3})}.$$

The first and second inferences suggest the empty move transition $[\overline{E}]_{\approx} \stackrel{\emptyset}{\to} [\overline{(\{a\},\natural_1^3)^2[](\{b\},\frac{1}{3})}]_{\approx} \neq [\overline{E}]_{\approx}$. *The intuition is that the timer of the enabled waiting multiaction* $(\{a\},\natural_1^3)$ *is decremented by one time unit in the both cases, whenever it is overlined or not. In the both cases, the respective waiting transition of the PN corresponding to* $\overline{E}$ *will be enabled at a "common" marking (that also enables a stochastic transition, matched up to* $(\{b\},\frac{1}{3})$*), so its timer should be decreased by one with a time tick while staying at* the same marking*, and such a time move will lead to a* different state *of the PN.*

The definition of $TS(G)$ is correct, i.e. for every state, the sum of the probabilities of all the transitions starting from it is 1. This is guaranteed by the note after the definition of $PT(\Upsilon, s)$. Thus, we have a *generative* model of probabilistic processes, according to the classification from [16]. The reason is that the sum of the probabilities of the transitions with all possible labels should be equal to 1, not only of those with the same labels (up to enumeration of activities they include) as in the *reactive* models, and we do not have a nested probabilistic choice as in the *stratified* models.

The transition system $TS(G)$ associated with a dynamic expression G describes all the steps (parallel executions) that occur at discrete time moments with some (one-step) probability and consist of multisets of activities. Every step consisting of stochastic (waiting, respectively) multiactions or the empty step (i.e. that consisting of the empty multiset of activities) occurs instantly after one discrete time unit delay. Each step consisting of immediate multiactions occurs instantly without any delay. The step can change the current state to a different one. The states are the structural equivalence classes of dynamic expressions obtained by application of action rules starting from the expressions belonging to $[G]_{\approx}$. A transition $(s,(\Upsilon,\mathcal{P}),\tilde{s}) \in \mathcal{T}_G$ will be written as $s \stackrel{\Upsilon}{\to}_{\mathcal{P}} \tilde{s}$. It is interpreted as follows: the probability to change the state s to $\tilde{s}$ as a result of executing Υ is $\mathcal{P}$.

Note that from every s-tangible state the empty multiset of activities can always be executed by rule **E**. Hence, for s-tangible states, Υ may be the empty multiset, and its execution only decrements by one the timer values (if any) of the current state (i.e. the equivalence class). Then we may have a transition $s \stackrel{\emptyset}{\to}_{\mathcal{P}} \circlearrowleft s$ from an s-tangible state s to the tangible (i.e. s-tangible or w-tangible) state $\circlearrowleft s = \bigcup\{[\circlearrowleft H]_{\approx} \mid H \in s \cap SatOpRegDynExpr\}$. Thus, $\circlearrowleft s$ is the union of the structural equivalence classes of all saturated operative dynamic expressions from s, whose timer values have been decremented by one, prior to combining them into the equivalence classes. This corresponds to applying the empty move rule to all saturated operative dynamic expressions from s, followed by unifying the structural equivalence classes of all the resulting expressions. We have to keep track of such executions, called the *empty moves*, because they affect the timers and have non-zero probabilities. The latter follows from the definition of $PF(\emptyset, s)$ and the fact that the probabilities of stochastic multiactions cannot be equal to 1 as they belong to the interval $(0;1)$. When it holds $\forall H \in s \cap SatOpRegDynExpr \;\; \circlearrowleft H = H$, we obtain $\circlearrowleft s = s$ by definition of $\circlearrowleft s$. Then the empty move from s is in the form of $s \stackrel{\emptyset}{\to}_{\mathcal{P}} s$, called the *empty loop*. For w-tangible and vanishing states Υ cannot be the empty multiset,

since we must execute some immediate (waiting, respectively) multiactions from them at the current (next, respectively) time moment.

The step probabilities belong to the interval $(0;1]$, being 1 in the case when we cannot leave an s-tangible state s and the only transition leaving it is the empty move one $s \stackrel{\emptyset}{\rightarrow}_1 \circlearrowleft s$, or if there is just a single transition from a w-tangible or a vanishing state to any other one.

We write $s \stackrel{\Upsilon}{\rightarrow} \tilde{s}$ if $\exists \mathcal{P}\ s \stackrel{\Upsilon}{\rightarrow}_{\mathcal{P}} \tilde{s}$ and $s \rightarrow \tilde{s}$ if $\exists \Upsilon\ s \stackrel{\Upsilon}{\rightarrow} \tilde{s}$.

The first equivalence we are going to introduce is isomorphism which is a coincidence of systems up to renaming of their components or states.

Definition 14. *Let G, G' be dynamic expressions and $TS(G)=(S_G, L_G, \mathcal{T}_G, s_G)$, $TS(G')=(S_{G'}, L_{G'}, \mathcal{T}_{G'}, s_{G'})$ be their transition systems. A mapping $\beta : S_G \rightarrow S_{G'}$ is an* isomorphism *between $TS(G)$ and $TS(G')$, denoted by $\beta : TS(G) \simeq TS(G')$, if*

(1) *β is a bijection such that $\beta(s_G) = s_{G'}$;*
(2) *$\forall s, \tilde{s} \in S_G\ \forall \Upsilon\ s \stackrel{\Upsilon}{\rightarrow}_{\mathcal{P}} \tilde{s}\ \Leftrightarrow\ \beta(s) \stackrel{\Upsilon}{\rightarrow}_{\mathcal{P}} \beta(\tilde{s})$.*

Two transition systems $TS(G)$ and $TS(G')$ are isomorphic*, denoted by $TS(G) \simeq TS(G')$, if $\exists \beta : TS(G) \simeq TS(G')$.*

Transition systems of static expressions can also be defined. For $E \in RegStatExpr$, let $TS(E) = TS(\overline{E})$.

Definition 15. *Two dynamic expressions G and G' are* equivalent with respect to transition systems*, denoted by $G =_{ts} G'$, if $TS(G) \simeq TS(G')$.*

3.4. **Examples of transition systems.** We now present a series of examples that demonstrate how to construct the transition systems of the dynamic expressions that include various compositions of stochastic, waiting and immediate multiactions.

Example 11. *Let $E = (\{a\}, \natural_1^2)[](\{b\}, \natural_2^3)$. $DR(\overline{E})$ consists of the equivalence classes*

$$s_1 = [\overline{(\{a\}, \natural_1^2)^2}[](\{b\}, \natural_2^3)^3]_{\approx} = [(\{a\}, \natural_1^2)^2[]\overline{(\{b\}, \natural_2^3)^3}]_{\approx},$$
$$s_2 = [\overline{(\{a\}, \natural_1^2)^1}[](\{b\}, \natural_2^3)^2]_{\approx} = [(\{a\}, \natural_1^2)^1[]\overline{(\{b\}, \natural_2^3)^2}]_{\approx},$$
$$s_3 = [\underline{(\{a\}, \natural_1^2)[](\{b\}, \natural_2^3)}]_{\approx}.$$

We have $DR_{ST}(\overline{E}) = \{s_1, s_3\}$, $DR_{WT}(\overline{E}) = \{s_2\}$ and $DR_V(\overline{E}) = \emptyset$.

In Figure 2, the transition system $TS(\overline{E})$ is shown. The s-tangible and w-tangible states are placed in ordinary and double ovals, respectively. To simplify the graphical representation, the singleton multisets of activities are written without outer braces.

This example demonstrates a choice between two waiting multiactions with different delays. It shows that the waiting multiaction $(\{a\}, \natural_1^2)$ with a less delay 2 is always executed first, hence, the choice is resolved in favour of it in any case and an absorbing state is then reached, so that the waiting multiaction $(\{b\}, \natural_2^3)$ with a greater delay 3 is never executed.

Example 12. *Let $E = (\{a\}, \natural_1^3)[](\{b\}, \frac{1}{3})$. $DR(\overline{E})$ consists of the equivalence classes*

$$s_1 = [\overline{(\{a\}, \natural_1^3)^3}[](\{b\}, \tfrac{1}{3})]_{\approx} = [(\{a\}, \natural_1^3)^3[]\overline{(\{b\}, \tfrac{1}{3})}]_{\approx},$$
$$s_2 = [\overline{(\{a\}, \natural_1^3)^2}[](\{b\}, \tfrac{1}{3})]_{\approx} = [(\{a\}, \natural_1^3)^2[]\overline{(\{b\}, \tfrac{1}{3})}]_{\approx},$$
$$s_3 = [\overline{(\{a\}, \natural_1^3)^1}[](\{b\}, \tfrac{1}{3})]_{\approx} = [(\{a\}, \natural_1^3)^1[]\overline{(\{b\}, \tfrac{1}{3})}]_{\approx},$$
$$s_4 = [\underline{(\{a\}, \natural_1^3)[](\{b\}, \tfrac{1}{3})}]_{\approx}.$$

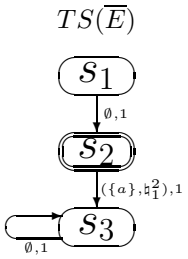


FIG. 2. The transition system of $\overline{E}$ for $E = (\{a\},\natural_1^2)[](\{b\},\natural_2^3)$

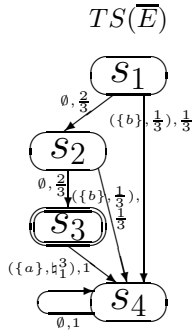


FIG. 3. The transition system of $\overline{E}$ for $E = (\{a\},\natural_1^3)[](\{b\},\frac{1}{3})$

We have $DR_{ST}(\overline{E}) = \{s_1, s_2, s_4\}$, $DR_{WT}(\overline{E}) = \{s_3\}$ *and* $DR_V(\overline{E}) = \emptyset$.

