Model Checking Stochastic Branching Processes[⋆]

Taolue Chen, Klaus Dräger, and Stefan Kiefer

University of Oxford, UK
{taolue.chen,klaus.draeger,stefan.kiefer}@cs.ox.ac.uk

Abstract. Stochastic branching processes are a classical model for describing random trees, which have applications in numerous fields including biology, physics, and natural language processing. In particular, they have recently been proposed to describe parallel programs with stochastic process creation. In this paper, we consider the problem of model checking stochastic branching process. Given a branching process and a deterministic parity tree automaton, we are interested in computing the probability that the generated random tree is accepted by the automaton. We show that this probability can be compared with any rational number in PSPACE, and with 0 and 1 in polynomial time. In a second part, we suggest a tree extension of the logic PCTL, and develop a PSPACE algorithm for model checking a branching process against a formula of this logic. We also show that the qualitative fragment of this logic can be model checked in polynomial time.

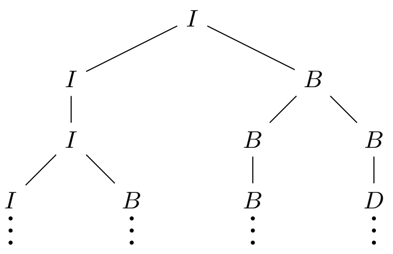
1 Introduction

Consider an interactive program featuring two types of threads: interruptible threads (type I) and blocking threads (type B) which perform a non-interruptible computation or database transaction. An I-thread responds to user commands which occasionally trigger the creation of a B-thread. A B-thread may either terminate, or continue, or spawn another B-thread in an effort to perform its tasks in parallel. Under probabilistic assumptions on the thread behaviour, this scenario can be modelled as a *stochastic branching process* as follows:

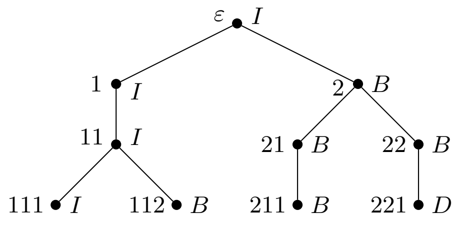
$$\begin{array}{lll} I \xhookrightarrow{0.9} I & B \xhookrightarrow{0.2} D & D \xhookrightarrow{1} D \\ I \xhookrightarrow{0.1} (I,B) & B \xhookrightarrow{0.5} B & \\ & B \xhookrightarrow{0.3} (B,B) & \end{array} \qquad (1)$$

This means, e.g., that a single step of an I-thread spawns a B-thread with probability 0.1. We have modelled the termination of a B-thread as a transformation

[⋆] This work was partially supported by the ERC Advanced Grant VERIWARE and EPSRC grant EP/F001096/1. Stefan Kiefer is supported by a DAAD postdoctoral fellowship.



(a) A prefix of a tree that the example process might create.



(b) A finite tree over $\{I, B, D\}$.

Fig. 1. Figures for Section 1 (left) and 2 (right)

into a "dead" state D.[1] A "run" of this process unravels an infinite tree whose branches record the computation of a thread and its ancestors. For example, Figure 1(a) shows the prefix of a tree that the example process might create. The probability of creating this tree prefix is the product of the probabilities of the applied rules, i.e., $0.1 \cdot 0.9 \cdot 0.1 \cdot 0.3 \cdot 0.5 \cdot 0.2$.

This example is an instance of a (stochastic multitype) branching process, which is a classical mathematical model with applications in numerous fields including biology, physics and natural language processing, see e.g. [12, 2]. In [13] an extension of branching processes was introduced to model parallel programs with stochastic process creation. The broad applicability of branching processes arises from their simplicity: each *type* models a class of threads (or tasks, animals, infections, molecules, grammatical structures) with the same probabilistic behaviour.

This paper is about model checking the random trees created by branching processes. Consider a specification that requires a linear-time property to hold along all tree branches. In the example above, e.g., we may specify that "no process should become forever blocking", more formally, "on all branches of the tree we see infinitely many I or D". We would like to compute the probability that all branches satisfy such a given ω-regular word property. Curiously, this problem generalises two seemingly very different classical problems:

(i) If all rules in the branching process are of the form $X \overset{p}{\hookrightarrow} Y$, i.e., each node has exactly one child, the branching process describes a finite-state Markov chain. Computing the probability that a run of such a Markov chain satisfies an ω-regular property is a standard problem in probabilistic verification, see e.g. [5, 16].

(ii) If for each type X in the branching process there is only one rule $X \overset{1}{\hookrightarrow} \alpha$ (where α is a nonempty sequence of types), then the branching process describes a unique infinite tree. Viewing the types in α as possible successor

[1] We disallow "terminating" rules like $B \xhookrightarrow{0.2} \varepsilon$. This is in contrast to classical branching processes, but technically more convenient for model checking, where absence of deadlocked states is customarily assumed.

states of X in a finite nondeterministic transition system, the branches in the created tree are exactly the possible runs in the finite transition systems. Of course, checking if all runs in such a transition system satisfy an ω-regular specification is also a well-understood problem.

One could expect that well-known Markov-chain based techniques for dealing with problem (i) can be generalised to branching processes. This is *not* the case: it follows from our results that in the example above, the probability that all branches satisfy the mentioned property is 0;[2] however, if the numbers 0.2 and 0.3 in (1) are swapped, the probability changes from 0 to 1. This is in sharp contrast to finite-state Markov chains, where qualitative properties (satisfaction with probability 0 resp. 1) do not depend on the exact probability of individual transitions.

