Time-Bounded Termination Analysis for Probabilistic Programs with Delays

Ming Xu[a,b], Yuxin Deng[a,*]

[a]*Shanghai Key Laboratory of Trustworthy Computing, MOE International Joint Lab of Trustworthy Software, and International Research Center of Trustworthy Software, East China Normal University, Shanghai 200062, China*
[b]*Institute of Intelligent Software, Guangzhou 511458, China*

Abstract

This paper investigates the model of probabilistic program with delays (PPD) that consists of a few program blocks. Performing each block has an additional time-consumption—waiting to be executed—besides the running time. We interpret the operational semantics of PPD by Markov automata with a cost structure on transitions. Our goal is to measure those individual execution paths of a PPD that terminates within a given time bound, and to compute the minimum termination probability, i. e. the termination probability under a demonic scheduler that resolves the nondeterminism inherited from probabilistic programs. When running time plus waiting time is bounded, the demonic scheduler can be determined by comparison between a class of well-formed real numbers. The method is extended to parametric PPDs. When only the running time is bounded, the demonic scheduler can be determined by real root isolation over a class of well-formed real functions under Schanuel's conjecture. Finally we give the complexity upper bounds of the proposed methods.

Keywords: Probabilistic program, program verification, termination analysis, quantitative evaluation, computer algebra, symbolic computation
2000 MSC: 68Q10, 68Q60, 68W30

1. Introduction

Program verification is an important way to ensure program correctness, i. e., programs should be designed to meet the predefined specification. The earliest verification work could date back to von Neumann's era [29]. A subsequent breakthrough [34] suggested "total correctness of (deterministic) programs equals partial correctness plus termination". Hence termination analysis is an essential part of program verification.

There has been a lot of work to reason about termination for deterministic programs, particularly for the challenging loop programs. A powerful approach is synthesising *ranking functions* that force program executions into a well-founded domain and thereby yield termination. By inductive assertion, linear ranking functions could be effectively synthesised [19, 49, 12]. The

*Corresponding author
Email addresses: mxu@cs.ecnu.edu.cn (Ming Xu), yxdeng@sei.ecnu.edu.cn (Yuxin Deng)

complexity of synthesis for linear loop programs was shown to be in **PTIME** when program variables range over rational numbers (or over real numbers), and be **coNP**-complete when program variables range over integers [10]. Verification tools like TERMINATOR [20] and APROVE [28] have been released to automatically prove termination and other liveness properties. Recently, ranking functions have been applied to recursion, resulting in *measure functions*, which turn out to be a sound and complete approach to proving worst-case bounds of nondeterministic recursive programs [16].

Randomised algorithms originated in 1970s, and have been widely applied to different fields in the last decade [30], such as machine learning (Bayesian inference), information security (randomised encryption), quantum computing. So the interest in probabilistic programs (PPs) that conveniently describe randomised algorithms is rapidly growing. The formal semantics and verification of PPs had been considered in [41] and [42, 51] respectively. As PPs have many new features—probabilistic choice, nondeterministic choice, sampling assignment and observation, the (sure) termination problem has many variants, partially listed as:

- **almost-sure termination**—Does an input PP terminate with probability one?
- **positive almost-sure termination**—Is the expected running time of an input PP finite?
- What is the termination probability of an input PP?

The former two are qualitative problems, and the last one is a quantitative problem. Unfortunately it has been revealed in [38] that all these termination problems for PPs are generally undecidable/incomputable, as well as the famous **halting** problem. That is, the three problems are far beyond the complexity hierarchy $\Sigma_1^0 \cap \Pi_1^0$. However, there still exist many methods to attack some subclasses of them in this active field.

Arons *et al.* proved liveness properties with probability one (e. g. almost-sure termination) over parametric programs using *planners* [4] that occasionally determine the outcome of a finite sequence of random choices, while the other random choices are performed nondeterministically. This idea was generalised in [26]. Another approach is computing a threshold for running time, after which the termination probability decreases exponentially [46]. Fioriti and Hermanns proposed a framework to prove almost-sure termination by *ranking super-martingales* [27], which is analogous to ranking functions on deterministic programs. Chakarov and Sankaranarayanan applied constraint-based techniques to generate linear ranking super-martingales [14]. Chatterjee *et al.* constructed polynomial ranking super-martingales through positivstellensatz's [15]. A **PTIME** procedure was given to synthesise lexicographic ranking super-martingales for linear PPs [3]. The method of ranking super-martingales is not only sound, but also semi-complete over some meaningful classes of PPs. As an application, they developed an efficient sound approach to derive expected-runtime bounds for the analysis of recurrence relations induced by randomised algorithms [17], and analysed expected sensitivity of PPs [54]. The method of ranking function also works for positive almost-sure termination of PPs [11].

On the other hand, McIver and Morgan generalised the *weakest preconditions* of Dijkstra (an approach to prove total correctness [23]) to the *weakest pre-expectations* [44] for analysing properties of probabilistic guarded command language [33], a dynamic logic of PPs and for establishing almost-sure termination [45]. Recently, Kaminski *et al.* presented a calculus of weakest pre-expectation style for obtaining bounds on the expected running time of PPs [39].

The running time of a program is an important measure of its performance. As we all know, a process in an operating system must be in one of the statuses: running, idle (waiting to be executed) or blocked. So the running time plus waiting time of program execution is also a worth-studying issue in performance analysis. A practical example is that the Linux system provides each process with three interval timers (see `https://linux.die.net/man/2/setitimer`):

- ITIMER_REAL that decrements in real time (on running time, waiting time and blocked time), and delivers the signal SIGALRM upon expiration;
- ITIMER_VIRTUAL that decrements only when the process is executing (on running time), and delivers the signal SIGVTALRM upon expiration;
- ITIMER_PROF that decrements both when the process executes and when the system is executing on behalf of the process (on running time and waiting time), and delivers the signal SIGPROF upon expiration.

A question in point is "what is the termination probability of a PP without delivering SIGPROF". In this paper we will consider the performance measure of PPs in terms of running time plus waiting time.

The waiting time can be reflected in the following manner. A program is split into finitely many blocks according to its parsing structure, so that a block would be loaded into memory only when it is necessary. For instance, consider the conditional branching statement

$$\texttt{if}(\xi)\,\{\mathbf{C}_1\}\,\texttt{else}\,\{\mathbf{C}_2\}\ .$$

If the guard ξ is evaluated to be true, only $\mathbf{C}_1$ is necessary to be loaded into memory; and otherwise only $\mathbf{C}_2$ is necessary. So it is wise to make $\mathbf{C}_1$ and $\mathbf{C}_2$ as two individual blocks following the judgement ξ. After splitting the program, every block has its own priority λ, which is usually set by program's annotations, operating systems or users. When a lot of blocks possibly from different programs are delivered to the processor, they have to wait for the processor to select one of them to execute. The waiting process is a continuous-time process of competitive selection that should satisfy the fairness requirements:

- each block would be surely selected (preferably) sooner or later,
- a block with a high priority is likely to be selected sooner than a block with a low priority.

To meet them, we adopt round-robin scheduling and lottery scheduling, such that the waiting process is supposed to follow an exponential distribution with rate λ (the priority). We call this model **probabilistic programs with delays** (PPDs).

In a PPD, if the time is unbounded, those waiting processes along any execution path must be completed with probability 1. It follows that the **time-unbounded termination** problem over PPDs is the same as that over PPs, and thus is incomputable by [38]. This paper, however, aims at the **time-bounded termination**, i. e. "what is the minimum termination probability for a PPD within a given time bound?" Here the bound is specified on the time of *individual* execution paths, rather than the average/expected time of *all* execution paths.

We first consider the case where running time plus waiting time is bounded. The PPD admits the nondeterminism that inherits from PP. The time-bounded termination problem can be resolved by determining the demonic scheduler that minimises the termination probability. We express the termination probabilities under schedulers as a class of well-formed real numbers, elements of the field extension $\mathbb{Q}(\mathrm{e}) : \mathbb{Q}$ (that is the smallest extension of the rational number field $\mathbb{Q}$ containing the Euler's constant e), so that the demonic scheduler can be determined by comparing those numbers. Furthermore, the method is extended to parametric PPDs that involve several parameters in probabilistic choices. By an extended usage of cylindrical algebraic decomposition [18], we can solve the constraints under the first-order logic $\mathfrak{L}(\mathbb{R}; >, =; +, \cdot; 0, 1, \mathrm{e})$ (that consists of polynomial equations and inequalities with coefficients taken from $\mathbb{Q}(\mathrm{e}) : \mathbb{Q}$ and variables ranging over $\mathbb{R}$). Based on that, the parameter space is eventually partitioned into finitely many subregions, on each of which a scheduler would be demonic everywhere. Then we consider the case where only the running time is bounded. After expressing the termination probabilities under schedulers as well-formed real functions in the form of the ring $\mathbb{Q}[\mathrm{e}, x, \mathrm{e}^x]$ (that is obtained by substituting $w = \mathrm{e}$ and $y = \mathrm{e}^x$ into the trivariate polynomial ring $\mathbb{Q}[w, x, y]$), we can determine the demonic scheduler by real root isolation over $\mathbb{Q}[\mathrm{e}, x, \mathrm{e}^x]$. We present a state-of-the-art algorithm for isolating real roots of those functions under Schanuel's conjecture. Based on that, the positive real line (interpreted as time) is partitioned into finitely many segments, on each of which a scheduler would be demonic everywhere. Finally we give the complexity upper bounds $\mathbf{EXP}^{\chi}$ of all the aforementioned proposed methods in the support of certain oracle machines χ.

The contribution of this paper is three-fold:

1. We propose the model of probabilistic program with delays (PPD).
2. We solve the time-bounded termination over PPDs by exact methods.
3. The termination problem motivates us to develop the real root isolation for a class of real functions under Schanuel's conjecture.

Other related work. The operational semantics of PP was interpreted by Markov decision processes (MDPs) with rewards/costs (interpreted as running time) in [39]. The PPD involves additional waiting processes. To express it, we adopt Markov automata (MA) with costs (interpreted as running and waiting time). The model of MA is too powerful to establish the general decidability/computability [25]. Fortunately, we will show that the time-bounded termination problem is solvable over those MA that interpret PPDs. A key feature of those MA is that there are only finitely many reachable states under a given time bound. Applying the technique of bounded model checking to the termination problem for PPs was considered in [36]. It can offer lower and upper bounds to the termination probability. The gap between the two bounds can be refined along with the increase in the search depth. But the gap may not converge zero for some instances, such as *non-decisive* probabilistic models [1]. In contrast, we consider the time-bounded termination problem in this paper, rather than the termination problem under bounded model checking.

The rest of the paper is structured as follows. Section 2 reviews some Markov models and the PP. Based on them, we introduce the model of PPD in Section 3. Section 4 considers the time-bounded termination problem, and solves it by comparing finitely many schedulers. We extend the above method to parametric PPDs in Section 5. Section 6 considers the case where only the running time is bounded. We analyse the complexity of our methods in Section 7 before concluding with Section 8.

2. Preliminaries

Here we review some basic notions on Markov models and probabilistic programs. Based on them, we will give a formal semantics of probabilistic programs with delays in the next section.

2.1. Markov chains, decision processes and automata

Let S be a countable set. A *(discrete) probability distribution* over S is a function $\mu : S \mapsto \mathbb{Q} \cap [0, 1]$, satisfying $\sum_{s \in S} \mu(s) = 1$. Let $Dist(S)$ denote the set of probability distributions over S.

Definition 2.1 (Markov chains)**.** *A Markov chain (MC for short) is a triple* $(S, \mathbf{P}, s_0)$*, where*

- S *is a countable set of states,*
- $\mathbf{P} : S \mapsto Dist(S)$ *is a probabilistic transition function, and*
- $s_0 \in S$ *is an initial state.*

The probabilistic transition function $\mathbf{P}$ prescribes the probability with which one state moves to another (in one step). For simplicity, we write $\mathbf{P}(s, s')$ instead of $\mathbf{P}(s)(s')$ for the transition probability from s to s'. The corresponding transition is written as $s \xrightarrow{\mathbf{P}(s,s')} s'$. A path is an infinite sequence of transitions such as $s_0 \xrightarrow{\mathbf{P}(s_0,s_1)} s_1 \xrightarrow{\mathbf{P}(s_1,s_2)} \cdots$ with $\mathbf{P}(s_k, s_{k+1}) > 0$ for each $k \geq 0$. It is well-known that a finite prefix, called a *fragment*, of such a path induces a *cylinder set* [8, Definition 10.9]; and all measurable sets of paths can be expressed as (countable) unions and complements of cylinder sets. For instance, the fragment $s_0 \xrightarrow{\mathbf{P}(s_0,s_1)} s_1 \xrightarrow{\mathbf{P}(s_1,s_2)} s_2 \xrightarrow{\mathbf{P}(s_2,s_3)} s_3$ corresponds to a cylinder set that is measured to have probability $\mathbf{P}(s_0, s_1) \cdot \mathbf{P}(s_1, s_2) \cdot \mathbf{P}(s_2, s_3)$.

Definition 2.2 (Markov decision processes)**.** *A Markov decision process (MDP for short) is a tuple* $(S, Act, \mathbf{P}, s_0)$*, where*

- S *is a countable set of states,*
- *Act is a nonempty set of actions,*
- $\mathbf{P} \subseteq S \times Act \times Dist(S)$ *is a probabilistic transition relation, and*
- $s_0 \in S$ *is an initial state.*

In an MDP, the probabilistic transition relation $\mathbf{P}$ gives the probability with which one state moves to another under an action: $\mathbf{P}(s, \alpha, s')$ represents the transition probability from s to s' under action $\alpha \in Act(s)$, where $Act(s)$ is the set of actions enabled at s, i. e. $\{\alpha \in Act \mid \exists \mu.\ (s, \alpha, \mu) \in \mathbf{P}\}$. The corresponding transition is now denoted by $s \xrightarrow{\alpha,\mathbf{P}(s,\alpha,s')} s'$. A path in an MDP is an infinite sequence of transitions, such as $s_0 \xrightarrow{\alpha_1,\mathbf{P}(s_0,\alpha_1,s_1)} s_1 \xrightarrow{\alpha_2,\mathbf{P}(s_1,\alpha_2,s_2)} \cdots$ with $\alpha_{k+1} \in Act(s_k)$ and $\mathbf{P}(s_k, \alpha_{k+1}, s_{k+1}) > 0$ for each $k \geq 0$. The nondeterminism in $\mathbf{P}$ could be resolved by a *scheduler* [8, Definition 10.91] that chooses one action each step a state faces a nondeterministic choice of outgoing transitions. Therefore, the behaviour of an MDP under a scheduler would degenerate as that of an MC.

Definition 2.3 (Markov automata)**.** *A Markov automaton (MA for short) is a tuple* $(S, Act, \mathbf{P}, \mathbf{R}, s_0)$*, where*

- $S = S_1 \cup S_2$ *with* $S_1 \cap S_2 = \emptyset$ *is a countable set of states,*
- *Act is a nonempty set of actions,*

- $\mathbf{P} \subseteq S_1 \times Act \times Dist(S)$ *is a probabilistic transition relation,*
- $\mathbf{R} \subseteq S_2 \times \mathbb{Z}^+ \times S$ *is a timed stochastic transition relation, and*
- $s_0 \in S$ *is an initial state.*

In an MA, each state has exactly one kind of outgoing transitions: probabilistic transitions or timed stochastic ones [25, 22]. So the set S of states is partitioned into two subsets S_1 and S_2 with $S_1 \cap S_2 = \emptyset$: for each $s \in S_1$, it has probabilistic transitions only; while for each $s \in S_2$, it has timed stochastic ones only. The transition rule $\mathbf{R}(s, \lambda, s')$ indicates that the move from s to s' follows the exponential distribution with rate λ, i. e. the probabilistic density function is $\lambda e^{-\lambda t}$, where t is a *random variable* interpreted as the sojourn time at s. The corresponding timed stochastic transition is written as $s \overset{\lambda}{\dashrightarrow} s'$. A path in an MA is an infinite sequence of transitions, such as $s_0 \xrightarrow{\alpha_1, \mathbf{P}(s_0,\alpha_1,s_1)} s_1 \overset{\lambda_2}{\dashrightarrow} s_2 \cdots$ with $\alpha_{k+1} \in Act(s_k)$ and $\mathbf{P}(s_k, \alpha_{k+1}, s_{k+1}) > 0$ for $s_k \in S_1$ while $(s_k, \lambda_{k+1}, s_{k+1}) \in \mathbf{R}$ for $s_k \in S_2$ and $k \geq 0$. The fragment $s_0 \xrightarrow{\alpha_1, \mathbf{P}(s_0,\alpha_1,s_1)} s_1 \overset{\lambda_2}{\dashrightarrow} s_2$ of those paths whose sojourn time at s_1 is bounded in I has probability $\mathbf{P}(s_0, \alpha_1, s_1) \cdot [e^{-\lambda_2 \inf I} - e^{-\lambda_2 \sup I}]$.

Let *Affine*(t) denote the set of affine expressions $a + b \cdot t$ where $a, b \in \mathbb{Z}$ are constants and t is an independent *random variable* for a timed stochastic transition. Generally speaking, the coefficients a and b in the affine expression $a + b \cdot t$ are rational numbers. Here, we impose the restriction that a and b are integers only for convenience, since the time variable t could be scalable to a small one $t' = t/c$ where c is the least common multiple of denominators of a and b, so that the resulting expression $a + b \cdot t = ac + bc \cdot t'$ has integer coefficients ac and bc. Then we can enrich MA by adding costs to transitions.

