A Progress Measure for Explicit-State Probabilistic Model-Checkers*

Xin Zhang and Franck van Breugel
DisCoVeri Group, Department of Computer Science and Engineering
York University, Toronto, Canada

Abstract

Verification of the source code of a probabilistic system by means of an explicit-state model-checker is challenging. In most cases, the model-checker will either run out of memory or will simply not terminate within any reasonable amount of time. We introduce the notion of a progress measure for such a model-checker. The progress measure returns a number in the interval [0, 1]. This number provides us a quantitative measure of the amount of progress the model-checker has made verifying a particular linear time property. The larger the number, the more progress the model-checker has made. We also show how to compute the progress measure for checking invariants.

Explicit-state model-checkers usually exploit search strategies such as depth-first search and breadth-first search to explore the transitions. We introduce several new search strategies that take the probabilities associated with the transitions into account. We compare the amount of progress made by the different search strategies.

1. Introduction

Model-checkers such as PRISM [4] have been successfully exploited to check properties of probabilistic systems. Such a verification tool considers a model of the system, rather than the actual source code of the system. A model is usually simpler than the source code and, hence, the model is generally easier to verify. However, the details from which the model abstracts are not considered in the verification effort and, hence, the results obtained when considering a model may be less precise. Whereas a tool that checks properties of a model is usually exploited to find errors in algorithms, a tool that considers the source code is generally used to detect coding errors.

In this paper, we consider the applicability of model-checkers to verify the source code of probabilistic systems. In particular, we focus on explicit-state model-checkers, that is, model-checkers in which the states of the systems are represented explicitly. For a comparison of explicit-state and symbolic model-checkers, we refer the reader to, for example, [3].

One may wonder if an explicit-state model-checker, such as Java PathFinder (JPF for short) [8], is suitable for verifying implementations of randomized algorithms. Consider the following Java snippet.

```
Random random = new Random();
long count = 0;
while (random.nextBoolean())
   count++;
```

The above snippet gives rise to a huge number of different states: more than 2^{64}. Hence, it will come as no surprise that an explicit-state model-checker either will run out of memory or will not complete its verification of the above very simple code snippet within any reasonable amount of time. The same applies for most implementations of randomized algorithms.

Since explicit-state model-checkers generally cannot fully verify implementations of randomized algorithms, it would be interesting to extend such a model-checker such that it keeps track of the amount of progress it has made with its verification effort. Simply counting the number of (states or) execution paths that have been checked is not very useful for several reasons. First of all, it may be very difficult or even impossible to determine the total number of potential execution paths. Hence, the number of execution paths that have been checked by the model-checker gives us very limited information about the amount of progress that has been made. Secondly, some execution paths are more likely to happen than others. For example, the nonterminating execution path of the above snippet occurs with probability zero. Checking this execution path amounts to no progress at all.

In this paper, we develop a theoretical framework to define the progress made by an explicit-state probabilistic model-checker when verifying a particular linear time property. Furthermore, we show how to compute the progress for the verification of invariants.

*This research is supported by NSERC.

The potential execution paths of the system being verified are represented by means of a probabilistic transition system. We use a sequence of transitions to represent the verification effort, or search, of the model-checker.

To capture the progress made by the model-checker, instead of counting the number of execution paths, we endow the set of potential execution paths with a σ-algebra and a probability measure. In this way, we obtain a probability space of execution paths. The measure of a particular set of execution paths relevant to the property being checked gives us a number in the interval [0, 1]. This number provides us a quantitative measure of the amount of progress the model-checker has made verifying the property. The larger the number, the more progress the model-checker has made.

For the case that the property being verified is an invariant, we will present an alternative characterization of the progress measure. We will show that the amount of progress for invariants is the measure of the set of execution paths that have been checked.

From the probabilistic transition system representing the system under verification, we construct another probabilistic transition system. To distinguish the two systems, we refer to the probabilistic transition system representing the system under verification as the complete system and the other system as the searched system. Assume that the transitions t_1, t_2, ..., t_n are those transitions of the complete system that have been explored by the model-checker. Then the states of the searched system are the source and target states of t_1, t_2, ..., t_n and a "sink" state $s_\perp$. For those states s of which the sum of the probabilities of the outgoing transitions is less than one, that is, those states that have outgoing transitions which have not been explored yet, we add a transition from s to $s_\perp$ with the remaining probability (so that the transition probabilities add up to one). The state $s_\perp$ has a transition to itself with probability one.

As we will show, to compute the amount of progress made by the model-checker after having explored the transitions $t_1, \ldots, t_n$ when checking an invariant, it suffices to consider the searched system as described in the previous paragraph. Let the "sink" state $s_\perp$ be the only state that satisfies the atomic proposition $\perp$. We will prove that the measure of the complement of the set of those executions of the searched system that satisfy the temporal logic formula true $\mathcal{U}$ $\perp$ corresponds to the progress measure of the search $t_1, t_2, \ldots, t_n$ in the complete system. Note that, according to [7, Corollary 2.4], this set of executions is measurable. To compute the measure of this set, we can use, for example, the algorithm of Courcoubetis and Yannakakis [2, Lemma 3.1.1.1].

Assume that we have constructed the searched system for the search $t_1, \ldots, t_n$. As we will show, rather than constructing a new searched system from scratch after transition t_{n+1} has been explored by the model-checker, we can construct the new searched system from the old one in constant time.

Explicit-state model-checkers such as JPF can check the transitions in different orders by using, for example, a depth-first search (DFS) or a breadth-first search (BFS). These search strategies do not take the probabilities of the transitions into account. In this paper, we will propose several new search strategies which use the probabilities associated with the transitions. To let transitions with the highest probability be searched first, our probability-first search (PFS) strategy sorts the enabled transitions by their probability. Our breadth-first probability-second search (BFPSS) is an enhancement of BFS in which transitions at the same level are sorted by their probability. Our randomized search (RS) randomly selects an enabled transition. The chance that a transition is selected is proportional to its probability.

Our progress measure allows us to compare the amount of progress these different search strategies make. As we will show, the different search strategies are incomparable. That is, for each pair of search strategies, we can construct a small complete system such that the one strategy makes faster progress than the other.

We have implemented several well-known randomized algorithms and compared their progress when being verified using different search strategies. We observed that different algorithms are best verified using different search strategies. Hence, being able to determine how much progress is made with a particular search strategy is useful for choosing an appropriate search strategy.