The rules of a branching process are reminiscent of the rules of probabilistic pushdown automata (pPDA) or the equivalent model of recursive Markov chains (RMCs). However, the model-checking algorithms for both linear-time and branching-time logics proposed for RMCs and pPDAs [8, 10, 11] do *not* work for branching processes, essentially because pPDA and RMCs specify Markov chains, whereas branching processes specify random trees. Branching processes cannot be transformed to pPDAs, at least not in a straightforward way. Note that if the rules in the example above are understood as pPDA rules with I as starting symbol, then B will never even occur as the topmost symbol.

To model check branching processes, we must leave the realm of Markov chains and consider the probability space in terms of tree prefixes [12, 2]. Consequently, we develop algorithms that are very different from the ones dealing with the special cases (i) and (ii) above. Nevertheless, for qualitative problems (satisfaction with probability 0 resp. 1) our algorithms also run in polynomial time with respect to the input models, even for branching processes that do not conform to the special cases (i) and (ii) above.

Instead of requiring a linear-time property to hold on all branches, we consider more general specifications in terms of *deterministic parity tree automata*. In a nutshell, our model-checking algorithms work as follows: (1) compute the "product" of the input branching process and the tree automaton; (2) reduce the analysis of the resulting product process to the problem of computing the probability that all branches reach a "good" symbol; (3) compute the latter probability by setting up and solving a nonlinear equation system. Step (1) can be seen as an instance of the automata-theoretic model-checking approach. The equation systems of step (3) are of the form $\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{x})$, where $\boldsymbol{x}$ is a vector of variables, and $\boldsymbol{f}(\boldsymbol{x})$ is a vector of polynomials with nonnegative coefficients. Solutions to such equation systems can be computed or approximated efficiently [10, 7, 9]. Step (2) is, from a technical point of view, the main contribution of the paper; it requires a delicate and nontrivial analysis of the behaviour of branching processes.

In Section 4 we also consider logic specifications. We propose a new logic, PTTL, which relates to branching processes in the same manner as the logic

[2] Intuitively, this is because a B-thread more often clones itself than dies.

PCTL relates to Markov chains. Recall that PCTL contains formulae such as $[\phi \mathsf{U} \psi]_{\geq p}$ which specifies that the probability of runs satisfying $\phi \mathsf{U} \psi$ is at least p. For PTTL we replace the linear-time subformulae such as $\phi \mathsf{U} \psi$ with tree subformulae such as $\phi \mathsf{EU} \psi$ or $\phi \mathsf{AU} \psi$, so that, e.g., $[\phi \mathsf{EU} \psi]_{\geq p}$ specifies that the probability of trees that have a branch satisfying $\phi \mathsf{U} \psi$ is at least p, and $[\phi \mathsf{AU} \psi]_{\geq p}$ specifies that the probability of trees all whose branches satisfy $\phi \mathsf{U} \psi$ is at least p. We show that branching processes can be model checked against this logic in PSPACE, and against its qualitative fragment in polynomial time.

Related work. The rich literature on branching processes (see e.g. [12, 2] and the references therein) does not consider model-checking problems. Probabilistic split-join systems [13] are branching processes with additional features for process synchronisation and communication. The paper [13] focuses on performance measures (such as runtime, space and work), and does not provide a functional analysis. The models of pPDAs and RMCs also feature dynamic task creation by means of procedure calls, however, as discussed above, the existing model-checking algorithms [8, 10, 11] do not work for branching processes. Several recent works [10, 7, 9] have studied the exact and approximative solution of fixed-point equations of the above mentioned form. Our work connects these algorithms with the model-checking problem for branching processes.

Organisation of the paper. After some preliminaries (Section 2), we present our results on parity specifications in Section 3. In Section 4 we propose the new logic PTTL and develop model-checking algorithms for it. We conclude in Section 5. Some proofs have been moved to a technical report [4].

2 Preliminaries

We let $\mathbb{N}$ and $\mathbb{N}_0$ denote the set of positive and nonnegative integers, respectively. Given a finite set Γ, we write $\Gamma^* := \bigcup_{k \in \mathbb{N}_0} \Gamma^k$ for the set of tuples and $\Gamma^+ := \bigcup_{k \in \mathbb{N}} \Gamma^k$ for the set of nonempty tuples over Γ.

Definition 1 (Branching process). *A* branching process *is a tuple $\Delta = (\Gamma, \hookrightarrow, Prob)$ where Γ is a finite set of types, $\hookrightarrow \subseteq \Gamma \times \Gamma^+$ is a finite set of transition rules, Prob is a function assigning positive probabilities to transition rules so that for every $X \in \Gamma$ we have that $\sum_{X \hookrightarrow \alpha} Prob(X \hookrightarrow \alpha) = 1$.*

We write $X \overset{p}{\hookrightarrow} \alpha$ if $Prob(X \hookrightarrow \alpha) = p$. Observe that since the set of transition rules is finite, there is a global upper bound K_Δ such that $|\alpha| \leq K_\Delta$ for all $X \hookrightarrow \alpha$.

A *tree* is a nonempty prefix-closed language $V \subseteq \mathbb{N}^*$ for which there exists a function $\beta_V : V \to \mathbb{N}_0$ such that for all $w \in V$ and $k \in \mathbb{N}$, $wk \in V$ if and only if $k \leq \beta_V(w)$. $\beta_V(w)$ is called the *branching degree* of w in V. We denote by B_f the set of finite trees, and by B_i the set of infinite trees without leaves (i.e. trees such that $\beta_V(w) > 0$ for all $w \in V$). A *prefix* of V is a tree $V' \subseteq V$ such that for all $w \in V'$, $\beta_{V'}(w) \in \{0, \beta_V(w)\}$.