Definition 2.4 (Markov automata with costs)**.** *A Markov automaton with costs is a tuple* $(S, Act, \mathbf{P}, \mathbf{R}, s_0, Cost)$*, where*

- $(S, Act, \mathbf{P}, \mathbf{R}, s_0)$ *is an MA, and*
- $Cost : \mathbf{P} \cup \mathbf{R} \mapsto Affine(t)$ *with* $Cost(\mathbf{P}) \subset \mathbb{Z}$ *is the cost function defined on transitions.*

Although the costs are defined for individual transitions, we can talk about the cumulative costs of several consecutive transitions. For instance, suppose we have three transitions:

$$T_1 : s_0 \xrightarrow{\alpha_1, \mathbf{P}(s_0,\alpha_1,s_1)} s_1, \qquad T_2 : s_1 \overset{\lambda_2}{\dashrightarrow} s_2, \qquad T_3 : s_2 \overset{\lambda_3}{\dashrightarrow} s_3 \ .$$

The cumulative costs of going from s_0 to s_3 by executing T_1, T_2, and then T_3 is

$$Cost(T_1) + Cost(T_2) + Cost(T_3) \ .$$

2.2. Probabilistic programs

The probabilistic programs (PPs) that we will consider are defined by the syntax displayed in Table 1, in which x is a program variable, μ is a distribution expression with finitely many samples v, and ξ is a Boolean guard. A formal operational semantics of PPs will be given in the next section. We denote by $\mu(v)$ the probability value of sampling μ with v. In the special case $x :\approx \mu$ and $\mu(v) = 1$, we simply write $x := v$. Each of `skip`, judgement (deciding the truth of ξ) and assignment together with possible sampling (executing $x :\approx \mu$) costs one unit of time; other operations cost none. In the (probabilistic and nondeterministic) choices and

Syntax ($\mathbf{C}$)	Description
`empty`	Empty program
`skip`	Effectless operation
`halt`	Immediate termination
$x :\approx \mu$	Probabilistic assignment that samples an expression from μ and assigns it to variable x
$\mathbf{C};\mathbf{C}$	Sequential composition of statements
$\{\mathbf{C}\}\ [p]\ \{\mathbf{C}\}$	Probabilistic choice that executes the left branch with probability p and executes the right branch with probability $1 - p$
$\{\mathbf{C}\} \ \square\ \{\mathbf{C}\}$	Nondeterministic choice between statements to be resolved by a scheduler
`if`(ξ) $\{\mathbf{C}\}$ `else` $\{\mathbf{C}\}$	Conditional branching
`while`(ξ) $\{\mathbf{C}\}$	While loop
`observe`(ξ)	Observation that blocks all executions violating ξ and rescales the probability of the remaining executions

Table 1: Syntax of probabilistic programs

the conditional branching, the left and the right branches are supposed to be nonempty. For algorithmic purposes, the probability values $\mu(v)$ in sampling and p in probabilistic choices are supposed to be rational numbers. Interested readers can refer to [39] for more details, while the `observe` statement is separately addressed as one main feature of PP in [40].

3. Probabilistic Programs with Delays

Now we propose the model of probabilistic program with delays, and interpret its operational semantics using Markov automata with costs. In this paper, we consider the performance of a PP in terms of its running time plus the waiting time before being executed. The waiting time is reflected in the manner. A program can be split into finitely many blocks; performing each block requires the time-consumption—waiting to be executed.

3.1. Program splitting

There are many ways to split programs [43, Section 3.1]. A typical treatment is to cut the program at every branching and accumulating points of choices, branchings and loops, so that every cyclic execution would pass through at least one cut point. Blocks are nonempty segments of the program between program entrance, program exit, and those cut points. For instance, consider the following while loop

$$||^1\ \mathbf{C}_0;\ \texttt{while}(\xi)\ ||^2\ \{\mathbf{C}_1\};\ ||^3\ \mathbf{C}_2\ ||^4\ ,$$

where we use the symbols $||^i$ to indicate cut points in the program. Specifically, the above program has the beginning $||^1$, a branching point $||^2$, an accumulating point $||^3$ and the end $||^4$. By cutting the program at those points, we may obtain the three blocks:

- $\mathbf{B}_1 = \mathbf{C}_0; \xi$ (from the beginning $||^1$ to the branching point $||^2$),
- $\mathbf{B}_2 = \mathbf{C}_1; \xi$ (from the branching point $||^2$ to itself),

- $\mathbf{B}_3 = \mathbf{C}_2$ (from the branching point $\|^2$ directly to the end $\|^4$, skipping $\mathbf{C}_1$).

Every execution path through the loop can be read as a string in the form $\mathbf{B}_1\mathbf{B}_2^*\mathbf{B}_3$. We now formalise the above intuition of splitting a program into several blocks.

Definition 3.1 (Block)**.** *A block is either a program* $\mathbf{C}$ *or a program appended with a judgement of the form* $\mathbf{C};\xi$*. A block is primitive if it is free of choice, branching and loop.*

Every program block $\mathbf{B}$ can be split into a set $\mathcal{B}$ of primitive blocks according to the following function *split*.

$$
split(\mathbf{B}) = \begin{cases}
\emptyset & \text{if } \mathbf{B} \equiv \texttt{empty} \\
split(\mathbf{C}_0) \cup split(\mathbf{C}_1) \cup split(\mathbf{C}_2) \cup split(\mathbf{B}') & \text{if } \mathbf{B} \equiv \mathbf{C}_0;\ \{\mathbf{C}_1\}\ [p]\ \{\mathbf{C}_2\};\ \mathbf{B}' \\
 & \text{or } \mathbf{B} \equiv \mathbf{C}_0;\ \{\mathbf{C}_1\} \Box \{\mathbf{C}_2\};\ \mathbf{B}' \\
split(\mathbf{C}_0;\xi) \cup split(\mathbf{C}_1) \cup split(\mathbf{C}_2) \cup split(\mathbf{B}') & \text{if } \mathbf{B} \equiv \mathbf{C}_0;\ \texttt{if}(\xi)\ \{\mathbf{C}_1\}\ \texttt{else}\ \{\mathbf{C}_2\};\ \mathbf{B}' \\
split(\mathbf{C}_0;\xi) \cup split(\mathbf{C}_1;\xi) \cup split(\mathbf{B}') & \text{if } \mathbf{B} \equiv \mathbf{C}_0;\ \texttt{while}(\xi)\ \{\mathbf{C}_1\};\ \mathbf{B}' \\
split(\mathbf{C};\xi) \cup split(\mathbf{B}') & \text{if } \mathbf{B} \equiv \mathbf{C};\ \texttt{observe}(\xi);\ \mathbf{B}' \\
\{\mathbf{B}\} & \text{otherwise .}
\end{cases}
\tag{1}
$$

By the inductive definition in (1), we see that a primitive block cannot be further split. Every program execution can be expressed by a string over those primitive blocks (as the alphabet). In the presence of probabilistic and nondeterministic choices, a program may have several possible executions. We are interested in the primitive blocks that are first scheduled to be executed. Formally, given a block $\mathbf{B}$, we define the set of primitive blocks that can be first scheduled as *head*($\mathbf{B}$).

$$
head(\mathbf{B}) = \begin{cases}
\emptyset & \text{if } \mathbf{B} \equiv \texttt{empty} \\
head(\mathbf{C}_0) & \text{if } \mathbf{B} \equiv \mathbf{C}_0;\ \{\mathbf{C}_1\}\ [p]\ \{\mathbf{C}_2\};\ \mathbf{B}' \text{ and } \mathbf{C}_0 \not\equiv \texttt{empty} \\
 & \text{or } \mathbf{B} \equiv \mathbf{C}_0;\ \{\mathbf{C}_1\} \Box \{\mathbf{C}_2\};\ \mathbf{B}' \text{ and } \mathbf{C}_0 \not\equiv \texttt{empty} \\
head(\mathbf{C}_1) \cup head(\mathbf{C}_2) & \text{if } \mathbf{B} \equiv \texttt{empty};\ \{\mathbf{C}_1\}\ [p]\ \{\mathbf{C}_2\};\ \mathbf{B}' \\
 & \text{or } \mathbf{B} \equiv \texttt{empty};\ \{\mathbf{C}_1\} \Box \{\mathbf{C}_2\};\ \mathbf{B}' \\
head(\mathbf{C}_0;\xi) & \text{if } \mathbf{B} \equiv \mathbf{C}_0;\ \texttt{if}(\xi)\ \{\mathbf{C}_1\}\ \texttt{else}\ \{\mathbf{C}_2\};\ \mathbf{B}' \\
 & \text{or } \mathbf{B} \equiv \mathbf{C}_0;\ \texttt{while}(\xi)\ \{\mathbf{C}_1\};\ \mathbf{B}' \\
 & \text{or } \mathbf{B} \equiv \mathbf{C}_0;\ \texttt{observe}(\xi);\ \mathbf{B}' \\
\{\mathbf{B}\} & \text{otherwise .}
\end{cases}
\tag{2}
$$

Let $\mathcal{B}$ be a set. We write $\#\mathcal{B}$ for its cardinality. From the definition in (2), we see that $\#head(\mathbf{B}) > 1$ only when $\mathbf{B}$ starts with a probabilistic or nondeterministic choice. After the splitting, any block $\mathbf{B}$ in a nondeterministic branching is loaded into the memory iff the scheduler selects this $\mathbf{B}$; any block $\mathbf{B}$ following a judgement ξ in a conditional branching is loaded into the memory iff the evaluation of ξ under the values of program variables supports this $\mathbf{B}$. Thus the splitting can avoid loading unnecessary blocks and save memory space. However, for large programs, the granularity of splitting would be adjusted.

3.2. Introducing waiting time

Every primitive block has its own priority λ, which is usually set by program annotations, operating systems or users. When a lot of primitive blocks $\mathbf{B}$ possibly from different programs are delivered to a processor, they have to wait to be selected by the processor before execution. The *waiting process*, denoted by $\Diamond\mathbf{B}$, is typically a process of competitive selection that should satisfy the following fairness requirements:

- each block would be surely selected (preferably) sooner or later,
- a block with a high priority is likely to be selected sooner than a block with a low priority.

To meet them, the processor could adopt *round-robin scheduling* and *lottery scheduling* [53, Section 2.4], so that:

1. the processing time is split into many time segments of equally small length, called *quanta*, in order to have the high responsibility to primitive blocks;
2. let $\mathbf{B}_i$ $(1 \leq i \leq k)$ be all primitive blocks to be selected, each of which is allocated with a number, saying λ_i (the priority), of lottery tickets;
3. let T be the total number of lottery tickets the processor have during each quantum, which is greater than $\sum_{i=1}^{k} \lambda_i$;
4. for each quantum a random number r in $(0, T]$ is generated, and then
   - if $r \in (\sum_{j=1}^{i-1} \lambda_j, \sum_{j=1}^{i} \lambda_j]$ $(1 \leq i \leq k)$, the processor selects $\mathbf{B}_i$ to execute,
   - if $r \in (\sum_{i=1}^{k} \lambda_i, T]$, the processor does nothing as it is preserved only for emergency.

   In other words, it has probability λ_i/T to select $\mathbf{B}_i$ during the quantum, and probability $1 - \sum_{i=1}^{k} \lambda_i/T$ to do nothing.

Since the length of quantum is usually very small under round-robin scheduling, λ_i/T reflects the change rate of the probability of selecting $\mathbf{B}_i$, while $1 - \sum_{i=1}^{k} \lambda_i/T$ reflects that of the probability of doing nothing. For convenience, we assume that each quantum costs $1/T$ unit of time, i. e. T quanta cost one unit of time. It entails that the change rate per unit of time is simply λ_i and thus that the selecting process follows an exponential distribution with rate λ_i (the priority), i. e. the probability density function is $\lambda_i \mathrm{e}^{-\lambda_i t}$.

In the area of stochastic scheduling [55, 50, 48], exponential distributions are much popular. This design choice follows the model of continuous-time Markov chain (CTMC) [52, 7], which has been widely used to evaluate system performance and reliability characteristics such as the throughput of production lines, the mean time between failure in safety-critical systems, and bottlenecks in communication networks. For instance, the inter-arrival times and the job sizes were assumed in [31, 35] to be exponentially distributed. Moreover, a justification given in [13] is that the exponential distribution is reasonable to model a processing time when the only available information known on the processing time is its mean. In that setting, an exponential distribution implies a uniform conditional distribution for the processing time over an interval $[0, t]$ if the job is completed once before time t. The exponential distribution not only provides a precise mathematical characterization of the selecting process, but also would facilitate our analysis in the next section.

3.3. The model of probabilistic programs with delays

Under the aforementioned priority function, we call the model **probabilistic programs with delays** (PPD), and interpret its operational semantics by MA with costs.

Definition 3.2 (Probabilistic programs with delays)**.** *A probabilistic program with delays (PPD for short) is a pair* $(\mathbf{C}, \mathit{Rate})$*, where*

- $\mathbf{C}$ *is a probabilistic program, and*
- $\mathit{Rate} : \mathit{split}(\mathbf{C}) \mapsto \mathbb{Z}^+$ *is the priority function defined on the primitive blocks of* $\mathbf{C}$.

During an execution, we sometimes need to determine to which primitive block an appointed statement belongs for a fixed program splitting. So the function of *head* is extended as: $\mathit{head}(\mathbf{B}') = \mathit{head}(\mathbf{B})$, where $\mathbf{B}$ is a primitive block and $\mathbf{B}'$ is a nonempty suffix of $\mathbf{B}$. Let $[\![v]\!]_{\mathbf{val}}$ denote the value of expression v under the valuation $\mathbf{val}$ which is a map from program variables to their domains.

Definition 3.3. *For a PPD* $(\mathbf{C}_0, \mathit{Rate})$*, the MA with costs that interprets it is a tuple* $(S, \mathit{Act}, \longrightarrow, \dashrightarrow, s_0, \mathit{Cost})$*, in which:*

- $S = \{(\mathbf{val}, \mathbf{C}), \mathit{sink}, \lightning\} \cup \{(\mathbf{val}, \Diamond\mathbf{B}; \mathbf{C})\}$ *is a set of states, where*
  - $\mathbf{val}$ *is the current valuation of program variables;*
  - $\mathbf{C}$ *is the PPD to be executed;*
  - $\Diamond\mathbf{B}$ *with* $\mathbf{B} \in \mathit{split}(\mathbf{C}_0)$ *denotes a waiting process that appears only when accessing certain primitive block in* $\mathbf{C}_0$*; and*
  - sink *and* $\lightning$ *are two distinguished states.*
- $\mathit{Act} = \{\mathrm{L}, \mathrm{R}, \mathrm{D}\}$ *is a set of actions, where* D *is the default action.*
- $\longrightarrow \subseteq S \times \mathit{Act} \times \mathit{Dist}(S)$ *is a probabilistic transition relation, and* $\dashrightarrow \subseteq S \times \mathbb{Z}^+ \times S$ *is a timed stochastic transition relation. Costs are attached (from below) to those transitions. The transitions and their costs are defined by the rules in Table 2.*
- *The initial state* s_0 *is* $(\mathbf{val}_0, \Diamond\mathbf{B}_0; \mathbf{C}_0)$ *if* $\mathit{head}(\mathbf{C}_0) = \{\mathbf{B}_0\}$*, and* $(\mathbf{val}_0, \mathbf{C}_0)$ *otherwise, where all program variables are assigned as the default values, say* 0*, in* $\mathbf{val}_0$.