In Figure 3, the transition system $TS(\overline{E})$ *is shown. The s-tangible and w-tangible states are depicted in ordinary and double ovals, respectively.*

This example demonstrates a choice between waiting and stochastic multiactions. It shows that the stochastic multiaction $(\{b\},\frac{1}{3})$ *can be executed until the timer value of the waiting multiaction* $(\{a\},\natural_1^3)$ *becomes* 1*, after which only the waiting multiaction can be executed in the next moment, leading to an absorbing state. Thus, in our setting, a waiting multiaction that cannot be executed in the next time moment and whose timer is still running may be interrupted (preempted) by executing a stochastic multiaction.*

Example 13. *Let* $E = ((\{a\},\natural_1^3)[](\{b\},\frac{1}{3}))$ rs a. $DR(\overline{E})$ *consists of the equivalence classes*

$$s_1 = [(\overline{(\{a\},\natural_1^3)^3}[](\{b\},\tfrac{1}{3}))\ \mathsf{rs}\ a]_{\approx} = [((\{a\},\natural_1^3)^3[]\overline{(\{b\},\tfrac{1}{3})})\ \mathsf{rs}\ a]_{\approx},$$
$$s_2 = [(\overline{(\{a\},\natural_1^3)^2}[](\{b\},\tfrac{1}{3}))\ \mathsf{rs}\ a]_{\approx} = [((\{a\},\natural_1^3)^2[]\overline{(\{b\},\tfrac{1}{3})})\ \mathsf{rs}\ a]_{\approx},$$
$$s_3 = [(\overline{(\{a\},\natural_1^3)^1}[](\{b\},\tfrac{1}{3}))\ \mathsf{rs}\ a]_{\approx} = [((\{a\},\natural_1^3)^1[]\overline{(\{b\},\tfrac{1}{3})})\ \mathsf{rs}\ a]_{\approx},$$
$$s_4 = [\overline{((\{a\},\natural_1^3)[](\{b\},\tfrac{1}{3}))\ \mathsf{rs}\ a}]_{\approx}.$$

We have $DR_{ST}(\overline{E}) = \{s_1, s_2, s_3, s_4\}$ *and* $DR_{WT}(\overline{E}) = \emptyset = DR_V(\overline{E})$.

In Figure 4, the transition system $TS(\overline{E})$ *is shown. The s-tangible states are depicted in ordinary ovals.*

This example is a modification of the previous Example 12 by applying a restriction operation by action a to the whole expression. The present example shows that

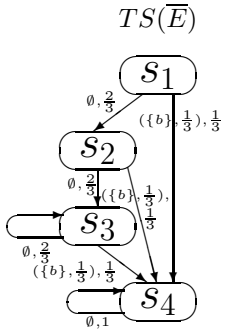


FIG. 4. The transition system of $\overline{E}$ for $E = ((\{a\}, \natural_1^3)[](\{b\}, \frac{1}{3}))$ rs a

the stochastic multiaction $(\{b\}, \frac{1}{3})$ *can be executed until the timer value of the "restricted" waiting multiaction* $(\{a\}, \natural_1^3)$ *becomes* 1*, after which the waiting multiaction also cannot be executed in the next moment, since it is affected by the restriction. Instead, the stochastic multiaction* $(\{b\}, \frac{1}{3})$ *can be executed again, leading to an absorbing state, or we return to the current state after one time tick (the empty loop in that state). Thus, a waiting multiaction that cannot be executed because of the restriction and whose timer runs until reaching its final value* 1 *may always be preempted by executing a stochastic multiaction. To verify that the timer value* 1 *remains unchanged with the time progress, recall the empty move rule* **E** *from Table 3 and the definition of* $\circlearrowleft G$ *with* $\max\{1, \delta - 1\} = \max\{1, 0\} = 1$ *when* $\delta = 1$.

Note that the timer decrement of the "restricted" waiting multiaction $(\{a\}, \natural_1^3)$ *induces a partial (for the first* 2 *time ticks) unfolding of the behaviour consisting in a choice between executing and non-executing the stochastic multiaction* $(\{b\}, \frac{1}{3})$*. In our setting, the timer values are kept even for the waiting multiactions that cannot be executed because of the restriction, since they can potentially participate in a synchronization, but the activities resulted from synchronization do not appear explicitly in the syntax of the process expressions, and their timer values can be detected only by observing those of the both synchronized waiting multiactions. We shall see an importance of such a construction, particularly, in Examples 17 and 21.*

Example 14. *Let* $E = [(\{a\}, \frac{1}{2}) * (\{b\}, \natural_1^3) * (\{c\}, \frac{1}{3})]$*.* $DR(\overline{E})$ *consists of the equivalence classes*

$$
\begin{aligned}
s_1 &= [[\overline{(\{a\}, \tfrac{1}{2})} * (\{b\}, \natural_1^3) * (\{c\}, \tfrac{1}{3})]]_{\approx},\\
s_2 &= [[(\{a\}, \tfrac{1}{2}) * \overline{(\{b\}, \natural_1^3)^3} * (\{c\}, \tfrac{1}{3})]]_{\approx} = [[(\{a\}, \tfrac{1}{2}) * (\{b\}, \natural_1^3)^3 * \overline{(\{c\}, \tfrac{1}{3})}]]_{\approx},\\
s_3 &= [[(\{a\}, \tfrac{1}{2}) * \overline{(\{b\}, \natural_1^3)^2} * (\{c\}, \tfrac{1}{3})]]_{\approx} = [[(\{a\}, \tfrac{1}{2}) * (\{b\}, \natural_1^3)^2 * \overline{(\{c\}, \tfrac{1}{3})}]]_{\approx},\\
s_4 &= [[(\{a\}, \tfrac{1}{2}) * \overline{(\{b\}, \natural_1^3)^1} * (\{c\}, \tfrac{1}{3})]]_{\approx} = [[(\{a\}, \tfrac{1}{2}) * (\{b\}, \natural_1^3)^1 * \overline{(\{c\}, \tfrac{1}{3})}]]_{\approx},\\
s_5 &= \underline{[[(\{a\}, \tfrac{1}{2}) * (\{b\}, \natural_1^3) * (\{c\}, \tfrac{1}{3})]]}_{\approx}.
\end{aligned}
$$

We have $DR_{ST}(\overline{E}) = \{s_1, s_2, s_3, s_5\}$*,* $DR_{WT}(\overline{E}) = \{s_4\}$ *and* $DR_V(\overline{E}) = \emptyset$*.*

In Figure 5, the transition system $TS(\overline{E})$ *is shown. The s-tangible and w-tangible states are depicted in ordinary and double ovals, respectively.*

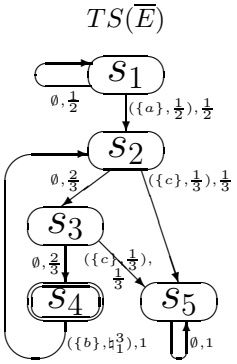


FIG. 5. The transition system of $\overline{E}$ for $E = [(\{a\},\frac{1}{2}) * (\{b\},\natural_1^3) * (\{c\},\frac{1}{3})]$

This example demonstrates an iteration loop with a waiting multiaction. The iteration initiation is modeled by a (initiating) stochastic multiaction $(\{a\},\frac{1}{2})$*. The iteration body that corresponds to the loop consists of a (looping) waiting multiaction* $(\{b\},\natural_1^3)$*. The iteration termination is represented by a (terminating) stochastic multiaction* $(\{c\},\frac{1}{3})$*. The terminating stochastic multiaction can be executed until the timer value of the waiting multiaction becomes* 1*, after which only the waiting multiaction can be executed in the next moment. Thus, the iteration termination can either complete the repeated execution of the iteration body or break its execution when the waiting multiaction timer shows some intermediate value (that is less than the initial value, being the multiaction delay, but greater than* 1*). The execution of the waiting multiaction* $(\{b\},\natural_1^3)$ *leads to the repeated start of the iteration body. The execution of the terminating stochastic multiaction* $(\{c\},\frac{1}{3})$ *brings to the final absorbing state of the iteration construction.*

Example 15. *Let* $E = (\{a\},\natural_1^0)\|(\{b\},\natural_2^2)\|(\{c\},\natural_3^3)$*.* $DR(\overline{E})$ *consists of the equivalence classes*

$$s_1 = [\overline{(\{a\},\natural_1^0)}\|\overline{(\{b\},\natural_2^2)^2}\|\overline{(\{c\},\natural_3^3)^3}]_{\approx}, \quad s_2 = [\underline{(\{a\},\natural_1^0)}\|\overline{(\{b\},\natural_2^2)^2}\|\overline{(\{c\},\natural_3^3)^3}]_{\approx},$$
$$s_3 = [\underline{(\{a\},\natural_1^0)}\|\overline{(\{b\},\natural_2^2)^1}\|\overline{(\{c\},\natural_3^3)^2}]_{\approx}, \quad s_4 = [\underline{(\{a\},\natural_1^0)}\|\underline{(\{b\},\natural_2^2)}\|\overline{(\{c\},\natural_3^3)^1}]_{\approx},$$
$$s_5 = [\underline{(\{a\},\natural_1^0)\|(\{b\},\natural_2^2)\|(\{c\},\natural_3^3)}]_{\approx}.$$

We have $DR_{ST}(\overline{E}) = \{s_2, s_5\}$*,* $DR_{WT}(\overline{E}) = \{s_3, s_4\}$ *and* $DR_V(\overline{E}) = \{s_1\}$*.*