A *tree over Γ* is a pair (V, ℓ) where V is a tree, and $\ell : V \to \Gamma$ is a labelling function on the nodes. Given a tree $t = (V, \ell)$ with a node $u \in V$, we write $t_u = (V_u, \ell_u)$ for the subtree of t rooted at u; here $V_u = \{w \in \mathbb{N}^* \mid uw \in V\}$ and $\ell_u(w) = \ell(uw)$ for $w \in V_u$. A tree (V', ℓ') is a *prefix* of (V, ℓ) if V' is a prefix of V and $\ell'(w) = \ell(w)$ for all $w \in V'$.

A *path* (resp. *branch*) in a tree $t = (V, \ell)$ is a finite (resp. infinite) sequence $u_0, u_1, \ldots$ with $u_i \in V$ such that $u_0 = \epsilon$ is the root of t, and $u_{i+1} = u_i k_i$ for $k_i \in \mathbb{N}$ is a child of u_i. A *branch label* of t is a sequence $\ell(u_0), \ell(u_1), \ldots$, where $u_0, u_1, \ldots$ is a branch. The *successor word* of a node $w \in V$ is $\sigma_t(w) = \ell(w1) \ldots \ell(w\beta_V(w))$.

Given a tree $t = (V, \ell)$ over Γ and a subset $W \subseteq V$, we write $t \models \mathsf{AF}W$ if all its branches go through W, i.e., for all $v \in V$ there is a $w \in W$ such that v is a predecessor of w or vice versa. If $\Lambda \subseteq \Gamma$, we write $t \models \mathsf{AF}\Lambda$ for $t \models \mathsf{AF}\{w \in V \mid \ell(w) \in \Lambda\}$. Similarly, we write $t \models \mathsf{AG}\Lambda$ if $\ell(w) \in \Lambda$ for all $w \in V$.

Example 2. We illustrate these notions. Figure 1(b) shows a finite tree $t = (V, \ell) \in B_f$ over Γ with $\Gamma = \{I, B, D\}$ and $V = \{\varepsilon, 1, 11, 111, 112, 2, 21, 211, 22, 221\}$ and, e.g., $\ell(\varepsilon) = I$ and $\ell(112) = B$. We have $\beta_V(\varepsilon) = 2$ and $\beta_V(21) = 1$ and $\beta_V(211) = 0$. The node 2 is a predecessor of 211. The tree $t' = (V', \ell')$ with $V' = \{\varepsilon, 1, 2, 21, 22\}$ and ℓ' being the restriction of ℓ on V' is a prefix of t. The sequence $\varepsilon, 2, 21$ is a path in t. We have $\sigma_t(11) = IB$. The tree satisfies $t \models \mathsf{AF}\{1, 21, 221\}$ and $t \models \mathsf{AF}\{I\}$.

A tree $t = (V, \ell)$ over Γ is *generated* by a branching process $\Delta = (\Gamma, \hookrightarrow, \mathit{Prob})$ if for every $w \in V$ with $\beta_V(w) > 0$ we have $\ell(w) \hookrightarrow \sigma_t(w)$. We write $(\!|\Delta|\!)$ and $[\![\Delta]\!]$ for the sets of trees (V, ℓ) generated by Δ with $V \in B_f$ and $V \in B_i$, respectively. For any $X \in \Gamma$, $(\!|\Delta|\!)_X \subseteq (\!|\Delta|\!)$ and $[\![\Delta]\!]_X \subseteq [\![\Delta]\!]$ contain those trees (V, ℓ) for which $\ell(\epsilon) = X$.

Definition 3 (Probability space of trees, cf. [12, Chap. VI]). *Let $\Delta = (\Gamma, \hookrightarrow, \mathit{Prob})$ be a branching process. For any finite tree $t = (V, \ell) \in (\!|\Delta|\!)$, let the* cylinder *over t be $\mathit{Cyl}_\Delta(t) := \{t' \in [\![\Delta]\!] \mid t \text{ is a prefix of } t'\}$, and define $p_\Delta(t) := \prod_{w \in V : \beta_V(w) > 0} \mathit{Prob}(\ell(w), \sigma_t(w))$. For each $X \in \Gamma$ we define a probability space $([\![\Delta]\!]_X, \Sigma_X, \Pr_X)$, where Σ_X is the σ-algebra generated by $\{\mathit{Cyl}_\Delta(t) \mid t \in (\!|\Delta|\!)_X\}$, and $\Pr_X$ is the probability measure generated by $\Pr_X(\mathit{Cyl}_\Delta(t)) = p_\Delta(t)$. Sometimes we write $\Pr_X^\Delta$ to indicate Δ. We may drop the subscript of $\Pr_X$ if X is understood. We often write t_X to mean a tree $t \in [\![\Delta]\!]_X$ randomly sampled according to the probability space above.*

Example 4. Let $\Delta = (\Gamma, \hookrightarrow, \mathit{Prob})$ be the branching process with $\Gamma = \{I, B, D\}$ and the rules as given in (1) on page 1. The tree t from Figure 1(b) is generated by Δ: we have $t \in (\!|\Delta|\!)_I$. We have $\Pr_I(\mathit{Cyl}_\Delta(t)) = p_\Delta(t) = 0.1 \cdot 0.9 \cdot 0.1 \cdot 0.3 \cdot 0.5 \cdot 0.2$; this is probability of those trees $t' \in [\![\Delta]\!]_I$ that have prefix t.