Table 2: The operational semantics of PPD

$$\frac{}{\mathit{sink} \xrightarrow[0]{\mathrm{D},1} \mathit{sink}} \qquad \frac{}{\lightning \xrightarrow[0]{\mathrm{D},1} \mathit{sink}} \qquad \frac{}{(\mathbf{val}, \mathtt{empty}) \xrightarrow[0]{\mathrm{D},1} \mathit{sink}} \qquad \frac{}{(\mathbf{val}, \mathtt{halt}; \mathbf{C}) \xrightarrow[0]{\mathrm{D},1} \mathit{sink}}$$

$$\frac{\mathit{head}(\mathtt{skip}; \mathbf{C}) = \mathit{head}(\mathbf{C}) \text{ or } \#\mathit{head}(\mathbf{C}) > 1}{(\mathbf{val}, \mathtt{skip}; \mathbf{C}) \xrightarrow[1]{\mathrm{D},1} (\mathbf{val}, \mathbf{C})} \qquad \frac{\mathit{head}(\mathtt{skip}; \mathbf{C}) \neq \mathit{head}(\mathbf{C}) = \{\mathbf{B}\}}{(\mathbf{val}, \mathtt{skip}; \mathbf{C}) \xrightarrow[1]{\mathrm{D},1} (\mathbf{val}, \Diamond\mathbf{B}; \mathbf{C})}$$

$$\frac{}{(\mathbf{val}, x :\approx \mu; \mathbf{C}) \xrightarrow[0]{\mathrm{D},\mu(v)} (\mathbf{val}, x := v; \mathbf{C})}$$

$$\frac{\mathit{head}(x := v; \mathbf{C}) = \mathit{head}(\mathbf{C}) \text{ or } \#\mathit{head}(\mathbf{C}) > 1}{(\mathbf{val}, x := v; \mathbf{C}) \xrightarrow[1]{\mathrm{D},1} (\mathbf{val}[[\![v]\!]_{\mathbf{val}}/x], \mathbf{C})} \qquad \frac{\mathit{head}(x := v; \mathbf{C}) \neq \mathit{head}(\mathbf{C}) = \{\mathbf{B}\}}{(\mathbf{val}, x := v; \mathbf{C}) \xrightarrow[1]{\mathrm{D},1} (\mathbf{val}[[\![v]\!]_{\mathbf{val}}/x], \Diamond\mathbf{B}; \mathbf{C})}$$

$$\frac{\#\mathit{head}(\mathbf{C}_1) > 1}{(\mathbf{val}, \{\mathbf{C}_1\}\ [p]\ \{\mathbf{C}_2\}; \mathbf{C}_3) \xrightarrow[0]{\mathrm{D},p} (\mathbf{val}, \mathbf{C}_1; \mathbf{C}_3)} \qquad \frac{\mathit{head}(\mathbf{C}_1) = \{\mathbf{B}_1\}}{(\mathbf{val}, \{\mathbf{C}_1\}\ [p]\ \{\mathbf{C}_2\}; \mathbf{C}_3) \xrightarrow[0]{\mathrm{D},p} (\mathbf{val}, \Diamond\mathbf{B}_1; \mathbf{C}_1; \mathbf{C}_3)}$$

$$\frac{\#\mathit{head}(\mathbf{C}_2) > 1}{(\mathbf{val}, \{\mathbf{C}_1\}\ [p]\ \{\mathbf{C}_2\}; \mathbf{C}_3) \xrightarrow[0]{\mathrm{D},1-p} (\mathbf{val}, \mathbf{C}_2; \mathbf{C}_3)}$$

$$\frac{\mathit{head}(\mathbf{C}_2) = \{\mathbf{B}_2\}}{(\mathbf{val}, \{\mathbf{C}_1\}\ [p]\ \{\mathbf{C}_2\}; \mathbf{C}_3) \xrightarrow[0]{\mathrm{D},1-p} (\mathbf{val}, \Diamond\mathbf{B}_2; \mathbf{C}_2; \mathbf{C}_3)}$$

$$\frac{\#\mathit{head}(\mathbf{C}_1) > 1}{(\mathbf{val}, \{\mathbf{C}_1\} \Box \{\mathbf{C}_2\}; \mathbf{C}_3) \xrightarrow[0]{\mathrm{L},1} (\mathbf{val}, \mathbf{C}_1; \mathbf{C}_3)} \qquad \frac{\mathit{head}(\mathbf{C}_1) = \{\mathbf{B}_1\}}{(\mathbf{val}, \{\mathbf{C}_1\} \Box \{\mathbf{C}_2\}; \mathbf{C}_3) \xrightarrow[0]{\mathrm{L},1} (\mathbf{val}, \Diamond\mathbf{B}_1; \mathbf{C}_1; \mathbf{C}_3)}$$

$$\frac{\#\mathit{head}(\mathbf{C}_2) > 1}{(\mathbf{val}, \{\mathbf{C}_1\} \Box \{\mathbf{C}_2\}; \mathbf{C}_3) \xrightarrow[0]{\mathrm{R},1} (\mathbf{val}, \mathbf{C}_2; \mathbf{C}_3)} \qquad \frac{\mathit{head}(\mathbf{C}_2) = \{\mathbf{B}_2\}}{(\mathbf{val}, \{\mathbf{C}_1\} \Box \{\mathbf{C}_2\}; \mathbf{C}_3) \xrightarrow[0]{\mathrm{R},1} (\mathbf{val}, \Diamond\mathbf{B}_2; \mathbf{C}_2; \mathbf{C}_3)}$$

$$\frac{[\![\xi]\!]_{\mathbf{val}} = \mathrm{true} \quad \#\mathit{head}(\mathbf{C}_1) > 1}{(\mathbf{val}, \texttt{if}(\xi)\ \{\mathbf{C}_1\}\ \texttt{else}\ \{\mathbf{C}_2\}; \mathbf{C}_3) \xrightarrow[1]{\mathrm{D},1} (\mathbf{val}, \mathbf{C}_1; \mathbf{C}_3)}$$

$$\frac{[\![\xi]\!]_{\mathbf{val}} = \mathrm{true} \quad \mathit{head}(\mathbf{C}_1) = \{\mathbf{B}_1\}}{(\mathbf{val}, \texttt{if}(\xi)\ \{\mathbf{C}_1\}\ \texttt{else}\ \{\mathbf{C}_2\}; \mathbf{C}_3) \xrightarrow[1]{\mathrm{D},1} (\mathbf{val}, \Diamond\mathbf{B}_1; \mathbf{C}_1; \mathbf{C}_3)}$$

$$\frac{[\![\xi]\!]_{\mathbf{val}} = \mathrm{false} \quad \#\mathit{head}(\mathbf{C}_2) > 1}{(\mathbf{val}, \texttt{if}(\xi)\ \{\mathbf{C}_1\}\ \texttt{else}\ \{\mathbf{C}_2\}; \mathbf{C}_3) \xrightarrow[1]{\mathrm{D},1} (\mathbf{val}, \mathbf{C}_2; \mathbf{C}_3)}$$

$$\frac{[\![\xi]\!]_{\mathbf{val}} = \mathrm{false} \quad \mathit{head}(\mathbf{C}_2) = \{\mathbf{B}_2\}}{(\mathbf{val}, \texttt{if}(\xi)\ \{\mathbf{C}_1\}\ \texttt{else}\ \{\mathbf{C}_2\}; \mathbf{C}_3) \xrightarrow[1]{\mathrm{D},1} (\mathbf{val}, \Diamond\mathbf{B}_2; \mathbf{C}_2; \mathbf{C}_3)}$$

$$\frac{[\![\xi]\!]_{\mathbf{val}} = \mathrm{true} \quad \#\mathit{head}(\mathbf{C}_1) > 1}{(\mathbf{val}, \texttt{while}(\xi)\ \{\mathbf{C}_1\}; \mathbf{C}_2) \xrightarrow[1]{\mathrm{D},1} (\mathbf{val}, \mathbf{C}_1; \texttt{while}(\xi)\ \{\mathbf{C}_1\}; \mathbf{C}_2)}$$

$$\frac{[\![\xi]\!]_{\mathbf{val}} = \mathrm{true} \quad \mathit{head}(\mathbf{C}_1) = \{\mathbf{B}_1\}}{(\mathbf{val}, \texttt{while}(\xi)\ \{\mathbf{C}_1\}; \mathbf{C}_2) \xrightarrow[1]{\mathrm{D},1} (\mathbf{val}, \Diamond\mathbf{B}_1; \mathbf{C}_1; \texttt{while}(\xi)\ \{\mathbf{C}_1\}; \mathbf{C}_2)}$$

$$\frac{[\![\xi]\!]_{\mathbf{val}} = \mathrm{false} \quad \#\mathit{head}(\mathbf{C}_2) > 1}{(\mathbf{val}, \texttt{while}(\xi)\ \{\mathbf{C}_1\}; \mathbf{C}_2) \xrightarrow[1]{\mathrm{D},1} (\mathbf{val}, \mathbf{C}_2)} \qquad \frac{[\![\xi]\!]_{\mathbf{val}} = \mathrm{false} \quad \mathit{head}(\mathbf{C}_2) = \{\mathbf{B}_2\}}{(\mathbf{val}, \texttt{while}(\xi)\ \{\mathbf{C}_1\}; \mathbf{C}_2) \xrightarrow[1]{\mathrm{D},1} (\mathbf{val}, \Diamond\mathbf{B}_2; \mathbf{C}_2)}$$

$$\frac{[\![\xi]\!]_{\mathbf{val}} = \mathrm{true} \quad \#\mathit{head}(\mathbf{C}) > 1}{(\mathbf{val}, \texttt{observe}(\xi); \mathbf{C}) \xrightarrow[1]{\mathrm{D},1} (\mathbf{val}, \mathbf{C})} \qquad \frac{[\![\xi]\!]_{\mathbf{val}} = \mathrm{true} \quad \mathit{head}(\mathbf{C}) = \{\mathbf{B}\}}{(\mathbf{val}, \texttt{observe}(\xi); \mathbf{C}) \xrightarrow[1]{\mathrm{D},1} (\mathbf{val}, \Diamond\mathbf{B}; \mathbf{C})}$$

$$\frac{[\![\xi]\!]_{\mathbf{val}} = \text{false}}{(\mathbf{val}, \texttt{observe}(\xi); \mathbf{C}_1) \xrightarrow[1]{\text{D},1} (\mathbf{val}, \lightning)} \qquad \frac{\mathit{Rate}(\mathbf{B}) = \lambda}{(\mathbf{val}, \Diamond\mathbf{B}; \mathbf{C}) \overset{\lambda}{\underset{t}{\dashrightarrow}} (\mathbf{val}, \mathbf{C})}$$

In the conclusion part of a rule, the superscript of the solid arrow gives the action and the probability of the probabilistic transition, whereas the subscript gives the cost. For instance, $\xrightarrow[0]{\text{D},1}$ gives action D, probability 1 and cost 0. In sum, `empty`, probabilistic sampling, probabilistic and nondeterministic choices have cost 0; while `skip`, assignment and judgement have cost 1. The superscript of the dashed arrow gives the rate of the timed stochastic transition, and the subscript gives the cost. For instance, $\overset{\lambda}{\underset{t}{\dashrightarrow}}$ gives rate λ and cost t.

In the above model, we can see that:

States (excluding $\lightning$ and sink) give the following information: (i) the current values of program variables prescribed by **val**, (ii) the statements **C** to be executed, and (iii) possibly the waiting processes $\Diamond\mathbf{B}$ when accessing primitive blocks in PPD. The execution of a PPD involves countably many states. Additionally, $\lightning$ is a distinguished state, reached after an unsuccessful `observe` statement; and $\Diamond\lightning$ denotes the set of paths that eventually reach the state $\lightning$. Similarly, sink is also a distinguished state, reached after a complete program execution, a `halt` statement, or $\lightning$; and $\Diamond\mathit{sink}$ denotes the set of paths, called *sinking* paths, that eventually reach the state sink.

Transitions take place when (i) `skip`, judgements and assignments are completed, (ii) `empty`, `halt` and sampling are completed, (iii) probabilistic and nondeterministic choices are resolved, and (iv) the waiting processes are completed. Let us have a close look at the rules for `skip` in Table 2. There are three disjoint cases:

1. When $\mathit{head}(\texttt{skip}; \mathbf{C}) = \mathit{head}(\mathbf{C})$, after executing `skip`, the remaining PPD **C** still stays in the same primitive block. So we proceed to tackle **C** without introducing any waiting process.
2. When $\#\mathit{head}(\mathbf{C}) > 1$, after executing `skip`, the remaining PPD **C** starts with a probabilistic or nondeterministic choice. We will resolve it later by the rules for $\{\mathbf{C}_1\}\ [p]\ \{\mathbf{C}_2\}$ or $\{\mathbf{C}_1\} \square \{\mathbf{C}_2\}$, and introduce the waiting process then.
3. When $\mathit{head}(\texttt{skip}; \mathbf{C}) \neq \mathit{head}(\mathbf{C})$ and $\mathit{head}(\mathbf{C}) = \{\mathbf{B}\}$, after executing `skip`, the PPD **C** definitely accesses a new primitive block **B**. So the waiting process $\Diamond\mathbf{B}$ is necessary to be introduced now.

Other rules can be analysed similarly.

Transition probabilities are taken from the corresponding sampling and probabilistic choices, actions are taken from nondeterministic choices, and the rates of timed stochastic transitions are taken from the priority λ of primitive blocks.

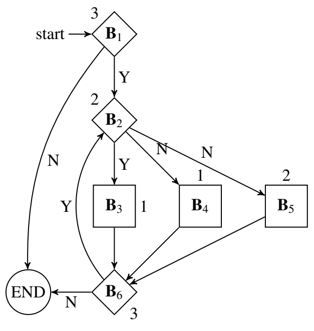
The cost of a probabilistic transition reflects the running time, and the cost of a timed stochastic transition reflects the waiting time.

So paths and schedulers of a PPD are those of the MA. All paths can be grouped into three disjoint classes [40]: *terminating*, *violating* and *nonterminating* ones. Denote them by $\Diamond\mathit{sink} \cap \neg\Diamond\lightning$, $\Diamond\mathit{sink} \cap \Diamond\lightning$, and $\neg\Diamond\mathit{sink}$, respectively. Note that the equality $\Pr(\Diamond\lightning) = \Pr(\Diamond\mathit{sink} \cap \Diamond\lightning)$ holds since the transition $\lightning \xrightarrow[0]{\text{D},1} \mathit{sink}$ is of probability 1.

Example 3.4 (Race between Tortoise and Hare)**.** *The program is given in Figure 1(a). Program variables h and t represent the positions of Hare and Tortoise respectively, r is a flag that is true iff Hare is just refreshed (then Hare can potentially jump farther).*

```
1: $h := 0$; $t := 4$; $r :=$ false
2: while $h < t$ do
3:     if $r$ then
4:         $h :\approx h + \text{Unif}[3..6]$; $r :=$ false
5:     else
6:         $\{h :\approx h + \text{Unif}[0..3]\} \,\Box\, \{r := \text{true}\}$
7:     $t := t + 1$
```

(a)



(b)

Figure 1: An example of probabilistic program

The program can be split inductively as:

$$\begin{aligned}
split(\mathbf{C}_0) &= split(\mathbf{B}_1) \cup split(\mathbf{C}_1;\ h < t) \\
&= \{\mathbf{B}_1\} \cup split(\mathbf{B}_2) \cup split(\mathbf{B}_3) \cup split(\mathbf{C}_2) \cup split(\mathbf{B}_6) \\
&= \{\mathbf{B}_1\} \cup \{\mathbf{B}_2\} \cup \{\mathbf{B}_3\} \cup split(\mathbf{B}_4) \cup split(\mathbf{B}_5) \cup \{\mathbf{B}_6\} \\
&= \{\mathbf{B}_1\} \cup \{\mathbf{B}_2\} \cup \{\mathbf{B}_3\} \cup \{\mathbf{B}_4\} \cup \{\mathbf{B}_5\} \cup \{\mathbf{B}_6\} \\
&= \{\mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3, \mathbf{B}_4, \mathbf{B}_5, \mathbf{B}_6\} \ ,
\end{aligned}$$

where

- $\mathbf{C}_0$ *denotes the whole program,*
- $\mathbf{C}_1$ *is the loop body (Lines 3–7),*
- $\mathbf{C}_2$ *is the nondeterministic choice (Line 6),*
- $\mathbf{B}_1 \equiv h := 0;\ t := 4;\ r := \text{false};\ h < t,$
- $\mathbf{B}_2 \equiv r,$
- $\mathbf{B}_3 \equiv h :\approx h + \text{Unif}[3..6];\ r := \text{false},$
- $\mathbf{B}_4 \equiv h :\approx h + \text{Unif}[0..3],$
- $\mathbf{B}_5 \equiv r := \text{true},$ *and*
- $\mathbf{B}_6 \equiv t := t + 1;\ h < t.$

Equipped with the priority function $Rate(\mathbf{B}_1) = Rate(\mathbf{B}_6) = 3$, $Rate(\mathbf{B}_2) = Rate(\mathbf{B}_5) = 2$ *and* $Rate(\mathbf{B}_3) = Rate(\mathbf{B}_4) = 1$, *the pair* $(\mathbf{C}_0, Rate)$ *forms a PPD. All these primitive blocks with the assumed priorities are displayed in the control flow graph (see Figure 1(b)), in which primitive blocks ending in judgements are denoted by diamond-shaped states.*

Consider the program execution that takes the left branch of Line 6 with sample 3 *for the first iteration and is interrupted after that assignment. It adopts the scheduler fragment "*DDDDDLDD*", or shortly "*L*" (the letter '*L*' is short for the left branch, '*R*' for the right branch, and the default latter '*D*' is omitted for conciseness), and accesses "*$\mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_4$*" in turn. The path fragment in PPD is shown in Figure 2.*

The path fragment has running time 6 *as the sum of costs attached to those probabilistic transitions prior to the state* $\boxed{\langle h = 3, t = 4, r = \text{false}\rangle,\ \textit{Lines 7 \& 2–7}}$. *In additional, it has three timed stochastic transitions. They represent the waiting processes for executing* $\mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_4$. *For instance, at the entrance of the program, we have to wait the processor to select the primitive block* $\mathbf{B}_1$ *for execution. We denote by* $\Diamond\mathbf{B}_1$ *this waiting process; and the current state is* $\boxed{\langle h = 0, t = 0, r = \text{false}\rangle, \Diamond\mathbf{B}_1;\ \textit{Lines 1–7}}$. *When it finishes, we would move to the next state* $\boxed{\langle h = 0, t = 0, r = \text{false}\rangle,\ \textit{Lines 1–7}}$. *The transition is timed stochastic with rate* 3 *(the priority of* $\mathbf{B}_1$*). So the waiting time is the sum of three independent random variables* t_1, t_2, t_3 *following the exponential distributions with rates* $3, 2, 1$ *respectively. In total, the running time plus waiting time of the fragment is* $6 + t_1 + t_2 + t_3$.

We remark that the model of PPD is a class of MAs equipped with the special cost structure; and that the time-bounded termination analysis for PPDs in the coming sections corresponds to the cost-bounded reachability analysis for those special MAs, which is different from the step-bounded reachability analysis, and is not for the general MAs.