In Figure 6, the transition system $TS(\overline{E})$ *is shown. The s-tangible and w-tangible states are depicted in ordinary and double ovals, respectively, and the vanishing ones are depicted in boxes.*

This example demonstrates a parallel composition of an immediate and two waiting multiactions with different delays. It shows that the immediate multiaction $(\{a\},\natural_1^0)$ *is always executed before any parallel with it waiting multiaction. Further, from the two parallel waiting multiactions, that* $(\{b\},\natural_2^2)$ *with a less delay* 2 *executed first in any case. Finally, the execution of the waiting multiaction* $(\{c\},\natural_3^3)$ *with a greater delay* 3 *leads to an absorbing state. Thus, in spite of parallelism of those three deterministic multiactions, they are executed sequentially in fact, in the increasing order of their (different) delays. That sequence also includes the empty set, executed after the immediate multiaction* $(\{a\},\natural_1^0)$*, since the waiting multiaction* $(\{b\},\natural_2^2)$

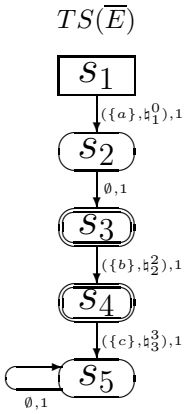


FIG. 6. The transition system of $\overline{E}$ for $E = (\{a\},\natural_1^0)\|(\{b\},\natural_2^2)\|(\{c\},\natural_3^3)$

with a less delay will then need a passage of one time unit (one time tick) for its timer value (RTE) become 1 *and it can be executed itself. Though the example is not complex, it shows a transition system with all three types of states: s-tangible, w-tangible and vanishing.*

Example 16. *Let* $E = (\{a\},\natural_1^3)\|(\{b\},\frac{1}{3})$. $DR(\overline{E})$ *consists of the equivalence classes*

$$s_1 = [\overline{(\{a\},\natural_1^3)^3}\|\overline{(\{b\},\tfrac{1}{3})}]_{\approx}, \quad s_2 = [\overline{(\{a\},\natural_1^3)^2}\|\overline{(\{b\},\tfrac{1}{3})}]_{\approx},$$
$$s_3 = [\overline{(\{a\},\natural_1^3)^2}\|\underline{(\{b\},\tfrac{1}{3})}]_{\approx}, \quad s_4 = [\overline{(\{a\},\natural_1^3)^1}\|\overline{(\{b\},\tfrac{1}{3})}]_{\approx},$$
$$s_5 = [\overline{(\{a\},\natural_1^3)^1}\|\underline{(\{b\},\tfrac{1}{3})}]_{\approx}, \quad s_6 = [\underline{(\{a\},\natural_1^3)}\|\overline{(\{b\},\tfrac{1}{3})}]_{\approx},$$
$$s_7 = [\underline{(\{a\},\natural_1^3)\|(\{b\},\tfrac{1}{3})}]_{\approx}.$$

We have $DR_{ST}(\overline{E}) = \{s_1, s_2, s_3, s_6, s_7\}$, $DR_{WT}(\overline{E}) = \{s_4, s_5\}$ *and* $DR_V(\overline{E}) = \emptyset$.

In Figure 7, the transition system $TS(\overline{E})$ *is shown. The s-tangible and w-tangible states are depicted in ordinary and double ovals, respectively.*

This example demonstrates a parallel composition of waiting and stochastic multiactions. It shows that the stochastic multiaction $(\{b\},\frac{1}{3})$ *can be executed until the timer value of the waiting multiaction* $(\{a\},\natural_1^3)$ *becomes* 1*, after which only the waiting multiaction can be executed in the next moment. The execution of the latter leads to an absorbing state either directly or indirectly, via executing a possible empty loop, followed (via sequential composition) by the stochastic multiaction* $(\{b\},\frac{1}{3})$ *that has not been executed in the preceding states.*

Example 17. *Let* $E = ((\{a\},\natural_1^2)\|(\{\hat{a}\},\natural_2^2))$ sy a rs a. $DR(\overline{E})$ *consists of the equivalence classes*

$$s_1 = [(\overline{(\{a\},\natural_1^2)^2}\|\overline{(\{\hat{a}\},\natural_2^2)^2})\ \mathsf{sy}\ a\ \mathsf{rs}\ a]_{\approx}, \quad s_2 = [(\overline{(\{a\},\natural_1^2)^1}\|\overline{(\{\hat{a}\},\natural_2^2)^1})\ \mathsf{sy}\ a\ \mathsf{rs}\ a]_{\approx},$$
$$s_3 = [\underline{((\{a\},\natural_1^2)\|(\{\hat{a}\},\natural_2^2))\ \mathsf{sy}\ a\ \mathsf{rs}\ a}]_{\approx}.$$

We have $DR_{ST}(\overline{E}) = \{s_1, s_3\}$, $DR_{WT}(\overline{E}) = \{s_2\}$ *and* $DR_V(\overline{E}) = \emptyset$.

In Figure 8, the transition system $TS(\overline{E})$ *is shown. The s-tangible and w-tangible states are depicted in ordinary and double ovals, respectively.*

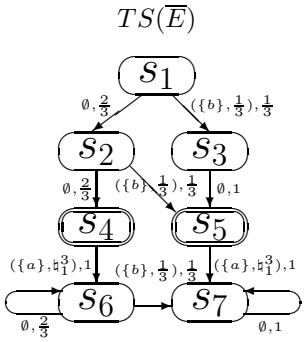


FIG. 7. The transition system of $\overline{E}$ for $E = (\{a\}, \natural_1^3) \| (\{b\}, \frac{1}{3})$

This example demonstrates a parallel composition of two waiting multiactions $(\{a\}, \natural_1^2)$ *and* $(\{\hat{a}\}, \natural_2^2)$*, whose multiaction parts are singleton multisets with an action* a *and its conjugate* $\hat{a}$*, respectively. The resulting composition is synchronized and then restricted by that action, which (and its conjugate) therefore "disappears" from the composite process behaviour. From the initial state, only the empty multiset of activities is executed that decrements by one the values of the timers. That evolution follows by the execution of a new waiting multiaction* $(\emptyset, \natural_3^2)$ *with the empty multiaction part, resulted from synchronization of the two waiting multiactions, which leads to an absorbing state.*

Note that the timer values of the two waiting multiactions and that of the new waiting multiaction $(\emptyset, \natural_3^2)$ *(being their synchronous product) coincide until all of them remain enabled with the time progress. Thus, it is very useful that the expression syntax preserves such two enabled synchronized waiting multiactions, removed by restriction from the behaviour, since their timer values suggest that of their synchronous product, which is not explicit in the syntax. Thus, the timer values of those two "virtual" enabled waiting multiactions cannot just be marked as undefined in the syntax, provided that one keeps track of the timer value of their synchronous product being only implicit in the syntax.*

If both synchronized waiting multiactions lose their enabledness with the time progress then their synchronous product $(\emptyset, \natural_3^2)$ *also loses its enabledness and all of them obviously loose their timer value annotations. It may happen that one of the synchronized waiting multiactions loses its enabledness (for example, when a conflicting waiting multiaction is executed) while the other one keeps its enabledness. Then their synchronous product also loses its enabledness, together with its timer value annotation. In such a case, the timer value of the enabled synchronized waiting multiaction does not suggest anymore that of the synchronous product. That "saved" timer value merely decrements with every time tick unless it becomes equal to* 1*, after which either the enabled synchronized waiting multiaction is executed or it cannot be executed by some reason (for example, when affected by restriction) and then the timer value* 1 *remains unchanged with the time progress. To verify this, recall the empty move rule* **E** *from Table 3 and the definition of* $\circlearrowleft G$ *with* $\max\{1, \delta - 1\} = \max\{1, 0\} = 1$ *when* $\delta = 1$*.*

Example 18. *Let* $E = ((\{a\}, \natural_1^1); (\{b\}, \natural_2^3)) \| (\{\hat{b}\}, \natural_3^3))$ sy b*.* $DR(\overline{E})$ *consists of the equivalence classes*

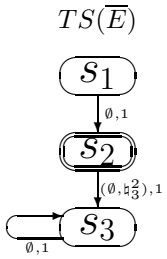


FIG. 8. The transition system of $\overline{E}$ for $E = ((\{a\},\natural_1^2)\|(\{\hat{a}\},\natural_2^2))$ sy a rs a

$$
\begin{aligned}
s_1 &= [(\overline{(\overline{(\{a\},\natural_1^1)^1};(\{b\},\natural_2^3))\|\overline{(\{\hat{b}\},\natural_3^3)^3}}) \text{ sy } b]_{\approx},\\
s_2 &= [((\overline{(\{a\},\natural_1^1);\overline{(\{b\},\natural_2^3)^3})\|\overline{(\{\hat{b}\},\natural_3^3)^2}}) \text{ sy } b]_{\approx},\\
s_3 &= [((\overline{(\{a\},\natural_1^1);\overline{(\{b\},\natural_2^3)^2})\|\overline{(\{\hat{b}\},\natural_3^3)^1}}) \text{ sy } b]_{\approx},\\
s_4 &= [((\overline{(\{a\},\natural_1^1);\overline{(\{b\},\natural_2^3)^1})\|\underline{(\{\hat{b}\},\natural_3^3)}) \text{ sy } b]_{\approx},\\
s_5 &= [(\underline{((\{a\},\natural_1^1);(\{b\},\natural_2^3))\|(\{\hat{b}\},\natural_3^3))} \text{ sy } b]_{\approx}.
\end{aligned}
$$

We have $DR_{ST}(\overline{E}) = \{s_2, s_5\}$, $DR_{WT}(\overline{E}) = \{s_1, s_3, s_4\}$ *and* $DR_V(\overline{E}) = \emptyset$.