We say that a quantity $q \in [0, 1]$ is *PPS-expressible* if one can compute, in polynomial time, an integer $m \in \mathbb{N}$ and a fixed-point equation system $\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{x})$, where $\boldsymbol{x}$ is a vector of m variables, $\boldsymbol{f}$ is a vector of m multivariate polynomials over $\boldsymbol{x}$ with nonnegative rational coefficients, $\boldsymbol{f}(\mathbf{1}) \leq \mathbf{1}$ where $\mathbf{1}$ denotes the vector $(1, \ldots, 1)$, and q is the first component of the least nonnegative solution $\boldsymbol{y} \in [0, \infty)^m$ of $\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{x})$.

Proposition 5. *Let q be PPS-expressible. We have:*

(a) For $\tau \in \{0,1\}$ one can decide in (strongly) polynomial time whether $q = \tau$.
(b) For $\tau \in \mathbb{Q}$ one can decide in polynomial space whether $q \bowtie \tau$, where $\bowtie \in \{<, >, \leq, \geq, =, \neq\}$.
(c) One can approximate q within additive error 2^{-j} in time polynomial in both j and the (binary) representation size of $\boldsymbol{f}$.

Part (a) follows from [10, 6]. Part (b) is shown in [10, section 4] by appealing to the existential fragment of the first-order theory of the reals, which is decidable in PSPACE, see [3, 14]. Part (c) follows from a recent result [9, Corollary 4.5]. The following proposition follows from a classical result on branching processes [12].

Proposition 6. *Let $\Delta = (\Gamma, \hookrightarrow, Prob)$ be a branching process. Let $X \in \Gamma$ and $\Lambda \subseteq \Gamma$. Then $\Pr[t_X \models \mathsf{AF}\Lambda]$ is PPS-expressible.*

3 Parity Specifications

In this section we show how to compute the probability of those trees that satisfy a given parity specification.

A *(top-down) deterministic (amorphous) parity tree automaton* (DPTA) is a tuple $\mathcal{A} = (Q, \Gamma, q_0, \delta, c)$, where Q is the finite set of states, $q_0 \in Q$ is the initial state, $\delta : Q \times \Gamma \times \mathbb{N} \to Q^*$ is the transition function satisfying $|\delta(q, X, n)| = n$ for all q, X, n, and $c : Q \to \mathbb{N}$ is a colouring function. Such an automaton $\mathcal{A}$ maps a tree $t = (V, \ell)$ over Γ to the (unique) tree $\mathcal{A}(t) = (V, \ell')$ over Q such that $\ell(\varepsilon) = q_0$ and for all $w \in V$, $\sigma_{A(t)}(w) = \delta(\ell'(w), \ell(w), \beta_V(w))$.

Automaton $\mathcal{A} = (Q, \Gamma, q_0, \delta, c)$ *accepts* a tree t over Γ if for all branch labels $q_0 q_1 \cdots \in Q^\omega$ of $\mathcal{A}(t)$ the highest colour that occurs infinitely often in $c(q_0), c(q_1), \ldots$ is even.

Example 7. Recall (e.g., from [15]) that any ω-regular *word* property (e.g., any LTL specification) can be translated into a deterministic parity *word* automaton. Such an automaton, in turn, can be easily translated into a DPTA which specifies that the labels of *all* branches satisfy the ω-regular word property. We do not spell out the translation, but let us note that in the resulting tree automaton, for all $(q, X) \in Q \times \Gamma$ there is $q' \in Q$ such that $\delta(q, X, k) = (q', \ldots, q')$ for all k.

Given a colouring function $c : \Gamma \to \mathbb{N}$, a tree (V, ℓ) over Γ is called *good* for c if for each branch $u_0, u_1, \cdots$ the largest number that occurs infinitely often in the sequence $c(\ell(u_0)), c(\ell(u_1)), \ldots$ is even. The following proposition is immediate.

Proposition 8. *Let $\Delta = (\Gamma, \hookrightarrow, Prob)$ be a branching process, and let $\mathcal{A} = (Q, \Gamma, q_0, \delta, c)$ be a DPTA. Define the* product *of Δ and $\mathcal{A}$ as the branching process $\Delta_\bullet = (\Gamma \times Q, \hookrightarrow_\bullet, Prob_\bullet)$ with $(X, q) \overset{p}{\hookrightarrow}_\bullet (Y_1, q_1) \ldots (Y_k, q_k))$ for $X \overset{p}{\hookrightarrow}$*

$Y_1 \ldots Y_k$, where $(q_1, \ldots, q_k) = \delta(q, X, k)$. Define $c_\bullet : \Gamma \times Q \to \mathbb{N}$ by $c_\bullet(X, q) := c(q)$. Then for all $X \in \Gamma$ we have

$$\Pr_X^{\Delta}[t \text{ is accepted by } \mathcal{A}] = \Pr_{(X,q_0)}^{\Delta_\bullet}[t \text{ is good for } c_\bullet]\,.$$

In view of Proposition 8, it suffices to compute the probability $\Pr[t_X \text{ is good for } c]$, where a branching process $\Delta = (\Gamma, \hookrightarrow, \mathit{Prob})$ with $X \in \Gamma$ and a colouring function $c : \Gamma \to \mathbb{N}$ are fixed for the rest of the section. We write $\Pr[t_X \text{ is good}]$ if c is understood. We distinguish between the *qualitative problem*, i.e., computing whether $\Pr[t_X \text{ is good}] = 1$ holds for a given $X \in \Gamma$, and the *quantitative problem*, i.e., the computation of $\Pr[t_X \text{ is good}]$.