From Figure 1(b) we also see one advantage of reasoning about PPDs rather than MAs. If we are only concerned about the high-level behaviour of the probabilistic program in Figure 1(a), the finite control flow graph with blocks as nodes suffices. But if we interpret that program as an MA without the special cost structure, we would need an infinite-state automaton because the uniform distribution Unif[0..3] in Line 6 might be executed infinitely many times and always yield very small values such that h never exceeds t, though this would happen with probability 0.

4. Time-Bounded Termination Analysis

In this section we study the following problem:

Problem 4.1 (Time-bounded termination)**.** *What is the minimum termination probability for a PPD when the total time (running time plus waiting time) is bounded?*

The nondeterminism in the PPD can be resolved by a demonic scheduler that minimises the termination problem for conservatism. So Problem 4.1 amounts to determining the demonic scheduler. We will do it by finitely many times of comparison between elements of the field extension $\mathbb{Q}(\mathrm{e}) : \mathbb{Q}$ (that is the smallest extension of the rational number field $\mathbb{Q}$ containing the Euler's constant e).

The PPDs would degenerate as PPs if the waiting processes are removed. That is, every PPD induces a PP. Correspondingly, a path in a PPD can be read as a series of probabilistic and timed stochastic transitions. If we remove those timed stochastic transitions, the resulting series of probabilistic transitions would be a path in the induced PP. Given a path of the induced PP, we can record the blocks with multiplicities that this path accesses in the control flow graph. To execute these blocks $\mathbf{B}$, one has to wait the timed stochastic transitions with their rates λ, like in a CTMC. Based on that, all paths in a PPD can be classified upon their projections in the induced PP; two paths of the same type in a PPD differ in their waiting time.

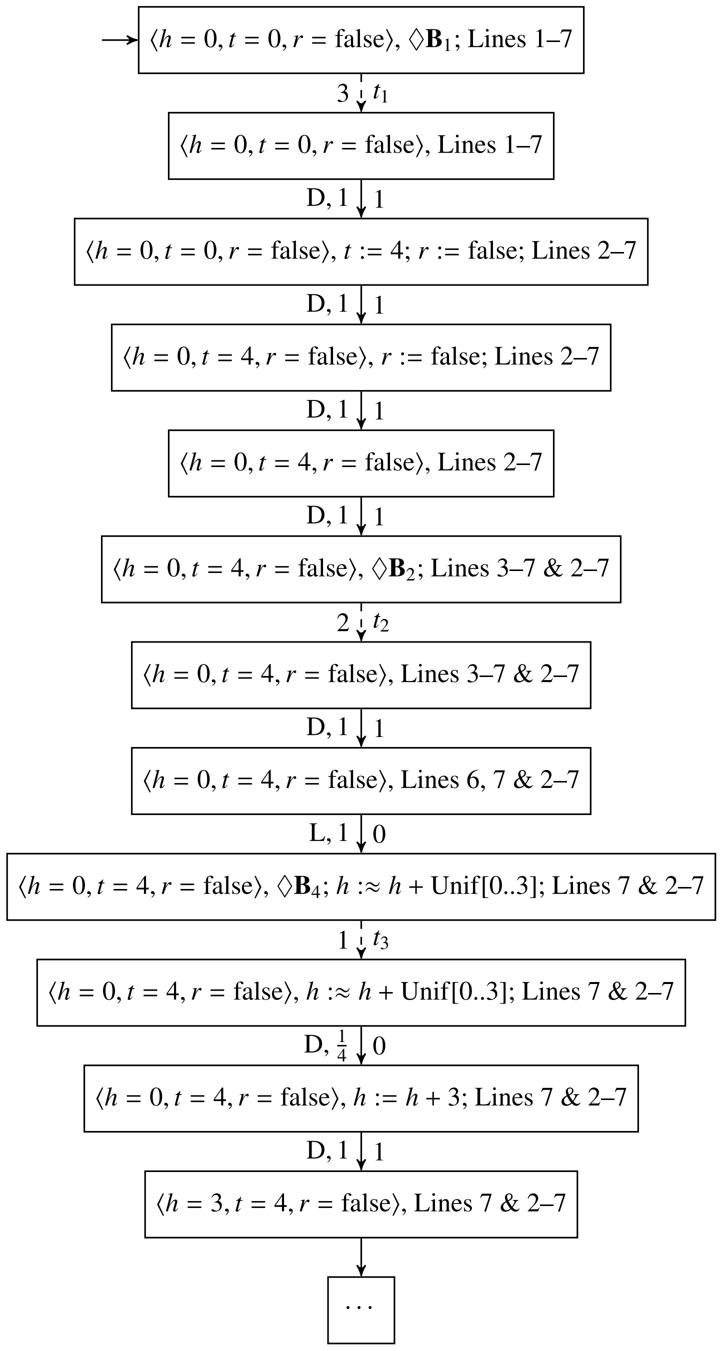

$\langle h = 0, t = 0, r = \text{false}\rangle$, $\Diamond\mathbf{B}_1$; Lines 1–7
3 t_1
$\langle h = 0, t = 0, r = \text{false}\rangle$, Lines 1–7
D, 1 1
$\langle h = 0, t = 0, r = \text{false}\rangle$, $t := 4$; $r :=$ false; Lines 2–7
D, 1 1
$\langle h = 0, t = 4, r = \text{false}\rangle$, $r :=$ false; Lines 2–7
D, 1 1
$\langle h = 0, t = 4, r = \text{false}\rangle$, Lines 2–7
D, 1 1
$\langle h = 0, t = 4, r = \text{false}\rangle$, $\Diamond\mathbf{B}_2$; Lines 3–7 & 2–7
2 t_2
$\langle h = 0, t = 4, r = \text{false}\rangle$, Lines 3–7 & 2–7
D, 1 1
$\langle h = 0, t = 4, r = \text{false}\rangle$, Lines 6, 7 & 2–7
L, 1 0
$\langle h = 0, t = 4, r = \text{false}\rangle$, $\Diamond\mathbf{B}_4$; $h :\approx h + \text{Unif}[0..3]$; Lines 7 & 2–7
1 t_3
$\langle h = 0, t = 4, r = \text{false}\rangle$, $h :\approx h + \text{Unif}[0..3]$; Lines 7 & 2–7
D, $\frac{1}{4}$ 0
$\langle h = 0, t = 4, r = \text{false}\rangle$, $h := h + 3$; Lines 7 & 2–7
D, 1 1
$\langle h = 3, t = 4, r = \text{false}\rangle$, Lines 7 & 2–7
$\cdots$

Figure 2: A path fragment in PPD

Let τ_1 be the running time, and τ_2 the waiting time. Time-bounded sinking (terminating or violating) paths in the PPD are of the total time $\tau = \tau_1 + \tau_2 \leq \Delta$ for a given time bound Δ. For conciseness, we will write the fragments of paths and schedulers shortly as paths and schedulers, and write primitive blocks as blocks afterwards. Particularly, we are interested in the *fully acted* schedulers that exactly resolve nondeterminism up to the time bound $\tau_1 \leq \Delta$. Let σ be a path of the PPD, and ς its projection in the induced PP. The outline to solve the time-bounded termination problem is given below.

1. As $\tau_1 \leq \Delta$ and the loop judgements are time-consuming, all loops in the induced PP can be executed only finitely many times. We unroll those loops and get a finite MA that covers all time-bounded paths with $\tau_1 \leq \Delta$ in the induced PP, among which all time-bounded sinking paths ς are finitely enumerable. It implies that the MA has only finitely many distinct fully acted schedulers.

2. For each aforementioned path ς in the induced PP, its probability $\mathcal{P}_1$ can be obtained by the standard analysis over MCs. The path ς can be augmented as time-bounded sinking paths σ in the PPD, provided that the waiting time τ_2 for executing all involved blocks are not greater than $\Delta - \tau_1$. The latter event can be specified by a multi-phase until formula under **continuous stochastic logic** (CSL) [6], saying

$$\mathbf{B}_1 \ \mathrm{U}^{(0,\Delta-\tau_1]} \ \mathbf{B}_2 \cdots \mathrm{U}^{(0,\Delta-\tau_1]} \ \mathbf{B}_k \ \mathrm{U}^{(0,\Delta-\tau_1]} \ \mathrm{END} \ . \tag{3}$$

   Thus its probability $\mathcal{P}_2$ can be computed by the existing methods [58, 57]. The product $\mathcal{P}_1 \times \mathcal{P}_2$ is the measure of all time-bounded sinking paths σ of the type ς in the PPD.

3. The set of all sinking paths ς with $\tau_1 \leq \Delta$ in the induced PP is finite. A scheduler only covers a subset of them. For the PPD without conditioning feature, demonic schedulers are the ones that minimise the measures of all terminating paths. For the PPD with conditioning feature, demonic schedulers minimise

$$\Pr(\Diamond \mathit{sink}_{\checkmark} \mid \neg\Diamond \lightning) = \frac{\Pr(\Diamond \mathit{sink}_{\checkmark} \cap \neg\Diamond \lightning)}{1 - \Pr(\Diamond \lightning)} \ . \tag{4}$$

   (We abuse the notations $\Diamond \mathit{sink}_{\checkmark} \cap \neg\Diamond \lightning$ and $\Diamond \lightning$ here to denote time-bounded termination and violation, respectively.) Hence the minimum time-bounded termination probability over PPDs can be obtained by comparing finitely many schedulers.

Note that the time-bounded termination probability and violation probability are monotonously increasing functions in the time bound Δ. When Δ increases, more and more paths are terminating (resp. violating), which results in a larger termination probability in absolute amount (resp. relative amount via the fraction (4)). In case of an `observe`(ξ) statement, the subsequent executions not satisfying ξ are blocked and the termination probability would be rescaled and thus increase, which is consistent with the monotonicity mentioned above.

Demonic schedulers of a PPD may differ from those of its induced PP, since the probabilities $\mathcal{P}_1$ of sinking paths ς are weighted by $\mathcal{P}_2$ then. We illustrate it via an example.

Example 4.2. *Consider the path ς in the PP of Figure 1(a) that executes the left branch of Line 6 with sample 3 for the first two iterations and terminates ($h = 6$ and $t = 6$ then). It adopts the scheduler fragment "LL", and passes through blocks "$\mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_4, \mathbf{B}_6, \mathbf{B}_2, \mathbf{B}_4, \mathbf{B}_6$" in turn and eventually* END. *If the given time bound Δ for τ is 20 units of time, its augmented terminating* *paths σ in the PPD have waiting time at most* $\Delta - \tau_1 = 8$ *units, since the running time is* $\tau_1 = 12$ *units. The weight* $\mathcal{P}_2$ *for this type in the PPD is specified by the probability of the CSL formula [6]:*

$$\mathbf{B}_1 \ \mathrm{U}^{(0,8]} \ \mathbf{B}_2 \ \mathrm{U}^{(0,8]} \ \mathbf{B}_4 \ \mathrm{U}^{(0,8]} \ \mathbf{B}_6 \ \mathrm{U}^{(0,8]} \ \mathbf{B}_2 \ \mathrm{U}^{(0,8]} \ \mathbf{B}_4 \ \mathrm{U}^{(0,8]} \ \mathbf{B}_6 \ \mathrm{U}^{(0,8]} \ \mathrm{END} \ ,$$

which can be computed by the multiple integral [57, Lemma 4.7]

$$\begin{aligned}&\int_0^8 \mathrm{d}\,t_1 \int_0^{8-T_1} \mathrm{d}\,t_2 \int_0^{8-T_2} \mathrm{d}\,t_3 \int_0^{8-T_3} \mathrm{d}\,t_4 \int_0^{8-T_4} \mathrm{d}\,t_5 \int_0^{8-T_5} \mathrm{d}\,t_6 \int_0^{8-T_6} \\ &\qquad 3\mathrm{e}^{-3t_1} \cdot 2\mathrm{e}^{-2t_2} \cdot \mathrm{e}^{-t_3} \cdot 3\mathrm{e}^{-3t_4} \cdot 2\mathrm{e}^{-2t_5} \cdot \mathrm{e}^{-t_6} \cdot 3\mathrm{e}^{-3t_7} \,\mathrm{d}\,t_7 \\ &= 1 - \tfrac{297}{4}\mathrm{e}^{-8} - 405\mathrm{e}^{-16} - \tfrac{2359}{4}\mathrm{e}^{-24} \ ,\end{aligned}$$

where $T_i = t_1 + \cdots + t_i$ *for* $i = 1, \ldots, 6$. *Totally the measure of those* σ *would be*

$$\mathcal{P}_1 \times \mathcal{P}_2 = \tfrac{1}{16} \cdot [1 - \tfrac{297}{4}\mathrm{e}^{-8} - 405\mathrm{e}^{-16} - \tfrac{2359}{4}\mathrm{e}^{-24}] \approx 0.0609404 \ ,$$

where $\mathcal{P}_1 = \frac{1}{16}$ *is computed as the probability of twice of independent sampling with* 3 *under the uniform distribution* Unif[0..3]. *The measure of those* σ *would tend to* $\frac{1}{16}$ *for sufficiently large* $\Delta - \tau_1$, *and tend to zero for sufficiently small* $\Delta - \tau_1$.

Different execution paths access different sequences of blocks, and their waiting time have different bounds $\Delta - \tau_1$. *The scheduler fragment* "LLLL" *is a fully acted scheduler, as it resolves nondeterminism up to the time bound* Δ *and any proper prefix of it does not suffice. All schedulers that are fully acted up to* Δ *are* "LLLL, LLLR, LLR, LRL, LRR, RLL, RLR, RR". *The demonic scheduler can be determined by comparing them. After a finite number of comparisons, we can see that*

- *the demonic scheduler for the induced PP is* "LLR", *under which the measure of all terminating paths with* $\tau_1 \leq 20$ *achieves the minimum* $\frac{1}{16}$*;*
- *the demonic schedulers for the PPD are* "LRL" *and* "LRR", *under which the measures of all terminating paths with* $\tau_1 + \tau_2 \leq 20$ *achieve the minimum* $\frac{3}{8} + \frac{243}{4}\mathrm{e}^{-3} - \frac{8991}{8}\mathrm{e}^{-6} - \frac{18969}{4}\mathrm{e}^{-9}$ *(*≈ 0.0285167*).*

The results on all fully acted schedulers can be found in Table 3. Although "LLLL" *covers* 30 *types of terminating paths more than* "LLLR", *those paths are of measure zero in PPD due to their waiting time* $\tau_2 \leq \Delta - \tau_1 = 0$. *So the time-bounded termination probabilities under the two schedulers are the same in PPD.*

Proposition 4.3. *Let ς be a sinking path in the induced PP that accesses blocks* $\mathbf{B}_1, \ldots, \mathbf{B}_k$ *in turn. Then the weight* $\mathcal{P}_2$ *for this type and for the waiting time* τ_2 *in the PPD can be expressed as the multiple integral*

$$\int_0^{\tau_2} \lambda_1 \mathrm{e}^{-\lambda_1 t_1} \,\mathrm{d}\,t_1 \int_0^{\tau_2 - T_1} \lambda_2 \mathrm{e}^{-\lambda_2 t_2} \,\mathrm{d}\,t_2 \cdots \int_0^{\tau_2 - T_{k-1}} \lambda_k \mathrm{e}^{-\lambda_k t_k} \,\mathrm{d}\,t_k \ , \tag{5}$$

where $\lambda_i \in \mathbb{Z}^+$ *are priorities of* $\mathbf{B}_i$ *for* $i = 1, \ldots, k$ *and* $T_i = t_1 + \cdots + t_i$ *for* $i = 1, \ldots, k-1$. *It can be further expressed as an element of the ring* $\mathbb{Q}[\tau_2, \mathrm{e}^{-\tau_2}]$.