In Figure 9, the transition system $TS(\overline{E})$ *is shown. The s-tangible and w-tangible states are depicted in ordinary and double ovals, respectively.*

This example demonstrates a parallel composition of two subprocesses. The first subprocess is a sequential composition of two waiting multiactions $(\{a\},\natural_1^1)$ *and* $(\{b\},\natural_2^3)$*. The second subprocess consists of a single waiting multiaction* $(\{\hat{b}\},\natural_3^3)$*. The resulting composition is synchronized by the action* b*, which (and its conjugate) therefore "disappears" from the behaviour of their synchronous product. From the initial state, only the waiting multiaction* $(\{a\},\natural_1^1)$ *is executed and the timer of the newly enabled waiting multiaction* $(\{b\},\natural_2^3)$ *starts with the value* 3 *while the timer value* 3 *of* $(\{\hat{b}\},\natural_3^3)$ *is decreased by one and becomes* 2*. That evolution follows by the execution of the empty multiset of activities that further decrements the values of those timers that become* 2 *and* 1*, respectively. Then the waiting multiaction* $(\{\hat{b}\},\natural_3^3)$ *is executed and its timer value annotation disappears while the timer value of* $(\{b\},\natural_2^3)$ *becomes* 1*. Then the execution of waiting multiaction* $(\{b\},\natural_2^3)$ *finally leads to an absorbing state.*

Thus, the new waiting multiaction $(\emptyset,\natural_5^3)$*, resulted from synchronization of* $(\{b\},\natural_2^3)$ *and* $(\{\hat{b}\},\natural_3^3)$*, cannot be executed, since those synchronized waiting multiactions cannot be executed together (in parallel) in any reachable state. Note that a synchronous product cannot be executed even if one (the latest, in case the timers are disbalanced) of the synchronized activities cannot be executed. Then only the* maximum *timer value of the two synchronized waiting multiactions suggests the timer value of their synchronous product* $(\emptyset,\natural_5^3)$*, until all of them remain enabled with the time progress. The enabledness keeps the corresponding timer value annotations present in the syntax and those values* defined*. Each defined timer value of* $(\{b\},\natural_2^3)$ *is always less by one than that of* $(\{\hat{b}\},\natural_3^3)$*, since the execution of the former waiting multiaction is delayed for one time unit due to the execution of the preceding* $(\{a\},\natural_1^1)$*. Then simultaneous starting the timers of the two synchronized waiting multiactions is prevented, resulting in the disbalanced timers. If just one timer*

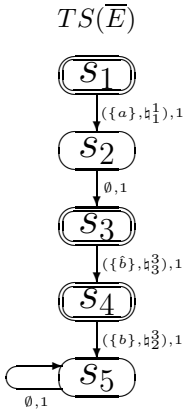


FIG. 9. The transition system of $\overline{E}$ for $E = ((((\{a\},\natural_1^1);(\{b\},\natural_2^3))\|(\{\hat{b}\},\natural_3^3))\ \mathsf{sy}\ b$

value of the two synchronized waiting multiactions is undefined then that of their synchronous product is undefined too, since it is not enabled in that case.

Example 19. *Let* $E = ((((\{a\},\natural_1^1);(\{b,\hat{x}\},\natural_2^0))\|(((\{x\},\natural_3^0)[](\{c\},\natural_4^1)))\ \mathsf{sy}\ x\ \mathsf{rs}\ x$. $DR(\overline{E})$ *consists of the equivalence classes*

$$s_1 = [(\overline{(((\{a\},\natural_1^1)^1};(\{b,\hat{x}\},\natural_2^0))\|((\{x\},\natural_3^0)[]\overline{(\{c\},\natural_4^1)^1}))\ \mathsf{sy}\ x\ \mathsf{rs}\ x]_{\approx} =$$
$$[(\overline{(((\{a\},\natural_1^1)^1};(\{b,\hat{x}\},\natural_2^0))\|\overline{((\{x\},\natural_3^0)[](\{c\},\natural_4^1)^1)})\ \mathsf{sy}\ x\ \mathsf{rs}\ x]_{\approx},$$
$$s_2 = [((((\{a\},\natural_1^1);\overline{(\{b,\hat{x}\},\natural_2^0)})\|\underline{(((\{x\},\natural_3^0)[](\{c\},\natural_4^1)))}\ \mathsf{sy}\ x\ \mathsf{rs}\ x]_{\approx}.$$

We have $DR_{ST}(\overline{E}) = \{s_2\},\ DR_{WT}(\overline{E}) = \{s_1\}$ *and* $DR_V(\overline{E}) = \emptyset$.

In Figure 10, the transition system $TS(\overline{E})$ *is shown. The s-tangible and w-tangible states are depicted in ordinary and double ovals, respectively.*

This example demonstrates a parallel composition of two subprocesses, synchronized and then restricted by an auxiliary action that (and its conjugate) hereupon "disappears" from the composite process behaviour. The first subprocess is a sequential composition of the waiting $(\{a\},\natural_1^1)$ *and immediate* $(\{b,\hat{x}\},\natural_2^0)$ *multiactions. The second subprocess is a choice between the immediate* $(\{x\},\natural_3^0)$ *and waiting* $(\{c\},\natural_4^1)$ *multiactions. The immediate multiactions* $(\{b,\hat{x}\},\natural_2^0)$ *and* $(\{x\},\natural_3^0)$ *in the first and second subprocesses are synchronized via an auxiliary action* x *that (and its conjugate) is then removed from the behaviour by the restriction operation. Since those immediate multiactions are within coverage of restriction by the auxiliary action, they cannot be executed. The new immediate multiaction* $(\{b\},\natural_5^0)$*, resulted from that synchronization can only be executed if the waiting multiaction* $(\{a\},\natural_1^1)$ *(preceding it via sequential composition) in the first subprocess has occurred and the waiting multiaction* $(\{c\},\natural_4^1)$ *(conflicting with it via the choice composition) in the second subprocess has not occurred. Since only maximal multisets of parallel waiting multiactions may be executed, the waiting multiactions in both the subprocesses must occur, thus preventing execution of the new immediate multiaction* $(\{b\},\natural_5^0)$*, generated by synchronization.*

Example 20. *Let* $E = ((((\{a\},\natural_1^2);(\{b,\hat{x}\},\natural_2^2))\|(((\{x\},\natural_3^2)[](\{c\},\natural_4^2)))\ \mathsf{sy}\ x\ \mathsf{rs}\ x$. $DR(\overline{E})$ *consists of the equivalence classes*

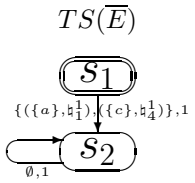


FIG. 10. The transition system of $\overline{E}$ for
$E = ((((\{a\},\natural_1^1);(\{b,\hat{x}\},\natural_2^0))\|(((\{x\},\natural_3^0)[](\{c\},\natural_4^1)))\ \mathsf{sy}\ x\ \mathsf{rs}\ x$

$$s_1 = [(((\overline{(\{a\},\natural_1^2)^2};(\{b,\hat{x}\},\natural_2^2))\|(((\{x\},\natural_3^2)^2[]\overline{(\{c\},\natural_4^2)^2}))\ \mathsf{sy}\ x\ \mathsf{rs}\ x]_{\approx} =$$
$$[(((\overline{(\{a\},\natural_1^2)^2};(\{b,\hat{x}\},\natural_2^2))\|(\overline{(\{x\},\natural_3^2)^2}[](\{c\},\natural_4^2)^2))\ \mathsf{sy}\ x\ \mathsf{rs}\ x]_{\approx},$$
$$s_2 = [(((\overline{(\{a\},\natural_1^2)^1};(\{b,\hat{x}\},\natural_2^2))\|(((\{x\},\natural_3^2)^1[]\overline{(\{c\},\natural_4^2)^1}))\ \mathsf{sy}\ x\ \mathsf{rs}\ x]_{\approx} =$$
$$[(((\overline{(\{a\},\natural_1^2)^1};(\{b,\hat{x}\},\natural_2^2))\|(\overline{(\{x\},\natural_3^2)^1}[](\{c\},\natural_4^2)^1))\ \mathsf{sy}\ x\ \mathsf{rs}\ x]_{\approx},$$
$$s_3 = [((((\{a\},\natural_1^2);\underline{\overline{(\{b,\hat{x}\},\natural_2^2)^2}})\|\underline{(((\{x\},\natural_3^2)[](\{c\},\natural_4^2)))}\ \mathsf{sy}\ x\ \mathsf{rs}\ x]_{\approx},$$
$$s_4 = [((((\{a\},\natural_1^2);\underline{\overline{(\{b,\hat{x}\},\natural_2^2)^1}})\|\underline{(((\{x\},\natural_3^2)[](\{c\},\natural_4^2)))}\ \mathsf{sy}\ x\ \mathsf{rs}\ x]_{\approx}.$$

We have $DR_{ST}(\overline{E}) = \{s_1, s_3, s_4\}$, $DR_{WT}(\overline{E}) = \{s_2\}$ *and* $DR_V(\overline{E}) = \emptyset$.