The two main contributions of this paper are the following. First of all, we introduce the notion of a progress measure. Secondly, we show how this progress measure can be computed when checking invariants. Furthermore, we propose several new search strategies. Our proposed theoretical framework has been implemented within JPF. The details of our implementation are discussed in [10].

When we verify the implementation of a randomized algorithm with JPF and the model-checker runs out of memory, our progress measure provides us with an indication how much progress the model-checker has made with its verification effort. In some examples, we have seen that DFS makes no progress (the progress measure is still zero when it runs out of memory) whereas the progress of PFS is very close to one. We believe that this type of information is useful when verifying implementations of randomized algorithms.

2. Probabilistic Transition Systems

We represent the system to be verified by the explicit-state model-checker as a probabilistic transition system. The model-checker explores the (states and) transitions of

the probabilistic transition system in a systematic way. Note that the set of transitions of a probabilistic transition system may be infinite. In that case, not all transitions can be explored by the model-checker.

Definition 1 *A probabilistic transition system is a tuple* $\langle S, T, AP, s_0, \text{source}, \text{target}, \text{prob}, \text{label}\rangle$ *consisting of*

- *a set* S *of states,*
- *a set* T *of transitions,*
- *a set AP of atomic propositions,*
- *an initial state* s_0*,*
- *a function* source : $T \to S$*,*
- *a function* target : $T \to S$*,*
- *a function* prob : $T \to (0, 1]$*, and*
- *a function* label : $S \to 2^{AP}$

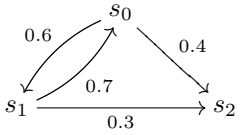
such that

- $s_0 \in S$ *and*
- *for all* $s \in S$, $\sum\{\, \text{prob}(t) \mid \text{source}(t) = s \,\} \in \{0, 1\}$.

Instead of $\langle S, T, s_0, \text{source}, \text{target}, \text{prob}, AP, L\rangle$ we usually write $\mathcal{S}$ and we denote, for example, its set of states by $S_{\mathcal{S}}$.

We will use the following probabilistic transition system as the running example for the rest of this paper.

Example 2 *The probabilistic transition system depicted by*



has 3 states and 4 transitions and a single atomic proposition p*. In this example, we use the indexes of the source and target to name the transitions. For example, the transition from* s_0 *to* s_2 *is named* t_{02}*. Given this naming convention, the functions* source *and* target *are defined in the obvious way. For example,* $\text{source}(t_{02}) = s_0$ *and* $\text{target}(t_{02}) = s_2$*. The function* prob *can be easily extracted from the above diagram. For example,* $\text{prob}(t_{02}) = 0.5$*. Atomic proposition* p *is satisfied in all states. Hence, for example,* $\text{label}(s_1) = \{p\}$.

For the remainder of this section, we fix a probabilistic transition system $\mathcal{S}$.

A state is final in $\mathcal{S}$ if it has no outgoing transitions. In Example 2, the state s_2 is final.

Definition 3 *A state* s *is final in* $\mathcal{S}$ *if* $\sum\{\, \text{prob}_{\mathcal{S}}(t) \mid \text{source}_{\mathcal{S}}(t) = s \,\} = 0$.

Next, we formalize the potential execution paths of a probabilistic transition system. We classify them into two categories: infinite execution paths and finite execution paths.

Definition 4 *An infinite execution path is an infinite sequence* $t_1 t_2 \ldots$ *such that*

- *for all* $i \geq 1$, $t_i \in T_{\mathcal{S}}$*,*
- $\text{source}_{\mathcal{S}}(t_1) = s_0$*, and*
- *for all* $i \geq 1$, $\text{target}_{\mathcal{S}}(t_i) = \text{source}_{\mathcal{S}}(t_{i+1})$.

The set of all infinite execution paths is denoted by $\text{Exec}^{\omega}_{\mathcal{S}}$.

A finite execution path is either a finite sequence of transitions $t_1 \ldots t_n$ *for some* $n \geq 1$ *such that*

- *for all* $1 \leq i \leq n$, $t_i \in T_{\mathcal{S}}$*,*
- $\text{source}_{\mathcal{S}}(t_1) = s_0$*,*
- $\text{target}_{\mathcal{S}}(t_n)$ *is final in* $\mathcal{S}$ *and*
- *for all* $1 \leq i < n$, $\text{target}_{\mathcal{S}}(t_i) = \text{source}_{\mathcal{S}}(t_{i+1})$*,*

or the empty sequence ϵ *if* s_0 *is final in* $\mathcal{S}$*. The set of all finite execution paths is denoted by* $\text{Exec}^{*}_{\mathcal{S}}$.

The set of all execution paths $\text{Exec}_{\mathcal{S}}$ *is defined by* $\text{Exec}_{\mathcal{S}} = \text{Exec}^{\omega}_{\mathcal{S}} \cup \text{Exec}^{*}_{\mathcal{S}}$.

For the probabilistic transition system of Example 2, the sequence $t_{01}t_{10}t_{01}t_{10}\ldots$ is an example of an infinite execution path and the sequence $t_{01}t_{12}$ is an example of a finite execution path.

We denote the set of finite prefixes of $\text{Exec}_{\mathcal{S}}$ by $\text{pref}(\text{Exec}_{\mathcal{S}})$.

Definition 5 *The function* $\text{target}_{\mathcal{S}} : \text{pref}(\text{Exec}_{\mathcal{S}}) \to S$ *is defined by*

$$\begin{array}{rcl} \text{target}_{\mathcal{S}}(\epsilon) & = & s_0 \\ \text{target}_{\mathcal{S}}(t_1 \ldots t_n) & = & \text{target}_{\mathcal{S}}(t_n) \end{array}$$

Many proofs, which are based on probability theory, rely on sets being countable. The sets $\text{pref}(\text{Exec}_{\mathcal{S}})$ and $\text{Exec}^{*}_{\mathcal{S}}$ play a key role in our theory. Both are countable.

Proposition 6 *The set* $\text{pref}(\text{Exec}_{\mathcal{S}})$ *is countable.*

Corollary 7 *The set* $\text{Exec}^{*}_{\mathcal{S}}$ *is countable.*

3. Searches and their Progress

Before formalizing the search of a model-checker and its progress measure, we first turn the set $\text{Exec}_{\mathcal{S}}$ into a probability space. The probability space we define below is similar to the ones studied by Segala [5] and Sokolova, De Vink and Woracek [6]. We start by identifying particular subsets of $\text{Exec}_{\mathcal{S}}$.