Proof. The construction of the integral (5) follows from [57, Lemma 4.7]. We then show the integrability by induction on the number k of blocks.

schedulers	$\#\{\varsigma\}$	$\sum_\varsigma \mathcal{P}_1$	$\sum_\varsigma \mathcal{P}_1 \cdot \mathcal{P}_2$
LLLL	38	$\frac{37}{128}$	$\frac{11}{64} - \frac{2835}{256}\mathrm{e}^{-4} + \frac{2727}{8}\mathrm{e}^{-8} - \frac{100597}{256}\mathrm{e}^{-12} - \frac{405}{16}\mathrm{e}^{-16} - \frac{2359}{64}\mathrm{e}^{-24}$
LLLR	8	$\frac{11}{64}$	$\frac{11}{64} - \frac{2835}{256}\mathrm{e}^{-4} + \frac{2727}{8}\mathrm{e}^{-8} - \frac{100597}{256}\mathrm{e}^{-12} - \frac{405}{16}\mathrm{e}^{-16} - \frac{2359}{64}\mathrm{e}^{-24}$
LLR	1	$\frac{1}{16}$	$\frac{1}{16} - \frac{297}{64}\mathrm{e}^{-8} - \frac{405}{16}\mathrm{e}^{-16} - \frac{2359}{64}\mathrm{e}^{-24}$
LRL	6	$\frac{3}{8}$	$\frac{3}{8} + \frac{243}{4}\mathrm{e}^{-3} - \frac{8991}{8}\mathrm{e}^{-6} - \frac{18969}{4}\mathrm{e}^{-9}$
LRR	6	$\frac{3}{8}$	$\frac{3}{8} + \frac{243}{4}\mathrm{e}^{-3} - \frac{8991}{8}\mathrm{e}^{-6} - \frac{18969}{4}\mathrm{e}^{-9}$
RLL	4	$\frac{7}{16}$	$\frac{7}{16} + \frac{243}{8}\mathrm{e}^{-3} - \frac{8991}{16}\mathrm{e}^{-6} - \frac{27}{4}\mathrm{e}^{-7} - \frac{18969}{8}\mathrm{e}^{-9} + \frac{1863}{4}\mathrm{e}^{-14} - \frac{2173}{4}\mathrm{e}^{-21}$
RLR	4	$\frac{7}{16}$	$\frac{7}{16} + \frac{243}{8}\mathrm{e}^{-3} - \frac{8991}{16}\mathrm{e}^{-6} - \frac{27}{4}\mathrm{e}^{-7} - \frac{18969}{8}\mathrm{e}^{-9} + \frac{1863}{4}\mathrm{e}^{-14} - \frac{2173}{4}\mathrm{e}^{-21}$
RR	1	$\frac{1}{4}$	$\frac{1}{4} - \frac{27}{4}\mathrm{e}^{-7} + \frac{1863}{4}\mathrm{e}^{-14} - \frac{2173}{4}\mathrm{e}^{-21}$

Table 3: Schedulers in PP and PPD

- Basically, when $k = 1$, the integral (5) is plainly integrable. After integration, it would be $1 - \mathrm{e}^{-\lambda_1 \tau_2}$, an element of $\mathbb{Q}[\tau_2, \mathrm{e}^{-\tau_2}]$.
- Inductively, we assume it holds for $k = K - 1$, and proceed to show it holds for $k = K$. After the inner $K - 1$ times of integration, the integral (5) becomes

  $$\int_0^{\tau_2} \lambda_1 \mathrm{e}^{-\lambda_1 t_1} \cdot \Phi(\tau_2 - t_1)\,\mathrm{d}\,t_1 \quad ,$$

  where $\Phi(x)$ is an element of $\mathbb{Q}[x, \mathrm{e}^{-x}]$ by inductive assumption. Then, after integration-by-parts, it could further be expressed as an element of $\mathbb{Q}[\tau_2, \mathrm{e}^{-\tau_2}]$. □

Theorem 4.4. *The time-bounded termination problem is solvable over PPDs.*

Proof. It suffices to determine the demonic scheduler. Fixing a scheduler θ, we first notice that every sinking path ς in the induced PP takes a rational number as its probability. By substituting $\tau_2 = \Delta - \tau_1$ (a known nonnegative integer) into Proposition 4.3, we can see that the measure of all sinking paths of this type can be expressed in the explicit form

$$\alpha_0 + \alpha_1 \mathrm{e}^{\beta_1} + \cdots + \alpha_m \mathrm{e}^{\beta_m} \quad , \tag{6}$$

where $\alpha_0, \ldots, \alpha_m$ are rational numbers and $\beta_1, \ldots, \beta_m$ are distinct negative integers, and that the time-bounded termination probability $\mathcal{P}(\theta)$ is the fraction (4), whose numerator and denominator are in the above form. That is, $\mathcal{P}(\theta)$ is a fraction with numerator and denominator taken from the ring $\mathbb{Q}[\mathrm{e}]$, equivalently an element of the field extension $\mathbb{Q}(\mathrm{e}) : \mathbb{Q}$. By the fact [9, Theorem 1.2] that e is transcendental, we can infer that an element of $\mathbb{Q}(\mathrm{e}) : \mathbb{Q}$ equals zero iff its numerator is the zero element of $\mathbb{Q}[\mathrm{e}]$.[1] Thus we can compute the value $\mathcal{P}(\theta)$ up to any precision by sufficiently approaching e, and compare $\mathcal{P}(\theta_1)$ and $\mathcal{P}(\theta_2)$ for any pair of schedulers θ_1 and θ_2 after ruling out the case of $\mathcal{P}(\theta_1) = \mathcal{P}(\theta_2)$. Hence the demonic scheduler can be eventually determined by comparing all fully acted schedulers. □

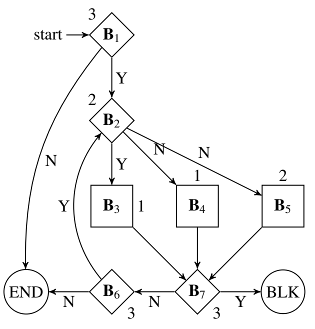
Finally we give an example of PPD with condition feature.

Example 4.5. *Here we insert the* `observe` *statement at Line 7 of the PPD in Figure 3(a). The control flow graph has an extra block* $\mathbf{B}_7 \equiv h \mod 3 \neq 1$ *with priority* $Rate(\mathbf{B}_7) = 3$ *to denote this statement.*

[1] A number α is *algebraic*, denoted by $\alpha \in \mathbb{A}$, if there is a nonzero univariate $\mathbb{Q}$-polynomial $f(x)$ such that $f(\alpha) = 0$; and is *transcendental* otherwise. Note that $(\sqrt{2})^2 - 2$ is a nonzero element of $\mathbb{Q}[\sqrt{2}]$, but it equals zero. We do not suffer this trouble, because a nonzero element of $\mathbb{Q}[\mathrm{e}]$ must not equal zero (otherwise it contradicts the fact that e is algebraic).

```
1: $h := 0$; $t := 4$; $r :=$ false
2: **while** $h < t$ **do**
3:     **if** $r$ **then**
4:         $h :\approx h + \mathrm{Unif}[3..6]$; $r :=$ false
5:     **else**
6:         $\{h :\approx h + \mathrm{Unif}[0..3]\} \,\square\, \{r := \mathrm{true}\}$
7:     observe($h \mod 3 \neq 1$)
8:     $t := t + 1$
```

(a)



(b)

Figure 3: An example of probabilistic program with condition feature

Consider the path ς in the induced PP that executes the left branch of Line 6 with sample 2 *for the first two iterations and is violating ($h = 4$ then). It accesses blocks "$\mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_4, \mathbf{B}_7, \mathbf{B}_6, \mathbf{B}_2, \mathbf{B}_4, \mathbf{B}_7$" in turn. If the time bound Δ for τ is* 22 *units of time, its augmented violating paths σ in the PPD have waiting time at most $\Delta - \tau_1 = 10$ units since the running time is $\tau_1 = 12$ units. So the measure of all violating paths of this type in the PPD can be similarly computed as in Example 4.2.*

After comparing all fully acted schedulers "LLLL, LLLR, LLR, LRL, LRR, RLL, RLR, RR"*, we get the demonic schedulers* "LRL" *and* "LRR" *for the PPD with condition feature, under which the time-bounded termination probabilities achieve the minimum*

$$\frac{\frac{3}{16} + \frac{59049}{256}\mathrm{e}^{-2} + \frac{828873}{16}\mathrm{e}^{-4} - \frac{506183721}{1280}\mathrm{e}^{-6}}{1 - (\frac{7}{16} + \frac{2187}{32}\mathrm{e}^{-4} - \frac{13851}{16}\mathrm{e}^{-8} - \frac{29260047}{160}\mathrm{e}^{-12} - \frac{9}{8}\mathrm{e}^{-15} + \frac{9}{4}\mathrm{e}^{-30} - \frac{101}{8}\mathrm{e}^{-45})} \approx 0.000161430 \ .$$

The results on all fully acted schedulers can be found in Table 4.

Besides, if we switch the statements in Line 7 and in Line 8, the resulting PPD is functionally identical with the original one, but would have lower time-bounded termination probability under any scheduler, since all violating paths have less one unit of time as to the waiting processes.

5. Parametric Extension

We now study the time-bounded termination problem for parametric PPDs. The parameters $\mathbf{y} = (y_1, \ldots, y_n)$ are introduced to the program in the following manner: Consider the probabilistic choice

$$\{\mathbf{C}_1\} \quad [p] \quad \{\mathbf{C}_2\} \tag{7a}$$

that executes $\mathbf{C}_1$ with probability p and executes $\mathbf{C}_2$ with probability $1 - p$, where $p \in \mathbb{Q} \cap (0, 1)$ is a constant. We extend it to the parametric form

$$\{\mathbf{C}_1\} \quad [y] \quad \{\mathbf{C}_2\} \tag{7b}$$

that executes $\mathbf{C}_1$ with probability y and executes $\mathbf{C}_2$ with probability $1 - y$, where $y \in (0, 1)$ is a real parameter.[2] So the time-bounded termination problem amounts to:

[2]It is not suggestable to assign the extreme values 0 and 1 to the parameter y, as the parametrisation should keep the structure of the original program and thus preserve the same qualitative properties. However, dropping this restriction

schedulers	$\#\Diamond sink \cap \neg\Diamond\lightning$	$\Pr(\Diamond sink \cap \neg\Diamond\lightning)$
	$\#\Diamond\lightning$	$\Pr(\Diamond\lightning)$
LLLL	3	$\frac{3}{32} - \frac{6561}{256}\mathrm{e}^{-3} + \frac{45927}{32}\mathrm{e}^{-6} - \frac{2187}{256}\mathrm{e}^{-8} - \frac{5996287}{320}\mathrm{e}^{-9} - \frac{2673}{16}\mathrm{e}^{-16} - \frac{1012549}{256}\mathrm{e}^{-24}$
	34	$\frac{9}{16} - \frac{8019}{256}\mathrm{e}^{-5} + \frac{179091}{64}\mathrm{e}^{-10} - \frac{8990099}{256}\mathrm{e}^{-15} - \frac{4131}{16}\mathrm{e}^{-20} - \frac{133743}{64}\mathrm{e}^{-30} - \frac{101}{8}\mathrm{e}^{-45}$
LLLR	3	$\frac{3}{32} - \frac{6561}{256}\mathrm{e}^{-3} + \frac{45927}{32}\mathrm{e}^{-6} - \frac{2187}{256}\mathrm{e}^{-8} - \frac{5996287}{320}\mathrm{e}^{-9} - \frac{2673}{16}\mathrm{e}^{-16} - \frac{1012549}{256}\mathrm{e}^{-24}$
	12	$\frac{9}{16} - \frac{8019}{256}\mathrm{e}^{-5} + \frac{179091}{64}\mathrm{e}^{-10} - \frac{8990099}{256}\mathrm{e}^{-15} - \frac{4131}{16}\mathrm{e}^{-20} - \frac{133743}{64}\mathrm{e}^{-30} - \frac{101}{8}\mathrm{e}^{-45}$
LLR	1	$\frac{1}{16} - \frac{2187}{256}\mathrm{e}^{-8} - \frac{2673}{16}\mathrm{e}^{-16} - \frac{1012549}{256}\mathrm{e}^{-24}$
	4	$\frac{7}{16} - \frac{1701}{64}\mathrm{e}^{-10} - \frac{9}{8}\mathrm{e}^{-15} - \frac{4131}{16}\mathrm{e}^{-20} - \frac{133743}{64}\mathrm{e}^{-30} - \frac{101}{8}\mathrm{e}^{-45}$
LRL	3	$\frac{3}{16} + \frac{59049}{256}\mathrm{e}^{-2} + \frac{828873}{16}\mathrm{e}^{-4} - \frac{506183721}{1280}\mathrm{e}^{-6}$
	4	$\frac{7}{16} + \frac{2187}{32}\mathrm{e}^{-4} - \frac{13851}{16}\mathrm{e}^{-8} - \frac{29260047}{160}\mathrm{e}^{-12} - \frac{9}{8}\mathrm{e}^{-15} + \frac{9}{4}\mathrm{e}^{-30} - \frac{101}{8}\mathrm{e}^{-45}$
LRR	3	$\frac{3}{16} + \frac{59049}{256}\mathrm{e}^{-2} + \frac{828873}{16}\mathrm{e}^{-4} - \frac{506183721}{1280}\mathrm{e}^{-6}$
	4	$\frac{7}{16} + \frac{2187}{32}\mathrm{e}^{-4} - \frac{13851}{16}\mathrm{e}^{-8} - \frac{29260047}{160}\mathrm{e}^{-12} - \frac{9}{8}\mathrm{e}^{-15} + \frac{9}{4}\mathrm{e}^{-30} - \frac{101}{8}\mathrm{e}^{-45}$
RLL	2	$\frac{5}{16} + \frac{19683}{256}\mathrm{e}^{-2} + \frac{276291}{16}\mathrm{e}^{-4} - \frac{168727907}{1280}\mathrm{e}^{-6} - \frac{243}{16}\mathrm{e}^{-7} + \frac{8991}{4}\mathrm{e}^{-14} - \frac{889207}{16}\mathrm{e}^{-21}$
	3	$\frac{3}{8} + \frac{729}{16}\mathrm{e}^{-4} - \frac{4617}{8}\mathrm{e}^{-8} - \frac{81}{8}\mathrm{e}^{-9} - \frac{9753349}{80}\mathrm{e}^{-12} + \frac{7695}{4}\mathrm{e}^{-18} - \frac{80975}{8}\mathrm{e}^{-27}$
RLR	2	$\frac{5}{16} + \frac{19683}{256}\mathrm{e}^{-2} + \frac{276291}{16}\mathrm{e}^{-4} - \frac{168727907}{1280}\mathrm{e}^{-6} - \frac{243}{16}\mathrm{e}^{-7} + \frac{8991}{4}\mathrm{e}^{-14} - \frac{889207}{16}\mathrm{e}^{-21}$
	3	$\frac{3}{8} + \frac{729}{16}\mathrm{e}^{-4} - \frac{4617}{8}\mathrm{e}^{-8} - \frac{81}{8}\mathrm{e}^{-9} - \frac{9753349}{80}\mathrm{e}^{-12} + \frac{7695}{4}\mathrm{e}^{-18} - \frac{80975}{8}\mathrm{e}^{-27}$
RR	1	$\frac{1}{4} - \frac{243}{16}\mathrm{e}^{-7} + \frac{8991}{4}\mathrm{e}^{-14} - \frac{889207}{16}\mathrm{e}^{-21}$
	1	$\frac{1}{4} - \frac{81}{8}\mathrm{e}^{-9} + \frac{7695}{4}\mathrm{e}^{-18} - \frac{80975}{8}\mathrm{e}^{-27}$

Table 4: Schedulers in PPD with condition feature

Problem 5.1. *What is the demonic scheduler for a parametric PPD when the total time (running time plus waiting time) is bounded?*

We formulate it as a quantifier-free constraint under the first-order logic $\mathfrak{L}(\mathbb{R}; >, =; +, \cdot; 0, 1, \mathrm{e})$ (that consists of polynomial equations and inequalities with coefficients taken from $\mathbb{Q}[\mathrm{e}]$ and variables ranging over $\mathbb{R}$), and solve them by an extended usage of cylindrical algebraic decomposition [18].

Proposition 5.2. *Let ς be a sinking path in the induced PP. Then its probability $\mathcal{P}_1$ in the PP can be expressed as an element of the ring $\mathbb{Q}[\mathbf{y}]$.*

Proof. It follows immediately by noting that ς is exactly a finite path of the parametric MA; and that its probability $\mathcal{P}_1$ is a finite product of the transition probabilities along with ς, each probability is an element of $\mathbb{Q}[\mathbf{y}]$. □

Lemma 5.3. *Problem 5.1 can be formulated as quantifier-free constraints under the first-order logic $\mathfrak{L}(\mathbb{R}; >, =; +, \cdot; 0, 1, \mathrm{e})$.*

Proof. The proof is similar to that of Theorem 4.4. The only difference lies in that every sinking path ς in the induced PP takes an element of $\mathbb{Q}[\mathbf{y}]$ as its probability by Proposition 5.2. Then, for a fixed scheduler θ, the time-bounded termination probability $\mathcal{P}(\theta)$ is the fraction (4), whose numerator and denominator are in the form

$$\alpha_0(\mathbf{y}) + \alpha_1(\mathbf{y})\mathrm{e}^{\beta_1} + \cdots + \alpha_m(\mathbf{y})\mathrm{e}^{\beta_m} \quad , \tag{8}$$

would not cause any technical hardness to our method. In fact, we can tackle those parameters appearing in arbitrary $\mathbb{Q}$-polynomial form.

where $\alpha_0(\mathbf{y}), \ldots, \alpha_m(\mathbf{y})$ are $\mathbb{Q}$-polynomials in parameters $\mathbf{y}$ and $\beta_1, \ldots, \beta_m$ are distinct negative integers. So comparing $\mathcal{P}(\theta_1)$ and $\mathcal{P}(\theta_2)$ for any pair of schedulers θ_1 and θ_2 are quantifier-free constraints under the first-order logic $\mathfrak{L}(\mathbb{R}; >, =; +, \cdot; 0, 1, \mathrm{e})$, extending **real closed fields** $\mathfrak{L}(\mathbb{R}; >, =; +, \cdot; 0, 1)$ [18] with the Euler's constant e. Once we know how to solve those constraints under $\mathfrak{L}(\mathbb{R}; >, =; +, \cdot; 0, 1, \mathrm{e})$ (particularly for $\mathbf{y}$ confined in its domain), we can eventually offer subregions w. r. t. $\mathbf{y}$, on each of which a scheduler would be demonic everywhere. $\square$

Example 5.4. *Let us consider the parametric version of the PPD in Figure 1(a). The sampling assignments in Lines 4 & 6 are replaced with probabilistic choices, whose probabilities are attached with two parameters y_1 and y_2, as shown in Figure 4. (Here we abuse the notation of probabilistic choices, which can be easily rewritten as usual ones.)*

```
1: $h := 0; t := 4; r :=$ false
2: while $h < t$ do
3:     if $r$ then
4:         $h :\approx \begin{cases} h+3 & [\frac{1}{4} - y_1] \\ h+4 & [\frac{1}{4} + y_1] \\ h+5 & [\frac{1}{4} - y_2] \\ h+6 & [\frac{1}{4} + y_2] \end{cases}$ ; $r :=$ false
5:     else
6:         $\left\{ h :\approx \begin{cases} h+0 & [\frac{1}{4} - y_1] \\ h+1 & [\frac{1}{4} + y_1] \\ h+2 & [\frac{1}{4} - y_2] \\ h+3 & [\frac{1}{4} + y_2] \end{cases} \right\} \;\square\; \{r := \text{true}\}$
7:     $t := t + 1$
```

Figure 4: An example of parametric probabilistic program

The domain for $\mathbf{y} = (y_1, y_2)$ is the default region $(-\frac{1}{4}, \frac{1}{4}) \times (-\frac{1}{4}, \frac{1}{4})$. Our goal is to partition the parameter domain into subregions, on each of which a scheduler would be demonic everywhere.