3.1 The Qualitative Problem

The outline of this subsection is the following: We will show that the qualitative problem can be solved in polynomial time (Theorem 12). First we show (Proposition 9) that it suffices to compute all *clean* types, where "clean" is defined below. We will show (Lemma 11) that a type X is clean if and only if $\Pr[t_X \models \mathsf{AF}\Lambda] = 1$ holds for suitable set $\Lambda \subseteq \Gamma$. By Proposition 6 the latter condition can be checked in polynomial time, completing the qualitative problem.

If there exists a tree $(V, \ell) \in [\![\Delta]\!]_X$ and a node $u \in V$ with $\ell(u) = Y$, then we say that Y is *reachable* from X. Given $X \in \Gamma$ and a finite word $w = X_0 \cdots X_m \in \Gamma^+$, we say that w is *X-closing* if $m \geq 1$ and $X_m = X$ and $c(X_i) \leq c(X)$ for $0 \leq i \leq m$. A branch with label $X_0 X_1 \cdots \in \Gamma^\omega$ is called *X-branch* if $X_0 = X$ and there is a sequence $0 = m_0 < m_1 < m_2 < \cdots$ such that $X_{m_i} \cdots X_{m_{i+1}}$ is X-closing for all $i \in \mathbb{N}$. We say that a type $Y \in \Gamma$ is *odd* (resp. *even*), if $c(Y)$ is odd (resp. even). Observe that a tree t is good if and only if for all its vertices u and all *odd* types Y the subtree t_u does not have a Y-branch. A type $Y \in \Gamma$ is *clean* if Y is even or $\Pr[t_Y \text{ has a } Y\text{-branch}] = 0$. The following proposition reduces the qualitative problem to the computation of all clean types.

Proposition 9. *We have that $\Pr[t_X \text{ is good}] = 1$ if and only if all Y reachable from X are clean.*

Proof. If there is an unclean reachable Y, then $\Pr[t_Y \text{ is good}] < 1$ and so $\Pr[t_X \text{ is good}] < 1$. Otherwise, for each node v in t_X and for each odd Y we have that $\Pr[(t_X)_v \text{ has a } Y\text{-branch}] = 0$. Since the set of nodes in a tree is countable, it follows that almost surely no subtree of t_X has a Y-branch for odd Y; i.e., t_X is almost surely good. ☐

Call a path in a tree *X-closing* if the corresponding label sequence is X-closing. Given $X \in \Gamma$, we define

$$N_X := \{Y \in \Gamma \mid \text{ no tree in } [\![\Delta]\!]_Y \text{ has an } X\text{-closing path}\}\,.$$

Note that $c(Y) > c(X)$ implies $Y \in N_X$ and that N_X is computable in polynomial time. A word $X_0 X_1 \cdots \in (\Gamma^* \cup \Gamma^\omega)$ is called *X-failing* if no prefix is

X-closing and there is $i \geq 0$ with $X_i \in N_X$. A branch in a tree is called X-failing if the corresponding branch label is X-failing. Given $X \in \Gamma$ and a tree t, let $\mathsf{Clos}_X(t)$ (resp. $\mathsf{Fail}_X(t)$) denote the set of those nodes w in t such that the path to w is X-closing (resp. X-failing) and no proper prefix of this path is X-closing (resp. X-failing). We will need the following lemma.

Lemma 10. *Define the events* $C := \{t_X \mid t_X \models \mathsf{AF}\,(\mathsf{Clos}_X(t_X) \cup \mathsf{Fail}_X(t_X))\}$ *and* $I := \{t_X \mid \mathsf{Clos}_X(t_X)$ *is infinite*$\}$. *Then* $C \cap I = \emptyset$ *and* $\Pr[C \cup I] = 1$.

The following lemma states in particular that an odd type X is clean if and only if $\Pr[t_X \models \mathsf{AF}N_X] = 1$. We prove something slightly stronger:

Lemma 11. *Define the events* $F := \{t_X \mid t_X \models \mathsf{AF}N_X\}$ *and* $H := \{t_X \mid t_X$ *has an* X*-branch*$\}$. *Then* $F \cap H = \emptyset$ *and* $\Pr[F \cup H] = 1$.

Now we have:

Theorem 12. *One can decide in polynomial time whether* $\Pr[t_X$ *is good*$] = 1$.

Proof. By Proposition 9 it suffices to show that cleanness can be determined in polynomial time. By Lemma 11 an odd type X is clean if and only if $\Pr[t_X \models \mathsf{AF}N_X] = 1$. The latter condition is decidable in polynomial time by Proposition 6. □

Example 13. Consider the branching process with $\Gamma = \{1, 2, 3, 4\}$ and the rules $1 \xhookrightarrow{1/3} 11$, $1 \xhookrightarrow{2/3} 4$, $2 \xhookrightarrow{1/2} 13$, $2 \xhookrightarrow{1/2} 23$, $3 \xhookrightarrow{2/3} 33$, $3 \xhookrightarrow{1/3} 1$, $4 \xhookrightarrow{1} 4$, and the colouring function c with $c(i) = i$ for $i \in \{1, 2, 3, 4\}$. Using a simple reachability analysis one can compute the sets $N_1 = \{2, 3, 4\}$, $N_2 = \{1, 3, 4\}$, $N_3 = \{1, 4\}$, $N_4 = \emptyset$. Applying Proposition 6 we find $\Pr[t_3 \models \mathsf{AF}N_3] < 1 = \Pr[t_1 \models \mathsf{AF}N_1]$. It follows by Lemma 11 that the only *un*clean type is 3. Since type 3 is only reachable from 2 and from 3, Proposition 9 implies that $\Pr[t_X$ is good$] = 1$ holds if and only if $X \in \{1, 4\}$.