In Figure 11, the transition system $TS(\overline{E})$ *is shown. The s-tangible and w-tangible states are depicted in ordinary and double ovals, respectively.*

This example is a modification of the previous Example 19 by replacing all the immediate multiactions with the waiting ones and by setting to 2 *the delays of all the waiting multiactions from the syntax. Thus, we examine a compound process, constructed with parallelism, synchronization and restriction operations from the following two subprocesses. The first subprocess is a sequential composition of two waiting multiactions* $(\{a\},\natural_1^2)$ *and* $(\{b,\hat{x}\},\natural_2^2)$. *The second subprocess is a choice between other two waiting multiactions* $(\{x\},\natural_3^2)$ *and* $(\{c\},\natural_4^2)$. *The second waiting multiaction* $(\{b,\hat{x}\},\natural_2^2)$ *in the first subprocess and the first waiting multiaction* $(\{x\},\natural_3^2)$ *in the second subprocess are synchronized via an auxiliary action* x *that (and its conjugate) is then removed from the behaviour by the restriction operation. The new waiting multiaction* $(\{b\},\natural_5^2)$, *resulted from that synchronization has the same delay* 2 *as the two synchronized waiting multiactions. It can only be executed if the first waiting multiaction* $(\{a\},\natural_1^2)$ *(preceding it via sequential composition) in the first subprocess has occurred and the second waiting multiaction* $(\{c\},\natural_4^2)$ *(conflicting with it via the choice composition) in the second subprocess has not occurred. Since only maximal multisets of parallel waiting multiactions may be executed, the mentioned ("first in first" and "second in second") waiting multiactions in both the subprocesses must occur, thus preventing execution of the new waiting multiaction* $(\{b\},\natural_5^2)$, *generated by synchronization.*

The overlined second waiting multiaction in the first subprocess is within coverage of restriction by the auxiliary action. Consider the state, reached from the initial state by execution of the empty multiset of activities, followed by the parallel execution of the mentioned ("first in first" and "second in second") waiting multiactions. After the empty multiset execution from the considered state, the associated timer value of that overlined waiting multiaction is decremented to 1. *Then an absorbing state is reached, from which only the empty loop is possible, which leaves that timer*

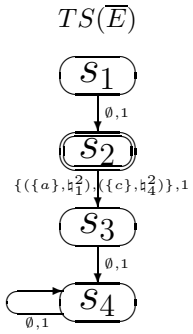


FIG. 11. The transition system of $\overline{E}$ for
$E = (((\{a\},\natural_1^2);(\{b,\hat{x}\},\natural_2^2))\|((\{x\},\natural_3^2)[](\{c\},\natural_4^2)))$ sy x rs x

value 1 *unchanged though. To verify this, recall the empty move rule* **E** *from Table 3 and the definition of* $\circlearrowleft G$ *with* $\max\{1,\delta-1\} = \max\{1,0\} = 1$ *when* $\delta = 1$.

Example 21. *Let* $E = (((\{a\},\natural_1^2);(\{b,\hat{x}\},\natural_2^2))\|((\{x\},\natural_3^2)[](\{c\},\natural_4^2)))$ sy x. $DR(\overline{E})$ *consists of the equivalence classes*

$$
\begin{aligned}
s_1 &= [(((\overline{(\{a\},\natural_1^2)^2};(\{b,\hat{x}\},\natural_2^2))\|(\overline{(\{x\},\natural_3^2)^2}[](\{c\},\natural_4^2)^2))\ \mathsf{sy}\ x]_{\approx} = \\
&\quad [(((\overline{(\{a\},\natural_1^2)^2};(\{b,\hat{x}\},\natural_2^2))\|((\{x\},\natural_3^2)^2[]\overline{(\{c\},\natural_4^2)^2}))\ \mathsf{sy}\ x]_{\approx}, \\
s_2 &= [(((\overline{(\{a\},\natural_1^2)^1};(\{b,\hat{x}\},\natural_2^2))\|(\overline{(\{x\},\natural_3^2)^1}[](\{c\},\natural_4^2)^1))\ \mathsf{sy}\ x]_{\approx} = \\
&\quad [(((\overline{(\{a\},\natural_1^2)^1};(\{b,\hat{x}\},\natural_2^2))\|((\{x\},\natural_3^2)^1[]\overline{(\{c\},\natural_4^2)^1}))\ \mathsf{sy}\ x]_{\approx}, \\
s_3 &= [((\{a\},\natural_1^2);\overline{(\{b,\hat{x}\},\natural_2^2)^2})\|\underline{((\{x\},\natural_3^2)[](\{c\},\natural_4^2))}\ \mathsf{sy}\ x]_{\approx}, \\
s_4 &= [((\{a\},\natural_1^2);\overline{(\{b,\hat{x}\},\natural_2^2)^1})\|\underline{((\{x\},\natural_3^2)[](\{c\},\natural_4^2))}\ \mathsf{sy}\ x]_{\approx}, \\
s_5 &= [\underline{((\{a\},\natural_1^2);(\{b,\hat{x}\},\natural_2^2))\|((\{x\},\natural_3^2)[](\{c\},\natural_4^2))\ \mathsf{sy}\ x}]_{\approx}.
\end{aligned}
$$

We have $DR_{ST}(\overline{E}) = \{s_1,s_3,s_5\}$, $DR_{WT}(\overline{E}) = \{s_2,s_4\}$ *and* $DR_V(\overline{E}) = \emptyset$.

In Figure 12, the transition system $TS(\overline{E})$ *is shown. The s-tangible and w-tangible states are depicted in ordinary and double ovals, respectively.*

This example is a modification of the previous Example 20 by removing restriction from the syntax. Thus, we examine a compound process, constructed with parallelism and synchronization operations from the two subprocesses being a sequential composition of two waiting multiactions $(\{a\},\natural_1^2)$ *and* $(\{b,\hat{x}\},\natural_2^2)$ *and a choice between other two waiting multiactions* $(\{x\},\natural_3^2)$ *and* $(\{c\},\natural_4^2)$*, respectively. All the four waiting multiactions have the same delay* 2*. The second waiting multiaction* $(\{b,\hat{x}\},\natural_2^2)$ *in the first subprocess and the first waiting multiaction* $(\{x\},\natural_3^2)$ *in the second subprocess are synchronized via an auxiliary action* x*. The new waiting multiaction* $(\{b\},\natural_5^2)$*, resulted from that synchronization has the same delay* 2 *as the two synchronized waiting multiactions. It can only be executed if the first waiting multiaction* $(\{a\},\natural_1^2)$ *(preceding it via sequential composition) in the first subprocess has occurred and the second waiting multiaction* $(\{c\},\natural_4^2)$ *(conflicting with it via the choice composition) in the second subprocess has not occurred. Since only maximal multisets of parallel waiting multiactions may be executed, the mentioned ("first in first" and "second in second") waiting multiactions in the subprocesses must occur, thus*

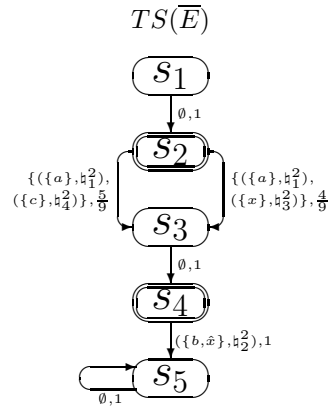


FIG. 12. The transition system of $\overline{E}$ for
$E = ((((\{a\},\natural_1^2);(\{b,\hat{x}\},\natural_2^2))\|((\{x\},\natural_3^2)[](\{c\},\natural_4^2)))\ \mathsf{sy}\ x$

preventing execution of the new waiting multiaction $(\{b\},\natural_5^2)$*, generated by synchronization. The alternative maximal multiset of parallel waiting multiactions that may be executed from the same state consists of the "first in first"* $(\{a\},\natural_1^2)$ *and "first in second"* $(\{x\},\natural_3^2)$ *waiting multiactions in the subprocesses, but the 'first in second" waiting multiaction* $(\{x\},\natural_3^2)$ *is the second of the two synchronized waiting multiactions, and its occurrence prevents execution of their synchronous product* $(\{b\},\natural_5^2)$*.*

Example 22. *Consider the expression* $\mathsf{Stop} = (\{g\},\frac{1}{2})\ \mathsf{rs}\ g$ *specifying the nonterminating process that performs only empty loops with probability* 1*.*

Let $E = [(\{a\},\frac{1}{2}) * ((\{b\},\natural_1^1)[](((\{c\},\natural_2^1);(\{d\},\frac{1}{3}))) * \mathsf{Stop}]$*.* $DR(\overline{E})$ *consists of the equivalence classes*

$$
\begin{aligned}
s_1 &= [[\overline{(\{a\},\tfrac{1}{2})} * ((\{b\},\natural_1^1)[](((\{c\},\natural_2^1);(\{d\},\tfrac{1}{3}))) * \mathsf{Stop}]]_{\approx},\\
s_2 &= [[(\{a\},\tfrac{1}{2}) * (\overline{(\{b\},\natural_1^1)^1}[](((\{c\},\natural_2^1)^1;(\{d\},\tfrac{1}{3}))) * \mathsf{Stop}]]_{\approx} =\\
&\quad [[(\{a\},\tfrac{1}{2}) * ((\{b\},\natural_1^1)^1[](\overline{((\{c\},\natural_2^1)^1;(\{d\},\tfrac{1}{3}))) * \mathsf{Stop}]]_{\approx},\\
s_3 &= [[(\{a\},\tfrac{1}{2}) * ((\{b\},\natural_1^1)[](((\{c\},\natural_2^1);\overline{(\{d\},\tfrac{1}{3})})) * \mathsf{Stop}]]_{\approx}.
\end{aligned}
$$

We have $DR_{ST}(\overline{E}) = \{s_1, s_3\},\ DR_{WT}(\overline{E}) = \{s_2\}$ *and* $DR_V(\overline{E}) = \emptyset$*.*

In Figure 13, the transition system $TS(\overline{E})$ *is presented. The s-tangible states are depicted in ovals and the vanishing ones are depicted in boxes.*