Let the time bound Δ for τ be 20 units of time. Then the results on all fully acted schedulers can be found in Table 5, in which distinct weights $\gamma_1, \ldots, \gamma_5$ are given below.

$$
\begin{aligned}
\gamma_1 &= 1 - \tfrac{297}{4}\mathrm{e}^{-8} - 405\mathrm{e}^{-16} - \tfrac{2359}{4}\mathrm{e}^{-24} \ , \\
\gamma_2 &= 1 - \tfrac{405}{4}\mathrm{e}^{-4} + 3159\mathrm{e}^{-8} - \tfrac{14371}{4}\mathrm{e}^{-12} \ , \\
\gamma_3 &= 1 + 162\mathrm{e}^{-3} - 2997\mathrm{e}^{-6} - 12646\mathrm{e}^{-9} \ , \\
\gamma_4 &= 1 - 27\mathrm{e}^{-7} + 1863\mathrm{e}^{-14} - 2173\mathrm{e}^{-21} \ , \\
\gamma_5 &= 1 + 162\mathrm{e}^{-3} - 2997\mathrm{e}^{-6} - 12646\mathrm{e}^{-9} \ .
\end{aligned}
$$

From Table 5 and the set inclusion, we can see that

- $\mathcal{P}(\mathrm{LLLL}) = \mathcal{P}(\mathrm{LLLR}) > \mathcal{P}(\mathrm{LLR})$*, as all terminating paths under* LLR *are those under* LL *and thus the set of terminating paths under* LLLL *is a superset of that under* LLR*;*
- $\mathcal{P}(\mathrm{LRL}) = \mathcal{P}(\mathrm{LRR})$*; and*
- $\mathcal{P}(\mathrm{RLL}) = \mathcal{P}(\mathrm{RLR}) > \mathcal{P}(\mathrm{RR})$*, as the set of terminating paths under* RLL *is a superset of that under* RR.

schedulers	$\sum_{\mathcal{S}} \mathcal{P}_1 \cdot \mathcal{P}_2$
LLLL	$(y_2^2 + \frac{1}{2}y_2 + \frac{1}{16}) \cdot \gamma_1 + (2y_1y_2^2 + y_2^3 + y_1y_2 - \frac{3}{4}y_2^2 + \frac{1}{8}y_1 + \frac{3}{16}y_2 + \frac{7}{64}) \cdot \gamma_2$
LLLR	$(y_2^2 + \frac{1}{2}y_2 + \frac{1}{16}) \cdot \gamma_1 + (2y_1y_2^2 + y_2^3 + y_1y_2 - \frac{3}{4}y_2^2 + \frac{1}{8}y_1 + \frac{3}{16}y_2 + \frac{7}{64}) \cdot \gamma_2$
LLR	$(y_2^2 + \frac{1}{2}y_2 + \frac{1}{16}) \cdot \gamma_1$
LRL	$(2y_1y_2 + \frac{1}{2}y_1 + \frac{1}{2}y_2 + \frac{3}{8}) \cdot \gamma_3$
LRR	$(2y_1y_2 + \frac{1}{2}y_1 + \frac{1}{2}y_2 + \frac{3}{8}) \cdot \gamma_3$
RLL	$(y_2 + \frac{1}{4}) \cdot \gamma_4 + (y_1y_2 + \frac{1}{4}y_1 - \frac{1}{4}y_2 + \frac{3}{16}) \cdot \gamma_5$
RLR	$(y_2 + \frac{1}{4}) \cdot \gamma_4 + (y_1y_2 + \frac{1}{4}y_1 - \frac{1}{4}y_2 + \frac{3}{16}) \cdot \gamma_5$
RR	$(y_2 + \frac{1}{4}) \cdot \gamma_4$

Table 5: Schedulers in parametric PPD

So it suffices to compare $\mathcal{P}$(LLR), $\mathcal{P}$(LRL) *and* $\mathcal{P}$(RR). *Since the critical line of* $\mathcal{P}(\text{LLR}) = \mathcal{P}(\text{RR})$ *lies in* $y_2 = \frac{\gamma_4}{\gamma_1} - \frac{1}{4} \approx 0.751928$, *the scheduler "*RR*" cannot be demonic anywhere of the domain* $(-\frac{1}{4}, \frac{1}{4}) \times (-\frac{1}{4}, \frac{1}{4})$. *On the other hand, the critical line of* $\mathcal{P}(\text{LLR}) = \mathcal{P}(\text{LRL})$ *is depicted by*

$$y_1 = \frac{4y_2 + 1}{8} \cdot \frac{\gamma_1}{\gamma_3} - \frac{1}{4} - \frac{1}{8y_2 + 2} \quad .$$

Therefore the domain is partitioned into two subregions (see Figure 5):

- *the schedulers "*LRL*" and "*LRR*" are demonic on the top subregion,*
- *the scheduler "*LLR*" is demonic on the bottom subregion, and*
- *both are demonic on the critical line.*

At the original point $(y_1, y_2) = (0, 0)$, *the result has been shown in Example 4.2.*

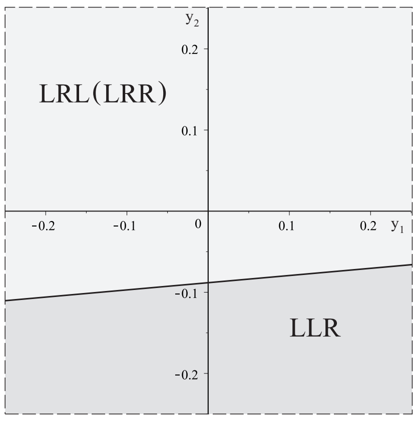


Figure 5: Demonic schedulers in the parameter domain

Note that the constraints formulated in Lemma 5.3 would involve polynomials of arbitrary degree along with the increase in Δ. So it needs a general method to solve them. We resort to cylindrical algebraic decomposition [18] (see Algorithm 1). It decomposes solutions of polynomial constraints into finitely many cells (to be specified below), which are cylindrically indexed.

Definition 5.5 (Cells). *A connected region in $\mathbb{R}^n$ is a cell w. r. t. $\mathbf{y} = (y_1, \ldots, y_n)$ if it is defined by:*

- *Basically, the first variable y_1 lies in a proper interval (l_1, u_1) with $l_1, u_1 \in \mathbb{R} \cap \mathbb{A}$, or (as boundary case) in a point interval $\{c_1\}$ with $c_1 \in \mathbb{R} \cap \mathbb{A}$.*
- *Inductively, when $(y_1, \ldots, y_{k-1})$ lies in a cell $C_{k-1} \subseteq \mathbb{R}^{k-1}$, the variable y_k is in a proper interval (l_k, u_k) or in a point interval $\{c_k\}$ (where the interval boundaries are $\mathbb{Q}$-polynomials on C_{k-1}, e. g., $l_k : C_{k-1} \to \mathbb{R} \cup \{-\infty\}$).*

Recall that those boundaries l_1, u_1, c_1 are real algebraic numbers, elements of $\mathbb{R} \cap \mathbb{A}$, defined as real roots of univariate $\mathbb{Q}$-polynomials in y_1.

For instance, let us consider the domain of (y_1, y_2) shown in Figure 5. Assume $m : (-\frac{1}{4}, \frac{1}{4}) \to \mathbb{R}$ is the boundary such that $y_2 = c(y_1)$ is defined by the critical line. Then there are three cells in the domain, which can be cylindrically indexed as:

- the top subregion $(-\frac{1}{4}, \frac{1}{4}) \times (c, \frac{1}{4})$,
- the critical line $(-\frac{1}{4}, \frac{1}{4}) \times \{c\}$, and
- the bottom subregion $(-\frac{1}{4}, \frac{1}{4}) \times (-\frac{1}{4}, c)$.

Algorithm 1 Cylindrical Algebraic Decomposition for $\mathfrak{L}(\mathbb{R}; >, =; +, \cdot; 0, 1)$

$$\mathfrak{G} \Leftarrow \mathsf{decompose}(\mathfrak{F})$$

Input: $\mathfrak{F}$ is a first-order formula consisting of $\mathbb{Q}$-polynomial equations and inequalities in $\mathbf{y}, \mathbf{z}$, where $\mathbf{y}$ are free variables and $\mathbf{z}$ are quantified ones.

Output: $\mathfrak{G}$ is a disjunctive normal form of $\mathbb{Q}$-polynomial equations and inequalities only in $\mathbf{y}$, such that the quantifier-free $\mathfrak{G}$ is equivalent to $\mathfrak{F}$. Specifically, each disjunct of $\mathfrak{G}$ defines a disjoint solution cell and all solution cells are cylindrically indexed.

Extended usage. In our current setting, we regard the input constraint formulated in Lemma 5.3 as a first-order formula $\mathfrak{F}(y_0, \mathbf{y})$, where y_0 is the first variable newly-introduced to symbolise e. Although $\mathfrak{F}(y_0, \mathbf{y})$ has no quantified variable, we cannot clearly know its solutions, even the existence of solutions. We invoke $\mathsf{decompose}(\mathfrak{F}(y_0, \mathbf{y}))$ to make it clear. The output $\mathfrak{G}(y_0, \mathbf{y})$ would cylindrically define all solution cells. As y_0 symbolises e, we focus only on those solution cells whose first components (l_0, u_0) or $\{c_0\}$ w. r. t. y_0 contain e, and drop others. Specifically, if the component for y_0 is a proper interval (l_0, u_0), the check $\mathrm{e} \in (l_0, u_0)$ can be completed by sufficiently approaching e, since l_0 and u_0 are real algebraic while e is transcendental; if the component for y_0 is a point interval $\{c_0\}$, we can immediately drop it due to $c_0 \neq \mathrm{e}$. After substituting $y_0 = \mathrm{e}$, we can simplify $\mathfrak{G}(\mathrm{e}, \mathbf{y})$ by deleting invalid disjuncts. So the resulting formula cylindrically defines all solutions of the input constraint, which is a clear representation. □

In fact, cylindrical algebraic decomposition has been well implemented in many computer algebra tools, such as REDLOG [24]. Besides, one can also use the efficient SMT tool **Z3** [21], that supports transcendental extensions, to find one solution for the input constraint, and thereby decides whether a scheduler would be demonic somewhere.

Finally, by Lemma 5.3 and the extended usage of Algorithm 1, we conclude with:

Theorem 5.6. *The time-bounded termination problem is solvable over parametric PPDs.*

6. Bounded Running Time

In the previous sections, we have studied the case where running time plus waiting time τ is bounded. Now we turn to the case where only the running time τ_1 is bounded while the total time τ (as well as the waiting time τ_2) varies. Under this setting one can see that the induced PP still corresponds to a finite MA. When τ_2 is sufficiently large, those weights $\mathcal{P}_2$ of sinking paths tend to 1. So the demonic scheduler in long-term could be easily determined. However, how to determine the demonic scheduler for not sufficiently large τ_2 is a more interesting problem. Namely, in this section, we study:

Problem 6.1. *What is the demonic scheduler for a PPD when the running time is bounded and the waiting time varies?*

To solve it, we first reduce it to the real root isolation problem over exponential polynomials, elements of the ring $\mathbb{Q}[\mathrm{e}, x, \mathrm{e}^x]$ (that is obtained by substituting $w = \mathrm{e}$ and $y = \mathrm{e}^x$ into the trivariate polynomial ring $\mathbb{Q}[w, x, y]$). Then we present an isolation algorithm for those functions, which develops the existing ones for $\mathbb{Q}[x, \mathrm{e}^x]$ [2, 56].

Lemma 6.2. *Problem 6.1 can be reduced to the real root isolation over the ring* $\mathbb{Q}[\mathrm{e}, x, \mathrm{e}^x]$.

Proof. The proof is similar to that of Theorem 4.4. The only difference lies in that τ is a real parameter and τ_1 is a known integer. For each sinking path ς in the induced PP, by substituting $\tau_2 = \tau - \tau_1$ into Proposition 4.3, the measure of all sinking paths of this type is an element of $\mathbb{Q}[\mathrm{e}, \tau, \mathrm{e}^{-\tau}]$. Let x be $-\tau$. Then, for a fixed scheduler θ, the termination probability $\mathcal{P}(\theta)$ with $\tau_1 \leq \Delta$ is the fraction (4), whose numerator and denominator are in the form

$$\alpha_0(x) + \alpha_1(x)\mathrm{e}^{\beta_1} + \cdots + \alpha_m(x)\mathrm{e}^{\beta_m} \ , \tag{9}$$

where $\alpha_0(x), \ldots, \alpha_m(x)$ are elements of $\mathbb{Q}[x, \mathrm{e}^x]$ and $\beta_1, \ldots, \beta_m$ are distinct positive integers. So comparing $\mathcal{P}(\theta_1)$ and $\mathcal{P}(\theta_2)$ for any pair of schedulers θ_1 and θ_2 amounts to finding real roots of an element of $\mathbb{Q}[\mathrm{e}, x, \mathrm{e}^x]$. Once we know how to find real roots for $\mathbb{Q}[\mathrm{e}, x, \mathrm{e}^x]$ (particularly for $x \in (-\infty, -\Delta)$), we can eventually offer finitely many intervals w. r. t. x, on each of which a scheduler would be demonic everywhere. $\square$

Example 6.3. *Here we reconsider the PPD in Figure 1(a). Let the running time* τ_1 *be bounded by* 20 *units of time, and the total time* τ *a real parameter. Then the results on all fully acted schedulers can be found in Table 6, in which* $\gamma_1, \ldots, \gamma_6$ *are given below. All of them are elements of* $\mathbb{Q}[\mathrm{e}, \tau, \mathrm{e}^{-\tau}]$.

$$\begin{aligned}\gamma_1 = {} & \tfrac{15}{128} + [\tfrac{82944405}{2048} - \tfrac{4785885}{1024}\tau + \tfrac{185895}{1024}\tau^2 - \tfrac{1215}{512}\tau^3]\mathrm{e}^{20-\tau} \\ & + [\tfrac{44642745}{128} - \tfrac{3131055}{64}\tau + \tfrac{149445}{64}\tau^2 - \tfrac{1215}{32}\tau^3]\mathrm{e}^{40-2\tau} \\ & + [-\tfrac{32579565}{2048} + \tfrac{557505}{128}\tau - \tfrac{233145}{512}\tau^2 + \tfrac{5535}{256}\tau^3 - \tfrac{405}{1024}\tau^4]\mathrm{e}^{60-3\tau} \ ,\end{aligned}$$

$$
\begin{aligned}
\gamma_2 &= \tfrac{7}{64} + [-\tfrac{229635}{256} + \tfrac{2835}{32}\tau - \tfrac{567}{256}\tau^2]\mathrm{e}^{16-\tau} + [\tfrac{316953}{64} - \tfrac{18711}{32}\tau + \tfrac{567}{32}\tau^2]\mathrm{e}^{32-2\tau} \\
&\quad + [\tfrac{190883}{256} - \tfrac{24297}{128}\tau + \tfrac{4221}{256}\tau^2 - \tfrac{63}{128}\tau^3]\mathrm{e}^{48-3\tau} \quad , \\
\gamma_3 &= \tfrac{1}{16} + [\tfrac{783}{64} - \tfrac{27}{32}\tau]\mathrm{e}^{12-\tau} + [\tfrac{675}{16} - \tfrac{27}{8}\tau]\mathrm{e}^{24-2\tau} + [-\tfrac{1399}{64} + \tfrac{39}{8}\tau - \tfrac{9}{32}\tau^2]\mathrm{e}^{36-3\tau} \quad , \\
\gamma_4 &= \tfrac{3}{8} + [\tfrac{2673}{4} - \tfrac{243}{8}\tau]\mathrm{e}^{17-\tau} + [\tfrac{2165049}{8} - \tfrac{169371}{4}\tau + \tfrac{8991}{4}\tau^2 - \tfrac{81}{2}\tau^3]\mathrm{e}^{34-2\tau} \\
&\quad + [\tfrac{47001}{4} - \tfrac{22797}{8}\tau + \tfrac{945}{4}\tau^2 - \tfrac{27}{4}\tau^3]\mathrm{e}^{51-3\tau} \quad , \\
\gamma_5 &= \tfrac{3}{16} + [\tfrac{2673}{8} - \tfrac{243}{16}\tau]\mathrm{e}^{17-\tau} + [\tfrac{2165049}{16} - \tfrac{169371}{8}\tau + \tfrac{8991}{8}\tau^2 - \tfrac{81}{4}\tau^3]\mathrm{e}^{34-2\tau} \\
&\quad + [\tfrac{47001}{8} - \tfrac{22797}{16}\tau + \tfrac{945}{8}\tau^2 - \tfrac{27}{8}\tau^3]\mathrm{e}^{51-3\tau} \quad , \\
\gamma_6 &= \tfrac{1}{4} - \tfrac{27}{4}\mathrm{e}^{13-\tau} + [\tfrac{11583}{4} - \tfrac{783}{2}\tau + \tfrac{27}{2}\tau^2]\mathrm{e}^{26-2\tau} + [-\tfrac{1573}{4} + \tfrac{165}{2}\tau - \tfrac{9}{2}\tau^2]\mathrm{e}^{39-3\tau} \quad .
\end{aligned}
$$

schedulers	$\sum_{\varsigma} \mathcal{P}_1 \cdot \mathcal{P}_2$	schedulers	$\sum_{\varsigma} \mathcal{P}_1 \cdot \mathcal{P}_2$
LLLL	$\gamma_1 + \gamma_2 + \gamma_3$	LRR	γ_4
LLLR	$\gamma_2 + \gamma_3$	RLL	$\gamma_5 + \gamma_6$
LLR	γ_3	RLR	$\gamma_5 + \gamma_6$
LRL	γ_4	RR	γ_6

Table 6: Schedulers in PPD with bounded running time

From Table 6 and the set inclusion, we can see that

- $\mathcal{P}(\text{LLLL}) > \mathcal{P}(\text{LLLR}) > \mathcal{P}(\text{LLR})$*, as the set of terminating paths under* LLLL *is a superset of that under* LLLR *and the set of terminating paths under* LLLR *is a superset of that under* LLR*;*
- $\mathcal{P}(\text{LRL}) = \mathcal{P}(\text{LRR})$*; and*
- $\mathcal{P}(\text{RLL}) = \mathcal{P}(\text{RLR}) > \mathcal{P}(\text{RR})$*, as the set of terminating paths under* RLL *is a superset of that under* RR.