Definition 8 *Let $t_1 \dots t_n \in \text{pref}(\text{Exec}_{\mathcal{S}})$. Its basic cylinder set $B_{t_1 \dots t_n}$ is defined by*

$$B_{t_1 \dots t_n} = \{\, e \in \text{Exec}_{\mathcal{S}} \mid e \textit{ starts with } t_1 \dots t_n \,\}.$$

The set $\mathcal{B}_{\mathcal{S}}$ is defined by

$$\mathcal{B}_{\mathcal{S}} = \{\, B_{t_1 \dots t_n} \mid t_1 \dots t_n \in \text{pref}(\text{Exec}_{\mathcal{S}}) \,\} \cup \{\emptyset\}.$$

Note that $B_\epsilon = \text{Exec}_{\mathcal{S}}$. The collection $\mathcal{B}_{\mathcal{S}}$ can be shown to satisfy the following properties:

- $\emptyset \in \mathcal{B}_{\mathcal{S}}$,
- if $A \in \mathcal{B}_{\mathcal{S}}$ and $B \in \mathcal{B}_{\mathcal{S}}$, then $A \cap B \in \mathcal{B}_{\mathcal{S}}$, and
- if $A \in \mathcal{B}_{\mathcal{S}}$ and $B \in \mathcal{B}_{\mathcal{S}}$, where $B \subset A$, then there exist finitely or countably many mutually disjoint $C_n \in \mathcal{B}_{\mathcal{S}}$ such that $A \setminus B = \bigcup C_n$.

A collection satisfying the above properties is called a semi-ring in [9, page 25].[1]

Proposition 9 *The set $\mathcal{B}_{\mathcal{S}}$ is a semi-ring.*

Next, we define a function that assigns to each set in $\mathcal{B}_{\mathcal{S}}$ a real number.

Definition 10 *The function $\nu_{\mathcal{S}} : \mathcal{B}_{\mathcal{S}} \to \mathbb{R}$ is defined by*

$$\begin{array}{rcl} \nu_{\mathcal{S}}(B_{t_1 \dots t_n}) & = & \prod_{1 \le i \le n} \text{prob}_{\mathcal{S}}(t_i) \\ \nu_{\mathcal{S}}(\emptyset) & = & 0 \end{array}$$

Note that $\nu_{\mathcal{S}}(B_\epsilon) = 1$.

Proposition 11 *The function $\nu_{\mathcal{S}}$ is a measure.*

According to [9, Chapter 2], the measure $\nu_{\mathcal{S}}$ on the semi-ring $\mathcal{B}_{\mathcal{S}}$ can be extended to a measure on a σ-algebra $\Sigma_{\mathcal{S}}$ containing the semi-ring. We denote the extended measure by $\mu_{\mathcal{S}}$. We will exploit the measurable space $\langle \text{Exec}_{\mathcal{S}}, \Sigma_{\mathcal{S}}, \mu_{\mathcal{S}} \rangle$ to capture our progress measure. But first we formalize the notion of a search of a model-checker.

Definition 12 *A search of a probabilistic transition system $\mathcal{S}$ is a sequence of distinct transitions $t_1, \dots, t_n$ for some $n \ge 0$ such that $t_i \in T_{\mathcal{S}}$ for all $1 \le i \le n$.*

[1] This definition of a semi-ring is non-standard.

For the probabilistic transition system introduced in Example 2, t_{01}, t_{02}, t_{12} is a search.

If the model-checker has checked the transitions $t_1, \dots, t_n$ of $\mathcal{S}$, it of course is not aware of the remaining transitions of $\mathcal{S}$. To capture this, we formalize how $t_1, \dots, t_n$ can be extended.

Definition 13 *The probabilistic transition system $\mathcal{S}'$ extends the search $t_1, \dots, t_n$ of the probabilistic transition system $\mathcal{S}$ if*

- $\{t_1, \dots, t_n\} \subseteq T_{\mathcal{S}'}$, *and for all* $1 \le i \le n$,
- $\text{source}_{\mathcal{S}'}(t_i) = \text{source}_{\mathcal{S}}(t_i)$,
- $\text{target}_{\mathcal{S}'}(t_i) = \text{target}_{\mathcal{S}}(t_i)$,
- $\text{prob}_{\mathcal{S}'}(t_i) = \text{prob}_{\mathcal{S}}(t_i)$,
- $\text{label}_{\mathcal{S}'}(\text{source}_{\mathcal{S}'}(t_i)) = \text{label}_{\mathcal{S}}(\text{source}_{\mathcal{S}}(t_i))$, *and*
- $\text{label}_{\mathcal{S}'}(\text{target}_{\mathcal{S}'}(t_i)) = \text{label}_{\mathcal{S}}(\text{target}_{\mathcal{S}}(t_i))$.

Obviously, the probabilistic transition system $\mathcal{S}$ extends the search $t_1, \dots, t_n$ of $\mathcal{S}$.

For the definition of linear time properties and the definition of $e \models_{\mathcal{S}} \phi$, capturing that execution path e of probabilistic transition system $\mathcal{S}$ satisfies linear time property ϕ, we refer the reader to, for example, [1].

Proposition 14 *Let the probabilistic transition system $\mathcal{S}'$ extend the search $t_1, \dots, t_n$ of the probabilistic transition system $\mathcal{S}$ and let ϕ be a linear time property. Then*

$$\text{Exec}_{\mathcal{S}} \cap \{t_1, \dots, t_n\}^* = \text{Exec}_{\mathcal{S}'} \cap \{t_1, \dots, t_n\}^*$$

and

$$\mu_{\mathcal{S}}(\text{Exec}_{\mathcal{S}} \cap \{t_1, \dots, t_n\}^*) = \mu_{\mathcal{S}'}(\text{Exec}_{\mathcal{S}'} \cap \{t_1, \dots, t_n\}^*)$$

and for all $e \in \text{Exec}_{\mathcal{S}} \cap \{t_1, \dots, t_n\}^*$,

$$e \models_{\mathcal{S}} \phi \textit{ iff } e \models_{\mathcal{S}'} \phi.$$

We only define our progress measure in case no violation of the property has been found yet. The latter is formalized as follows.