3.2 The Quantitative Problem

Define $G := \{X \in \Gamma \mid$ all Y reachable from X are clean$\}$. The following Proposition 14 states that $\Pr[t_X$ is good$] = \Pr[t_X \models \mathsf{AF}G]$. This implies, by Proposition 6, that the probability is PPS-expressible (see Theorem 15).

Proposition 14. *We have* $\Pr[t_X$ *is good*$] = \Pr[t_X \models \mathsf{AF}G]$.

This implies the following theorem.

Theorem 15. *For any* $X \in \Gamma$ *we have that* $\Pr[t_X$ *is good*$]$ *is PPS-expressible.*

Proof. By Proposition 14 we have $\Pr[t_X$ is good$] = \Pr[t_X \models \mathsf{AF}G]$. So we can apply Proposition 6 with $\Lambda := G$. Note that G can be computed in polynomial time, as argued in the proof of Theorem 12. □

Example 16. We continue Example 13, where we have effectively computed $G = \{1, 4\}$, and thus established that $\Pr[t_1 \text{ is good}] = \Pr[t_4 \text{ is good}] = 1$. By Proposition 14 the probabilities $\Pr[t_2 \text{ is good}]$ and $\Pr[t_3 \text{ is good}]$ are given by $\Pr[t_2 \models \mathsf{AF}G]$ and $\Pr[t_3 \models \mathsf{AF}G]$. Proposition 6 assures that these probabilities are PPS-expressible; in fact they are given by the least nonnegative solution of the equation system $[x_2 = \frac{1}{2}x_3 + \frac{1}{2}x_2x_3,\ x_3 = \frac{2}{3}x_3^2 + \frac{1}{3}]$, which is $x_2 = \frac{1}{3}$ and $x_3 = \frac{1}{2}$. Hence, we have $\Pr[t_2 \text{ is good}] = \frac{1}{3}$ and $\Pr[t_3 \text{ is good}] = \frac{1}{2}$.

A Lower Bound. We close the section with a hardness result in terms of the PosSLP problem, which asks whether a given straight-line program or, equivalently, arithmetic circuit with operations $+$, $-$, $\cdot$, and inputs 0 and 1, and a designated output gate, outputs a positive integer or not. PosSLP is in PSPACE, but known to be in NP. The PosSLP problem is a fundamental problem for numerical computation, see [1] for more details.

For given Γ with $D \in \Gamma$, consider the DPTA $\mathcal{A}_{hit} = (\{q, r\}, \Gamma, a, \delta, c)$ with $c(q) = 1$ and $c(r) = 2$; $\delta(q, X, 1) = (q)$ and $\delta(q, X, 2) = (q, q)$ for $X \in \Gamma \setminus \{D\}$; $\delta(q, D, 1) = (r)$ and $\delta(q, D, 2) = (r, r)$; $\delta(r, X, 1) = (r)$ and $\delta(r, X, 2) = (r, r)$ for $X \in \Gamma$. Automaton $\mathcal{A}_{hit}$ specifies that all branches satisfy the LTL property $\mathsf{F}D$, i.e., all branches eventually hit D. Let QUANT-HIT denote the problem to decide whether $\Pr_X^{\Delta}[t \text{ is accepted by } \mathcal{A}_{hit}] > p$ holds for a given branching process $\Delta = (\Gamma, \hookrightarrow, Prob)$ with $X \in \Gamma$ and a given rational $p \in (0, 1)$. By Theorem 15 and Proposition 5, QUANT-HIT is in PSPACE. We have the following proposition:

Proposition 17 (see Theorem 5.3 of [10]). *QUANT-HIT is PosSLP-hard.*

4 Logic Specifications

In this section, we propose a logic akin to PCTL, called *probabilistic tree temporal logic*, to specify the properties of random trees generated from a branching process. We also present model-checking algorithms for this logic.

Definition 18 (PTTL). Probabilistic Tree Temporal Logic (PTTL) *formulae over a set Σ of atomic propositions are defined by the following grammar:*

$$\begin{aligned}\phi, \phi' &::= \top \mid a \mid \neg\phi \mid \phi \wedge \phi' \mid [\psi]_{\bowtie r} \\ \psi &::= \mathsf{AX}\phi \mid \mathsf{EX}\phi \mid \phi\mathsf{AU}\phi' \mid \phi\mathsf{EU}\phi' \mid \phi\mathsf{AR}\phi' \mid \phi\mathsf{ER}\phi'\,,\end{aligned}$$

where $a \in \Sigma$, $\bowtie \in \{<, \le, \ge, >\}$, and $r \in \mathbb{Q} \cap [0, 1]$. If $r \in \{0, 1\}$ holds for all subformulae of a PTTL formula ϕ, we say that ϕ is in the qualitative fragment *of PTTL. We use standard abbreviations such as $\bot$ for $\neg\top$, $\mathsf{AF}\phi$ for $\top\mathsf{AU}\phi$, $\mathsf{EG}\phi$ for $\bot\mathsf{ER}\phi$, etc.*

For the PTTL semantics we need the notion of a *labelled* branching process, which is a branching process $\Delta = (\Gamma, \hookrightarrow, Prob)$ extended by a function $\chi : \Gamma \to 2^{\Sigma}$, where $\chi(X)$ indicates which atomic propositions the type X satisfies.