Hence, to determine the demonic scheduler, it suffices to compare $\mathcal{P}(\text{LLR})$*,* $\mathcal{P}(\text{LRL})$ *and* $\mathcal{P}(\text{RR})$*, which will be finished later.*

There exist some real root isolation algorithms for exponential polynomials $\mathbb{Q}[x, \mathrm{e}^x]$, e. g. [2, 56]. But they are not suitable for $\mathbb{Q}[\mathrm{e}, x, \mathrm{e}^x]$ due to the Euler's constant e appearing in coefficients. The hardness lies in ruling out multiple real roots, without which the algorithm may not halt. We will overcome it by a profound number-theoretic conjecture on *transcendence degree* that is the maximum number of algebraically independent elements in a given number field.

Conjecture 6.4 (Schanuel (1960s), see [5])**.** *Let* $\beta_1, \ldots, \beta_m$ *be* $\mathbb{Q}$*-linearly independent complex numbers. Then the field extension*

$$\mathbb{Q}(\beta_1, \mathrm{e}^{\beta_1}, \ldots, \beta_m, \mathrm{e}^{\beta_m}) : \mathbb{Q}$$

has transcendence degree at least m.

Proposition 6.5. *Let φ_1 and φ_2 be two co-prime elements of* $\mathbb{Q}[\mathrm{e}, x] \setminus \mathbb{Q}[\mathrm{e}]$. *Then φ_1 and φ_2 have no common real root.*

Proof. Suppose that φ_1 and φ_2 have a common real root x_0. Then $\varphi_1(x_0) = \varphi_2(x_0) = 0$ hold. Let ϕ be the Sylvester's resultant [18, Page 141] of φ_1 and φ_2 that eliminates the component x in $\mathbb{Q}[\mathrm{e}, x]$. Since φ_1 and φ_2 are co-prime, ϕ is a nonzero $\mathbb{Q}$-polynomial in e, which does not equal zero by the fact that e is transcendental. However, $\varphi_1(x_0) = \varphi_2(x_0) = 0$ imply $\phi(x_0) = 0$, which yields a contradiction. Hence φ_1 and φ_2 have no common real root. □

Proposition 6.6. *Let φ_1 and φ_2 be two co-prime elements of* $\mathbb{Q}[\mathrm{e}, x, \mathrm{e}^x] \setminus \mathbb{Q}[\mathrm{e}]$. *Then all common real roots of φ_1 and φ_2 are rational.*

Proof. Suppose that φ_1 and φ_2 have a common irrational root x_0. Then $\varphi_1(x_0) = \varphi_2(x_0) = 0$ hold. Assume w. l. o. g. that at least one of φ_1 and φ_2, saying φ_1, has positive degree in e^x. (Otherwise φ_1 and φ_2 have no common real root by Proposition 6.5, and thereby this proposition holds trivially.) Let ϕ be the Sylvester's resultant of φ_1 and φ_2 that eliminates the component e^x in $\mathbb{Q}[\mathrm{e}, x, \mathrm{e}^x]$. Since φ_1 and φ_2 are co-prime, ϕ is a nonzero $\mathbb{Q}$-polynomial in e, x, satisfying $\phi(x_0) = 0$. By $\phi(x_0) = 0$ we have x_0 algebraically dependents on e, and by $\varphi_1(x_0) = 0$ we have e^{x_0} algebraically dependents on e, x_0. In other words, the field extension

$$\mathbb{Q}(1, \mathrm{e}, x_0, \mathrm{e}^{x_0}) : \mathbb{Q} = \mathbb{Q}(\mathrm{e}, x_0, \mathrm{e}^{x_0}) : \mathbb{Q} = \mathbb{Q}(\mathrm{e}, x_0) : \mathbb{Q} = \mathbb{Q}(\mathrm{e}) : \mathbb{Q}$$

has transcendence degree 1. However, since x_0 is irrational, by Conjecture 6.4, we have that $\mathbb{Q}(1, \mathrm{e}, x_0, \mathrm{e}^{x_0}) : \mathbb{Q}$ has transcendence degree at least 2, which yields a contradiction. Hence φ_1 and φ_2 have no common irrational root. □

Proposition 6.7. *Let φ be a square-free element of* $\mathbb{Q}[\mathrm{e}, x, \mathrm{e}^x] \setminus \mathbb{Q}[\mathrm{e}]$. *Then all multiple real roots of φ are rational.*

Proof. It suffices to show that an irreducible element φ of $\mathbb{Q}[\mathrm{e}, x, \mathrm{e}^x] \setminus \mathbb{Q}[\mathrm{e}, x]$ has no multiple irrational root. We discuss it in two cases.

- If φ has the divisor e^x, it must be a constant multiple of e^x by irreducibility. Then φ has no real root, and this corollary holds trivially.
- Otherwise, the *tail polynomial* of φ that consists of those terms with degree 0 in e^x is nonzero. Specifically, it either has positive degree in x or is a nonzero constant. Let $\psi = \varphi'$ be the derivative of φ. Then ψ has the same positive degree in e^x as φ, and the tail polynomial of ψ either has less degree in x than that of φ or is the zero constant. So φ and ψ are co-prime. By Proposition 6.6 we have that φ and ψ have no common irrational root, which entails that φ has no multiple irrational root. □

Now we are ready to find all multiple real roots of a square-free element φ of $\mathbb{Q}[\mathrm{e}, x, \mathrm{e}^x]$. Let ϕ be the Sylvester's resultant of φ and φ' that eliminates the component e^x in $\mathbb{Q}[\mathrm{e}, x, \mathrm{e}^x]$. Multiple real roots of φ must be rational roots of ϕ. The latter comes from $\mathbb{Q}$-linear divisors of ϕ (elements of $\mathbb{Q}[x]$), which can be easily obtained by factorisation over $\mathbb{Q}[\mathrm{e}, x]$.

In what follows, we present the real root isolation algorithm for $\mathbb{Q}[\mathrm{e}, x, \mathrm{e}^x]$ (see Algorithm 2), which develops the ones in [2, 56]. Its rationale is that we isolate real roots of the original function after those of the simpler derivative have been isolated. All technical essentials and the correctness analysis will be provided below the statement of Algorithm 2.

Algorithm 2 Real Root Isolation for $\mathbb{Q}[\mathrm{e}, x, \mathrm{e}^x]$

$$\{\mathcal{I}_1, \ldots, \mathcal{I}_n\} \Leftarrow \mathsf{Isolate}(\varphi, \mathcal{I})$$

Input: φ is a square-free element of $\mathbb{Q}[\mathrm{e}, x, \mathrm{e}^x]$, satisfying that e^x is not a divisor of φ, defined on the rational interval $\mathcal{I} = (a, b)$.

Output: $\mathcal{I}_1, \ldots, \mathcal{I}_n$ are finitely many disjoint rational intervals (in ascending order), such that each contains exactly one real root of φ in $\mathcal{I}$, and together contain all.

```
1: if $\varphi \in \mathbb{Q}[\mathrm{e}, \mathrm{e}^x] \cup \mathbb{Q}[\mathrm{e}, x]$ then
2:     isolate all real roots of $\varphi$ in $\mathcal{I}$;
3:     return the above isolation intervals in ascending order;
4: if $\varphi(a) \cdot \varphi(b) = 0$ then
5:     adjust slightly the endpoints of $\mathcal{I}$, such that $\varphi(a) \cdot \varphi(b) \neq 0$;
6: let $\psi$ be the greatest square-free divisor of $\varphi'$ over $\mathbb{Q}[\mathrm{e}, x, \mathrm{e}^x]$;
7: if $\mathrm{e}^x$ is a divisor of $\psi$ then
8:     reset $\psi \leftarrow \psi/\mathrm{e}^x$;
9: let $a_1, \ldots, a_k$ be all common real roots of $\varphi$ and $\psi$ in $\mathcal{I}$;
10: set $a_0 \leftarrow a$ and $a_{k+1} \leftarrow b$;
11: for $i = 0, \ldots, k$ do
12:     $\{\mathcal{J}_1, \ldots, \mathcal{J}_{m_i}\} \Leftarrow \mathsf{Isolate}(\psi, (a_i, a_{i+1}))$;
13:     for $j = 1, \ldots, m_i$ do
14:         repeat
15:             refine isolation interval $\mathcal{J}_j$ for $\psi$;
16:         until $0 \notin \mathrm{Range}(\varphi, \mathrm{closure}(\mathcal{J}_j))$
17:     let $\mathcal{L}_i \leftarrow \emptyset$, $\mathcal{J}_0 \leftarrow [a_i, a_i]$ and $\mathcal{J}_{m_i+1} \leftarrow [a_{i+1}, a_{i+1}]$;
18:     for $j = 0, \ldots, m_i$ do
19:         if $\varphi(\sup \mathcal{J}_j) \cdot \varphi(\inf \mathcal{J}_{j+1}) < 0$ then
20:             update $\mathcal{L}_i \leftarrow \mathcal{L}_i \sqcup \{(\sup \mathcal{J}_j, \inf \mathcal{J}_{j+1})\}$;   ▷ '$\sqcup$' denotes the union on ordered sets
21: return $\mathcal{L}_0 \sqcup \{[a_1, a_1]\} \sqcup \mathcal{L}_1 \sqcup \cdots \sqcup \{[a_k, a_k]\} \sqcup \mathcal{L}_k$.
```

Usage. We first compute the greatest square-free divisor φ of the input exponential polynomial by factorisation over $\mathbb{Q}[\mathrm{e}, x, \mathrm{e}^x]$, and drop the divisor e^x if it has. The upper (resp. lower) bound for all real roots of φ can be found as the threshold, after (resp. before) which there is a dominating term in φ whose sign is the sign of the whole φ. After obtaining a lower bound a and an upper bound b for all real roots of φ, we turn to invoke $\mathsf{Isolate}(\varphi, (a, b))$. □

More specifically, we note that:

The sign-determination in Lines 4 & 19 can be finished using the fact that e is transcendental. For instance, if $\varphi(a)$ is the zero constant, the sign is clear; otherwise the sign would be clear when e is sufficiently approached.

The shifting perturbation ϵ in Line 5, saying for $a \leftarrow a + \epsilon$, can be chosen as any positive number less than

$$\min\left(\frac{\varphi^{(K)}(a)}{\sup\limits_{x\in[a,a+\varepsilon]} |\varphi^{(K+1)}(x)|}, \varepsilon\right) \quad \text{for certain } \varepsilon > 0 \ ,$$

where K is the minimum integer, satisfying the Kth derivative $\varphi^{(K)}(a) \neq 0$.

Frankly, we cannot obtain the exact range estimate in Line 16. However, by interval arithmetic [56, Formula (9)], the function range confined in $\mathcal{I}$ would be over-approximated within any error bound along with the refinement in interval estimates for e and e^x, which can be offered by classic continued fractions [56, Corollary 2.6].

Correctness. We first notice that the depth of recursion in Line 12 is finite, since either ψ has lower degree in e^x than that of φ, or the tail polynomial of ψ has lower degree in x than that of φ while ψ keeps the same degree in e^x as φ from the construction in Lines 6–8. We then prove the correctness by induction.

- Basically, when φ is a polynomial only in x or only in e^x, its real roots can be easily isolated by the existing methods, e. g. the one in [18, Page 148].
- Inductively, we can find all common real roots $a_1, \ldots, a_k$ of φ and ψ by Proposition 6.7. In each (a_i, a_{i+1}), once isolation intervals $\mathcal{J}_1, \ldots, \mathcal{J}_{m_i}$ are known for real roots of ψ, we refine them by the loop in Lines 14–16 until their closures contain no real root of φ. Then we check all intervals $(\sup \mathcal{J}_0, \inf \mathcal{J}_1), \ldots, (\sup \mathcal{J}_{m_i}, \inf \mathcal{J}_{m_i+1})$, complementing to closure$(\mathcal{J}_1), \ldots,$ closure$(\mathcal{J}_{m_i})$ in (a_i, a_{i+1}), in each of which φ is monotonous, and thus easily isolate all real roots of φ in (a_i, a_{i+1}) as an individual list $\mathcal{L}_i$.
- Finally all real roots of φ in $\mathcal{I}$ would be collected in ascending order as the union in Line 21. □

Example 6.8. *Let us compare* $\mathcal{P}(\mathrm{LLR})$, $\mathcal{P}(\mathrm{LRL})$ *and* $\mathcal{P}(\mathrm{RR})$ *that was left in Example 6.3.*

The critical points of $\mathcal{P}(\mathrm{LLR}) = \mathcal{P}(\mathrm{LRL})$ *are the real roots of* $\varphi = \mathrm{e}^{3\tau} \cdot [\gamma_3 - \gamma_4]$, *an element of* $\mathbb{Q}[\mathrm{e}, \tau, \mathrm{e}^\tau]$. *We first choose* 40 *as an upper bound for all real roots of* φ. *Then we turn to isolate all real roots of* φ' *in* $(20, 40)$, *which amount to those of the greatest square-free divisor* ψ *of* φ' *without the divisor* e^x. *As* φ *and* ψ *have no common real root, no splitting in* $(20, 40)$ *is necessary. After a finite depth of recursion, we eventually get the unique real root of* ψ *in* $(20, 40)$, *which can be refined to* $(\frac{5314365}{262144}, \frac{10628735}{524288})$, *so that* φ *has no real root in the closure* $[\frac{5314365}{262144}, \frac{10628735}{524288}]$. *Finally we can get the desired unique real root of* φ *in* $(20, 40)$. *In fact, we have implemented Algorithm 2 to automatically provide the aforementioned isolation interval* $(\frac{10628735}{524288}, 40)$ *of* φ, *which can be refined to any precision.*

Additionally, $\mathcal{P}(\mathrm{LLR}) = \mathcal{P}(\mathrm{RR})$ *has no critical point in* $(20, +\infty)$. *The interval* $(20, +\infty)$ *is finally partitioned into two subintervals:*

- *the schedulers "*LRL*" and "*LRR*" are demonic on the left subinterval,*
- *the scheduler "*LLR*" is demonic on the right subinterval, and*
- *both are demonic on that critical point (*≈ 20.5734*).*

Remark 6.9. *The critical point in the above example is "obvious", so that it can be efficiently approached using traditional numerical computation. But numerical computation usually fails to identify a tangent real root of* $\mathcal{P}(\theta_1) - \mathcal{P}(\theta_2)$ *(saying, distinguish a tangent real root with a pair of very close distinct real roots) in advance, at which it suffers the trouble of missing real roots. In other words, comparing* $\mathcal{P}(\theta_1)$ *and* $\mathcal{P}(\theta_2)$ *are not effective then. Our method overcomes this trouble by Proposition 6.7.*

Finally, by Lemma 6.2 and the usage of Algorithm 2, we conclude with:

Theorem 6.10. *The termination problem 6.1 is solvable under Schanuel's conjecture.*

7. Complexity Analysis

Now we are to analyse the complexity of the proposed methods along with the time bound Δ for an input PPD. The size of the PPD is constant, and only Δ is the size parameter of the input. Our methods widely carry on the sign-determination, i. e. determining the sign of an element φ of $\mathbb{Q}[\mathrm{e}, x, \mathrm{e}^x]$ when x is confined in a rational interval $\mathcal{I}$ that contains a real root of another element of $\mathbb{Q}[\mathrm{e}, x, \mathrm{e}^x]$. Namely, we regard $\mathrm{Range}(\varphi, \mathcal{I})$ as a trivariate function, varying with the interval estimates for $\mathrm{e}, \mathcal{I}, \mathrm{e}^x$. After the case of $\varphi = 0$ is ruled out by pretreatment, the sign-determination can be finished by sufficiently precise interval estimates for $\mathrm{e}, \mathcal{I}, \mathrm{e}^x$. Otherwise we would either refine the isolation interval $\mathcal{I}$ or refine the interval estimates for e, e^x. Although these procedures are effective, how precise interval estimates are enough is unknown. In other words, we do not know the required times of refinements in advance. To complete the complexity analysis, we define two computable oracle machines as follows.