Definition 15 *The search $t_1, \dots, t_n$ of the probabilistic transition system $\mathcal{S}$ has not found a violation of the linear time property ϕ if*

- *for all $e \in \{t_1, \dots, t_n\}^\omega$, if $e \in \text{Exec}^\omega_{\mathcal{S}}$ then $e \models_{\mathcal{S}} \phi$, and*
- *for all $e \in \{t_1, \dots, t_n\}^*$, if $e \in \text{pref}(\text{Exec}_{\mathcal{S}})$ then there exists a probabilistic transition system $\mathcal{S}'$ that extends $t_1, \dots, t_n$ such that $e' \models_{\mathcal{S}'} \phi$ for all $e' \in B_e$ and $B_e \in \mathcal{B}_{\mathcal{S}'}$.*

Note that the search $t_1, \ldots, t_n$ has found a violation of the linear time property ϕ if

- either we can form an infinite execution path from the transitions $t_1, \ldots, t_n$ that violates ϕ
- or we can form a prefix of an execution path from the transitions $t_1, \ldots, t_n$ which gives rise to a violation of ϕ no matter how we extend it.

For the probabilistic transition system of Example 2, the search t_{01}, t_{10} has found a violation of the property $\Box p$ and the search t_{01}, t_{12} has not found a violation of $\Box p$.

In all our proofs, we only need the following weaker condition.

Proposition 16 *If the search $t_1, \ldots, t_n$ of the probabilistic transition system $\mathcal{S}$ has not found a violation of the linear time property ϕ, then $e \models_{\mathcal{S}} \phi$ for all $e \in \text{Exec}_{\mathcal{S}} \cap (\{t_1, \ldots, t_n\}^* \cup \{t_1, \ldots, t_n\}^\omega)$.*

In the definition of our progress measure we make use of the following sets.

Definition 17 *Let the probabilistic transition system $\mathcal{S}'$ extend the search $t_1, \ldots, t_n$ of probabilistic transition system $\mathcal{S}$ and let ϕ be a linear time property. The set $\mathcal{B}^{\phi}_{\mathcal{S}'}(t_1, \ldots, t_n)$ is defined by*

$$\mathcal{B}^{\phi}_{\mathcal{S}'}(t_1, \ldots, t_n) = \bigcup \{\, B_e \in \mathcal{B}_{\mathcal{S}'} \mid e \in \{t_1, \ldots, t_n\}^* \wedge \forall e' \in B_e : e' \models_{\mathcal{S}'} \phi \,\}$$

The set $\mathcal{B}^{\phi}_{\mathcal{S}'}(t_1, \ldots, t_n)$ contains a basic cylinder set B_e if all its execution paths, that is, all extensions of e, satisfy ϕ. In that case, B_e does not contain a counterexample of ϕ. Hence, the set $\mathcal{B}^{\phi}_{\mathcal{S}'}(t_1, \ldots, t_n)$ consists of those basic cylinder sets that do not contain a counterexample of ϕ. This set is measurable, as is stated in the following proposition.

Proposition 18 *Let the probabilistic transition system $\mathcal{S}'$ extend the search $t_1, \ldots, t_n$ of probabilistic transition system $\mathcal{S}$ and let ϕ be a linear time property. Then $\mathcal{B}^{\phi}_{\mathcal{S}'}(t_1, \ldots, t_n) \in \Sigma_{\mathcal{S}'}$.*

Since the set $\mathcal{B}^{\phi}_{\mathcal{S}'}(t_1, \ldots, t_n)$ is measurable, its measure $\mu_{\mathcal{S}'}(\mathcal{B}^{\phi}_{\mathcal{S}'}(t_1, \ldots, t_n))$ is defined. The latter is a number in the interval $[0, 1]$ which represents the "size" of the basic cylinder sets that do not contain a counterexample of ϕ. This number captures the amount of progress of $t_1, \ldots, t_n$ for ϕ, *provided that* the probabilistic transition system under consideration is $\mathcal{S}'$.

Now consider the search $t_1, \ldots, t_n$ of the probabilistic transition system $\mathcal{S}$ for the linear time property ϕ. Since the model-checker has only checked the transitions $t_1, \ldots, t_n$, it only knows that it is checking some probabilistic transition system $\mathcal{S}'$ that extends $t_1, \ldots, t_n$. For each of those extensions, $\mu_{\mathcal{S}'}(\mathcal{B}^{\phi}_{\mathcal{S}'}(t_1, \ldots, t_n))$ captures the amount of progress of $t_1, \ldots, t_n$ for ϕ. Those amounts of progress can be combined as follows.

Definition 19 *Consider the search $t_1, \ldots, t_n$ of the probabilistic transition system $\mathcal{S}$. Assume that $t_1, \ldots, t_n$ has not found a violation of the linear time property ϕ. The progress of $t_1, \ldots, t_n$ for ϕ is defined by*

$$\text{prog}_{\mathcal{S}}(t_1, \ldots, t_n, \phi) = \inf \left\{\, \mu_{\mathcal{S}'}(\mathcal{B}^{\phi}_{\mathcal{S}'}(t_1, \ldots, t_n)) \mid \mathcal{S}' \textit{ extends } t_1, \ldots, t_n \textit{ of } \mathcal{S} \,\right\}.$$

In the above definition, we consider the worst case by taking the infimum. Hence, no matter how the search $t_1, \ldots, t_n$ is extended, it has made at least the given amount of progress. We could also consider the best case by replacing inf with sup in the above definition. However, in that case we can show that the progress is always one.

Consider the probabilistic transition system of Example 2 and the linear time property $\bigcirc p$. At the beginning, when the model-checker has not explored any transitions, the progress is zero. After checking the transition t_{01}, the progress increases to 0.6. Checking the transition t_{12} does not increase the progress, and the progress of the search t_{01}, t_{12}, t_{02} is one. Now consider the linear time property $\Box p$. Again, the empty search has not made any progress. Checking the transition t_{01} does not increase the progress. After also checking the transition t_{12}, the progress increases to $0.6 \times 0.3 = 0.18$. Furthermore, the progress of the search t_{01}, t_{12}, t_{02} is $0.18 + 0.4 = 0.58$.

Our progress measure provides a lowerbound of the probability that the linear time property is satisfied.

Proposition 20 *If the search $t_1, \ldots, t_n$ of the probabilistic transition system $\mathcal{S}$ has not found a violation of the linear time property ϕ, then*

$$\text{prog}_{\mathcal{S}}(t_1, \ldots, t_n, \phi) \leq \mu_{\mathcal{S}}(\{\, e \in \text{Exec}_{\mathcal{S}} \mid e \models_{\mathcal{S}} \phi \,\}).$$

4. Progress for Invariants

In case the property under verification is an invariant, we can characterize our progress measure in terms of the set of execution paths that solely consist of transitions that have been explored by the model-checker. This set is measurable, as is stated in the following proposition.