This example demonstrates an infinite iteration loop. The loop is preceded with the iteration initiation, modeled by a (first) stochastic multiaction $(\{a\},\frac{1}{2})$*. The iteration body that corresponds to the loop consists of the choice between two conflicting waiting multiactions* $(\{b\},\natural_1^1)$ *and* $(\{c\},\natural_2^1)$ *with the same delay* 1*, the second of them followed (via sequential composition) by a (second) stochastic multiaction* $(\{d\},\frac{1}{3})$*. Hence, the iteration loop actually consists of the two alternative subloops, such that the first one is a self-loop (one-state loop from a state to itself) with the first waiting multiaction* $(\{b\},\natural_1^1)$*, and the second one* $(\{c\},\natural_2^1)$ *is a two-state loop with an intermediate state, reached after the second waiting multiaction has been executed, and from which the second stochastic multiaction* $(\{d\},\frac{1}{3})$ *is then started. Thus, the iteration generates the self-loop with probability less than one*

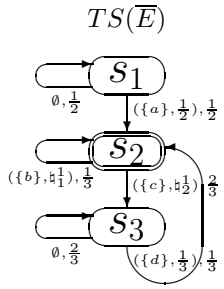


FIG. 13. The transition system of $\overline{E}$ for $E = [(\{a\}, \frac{1}{2}) * ((\{b\}, \natural_1^1)[] ((\{c\}, \natural_2^1); (\{d\}, \frac{1}{3}))) * \mathsf{Stop}]$

(since the two-state loop from the same state has a non-zero probability) from the states in which only waiting multiactions are executed. The iteration termination Stop *demonstrates an empty behaviour, assuring that the iteration does not reach its final state after any number of repeated executions of its body.*

Example 23. *Let* $E = [(\{a\}, \rho) * ((\{b\}, \natural_k^1); ((((\{c\}, \natural_l^0); (\{d\}, \theta))[](((\{e\}, \natural_m^0); (\{f\}, \phi)))) * \mathsf{Stop}]$*, where* $\rho, \theta, \phi \in (0; 1)$ *and* $k, l, m \in \mathbb{R}_{>0}$*.* $DR(\overline{E})$ *consists of the equivalence classes*

$$s_1 = [[\overline{(\{a\}, \rho)} * ((\{b\}, \natural_k^1); ((((\{c\}, \natural_l^0); (\{d\}, \theta))[](((\{e\}, \natural_m^0); (\{f\}, \phi)))) * \mathsf{Stop}]]_{\approx},$$
$$s_2 = [[(\{a\}, \rho) * \overline{((\{b\}, \natural_k^1)^1}; ((((\{c\}, \natural_l^0); (\{d\}, \theta))[](((\{e\}, \natural_m^0); (\{f\}, \phi)))) * \mathsf{Stop}]]_{\approx},$$
$$s_3 = [[(\{a\}, \rho) * ((\{b\}, \natural_k^1); (\overline{(((\{c\}, \natural_l^0)}; (\{d\}, \theta))[](((\{e\}, \natural_m^0); (\{f\}, \phi)))) * \mathsf{Stop}]]_{\approx} =$$
$$[[(\{a\}, \rho) * ((\{b\}, \natural_k^1); ((((\{c\}, \natural_l^0); (\{d\}, \theta))[]\overline{((\{e\}, \natural_m^0)}; (\{f\}, \phi)))) * \mathsf{Stop}]]_{\approx},$$
$$s_4 = [[(\{a\}, \rho) * ((\{b\}, \natural_k^1); ((((\{c\}, \natural_l^0); \overline{(\{d\}, \theta)})[](((\{e\}, \natural_m^0); (\{f\}, \phi)))) * \mathsf{Stop}]]_{\approx},$$
$$s_5 = [[(\{a\}, \rho) * ((\{b\}, \natural_k^1); ((((\{c\}, \natural_l^0); (\{d\}, \theta))[](((\{e\}, \natural_m^0); \overline{(\{f\}, \phi)})))) * \mathsf{Stop}]]_{\approx}.$$

We have $DR_{ST}(\overline{E}) = \{s_1, s_4, s_5\}$*,* $DR_{WT}(\overline{E}) = \{s_2\}$ *and* $DR_V(\overline{E}) = \{s_3\}$*.*

In Figure 14, the transition system $TS(\overline{E})$ *is presented. The s-tangible and w-tangible states are depicted in ordinary and double ovals, respectively, and the vanishing ones are depicted in boxes.*

This example demonstrates an infinite iteration loop. The loop is preceded with the iteration initiation, modeled by a stochastic multiaction $(\{a\}, \rho)$*. The iteration body that corresponds to the loop consists of a waiting multiaction* $(\{b\}, \natural_k^1)$*, followed (via sequential composition) by the probabilistic choice, modeled via two conflicting immediate multiactions* $(\{c\}, \natural_l^0)$ *and* $(\{e\}, \natural_m^0)$*, followed by different stochastic multiactions* $(\{d\}, \theta)$ *and* $(\{f\}, \phi)$*. The iteration termination* Stop *demonstrates an empty behaviour, assuring that the iteration does not reach its final state after any number of repeated executions of its body.*

Let us interpret E *as a specification of the travel system. A tourist visits regularly new cities. After seeing the sights of the current city, he goes to the next city by the nearest train or bus available at the station. Buses depart less frequently than trains, but the next city is quicker reached by bus than by train. We suppose that the stay duration in every city (being a constant), the departure numbers of trains and buses, as well as their speeds* do not depend *on a particular city, bus or train. The travel route has been planned so that the distances between successive cities* coincide.

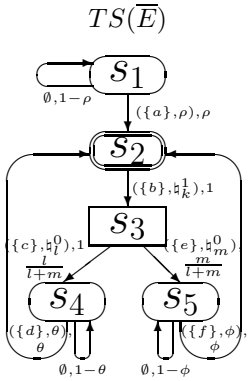


FIG. 14. The transition system of $\overline{E}$ for $E = [(\{a\},\rho) * ((\{b\},\natural_k^1); ((((\{c\},\natural_l^0); (\{d\},\theta))[](\{e\},\natural_m^0); (\{f\},\phi)))) * \mathsf{Stop}]$

The meaning of actions from the syntax of E is as follows. The action a corresponds to the system activation (the travel route has been planned) that takes a time, geometrically distributed with the parameter ρ. The action b represents the completion of looking round the current city and coming to the city station that takes a fixed time equal to 1 *(say, one hour) for every city. The actions c and e correspond to the urgent getting on bus and train, respectively, and thus model the choice between these two transport facilities. The weights of the two corresponding immediate multiactions suggest that every l departures of buses take the same time as m departures of trains ($l < m$), hence, a bus departs with the probability $\frac{l}{l+m}$ while a train departs with the probability $\frac{m}{l+m}$. The actions d and f correspond to the coming in a city by bus and train, respectively, that takes a time, geometrically distributed with the parameters θ and ϕ, respectively ($\theta > \phi$).*

The meaning of states from $DR(\overline{E})$ is the following. The s-tangible state s_1 corresponds to staying at home and planning the future travel. The w-tangible state s_2 means residence in a city for exactly one time unit (hour). The vanishing state s_3 with zero residence time represents instantaneous stay at the city station, signifying that the tourist does not wait there for departure of the transport. The s-tangible states s_4 and s_5 correspond to going by bus and train, respectively.

Due to the time constraints and since waiting multiactions may be preempted by stochastic ones, some simple dynamic expressions can have complex transition systems (Examples 11–16, 18, 21), or vice versa (Examples 17, 19, 20, 22, 23).

4. Conclusion

In this paper, we have proposed a discrete time stochastic extension dtsdPBC of PBC, enriched with deterministic multiactions. The calculus has a parallel step operational semantics, based on labeled probabilistic transition systems. A number of examples has demonstrated as construction of such transition systems with s-tangible, w-tangible and vanishing states for the dynamic expressions with different types of multiactions (stochastic, immediate and waiting) and various operations, as the specification capabilities of the calculus and particular features of its semantics.

The advantage of our framework is twofold. First, one can specify in it concurrent composition and synchronization of (multi)actions, whereas this is not possible in

classical Markov chains. Second, algebraic formulas represent processes in a more compact way than PNs and allow one to apply syntactic transformations and comparisons. Process algebras are compositional by definition and their operations naturally correspond to operators of programming languages. Hence, it is much easier to construct a complex model in the algebraic setting than in PNs. The complexity of PNs generated for practical models in the literature demonstrates that it is not straightforward to construct such PNs directly from the system specifications. dtsdPBC is well suited for the discrete time applications, whose discrete states change with a global time tick, such as business processes, neural and transportation networks, computer and communication systems, timed web services [53], as well as for those, in which the distributed architecture or the concurrency level should be preserved while modeling and analysis, such as genetic regulatory and cellular signalling networks (featuring maximal parallelism) in biology [11] (remember that, in step semantics, we have additional transitions due to concurrent executions). dtsdPBC is also capable to model parallel systems with fixed durations of the typical activities (loading, processing, transfer, repair, low-level events) and stochastic durations of the randomly occurring activities (arrival, departure, failure), including industrial, manufacturing, queueing, computing and network systems.