- The oracle machine χ_1 completes the refinement in e, such that the sign of $\varphi(a)$ is clear for a given constant a.
- The oracle machine χ_2 completes the refinement in $\mathcal{I}, \mathrm{e}, \mathrm{e}^x$, such that the sign of $\varphi(\mathcal{I})$ is clear. (Obviously χ_2 subsumes χ_1.)

For PPDs, we first notice that the number of sinking paths ς in the induced PP is exponential w. r. t. Δ, as well as the number of fully acted schedulers. For every type ς, the probability of sinking paths σ in PPD can be computed in polynomial time by integration-by-parts. So, for a fixed scheduler, we can obtain the time-bounded termination probability as an element fraction, whose numerator and denominator (elements of $\mathbb{Q}[\mathrm{e}]$) involve at most exponentially many ς. Then we compare probabilities under different schedulers, which uses χ_1 on an exponentially-sized element of $\mathbb{Q}[\mathrm{e}]$. The demonic scheduler can be eventually determined after at most exponentially many times of such comparison. Hence Theorem 4.4 is in $\mathbf{EXP}^{\chi_1}$.

For parametric PPDs, each time of comparison performs cylindrical algebraic decomposition [18] on exponentially-sized elements φ of $\mathbb{Q}[y_0, \mathbf{y}]$, which is double-exponential time in the number of parameters and polynomial time in $\|\varphi\|$. Since the size of the input PPD is constant, as well as the number of parameters, the decomposition is in **EXP**. After decomposition, it produces at most exponentially many solution cells that are defined by exponentially-sized elements of $\mathbb{Q}[y_0, \mathbf{y}]$. To recognise those solution cells with first components w. r. t. y_0 containing e, we carry on at most exponentially many times of sign-determination for boundaries of the first components at $y_0 = \mathrm{e}$, each of which uses χ_1 on exponentially-sized elements of $\mathbb{Q}[\mathrm{e}]$. Hence Theorem 5.6 is still in $\mathbf{EXP}^{\chi_1}$.

For PPDs with bounded running time, each time of comparison performs real root isolation on an exponentially-sized element φ of $\mathbb{Q}[\mathrm{e}, x, \mathrm{e}^x]$. Then the depth of recursion is bounded by $\|\varphi\|$. Specifically, for an exponential polynomial $\varphi = \sum_i p_i(x)\mathrm{e}^{k_i x}$ with $p_i \in \mathbb{Q}[\mathrm{e}, x] \setminus \{0\}$, the depth is bounded by $\sum_i(\deg_x(p_i) + 1)$. During the whole isolation, the number of all occurring isolation intervals (i. e. the number of real roots of all derivatives ψ occurring in the recursion procedure) is bounded by $[\sum_i(\deg_x(p_i) + 1)] \cdot [\sum_i(\deg_x(p_i) + 1) + 1]/2 \leq \|\varphi\|^2/2$. So the isolation uses at most exponentially many times of χ_1 and χ_2, and other operations (including factorisation over

$\mathbb{Q}[\mathrm{e}, x, \mathrm{e}^x]$ [37]) are totally polynomial time in $\|\varphi\|$. Hence Theorem 6.10 is in $\mathbf{EXP}^{\chi_1} + \mathbf{EXP}^{\chi_2} = \mathbf{EXP}^{\chi_2}$.

8. Conclusion

We have studied several time-bounded termination problems over PPDs. All these problems can be solved by number-theoretical and algebraic methods, which are of exact quantitative reasoning. We have also implemented a prototype of our methods. Finally we have given the complexity upper bounds for the proposed methods. Furthermore, we believe that the *angelic* scheduler that maximises the termination probability could be determined in a symmetric way.

For further work, we are interested in extending the proposed methods to recursive PPDs [47, 16] and the general parameter-synthesis and model-repair problems [32], and applying them to some case studies. The prototype could be incorporated with algorithmic learning to reuse the middle results, thus improve the practical efficiency.

Acknowledgements

The authors are grateful to the anonymous reviewers whose careful and insightful comments significantly improve the presentation and resolve inconsistencies. The authors also thank Chengchao Huang for drawing the figures. M. Xu is funded by the National Natural Science Foundation of China (11871221), the National Key R&D Program of China (2018YFA0306704), and the Fundamental Research Funds for the Central Universities. Y. Deng is funded by the National Natural Science Foundation of China (61672229, 61832015, 62072176) and the Inria-CAS joint project Quasar.

References

[1] Abdulla, P. A., 2011. Carrying probabilities to the infinite world. In: Katoen, J.-P., König, B. (Eds.), Proceedings of the 22nd International Conference on Concurrency Theory. Vol. 6901 of LNCS. Springer, pp. 1–16.

[2] Achatz, M., McCallum, S., Weispfenning, V., 2008. Deciding polynomial–exponential problems. In: Proceedings of the 33rd International Symposium on Symbolic and Algebraic Computation. ACM, pp. 215–222.

[3] Agrawal, S., Chatterjee, K., Novotný, P., 2018. Lexicographic ranking supermartingales: An efficient approach to termination of probabilistic programs. Proceedings of the ACM on Programming Languages 2 (POPL), 34:1–34:32.

[4] Arons, T., Pnueli, A., Zuck, L. D., 2003. Parameterized verification by probabilistic abstraction. In: Gordon, A. D. (Ed.), Proceedings of the 6th International Conference on Foundations of Software Science and Computational Structures. Vol. 2620 of LNCS. Springer, pp. 87–102.

[5] Ax, J., 1971. On Schanuel's conjectures. Annals of Mathematics 93 (2), 252–268.

[6] Aziz, A., Sanwal, K., Singhal, V., Brayton, R., 1996. Verifying continuous time Markov chains. In: Alur, R., Henzinger, T. A. (Eds.), Proceedings of the 8th International Conference on Computer Aided Verification. Vol. 1102 of LNCS. Springer, pp. 269–276.

[7] Baier, C., Haverkort, B. R., Hermanns, H., Katoen, J.-P., 2000. Model checking continuous-time markov chains by transient analysis. In: Emerson, E. A., Sistla, A. P. (Eds.), Proceedings of the 12th International Conference on Computer Aided Verification. Vol. 1855 of LNCS. Springer, pp. 358–372.

[8] Baier, C., Katoen, J.-P., 2008. Principles of Model Checking. MIT Press.

[9] Baker, A., 1975. Transcendental Number Theory. Cambridge University Press.

[10] Ben-Amram, A. M., Genaim, S., 2014. Ranking functions for linear-constraint loops. Journal of the ACM 61 (4), 26:1–26:55.

[11] Bournez, O., Garnier, F., 2005. Proving positive almost-sure termination. In: Giesl, J. (Ed.), Proceedings of the 16th International Conference on Term Rewriting and Applications. Vol. 3467 of LNCS. Springer, pp. 323–337.

[12] Bradley, A. R., Manna, Z., Sipma, H. B., 2005. The polyranking principle. In: Caires, L., Italiano, G. F., Monteiro, L., Palamidessi, C., Yung, M. (Eds.), Proceedings of the 32nd International Colloquium on Automata, Languages and Programming. Vol. 3580 of LNCS. Springer, pp. 1349–1361.

[13] Cai, X., Zhou, X., 2000. Asymmetric earliness and tardiness scheduling with exponential processing times on an unreliable machine. Annals of Operations Research 98 (1-4), 313–331.

[14] Chakarov, A., Sankaranarayanan, S., 2013. Probabilistic program analysis with martingales. In: Sharygina, N., Veith, H. (Eds.), Proceedings of the 25th International Conference on Computer Aided Verification. Vol. 8044 of LNCS. Springer, pp. 511–526.

[15] Chatterjee, K., Fu, H., Goharshady, A. K., 2016. Termination analysis of probabilistic programs through positivstellensatz's. In: Chaudhuri, S., Farzan, A. (Eds.), Proceedings of the 28th International Conference on Computer Aided Verification, Part I. Vol. 9779 of LNCS. Springer, pp. 3–22.

[16] Chatterjee, K., Fu, H., Goharshady, A. K., 2017. Non-polynomial worst-case analysis of recursive programs. In: Majumdar, R., Kuncak, V. (Eds.), Proceedings of the 29th International Conference on Computer Aided Verification, Part II. Vol. 10427 of LNCS. Springer, pp. 41–63.

[17] Chatterjee, K., Fu, H., Murhekar, A., 2017. Automated recurrence analysis for almost-linear expected-runtime bounds. In: Majumdar, R., Kuncak, V. (Eds.), Proceedings of the 29th International Conference on Computer Aided Verification, Part I. Vol. 10426 of LNCS. Springer, pp. 118–139.

[18] Collins, G. E., 1975. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In: Brakhage, H. (Ed.), Proceedings of the 2nd GI Conference on Automata Theory and Formal Languages. Vol. 33 of LNCS. Springer, pp. 134–183.

[19] Colón, M. A., Sipma, H. B., 2001. Synthesis of linear ranking functions. In: Margaria, T., Yi, W. (Eds.), Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Vol. 2031 of LNCS. Springer, pp. 67–81.

[20] Cook, B., Podelski, A., Rybalchenko, A., 2006. TERMINATOR: Beyond safety. In: Ball, T., Jones, R. B. (Eds.), Proceedings of the 18th International Conference on Computer Aided Verification. Vol. 4144 of LNCS. Springer, pp. 415–418.

[21] de Moura, L., Passmore, G. O., 2013. Computation in real closed infinitesimal and transcendental extensions of the rationals. In: Bonacina, M. P. (Ed.), Proceedings of the 24th International Conference on Automated Deduction. Vol. 7898 of LNCS. Springer, pp. 178–192.

[22] Deng, Y., Hennessy, M., 2013. On the semantics of markov automata. Information and Computation 222, 139–168.

[23] Dijkstra, E. W., 1976. A Discipline of Programming. Prentice-Hall.

[24] Dolzmann, A., Sturm, T., 1997. REDLOG: Computer algebra meets computer logic. ACM SIGSAM Bulletin 31 (2), 2–9.

[25] Eisentraut, C., Hermanns, H., Zhang, L., 2010. On probabilistic automata in continuous time. In: Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science. IEEE Computer Society, pp. 342–351.

[26] Esparza, J., Gaiser, A., Kiefer, S., 2012. Proving termination of probabilistic programs using patterns. In: Proceedings of the 24th International Conference on Computer Aided Verification. Vol. 7358 of LNCS. Springer, pp. 123–138.

[27] Fioriti, L. M. F., Hermanns, H., 2015. Probabilistic termination: Soundness, completeness, and compositionality. In: Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. ACM, pp. 489–501.

[28] Giesl, J., Brockschmidt, M., Emmes, F., Frohn, F., Fuhs, C., Otto, C., Plücker, M., Schneider-Kamp, P., Ströder, T., Swiderski, S., Thiemann, R., 2014. Proving termination of programs automatically with AProVE. In: Demri, S., Kapur, D., Weidenbach, C. (Eds.), Proceedings of the 7th International Joint Conference on Automated Reasoning. Vol. 8562 of LNCS. Springer, pp. 184–191.

[29] Goldstine, H. H., von Neumann, J., 1963. Planning and coding problems for an electronic computing instrument. In: von Neumann, J., Traub, A. H. (Eds.), The Collected Works of John von Neumann. Vol. 5. Pergamon Press, pp. 80–235.

[30] Gordon, A. D., Henzinger, T. A., Nori, A. V., Rajamani, S. K., 2014. Probabilistic programming. In: Proceedings on Future of Software Engineering. ACM, pp. 167–181.

[31] Hajek, B. E., 1983. The proof of a folk theorem on queuing delay with applications to routing in networks. Journal of the ACM 30 (4), 834–851.

[32] Han, T., Katoen, J.-P., Mereacre, A., 2008. Approximate parameter synthesis for probabilistic time-bounded reachability. In: Proceedings of the 29th IEEE Real-Time Systems Symposium. IEEE Computer Society, pp. 173–182.

[33] He, J., Seidel, K., McIver, A., 1997. Probabilistic models for the guarded command language. Science of Computer Programming 28 (2–3), 171–192.

[34] Hoare, C. A. R., 1969. An axiomatic basis for computer programming. Communications of the ACM 12 (10), 576–580.

[35] Hyytiä, E., Aalto, S., 2016. On round–robin routing with FCFS and LCFS scheduling. Performance Evaluation 97, 83–103.

[36] Jansen, N., Dehnert, C., Kaminski, B. L., Katoen, J.-P., Westhofen, L., 2016. Bounded model checking for probabilistic programs. In: Artho, C., Legay, A., Peled, D. (Eds.), Proceedings of the 14th International Symposium on

Automated Technology for Verification and Analysis. Vol. 9938 of LNCS. Springer, pp. 68–85.

[37] Kaltofen, E., 1985. Polynomial-time reductions from multivariate to bi- and univariate integral polynomial factorization. SIAM Journal on Computing 14 (2), 469–489.

[38] Kaminski, B. L., Katoen, J.-P., 2015. On the hardness of almost-sure termination. In: Italiano, G. F., Pighizzini, G., Sannella, D. (Eds.), Proceedings of the 40th International Symposium on Mathematical Foundations of Computer Science, Part I. Vol. 9234 of LNCS. Springer, pp. 307–318.

[39] Kaminski, B. L., Katoen, J.-P., Matheja, C., Olmedo, F., 2016. Weakest precondition reasoning for expected runtimes of probabilistic programs. In: Thiemann, P. (Ed.), Proceedings of the 25th European Symposium on Programming. Vol. 9632 of LNCS. Springer, pp. 364–389.

[40] Katoen, J.-P., Gretz, F., Jansen, N., Kaminski, B. L., Olmedo, F., 2015. Understanding probabilistic programs. In: Meyer, R., Platzer, A., Wehrheim, H. (Eds.), Correct System Design. Vol. 9360 of LNCS. Springer, pp. 15–32.

[41] Kozen, D., 1979. Semantics of probabilistic programs. In: Proceedings of the 20th Annual Symposium on Foundations of Computer Science. IEEE Computer Society, pp. 101–114.

[42] Kozen, D., 1983. A probabilistic PDL. In: Proceedings of the 15th Annual ACM Symposium on Theory of Computing. ACM, pp. 291–297.

[43] Manna, Z., 1974. Mathematical Theory of Computation. McGraw-Hill.

[44] McIver, A., Morgan, C., 2006. Abstraction, Refinement and Proof for Probabilistic Systems. Springer.

[45] McIver, A., Morgan, C., Kaminski, B. L., Katoen, J., 2018. A new proof rule for almost-sure termination. Proceedings of the ACM on Programming Languages 2 (POPL), 33:1–33:28.

[46] Monniaux, D., 2001. An abstract analysis of the probabilistic termination of programs. In: Cousot, P. (Ed.), Proceedings of the 8th International Symposium on Static Analysis. Vol. 2126 of LNCS. Springer, pp. 111–126.

[47] Olmedo, F., Kaminski, B. L., Katoen, J.-P., Matheja, C., 2016. Reasoning about recursive probabilistic programs. In: Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science. ACM, pp. 672–681.

[48] Pinedo, M., 1995. Scheduling: Theory, Algorithms, and Systems. Prentice-Hall.

[49] Podelski, A., Rybalchenko, A., 2004. A complete method for the synthesis of linear ranking functions. In: Steffen, B., Levi, G. (Eds.), Proceedings of the 5th International Conference on Verification, Model Checking, and Abstract Interpretation. Vol. 2937 of LNCS. Springer, pp. 239–251.

[50] Ross, S., 1983. Introduction to Stochastic Dynamic Programming. Adacemic Press.

[51] Sharir, M., Pnueli, A., Hart, S., 1984. Verification of probabilistic programs. SIAM Journal on Computing 13 (2), 292–314.

[52] Stewart, W. J., 1994. Introduction to the Numerical Solution of Markov Chains. Princeton University Press.

[53] Tanenbaum, A. S., 2007. Modern Operating Systems, 3rd Edition. Pearson.

[54] Wang, P., Fu, H., Chatterjee, K., Deng, Y., Xu, M., 2020. Proving expected sensitivity of probabilistic programs with randomized variable-dependent termination. Proceedings of the ACM on Programming Languages 4 (POPL), 25:1–25:30.

[55] Weiss, G., Pinedo, M., 1980. Scheduling tasks with exponential service times on nonidentical processors to minimize makespan or flow time. Journal of Applied Probability 17 (1), 187–202.

[56] Xu, M., Chen, L., Zeng, Z., Li, Z.-B., 2010. Reachability analysis of rational eigenvalue linear systems. International Journal of Systems Science 41 (12), 1411–1419.

[57] Xu, M., Zhang, L., Jansen, D. N., Zhu, H., Yang, Z., 2016. Multiphase until formulas over Markov reward models: An algebraic approach. Theoretical Computer Science 611, 116–135.

[58] Zhang, L., Jansen, D. N., Nielson, F., Hermanns, H., 2012. Efficient CSL model checking using stratification. Logical Methods in Computer Science 8 (2), 17:1–17:18.