Proposition 21 *Let $t_1, \ldots t_n$ be a search of the probabilistic transition system $\mathcal{S}$. Then* $\text{Exec}_{\mathcal{S}} \cap (\{t_1, \ldots, t_n\}^* \cup \{t_1, \ldots, t_n\}^\omega) \in \Sigma_{\mathcal{S}}$.

Since the above set is measurable, its measure is defined. As shown in the following theorem, it captures the progress of the search for invariants.

Theorem 22 *If the search $t_1, \ldots, t_n$ of the probabilistic transition system $\mathcal{S}$ has not found a violation of the invariant ϕ, then*

$$\begin{aligned}&\text{prog}_{\mathcal{S}}(t_1, \ldots, t_n, \phi)\\ &\quad = \quad \mu_{\mathcal{S}}(\text{Exec}_{\mathcal{S}} \cap (\{t_1, \ldots, t_n\}^* \cup \{t_1, \ldots, t_n\}^\omega)).\end{aligned}$$

Proof Due to lack of space, we only present a sketch of the proof. We prove the above equality by proving two inequalities.

To prove that $\text{prog}_{\mathcal{S}}(t_1, \ldots, t_n, \phi)$ is smaller than or equal to $\mu_{\mathcal{S}}(\text{Exec}_{\mathcal{S}} \cap (\{t_1, \ldots, t_n\}^* \cup \{t_1, \ldots, t_n\}^\omega))$, we construct a probabilistic transition system $\mathcal{S}'$ that extends $t_1, \ldots, t_n$ of $\mathcal{S}$ such that $\mu_{\mathcal{S}'}(\mathcal{B}^{\phi}_{\mathcal{S}'}(t_1, \ldots, t_n))$ equals $\mu_{\mathcal{S}}(\text{Exec}_{\mathcal{S}} \cap (\{t_1, \ldots, t_n\}^* \cup \{t_1, \ldots, t_n\}^\omega))$.

The proof that $\mu_{\mathcal{S}}(\text{Exec}_{\mathcal{S}} \cap (\{t_1, \ldots, t_n\}^* \cup \{t_1, \ldots, t_n\}^\omega))$ is smaller than or equal to $\text{prog}_{\mathcal{S}}(t_1, \ldots, t_n, \phi)$ consists of two major steps. Let $\mathcal{S}'$ extend $t_1, \ldots, t_n$ of $\mathcal{S}$. Assume that $t_1, \ldots, t_n$ has not found a violation of ϕ. First, we show that $\text{Exec}_{\mathcal{S}} \cap \{t_1, \ldots, t_n\}^*$ is a subset of $\mathcal{B}^{\phi}_{\mathcal{S}'}(t_1, \ldots, t_n, \phi)$. Since it is in general not the case that $\text{Exec}_{\mathcal{S}} \cap \{t_1, \ldots, t_n\}^\omega$ is a subset of $\mathcal{B}^{\phi}_{\mathcal{S}'}(t_1, \ldots, t_n, \phi)$, we show that the set $\mathcal{B}^{\phi}_{\mathcal{S}'}(t_1, \ldots, t_n, \phi) \setminus (\text{Exec}_{\mathcal{S}} \cap \{t_1, \ldots, t_n\}^\omega)$ has measure zero. The latter proof is quite involved. We define δ as $\min\{\, \text{prob}_{\mathcal{S}'}(t) \mid \text{source}_{\mathcal{S}'}(t) \in \{\, \text{target}_{\mathcal{S}'}(t_i) \mid 1 \leq i \leq n \,\} \cup \{s_0\} \,\} \vee t \in \{t_1, \ldots, t_n\} \,\}$. For $e \in \text{pref}(\text{Exec}_{\mathcal{S}'})$ and $n \in \mathbb{N}$, we use $e[n]$ to denote e truncated at length n. One of the key ingredients of this part of the proof is the proof that for all $k \in \mathbb{N}$, $\mu_{\mathcal{S}'}(\bigcup\{\, B_{e[k(n+1)]} \in \mathcal{B}_{\mathcal{S}'} \mid e \in \mathcal{B}^{\phi}_{\mathcal{S}'}(t_1, \ldots, t_n) \setminus (\text{Exec}_{\mathcal{S}} \cap \{t_1, \ldots, t_n\}^\omega) \,\}) \leq (1 - \delta^{n+1})^k$. □

Note that the above theorem does not hold for all linear time properties. For example, consider probabilistic transition system of Example 2, the search t_{01} and the property $\bigcirc p$. As we have already seen, the progress of t_{01} for $\bigcirc p$ is 0.6. However, the set $\text{Exec}_{\mathcal{S}} \cap (\{t_{01}\}^* \cup \{t_{01}\}^\omega)$ is empty and, hence, its measure is zero.

5. Progress Computation for Invariants

In this section, we fix a probabilistic transition system $\mathcal{S}$, which we call the complete system, a search $t_1, \ldots, t_n$ of this system and an invariant ϕ. Next, we construct a probabilistic transition system $\mathcal{S}'$, which we call the searched system. This searched system is usually considerably smaller than the complete system. We will exploit the searched system to compute the progress of $t_1, \ldots, t_n$ for ϕ.

Definition 23 *The set $S_{\mathcal{S}'}$ is defined by*

$$S_{\mathcal{S}'} = \bigcup_{1 \leq i \leq n} \{\text{source}_{\mathcal{S}}(t_i), \text{target}_{\mathcal{S}}(t_i)\} \cup \{s_0, s_\perp\}.$$

A state s is partial if s is not final in $\mathcal{S}$ and

$$\text{out}_{\mathcal{S}}(s) = \sum\{\, \text{prob}_{\mathcal{S}}(t_i) \mid 1 \leq i \leq n \wedge \text{source}_{\mathcal{S}}(t_i) = s \,\} \in [0, 1).$$

The set $T_{\mathcal{S}'}$ is defined by

$$T_{\mathcal{S}'} = \{t_1, \ldots, t_n\} \cup \{\, t_s \mid s \in S_{\mathcal{S}'} \text{ is final in } \mathcal{S} \text{ or partial} \,\} \cup \{t_\perp\}.$$