Definition 19 (Semantics of PTTL). *Given a labelled branching process $\Delta = (\Gamma, \hookrightarrow, Prob, \chi)$, we inductively define a* satisfaction relation $\models$ *as follows, where $X \in \Gamma$:*

$$\begin{array}{ll}
X \models \top & \\
X \models a & \Leftrightarrow a \in \chi(X) \\
X \models \neg\phi & \Leftrightarrow X \not\models \phi \\
X \models \phi \wedge \phi' & \Leftrightarrow X \models \phi \textit{ and } X \models \phi' \\
X \models [\psi]_{\bowtie r} & \Leftrightarrow \Pr^{\Delta}_X[t_X \models \psi] \bowtie r \\
t \models \mathsf{AX}\phi & \Leftrightarrow \textit{for all branches } u_0u_1\cdots \textit{ of } t \textit{ we have } \ell(u_1) \models \phi \\
t \models \phi\mathsf{AU}\phi' & \Leftrightarrow \textit{for all branches } u_0u_1\cdots \textit{ of } t \textit{ there exists } i \in \mathbb{N} \textit{ with} \\
 & \quad \ell(u_i) \models \phi' \textit{ and for all } 0 \le j < i \textit{ we have } \ell(u_j) \models \phi \\
t \models \phi\mathsf{AR}\phi' & \Leftrightarrow \textit{for all branches } u_0u_1\cdots \textit{ of } t \textit{ and for all } i \in \mathbb{N} \textit{ we have} \\
 & \quad \ell(u_i) \models \phi' \textit{ or there exists } 0 \le j < i \textit{ with } \ell(u_j) \models \phi
\end{array}$$

The modalities EX, EU *and* ER *are defined similarly, with "for all branches" replaced by "there exists a branch".*

We now present the model checking algorithm. The algorithm shares its basic structure with the well-known algorithm for (P)CTL and finite (probabilistic) transition systems. Given a PTTL formula ϕ, the algorithm recursively evaluates the truth values of the PTTL subformulae ψ of ϕ for all types. The boolean operators can be dealt with as in the CTL algorithm. Hence, it suffices to examine formulae of the form $[\psi]_{\bowtie r}$. Observe that we have $\mathsf{EX}\phi \equiv \neg\mathsf{AX}\neg\phi$ and $\phi\mathsf{ER}\phi' \equiv \neg(\neg\phi\mathsf{AU}\neg\phi')$ and $\phi\mathsf{EU}\phi' \equiv \neg(\neg\phi\mathsf{AR}\neg\phi')$ and

$$X \models [\neg\phi]_{\bowtie r} \text{ if and only if } X \models [\phi]_{\bar{\bowtie} 1-r}\,,$$

where $\bar{\bowtie} \in \{\ge, >, <, \le\}$ is the complement operator of $\bowtie \in \{<, \le, \ge, >\}$. Hence, it suffices to deal with the following three cases: (i) $X \models [\mathsf{AX}\phi]_{\bowtie r}$; (ii) $X \models [\phi\mathsf{AU}\psi]_{\bowtie r}$; (iii) $X \models [\phi\mathsf{AR}\psi]_{\bowtie r}$. We assume in the following case distinction that the algorithm has already computed the truth values of the subformulae ϕ, ψ. One could now construct a suitable DPTA for each of the cases (i)–(iii), and proceed according to the machinery of Section 3. Instead we present in the following a more direct and more efficient algorithm which takes advantage of the special shape of the linear-time operators X, U and R.

Case (i): We have $\Pr[t_X \models \mathsf{AX}\phi] = \sum_{\substack{X \overset{p}{\hookrightarrow} Y_1 \ldots Y_k \\ Y_1,\ldots,Y_k \models \phi}} p$, which is easy to compute. So one can decide in polynomial time whether $X \models [\mathsf{AX}\phi]_{\bowtie r}$.

Case (ii): We reduce the check of the $\phi\mathsf{AU}\psi$ modality to a check of AF. To this end, we define a branching process $\Delta' = (\Gamma \times \{0, \frac{1}{2}, 1\}, \hookrightarrow', Prob')$ which tracks the "status" of $\phi\mathsf{AU}\psi$. We define Δ' in terms of an auxiliary function $f_{\phi,\psi} : \Gamma \to \{0, \frac{1}{2}, 1\}$ with $f_{\phi,\psi}(Y) = 0$ if $Y \models \neg\phi \wedge \neg\psi$, $f_{\phi,\psi}(Y) = \frac{1}{2}$ if $Y \models \phi \wedge \neg\psi$, and $f_{\phi,\psi}(Y) = 1$ if $Y \models \psi$. For any rule $X \overset{p}{\hookrightarrow} Y_1 \ldots Y_k$ in Δ, there are three corresponding rules in Δ', namely $(X, 0) \overset{p}{\hookrightarrow} (Y_1, 0) \ldots (Y_k, 0)$,

$(X,1) \overset{p}{\hookrightarrow} (Y_1,1)\ldots(Y_k,1)$, and $(X,\frac{1}{2}) \overset{p}{\hookrightarrow} (Y_1, f_{\phi,\psi}(Y_1))\ldots(Y_k, f_{\phi,\psi}(Y_k))$. By this construction we achieve $\Pr^{\Delta}_X[t_X \models \phi \mathsf{AU} \psi] = \Pr^{\Delta'}_{X'}[t_{X'} \models \mathsf{AF}\Lambda]$ for $X' = (X, f_{\phi,\psi}(X))$ and $\Lambda := \Gamma \times \{1\}$. Hence, using Propositions 5 and 6 we obtain that whether $X \models [\phi \mathsf{AU} \psi]_{\bowtie r}$ holds is decidable in PSPACE; and in polynomial time for $r \in \{0,1\}$.
Case (iii): Similarly to case (ii) we reduce the check of $\phi \mathsf{AR} \psi$ to a check of AG. This time we define $\Delta' = (\Gamma \times \{0,\frac{1}{2},1\}, \hookrightarrow', \mathit{Prob}')$ in terms of an auxiliary function $g_{\phi,\psi} : \Gamma \to \{0,\frac{1}{2},1\}$ with $g_{\phi,\psi}(Y) = 0$ if $Y \models \neg\psi$, $g_{\phi,\psi}(Y) = \frac{1}{2}$ if $Y \models \neg\phi \wedge \psi$, $g_{\phi,\psi}(Y) = 1$ if $Y \models \phi \wedge \psi$. The rules of Δ' are defined as in case (ii), except that $f_{\phi,\psi}$ is replaced with $g_{\phi,\psi}$. By this construction we achieve $\Pr^{\Delta}_X[t_X \models \phi \mathsf{AR} \psi] = \Pr^{\Delta'}_{X'}[t_{X'} \models \mathsf{AG}\Lambda]$ for $X' = (X, g_{\phi,\psi}(X))$ and $\Lambda := \Gamma \times \{\frac{1}{2},1\}$. The following lemma allows to express this probability in terms of AF instead of AG:

Lemma 20. *Let $\Delta = (\Gamma, \hookrightarrow, \mathit{Prob})$ be a branching process. Let $\Lambda \subseteq \Gamma$ such that no type in Λ is reachable from any type in $\Gamma \setminus \Lambda$. Define $G := \{Y \in \Lambda \mid$ all types reachable from Y are in $\Lambda\}$. Let $X \in \Gamma$. Then $\Pr[t_X \models \mathsf{AG}\Lambda] = \Pr[t_X \models \mathsf{AF}G]$.*

To summarize case (iii): we have reduced AR to AG and then AG to AF. Hence, using Propositions 5 and 6 we obtain that whether $X \models [\phi \mathsf{AR} \psi]_{\bowtie r}$ holds is decidable in PSPACE; and in polynomial time for $r \in \{0,1\}$.

As the overall algorithm computes the truth values of the subformulae recursively, we have proved the following theorem:

Theorem 21. *Model checking branching processes against PTTL is in* PSPACE*. Model checking branching processes against the qualitative fragment of PTTL is in* P*.*

5 Conclusions and Future Work

Branching processes are a basic formalism for modelling probabilistic parallel programs with dynamic process creation. This paper is the first to consider the verification of branching processes, We have shown how to model check specifications given in terms of deterministic parity automata, a problem that unifies and strictly generalises linear-time model-checking problems for Markov chains and for (nonprobabilistic) nondeterministic transition systems. We have also provided model-checking algorithms for a new logic, PTTL, suitable for specifying probabilistic properties of random trees. To obtain these results we have provided reductions to computing the probability of hitting "good" states along all branches.

Future research in this area should involve:

– the complexity of the problem where the specification is an LTL formula required to hold on all branches;
– the problem where deterministic parity automata are replaced by other tree specification formalisms, such as CTL (or CTL*) formulae;

– extending the model-checking algorithms to accommodate the synchronisation and communication features of probabilistic split-join systems.

It seems that at least the latter two problems require additional techniques, as the children of a node in the branching process can no longer be treated independently.

Acknowledgements. We thank anonymous reviewers for their valuable feedback.

References

1. E. Allender, P. Bürgisser, J. Kjeldgaard-Pedersen, and P. B. Miltersen. On the complexity of numerical analysis. In *IEEE Conference on Computational Complexity*, pages 331–339, 2006.
2. K.B. Athreya and P.E. Ney. *Branching Processes.* Springer, 1972.
3. J. Canny. Some algebraic and geometric computations in PSPACE. In *STOC'88*, pages 460–467, 1988.
4. T. Chen, K. Dräger, and S. Kiefer. Model checking stochastic branching processes. Technical report, arxiv.org, 2012. Available at `http://arxiv.org/abs/1206.1317`.
5. C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42:857–907, 1995.
6. J. Esparza, A. Gaiser, and S. Kiefer. Computing least fixed points of probabilistic systems of polynomials. In *Proceedings of STACS*, pages 359–370, 2010.
7. J. Esparza, S. Kiefer, and M. Luttenberger. Computing the least fixed point of positive polynomial systems. *SIAM Journal on Computing*, 39(6):2282–2335, 2010.
8. J. Esparza, A. Kučera, and R. Mayr. Model checking probabilistic pushdown automata. In *LICS'04*, pages 12–21. IEEE, 2004.
9. K. Etessami, A. Stewart, and M. Yannakakis. Polynomial-time algorithms for multi-type branching processes and stochastic context-free grammars. In *Proceedings of STOC*, pages 579–588, 2012.
10. K. Etessami and M. Yannakakis. Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations. *Journal of the ACM*, 56(1):1–66, 2009.
11. K. Etessami and M. Yannakakis. Model checking of recursive probabilistic systems. *ACM Transactions on Computational Logic*, 13(2), 2012. To appear.
12. T.E. Harris. *The Theory of Branching Processes.* Springer, 1963.
13. S. Kiefer and D. Wojtczak. On probabilistic parallel programs with process creation and synchronisation. In *Proceedings of TACAS*, volume 6605 of *LNCS*, pages 296–310. Springer, 2011.
14. J. Renegar. On the computational complexity and geometry of the first-order theory of the reals. Parts I–III. *Journal of Symbolic Computation*, 13(3):255–352, 1992.
15. W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, Beyond Words, pages 389–455. Springer, 1997.
16. M.Y. Vardi. Probabilistic linear-time model checking: An overview of the automata-theoretic approach. In *Formal Methods for Real-Time and Probabilistic Systems*, volume 1601 of *LNCS*, pages 265–276. Springer, 1999.