In particular, we have adopted for dtsdPBC all examples of the expressions, ct-boxes and inferences by the transition rules from tPBC [23]. Whereas the examples from that paper explore only some selected state-transition sequences (paths), we always construct the complete transition systems of the expressions. We have observed that in our framework we have no difficulties like those in tPBC, which have forced to allow illegal transition sequences. In tPBC, the increasing timers are associated with the overlines and underlines of multiactions and suggest the ages of the corresponding markings in the respective boxes. In dtsdPBC, the decreasing (up to the value 1) timers are associated with the enabled waiting multiactions and specify their remaining times to execute (RTEs), like the timers of the enabled deterministic transitions in DTDSPNs from [60, 61, 59]. Besides such a PNs intuition, making difference between markings (overlines and underlines) and timers of (waiting) multiactions offers more syntactical flexibility to express their progress in time. The decreasing timers allow us to avoid problems with infinitely growing timer values in the deadlocked and final (absorbing) states. Each decreasing timer should start with a certain value that cannot be suggested by the current marking, but such an initial value is the delay of the waiting multiaction the timer is associated with.

It is known that combining time restrictions, parallelism and compositionality usually leads to many technical difficulties, so that the formal models possessing all the mentioned properties have almost not been proposed in the literature, in spite of the investigations in the related areas (for example, discrete time, generally distributed delays, non-interleaving functional semantics in the SPA framework). To solve the mentioned problem, some new (not existing in dtsiPBC) notions and constructions have been introduced in dtsdPBC, such as deterministic multiactions, decreasing timers of waiting multiactions, enabledness of activities, saturation with the timer values, timers discarding and decreasing operations, extended Can and Now functions, s-tangible and w-tangible dynamic expressions and states, inaction and action rules respecting waiting multiactions, empty moves, reachability of dynamic expressions, transition systems with 3 types of states and 4 types of transitions (unlike 2 types of states and 3 types of transitions in dtsiPBC). Thus,

the main advantages of dtsdPBC are the flexible multiaction labels, deterministic and stochastic multiactions, powerful operations, as well as its step operational semantics permitting parallel (simultaneous) execution of activities at time ticks.

In the following research, we intend to propose a Petri net denotational semantics of dtsdPBC, like it has been done for dtsiPBC in terms of LDTSIPNs and dtsi-boxes. For this purpose, we shall present a subclass of labeled discrete time stochastic and deterministic Petri nets (LDTSDPNs), based on the extension of DTSPNs with transition labeling and deterministic transitions, called dtsd-boxes. Further, a technique of performance evaluation in the framework of the calculus will be presented that will explore the corresponding stochastic process, which is a semi-Markov chain (SMC). It will be proved that the underlying discrete time Markov chain (DTMC) or its reduction (RDTMC) by eliminating vanishing states may alternatively and suitably be studied for that purpose. We plan to define behavioural equivalences for dtsdPBC, such as the step stochastic bisimulation one, aiming to reduce behaviour of the algebraic processes by quotienting their transition systems and Markov chains. Such a reduction should simplify the functional (qualitative) and performance (quantitative) analysis. We would like to construct some application examples demonstrating expressiveness of the calculus and application of the behavioural analysis and performance evaluation, both simplified using quotienting by step stochastic bisimulation. Future work could also consist in constructing a congruence relation for dtsdPBC, i.e. the equivalence that withstands application of all operations of the algebra. The first possible candidate is a stronger version of step stochastic bisimulation equivalence, defined via transition systems equipped with two extra transitions skip and redo, like those from sPBC [25]. Moreover, recursion operation could be added to dtsdPBC to increase further specification power of the algebra. It would be very interesting to implement the class of DTSDPNs, to be able to specify them and then model their behaviour by constructing the reahability graphs. Note that even DTSPNs of M.K. Molloy [38, 39] have never been implemented. Mostly interleaving and continuous-time variants of stochastic or timed PNs have been implemented so far.

References

[1] W.M.P. van der Aalst, K.M. van Hee, H.A. Reijers, *Analysis of discrete-time stochastic Petri nets*, Statistica Neerlandica, **54**:2 (2000), 237–255. http://tmitwww.tm.tue.nl/staff/hreijers/H.A. Reijers Bestanden/Statistica.pdf. MR1794979

[2] G. Balbo, *Introduction to stochastic Petri nets*, Lecture Notes in Computer Science, **2090** (2001), 84–155. Zbl 0990.68092

[3] G. Balbo, *Introduction to generalized stochastic Petri nets*, Lecture Notes in Computer Science, **4486** (2007), 83–131. Zbl 1323.68400

[4] J.A. Bergstra, J.W. Klop, *Algebra of communicating processes with abstraction*, Theoretical Computer Science, **37** (1985), 77–121. MR0796314

[5] M. Bernardo, M. Bravetti, *Reward based congruences: can we aggregate more?* Lecture Notes in Computer Science, **2165** (2001), 136–151. MR1904353

[6] M. Bernardo, R. Gorrieri, *A tutorial on EMPA: a theory of concurrent processes with nondeterminism, priorities, probabilities and time*, Theoretical Computer Science, **202** (1998), 1–54. MR1626813

[7] E. Best, R. Devillers, J.G. Hall, *The box calculus: a new causal algebra with multi-label communication*, Lecture Notes in Computer Science, **609** (1992), 21–69. MR1253529

[8] E. Best, R. Devillers, M. Koutny, *Petri net algebra*, EATCS Monographs on Theoretical Computer Science, Springer, 2001. MR1932732

[9] E. Best, M. Koutny, *A refined view of the box algebra*, Lecture Notes in Computer Science, **935** (1995), 1–20. MR1461021

[10] T. Bolognesi, F. Lucidi, S. Trigila, *From timed Petri nets to timed LOTOS*, Proc. IFIP WG 6.1 10^{th} Int. Symposium on Protocol Specification, Testing and Verification 1990, Ottawa, Canada, 1–14, North-Holland, Amsterdam, The Netherlands, 1990.

[11] N. Bonzanni, K.A. Feenstra, W. Fokkink, E. Krepska, *What can formal methods bring to systems biology?* Lecture Notes in Computer Science, **5850** (2009), 16–22.

[12] E. Brinksma, H. Hermanns, *Process algebra and Markov chains*, Lecture Notes in Computer Science, **2090** (2001), 183–231. Zbl 0990.68021

[13] E. Brinksma, J.-P. Katoen, R. Langerak, D. Latella, *A stochastic causality-based process algebra*, The Computer Journal, **38**:7 (1995), 552–565.

[14] G. Bucci, L. Sassoli, E. Vicario, *Correctness verification and performance analysis of real-time systems using stochastic preemptive time Petri nets*. IEEE Transactions on Software Engineering, **31**:11 (2005), 913–927.

[15] G. Ciardo, *Discrete-time Markovian stochastic Petri nets*, *Computations with Markov Chains: Proc. 2^{nd} Int. Workshop on the Numerical Solution of Markov Chains (NSMC) 1995* (W.J. Stewart, ed.), Raleigh, NC, USA, January 1995, 339–358, Kluwer, Boston, MA, USA, 1995. http://www.cs.ucr.edu/~ciardo/pubs/1995NSMC-Discrete.pdf Zbl 0862.60079

[16] R.J. van Glabbeek, S.A. Smolka, B. Steffen, *Reactive, generative, and stratified models of probabilistic processes*, Information and Computation, **121**:1 (1995), 59–80. MR1347332

[17] H.M. Hanish, *Analysis of place/transition nets with timed-arcs and its application to batch process control*, Lecture Notes in Computer Science, **691** (1993), 282–299.

[18] H. Hermanns, M. Rettelbach, *Syntax, semantics, equivalences and axioms for MTIPP*, Proc. 2^{nd} Int. Workshop on Process Algebras and Performance Modelling (PAPM) 1994 (U. Herzog, M. Rettelbach, eds.), Regensberg / Erlangen, Germany, July 1994, Arbeitsberichte des IMMD, **27**:4 (1994), 71–88. http://ftp.informatik.uni-erlangen.de/local/inf7/papers/Hermanns/syntax_semantics_equivalences_axioms_for_MTIPP.ps.gz

[19] J. Hillston, *The nature of synchronisation*, Proc. 2^{nd} Int. Workshop on Process Algebra and Performance Modelling (PAPM) 1994 (U. Herzog, M. Rettelbach, eds.), Regensberg / Erlangen, Germany, July 1994, Arbeitsberichte des IMMD, **27**:4 (1994), 51–70. http://www.dcs.ed.ac.uk/pepa/synchronisation.pdf

[20] J. Hillston, *A compositional approach to performance modelling*, Cambridge University Press, Cambridge, UK, 1996. http://www.dcs.ed.ac.uk/pepa/book.pdf MR1427945

[21] C.A.R. Hoare, *Communicating sequential processes*, Prentice-Hall, London, UK, 1985. http://www.usingcsp.com/cspbook.pdf MR0805324

[22] A. Horváth, A. Puliafito, M. Scarpa, M. Telek, *Analysis and evaluation of non-Markovian stochastic Petri nets*, Lecture Notes in Computer Science, 1786 (2000), 171–187. Zbl 0967.68114

[23] M. Koutny, *A compositional model of time Petri nets*, Lecture Notes in Computer Science, 1825 (2000), 303–322.

[24] H. Macià, V. Valero, D.C. Cazorla, F. Cuartero, *Introducing the iteration in sPBC*, Lecture Notes in Computer Science, **3235** (2004), 292–308. Zbl 1110.68420

[25] H. Macià, V. Valero, F. Cuartero, D. de Frutos, *A congruence relation for sPBC*, Formal Methods in System Design, **32**:2 (2008), 85–128. Zbl 1138.68040

[26] H. Macià, V. Valero, F. Cuartero, M.C. Ruiz, *sPBC: a Markovian extension of Petri box calculus with immediate multiactions*, Fundamenta Informaticae, **87**:3–4 (2008), 367–406. Zbl 1154.68092

[27] H. Macià, V. Valero, F. Cuartero, M.C. Ruiz, I.V. Tarasyuk, *Modelling a video conference system with sPBC*, Applied Mathematics and Information Sciences **10**:2 (2016), 475–493.