The set $AP_{\mathcal{S}'}$ is defined by

$$AP_{\mathcal{S}'} = AP_{\mathcal{S}} \cup \{\perp\}.$$

The function $\text{source}_{\mathcal{S}'} : T_{\mathcal{S}'} \to S_{\mathcal{S}'}$ *is defined by*

$$\text{source}_{\mathcal{S}'}(t) = \begin{cases} s_\perp & \text{if } t = t_\perp \\ s & \text{if } t = t_s \\ \text{source}_{\mathcal{S}}(t) & \text{if } t \in \{t_1, \ldots, t_n\} \end{cases}$$

The function $\text{target}_{\mathcal{S}'} : T_{\mathcal{S}'} \to S_{\mathcal{S}'}$ *is defined by*

$$\text{target}_{\mathcal{S}'}(t) = \begin{cases} s_\perp & \text{if } t = t_\perp \\ s & \text{if } t = t_s \text{ and } s \text{ is final} \\ s_\perp & \text{if } t = t_s \text{ and } s \text{ is partial} \\ \text{target}_{\mathcal{S}}(t) & \text{if } t \in \{t_1, \ldots t_n\} \end{cases}$$

The function $\text{prob}_{\mathcal{S}'} : T_{\mathcal{S}'} \to (0, 1]$ *is defined by*

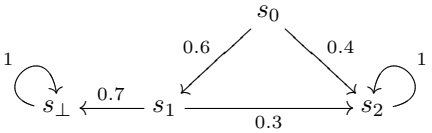
$$\text{prob}_{\mathcal{S}'}(t) = \begin{cases} 1 & \text{if } t = t_\perp \\ 1 & \text{if } t = t_s \text{ and } s \text{ is final} \\ 1 - \text{out}_{\mathcal{S}}(s) & \text{if } t = t_s \text{ and } s \text{ is partial} \\ \text{prob}_{\mathcal{S}}(t) & \text{if } t \in \{t_1, \ldots t_n\} \end{cases}$$

The function $\text{label}_{\mathcal{S}'} : S_{\mathcal{S}'} \to 2^{AP_{\mathcal{S}'}}$ *is defined by*

$$\text{label}_{\mathcal{S}'}(s) = \begin{cases} \{\perp\} & \text{if } s = s_\perp \\ \text{label}_{\mathcal{S}}(s) & \text{otherwise} \end{cases}$$

Proposition 24 *$\mathcal{S}'$ is a probabilistic transition system.*

For the complete system of Example 2 and the search t_{01}, t_{02}, t_{12}, the corresponding searched system can be depicted as



Note that the probabilistic transition system $\mathcal{S}'$ only gives rise to infinite execution paths. Let us denote the set of execution paths of this system by $\text{Exec}_{\mathcal{S}'}$. As we turned the set $\text{Exec}_{\mathcal{S}}$ into a measurable space, we can also turn the set $\text{Exec}_{\mathcal{S}'}$ into a measurable space $\langle \text{Exec}_{\mathcal{S}'}, \Sigma_{\mathcal{S}'}, \mu_{\mathcal{S}'} \rangle$.

To prove that we can indeed use the searched system to compute the progress of the search in the complete system, we relate the complete and the searched system. We relate the systems by linking the execution paths of the two systems.

Definition 25 *The function* $\eta : \text{Exec}_{\mathcal{S}} \to \text{Exec}_{\mathcal{S}'}$ *is defined by*

$$\eta(e) = \begin{cases} e(t_{\text{target}(e)})^\omega & \text{if } e \in \{t_1, \ldots, t_n\}^* \\ e & \text{if } e \in \{t_1, \ldots, t_n\}^\omega \\ e' t_{\text{target}(e')}(t_\perp)^\omega & \text{if } e = e' t e'' \text{ and } e' \in \{t_1, \ldots, t_n\}^* \\ & \text{and } t \notin \{t_1, \ldots, t_n\} \end{cases}$$

Next, we characterize those execution paths of the complete system, which solely consist of transitions of the search, in terms of execution paths in the searched system.

Proposition 26 *The sets* $\text{Exec}_{\mathcal{S}} \cap (\{t_1, \ldots, t_n\}^* \cup \{t_1, \ldots, t_n\}^\omega)$ *and* $\{\, e' \in \text{Exec}_{\mathcal{S}'} \mid e' \text{ does not contain } t_\perp \,\}$ *are isomorphic via* η*.*

As we show next, to compute the progress of the search in the complete system, it suffices to compute the measure of the set of those execution paths of the searched system which do not contain the transition $t_\perp$.

Theorem 27 $\text{prog}_{\mathcal{S}}(t_1, \ldots, t_n, \phi) = \mu_{\mathcal{S}'}(\{\, e' \in \text{Exec}_{\mathcal{S}'} \mid e' \text{ does not contain } t_\perp \,\})$.

The set of those execution paths of the searched system which contain the transition $t_\perp$ can be captured as the set of execution paths satisfying the linear time property true $\mathcal{U}\ \perp$.

Proposition 28 *For all* $e' \in \text{Exec}_{\mathcal{S}'}$,

$$e' \text{ contains } t_\perp \text{ iff } e' \models_{\mathcal{S}'} \text{true}\ \mathcal{U}\ \perp .$$

From Theorem 27 and Proposition 28 we can conclude the following.

Corollary 29 $\text{prog}_{\mathcal{S}}(t_1, \ldots, t_n, \phi) = 1 - \mu_{\mathcal{S}'}(\{\, e' \in \text{Exec}_{\mathcal{S}'} \mid e' \models_{\mathcal{S}'} \text{true}\ \mathcal{U}\ \perp \,\})$.

Hence, to compute the progress of the search in the complete system, it suffices to compute the measure of the set of those execution paths in the searched system that satisfy the property true $\mathcal{U}\ \perp$. An algorithm to compute the latter has been presented by Courcoubetis and Yannakakis [2, Lemma 3.1.1.1].

6. Maintaining the Searched System

As we have seen in the previous section, to compute the progress measure of a search of a complete system for an invariant, we construct the corresponding searched system (and compute the measure of the set of those execution paths in the searched system that satisfy a particular property). As the search of the model-checker continues, we would like to keep track of the progress. For a given complete system $\mathcal{S}$, this can be captured by the following diagram.

$$\begin{array}{ccc} t_1, \ldots, t_n & \longrightarrow & t_1, \ldots, t_n, t_{n+1} \\ \downarrow & & \downarrow \\ \mathcal{S}_n & & \mathcal{S}_{n+1} \end{array}$$

Rather than constructing the searched system from scratch after a new transition has been explored by the model-checker, we show that we can construct the new searched system $\mathcal{S}_{n+1}$ from the old searched system $\mathcal{S}_n$ and the transition t_{n+1} in constant time.

But first we characterize the searched system corresponding to the empty search.

Proposition 30 *Let* $\mathcal{S}_0$ *be the searched system corresponding to the empty search. Then* $S_0 = \{s_0, s_\perp\}$, $T_0 = \{t_{s_0}, t_\perp\}$, $AP_0 = AP_{\mathcal{S}} \cup \{\perp\}$, $\text{source}_0(t_{s_0}) = s_0$, $\text{source}_0(t_\perp) = s_\perp$, $\text{target}_0(t_\perp) = s_\perp$, $\text{prob}_0(t_{s_0}) = 1$, $\text{prob}_0(t_\perp) = 1$, $\text{label}_0(s_0) = \text{label}_{\mathcal{S}}(s_0)$ *and* $\text{label}_0(s_\perp) = \{\perp\}$. *If* s_0 *is final then* $\text{target}(t_{s_0}) = s_0$. *Otherwise,* $\text{target}(t_{s_0}) = s_\perp$.

Theorem 31 *Let* $\mathcal{S}_n$ *and* $\mathcal{S}_{n+1}$ *be the searched systems related to the searches* $t_1, \ldots, t_n$ *and* $t_1, \ldots, t_n, t_{n+1}$*, respectively. Let* $s_s = \text{source}(t_{n+1})$ *and* $s_t = \text{target}(t_{n+1})$*. Then*

$$S_{n+1} = S_n \cup \{s_s, s_t\}.$$

and

$$T_{n+1} = \begin{cases} T_n \cup \{t_{n+1}\} \cup \{t_{s_s}\} \cup \{t_{s_t}\} & \text{if } s_s \notin S_n \wedge s_t \notin S_n \wedge \\ & \text{prob}_{\mathcal{S}}(t_{n+1}) < 1 \\ T_n \cup \{t_{n+1}\} \cup \{t_{s_t}\} & \text{if } s_s \in S_n \wedge s_t \notin S_n \wedge \\ & \text{prob}_n(t_{s_s}) > \text{prob}_{\mathcal{S}}(t_{n+1}) \\ T_n \cup \{t_{n+1}\} \cup \{t_{s_t}\} & \text{if } s_s \notin S_n \wedge s_t \notin S_n \wedge \\ & \text{prob}_{\mathcal{S}}(t_{n+1}) = 1 \\ T_n \cup \{t_{n+1}\} \cup \{t_{s_t}\} \setminus \{t_{s_s}\} & \text{if } s_s \in S_n \wedge s_t \notin S_n \wedge \\ & \text{prob}_n(t_{s_s}) = \text{prob}_{\mathcal{S}}(t_{n+1}) \\ T_n \cup \{t_{n+1}\} \cup \{t_{s_s}\} & \text{if } s_s \notin S_n \wedge s_t \in S_n \wedge \\ & \text{prob}_{\mathcal{S}}(t_{n+1}) < 1 \\ T_n \cup \{t_{n+1}\} & \text{if } s_s \in S_n \wedge s_t \in S_n \wedge \\ & \text{prob}_n(t_{s_s}) > \text{prob}_{\mathcal{S}}(t_{n+1}) \\ T_n \cup \{t_{n+1}\} & \text{if } s_s \notin S_n \wedge s_t \in S_n \wedge \\ & \text{prob}_{\mathcal{S}}(t_{n+1}) = 1 \\ T_n \cup \{t_{n+1}\} \setminus \{t_{s_s}\} & \text{if } s_s \in S_n \wedge s_t \in S_n \wedge \\ & \text{prob}_n(t_{s_s}) = \text{prob}_{\mathcal{S}}(t_{n+1}) \end{cases}$$

and

$$AP_{n+1} = AP_n$$

and the function $\text{source}_{n+1} : T_{n+1} \to S_{n+1}$ *satisfies*

$$\text{source}_{n+1}(t) = \begin{cases} s_s & \text{if } t = t_{n+1} \\ s_s & \text{if } t = t_{s_s} \\ s_t & \text{if } t = t_{s_t} \\ \text{source}_n(t) & \text{otherwise} \end{cases}$$

and the function $\text{target}_{n+1} : T_{n+1} \to S_{n+1}$ *satisfies*

$$\text{target}_{n+1}(t) = \begin{cases} s_t & \textit{if } t = t_{n+1} \\ s_\perp & \textit{if } t = t_{s_s} \\ s_t & \textit{if } t = t_{s_t} \wedge s_t \textit{ is final} \\ s_\perp & \textit{if } t = t_{s_t} \wedge s_t \textit{ is partial} \\ \text{target}_n(t) & \textit{otherwise} \end{cases}$$

and the function $\text{prob}_{n+1} : T_{n+1} \to (0, 1]$ *satisfies*

$$\text{prob}_{n+1}(t) = \begin{cases} \text{prob}_{\mathcal{S}}(t_{n+1}) \\ \quad \textit{if } t = t_{n+1} \\ \text{prob}_n(t_{s_s}) - \text{prob}_{\mathcal{S}}(t_{n+1}) \\ \quad \textit{if } t = t_{s_s} \wedge s_s \in S_n \wedge \\ \quad \text{prob}_n(t_{s_s}) > \text{prob}_{\mathcal{S}}(t_{n+1}) \\ 1 - \text{prob}_{\mathcal{S}}(t_{n+1}) \\ \quad \textit{if } t = t_{s_s} \wedge s_s \notin S_n \wedge \\ \quad \text{prob}_{\mathcal{S}}(t_{n+1}) < 1 \\ 1 \\ \quad \textit{if } t = t_{s_t} \wedge s_t \notin S_n \\ \text{prob}_n(t) \\ \quad \textit{otherwise} \end{cases}$$

and the function $\text{label}_{n+1} : S_{n+1} \to 2^{AP_{n+1}}$ *satisfies*

$$\text{label}_{n+1}(s) = \begin{cases} \text{label}_n(s) & \textit{if } s \in S_n \\ \text{label}_{\mathcal{S}}(s) & \textit{otherwise} \end{cases}$$

The above results provide the correctness proof of the algorithm below.