[28] H. Macià, V. Valero, D. de Frutos, *sPBC: a Markovian extension of finite Petri box calculus*, Proc. 9^{th} IEEE Int. Workshop on Petri Nets and Performance Models (PNPM) 2001, Aachen, Germany, 207–216, IEEE Computer Society Press, 2001. http://www.info-ab.uclm.es/retics/publications/2001/pnpm01.ps

[29] J. Markovski, P.R. D'Argenio, J.C.M. Baeten, E.P. de Vink, *Reconciling real and stochastic time: the need for probabilistic refinement*, Formal Aspects of Computing, **24**:4–6 (2012), 497–518. MR2947264

[30] J. Markovski, E.P. de Vink, *Extending timed process algebra with discrete stochastic time*, Lecture Notes of Computer Science, **5140** (2008), 268–283. Zbl 1170.68542

[31] J. Markovski, E.P. de Vink, *Performance evaluation of distributed systems based on a discrete real- and stochastic-time process algebra*, Fundamenta Informaticae, **95**:1 (2009), 157–186. MR2590801

[32] O. Marroquín, D. de Frutos, *TPBC: timed Petri box calculus*, Technical Report, Departamento de Sistemas Infofmáticos y Programación, Universidad Complutense de Madrid, Madrid, Spain, 2000 (in Spanish).

[33] O. Marroquín, D. de Frutos, *Extending the Petri box calculus with time*, Lecture Notes in Computer Science, **2075** (2001), 303–322. Zbl 0986.68082

[34] M.A. Marsan, *Stochastic Petri nets: an elementary introduction*, Lecture Notes in Computer Science, **424** (1990), 1–29.

[35] M.A. Marsan, G. Balbo, G. Conte, S. Donatelli, G. Franceschinis, *Modelling with generalised stochastic Petri nets*, Wiley Series in Parallel Computing, John Wiley and Sons, 1995. http://www.di.unito.it/~greatspn/GSPN-Wiley/ Zbl 0843.68080

[36] Ph.M. Merlin, D.J. Farber, *Recoverability of communication protocols: implications of a theoretical study*, IEEE Transactions on Communications, **24**:9 (1976), 1036–1043. Zbl 0362.68096

[37] R.A.J. Milner, *Communication and concurrency*, Prentice-Hall, Upper Saddle River, NJ, USA, 1989. Zbl 0683.68008

[38] M.K. Molloy, *On the integration of the throughput and delay measures in distributed processing models*, Ph.D. thesis, Report, **CSD-810-921**, 108 p., University of California, Los Angeles, USA, 1981.

[39] M.K. Molloy, *Discrete time stochastic Petri nets*, IEEE Transactions on Software Engineering, **11**:4 (1985), 417–423. MR0788999

[40] A. Niaouris, *An algebra of Petri nets with arc-based time restrictions*, Lecture Notes in Computer Science, **3407** (2005), 447–462. Zbl 1109.68076

[41] A. Niaouris, M. Koutny, *An algebra of timed-arc Petri nets*, Technical Report, **CS-TR-895**, 60 p., School of Computer Science, University of Newcastle upon Tyne, UK, 2005. http://www.cs.ncl.ac.uk/publications/trs/papers/895.pdf

[42] C. Ramchandani, *Performance evaluation of asynchronous concurrent systems by timed Petri nets*, Ph.D. thesis, Department of Electrical Engineering, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, 1973.

[43] S.M. Ross, *Stochastic processes*, John Wiley and Sons, New York, USA, 1996. MR1373653

[44] I.V. Tarasyuk, *Discrete time stochastic Petri box calculus*, Berichte aus dem Department für Informatik, **3/05**, 25 p., Carl von Ossietzky Universität Oldenburg, Germany, 2005. http://itar.iis.nsk.su/files/itar/pages/dtspbcib_cov.pdf

[45] I.V. Tarasyuk, *Iteration in discrete time stochastic Petri box calculus*, Bulletin of the Novosibirsk Computing Center, Series Computer Science, IIS Special Issue, **24** (2006), 129–148. Zbl 1249.68132

[46] I.V. Tarasyuk, *Stochastic Petri box calculus with discrete time*, Fundamenta Informaticae, **76**:1–2 (2007), 189–218. MR2293057

[47] I.V. Tarasyuk, *Equivalence relations for modular performance evaluation in dtsPBC*, Mathematical Structures in Computer Science, **24**:1 (2014), e240103. MR3183269

[48] I.V. Tarasyuk, H. Macià, V. Valero, *Discrete time stochastic Petri box calculus with immediate multiactions*, Technical Report, **DIAB-10-03-1**, 25 p., Department of Computer Systems, High School of Computer Science Engineering, University of Castilla - La Mancha, Albacete, Spain, 2010. http://www.dsi.uclm.es/descargas/technicalreports/DIAB-10-03-1/dtsipbc.pdf

[49] I.V. Tarasyuk, H. Macià, V. Valero, *Discrete time stochastic Petri box calculus with immediate multiactions dtsiPBC*, Proc. 6^{th} Int. Workshop on Practical Applications of Stochastic Modelling (PASM) 2012 and 11^{th} Int. Workshop on Parallel and Distributed Methods in Verification (PDMC) 2012 (J. Bradley, K. Heljanko, W. Knottenbelt, N. Thomas, eds.), London, UK, 2012, Electronic Notes in Theoretical Computer Science, **296** (2013), 229–252.

[50] I.V. Tarasyuk, H. Macià, V. Valero, *Performance analysis of concurrent systems in algebra dtsiPBC*, Programming and Computer Software, **40**:5 (2014), 229–249.

[51] I.V. Tarasyuk, H. Macià, V. Valero, *Stochastic process reduction for performance evaluation in dtsiPBC*, Siberian Electronic Mathematical Reports, **12** (2015), 513–551. MR3493774

[52] I.V. Tarasyuk, H. Macià, V. Valero, *Stochastic equivalence for performance analysis of concurrent systems in dtsiPBC*, Siberian Electronic Mathematical Reports, **15** (2018), 1743–1812. Zbl 1414.60062

[53] V. Valero, M.E. Cambronero, *Using unified modelling language to model the publish/subscribe paradigm in the context of timed Web services with distributed resources*, Mathematical and Computer Modelling of Dynamical Systems, **23**:6 (2017), 570–594.

[54] R. Zijal, *Discrete time deterministic and stochastic Petri nets*, Proc. Int. Workshop on Quality of Communication-Based Systems 1994, Technical University of Berlin, Germany, 123–136, Kluwer Academic Publishers, 1995. Zbl 0817.68111

[55] R. Zijal, *Analysis of discrete time deterministic and stochastic Petri nets*, Ph.D. thesis, Technical University of Berlin, Germany, 1997.

[56] R. Zijal, G. Ciardo, *Discrete deterministic and stochastic Petri nets*, ICASE Report, **96-72**, 23 p., Institute for Computer Applications in Science and Engineering (ICASE), NASA, Langley Research Centre, Hampton, VA, USA, 1996. http://www.cs.odu.edu/~mln/ltrs-pdfs/icase-1996-72.pdf, http://www.dtic.mil/dtic/tr/fulltext/u2/a322409.pdf

[57] R. Zijal, G. Ciardo, G. Hommel, *Discrete deterministic and stochastic Petri nets*, Proc. 9^{th} ITG/GI Professional Meeting on Measuring, Modeling and Evaluation of Computer and Communication Systems (MMB) 1997 (K. Irmscher, Ch. Mittasch, K. Richter, eds.), Freiberg, Germany, 1997, Vol. **1**, 103–117, VDE-Verlag, Berlin, Germany, 1997. http://www.cs.ucr.edu/~ciardo/pubs/1997MMB-DDSPN.pdf

[58] R. Zijal, R. German, *A new approach to discrete time stochastic Petri nets*, Proc. 11^{th} Int. Conf. on Analysis and Optimization of Systems, Discrete Event Systems (DES) 1994 (G. Cohen, J.-P. Quadrat, eds.), Sophia-Antipolis, France, 1994, Lecture Notes in Control and Information Sciences, **199** (1994), 198–204.

[59] A. Zimmermann, *Modeling and evaluation of stochastic Petri nets with TimeNET 4.1*, Proc. 6^{th} Int. ICST Conf. on Performance Evaluation Methodologies and Tools (VALUETOOLS) 2012 (B. Gaujal, A. Jean-Marie, E. Jorswieck, A. Seuret, eds.), Cargèse, France, October 2012, 1–10, IEEE Computer Society Press, 2012. https://www.tu-ilmenau.de/fileadmin/public/sse/Veroeffentlichungen/2012/VALUETOOLS2012.pdf

[60] A. Zimmermann, J. Freiheit, R. German, G. Hommel, *Petri net modelling and performability evaluation with TimeNET 3.0*, Lecture Notes in Computer Science, **1786** (2000), 188–202. Zbl 0970.68665

[61] A. Zimmermann, J. Freiheit, G. Hommel, *Discrete time stochastic Petri nets for modeling and evaluation of real-time systems*, Proc. 9^{th} Int. Workshop on Parallel and Distributed Real Time Systems (WPDRTS) 2001, San Francisco, USA, 282–286, 2001. http://pdv.cs.tu-berlin.de/~azi/texte/WPDRTS01.pdf

Igor Valerievich Tarasyuk
A.P. Ershov Institute of Informatics Systems,
Siberian Branch of the Russian Academy of Sciences,
Acad. Lavrentiev pr. 6,
630090 Novosibirsk, Russian Federation
Email address: itar@iis.nsk.su