Initialize()
$S \leftarrow$ empty set
$T \leftarrow$ empty set
insert(S, s_0); insert$(S, s_\perp)$
source$(t_\perp) \leftarrow s_\perp$; target$(t_\perp) \leftarrow s_\perp$; prob$(t_\perp) \leftarrow 1$
insert$(T, t_\perp)$
if s_0 is final
  target$(t_{s_0}) \leftarrow s_0$
else
  target$(t_{s_0}) \leftarrow s_\perp$
source$(t_{s_0}) \leftarrow s_0$; prob$(t_{s_0}) \leftarrow 1$
insert(T, t_{s_0})

Precondition: $t \notin T$
Add(t)
$s_s \leftarrow$ source(t); $s_t \leftarrow$ target(t)
insert(T, t)
if $s_s \notin S$
  insert(S, s_s)
  if $\text{prob}_{\mathcal{S}}(t) < 1$
    source$(t_{s_s}) \leftarrow s_s$; target$(t_{s_s}) \leftarrow s_\perp$
    prob$(t_{s_s}) \leftarrow 1 - \text{prob}_{\mathcal{S}}(t)$
    insert(T, t_{s_s})
else
  if $\text{prob}_{\mathcal{S}}(t) < \text{prob}(t_{s_s})$
    prob$(t_{s_s}) \leftarrow$ prob$(t_{s_s}) - \text{prob}_{\mathcal{S}}(t)$
  else
    remove(T, t_{s_s})
if $s_t \notin S$
  insert(S, s_t)
if s_t is final
  target$(t_{s_t}) \leftarrow s_t$
else
  target$(t_{s_t}) \leftarrow s_\perp$
source$(t_{s_t}) \leftarrow s_t$; prob$(t_{s_t}) \leftarrow 1$
insert(T, t_{s_t})

Proposition 32 *The worst-case running time of Initialize is constant. The amortized expected running time per Add is constant.*

7. Search Strategies

Explicit-state model-checkers like JPF exploit search strategies such as depth-first search (DFS) and breadth-first search (BFS) to visit the transitions of the system under verification. Next, we present three new search strategies. In contrast to DFS and BFS, our search strategies take the probabilities of the transitions into account.

7.1. Probability-First Search

To let transitions with the highest probability be searched first, our probability-first search (PFS) strategy sorts the enabled transitions by their probability. For the probabilistic transition system of Example 2, PFS visits the transitions in the following order: t_{01}, t_{10}, t_{02}, t_{12}.

To implement PFS, we use a priority queue q. Each key is a real, which represents a probability. The keys are ordered as follows:

$$p_1 \preceq p_2 \text{ if } p_1 \geq p_2.$$

The elements of the priority queue q are transitions. An algorithm for PFS can be found below.

Precondition: no state is marked visited
$q \leftarrow$ empty priority queue
for all transitions t from s_0
  insert$(q, \langle \text{prob}(t), t \rangle)$
mark s_0 visited
while q is nonempty
  $\langle p, t \rangle \leftarrow$ deleteMin(q) ▷ t is visited
  if target(t) is not visited
    for all transitions t' from target(t)
      insert$(q, \langle \text{prob}(t') \times p, t' \rangle)$
  mark target(t) visited

7.2. Breadth-First Probability-Second Search

Our breadth-first probability-second search (BFPSS) is an enhancement of BFS in which transitions at the same level are sorted by their probability. For the probabilistic transition system of Example 2, BFPSS visits the transitions in the following order: t_{01}, t_{02}, t_{10}, t_{12}.

To implement BFPSS, we also use a priority queue q. This time, each key is a pair consisting of an integer, which represents a level, and a real, which represents a probability. The keys are ordered as follows:

$$[\ell_1, p_1] \preceq [\ell_2, p_2] \text{ if } \ell_1 < \ell_2 \vee (\ell_1 = \ell_2 \wedge p_1 \geq p_2).$$

The elements of the priority queue q are transitions. An algorithm for BFPSS can be found below.

Precondition: no state is marked visited
$q \leftarrow$ empty priority queue
for all transitions t from s_0
  insert($q, \langle[1, \text{prob}(t)], t\rangle$)
mark s_0 visited
while q is nonempty
  $\langle[\ell, -], t\rangle \leftarrow$ deleteMin(q) ▷ t is visited
  if target(t) is not visited
    for all transitions t' from target(t)
      insert($q, \langle[\ell + 1, \text{prob}(t')], t'\rangle$)
    mark target(t) visited

7.3. Randomized Search

Our randomized search (RS) randomly selects an enabled transition. The chance that a transition is selected is proportional to its probability.

To implement RS, we use a set s. The elements of the set are pairs, each consisting of a real, representing a probability, and a transition. The operation select(s) removes an element $\langle p, t\rangle$ from the set s and returns the element. The probability that the element $\langle p, t\rangle$ is selected is $\frac{p}{\sum_{\langle p', t'\rangle \in s} p'}$. An algorithm for RS is given below.

Precondition: no state is marked visited
$s \leftarrow$ empty set
for all transitions t from s_0
  insert($s, \langle\text{prob}(t), t\rangle$)
mark s_0 visited
while s is nonempty
  $\langle p, t\rangle \leftarrow$ select(s) ▷ t is visited
  if target(t) is not visited
    for all transitions t' from target(t)
      insert($s, \langle\text{prob}(t') \times p, t'\rangle$)
    mark target(t) visited

7.4. Properties of Search Strategies

Like DFS and BFS, also PFS, BFPSS and RS visit each state at most once.

Proposition 33 *In PFS, BFPSS and RS, each transition is visited at most once. Let T_0 be the set of those transitions that can be reached from s_0. If T_0 is finite then PFS, BFPSS and RS visit all transitions in T_0.*

Since PFS, BFPSS and RS take the probabilities into account, these search strategies are not as efficient as DFS and BFS.

Proposition 34 *If*

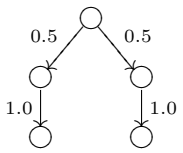
- *a state can be marked visited in $O(1)$,*
- *a state can be checked to be marked in $O(1)$,*
- *the outgoing transitions of a state s can be enumerated in $O(n)$, where n is the number of outgoing transitions of s,*

then the worst-case running time of PFS, BFPSS and RS is $O(|T_0| \log |T_0|)$.

7.5. Comparison

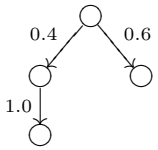
Our progress measure allows us to compare the amount of progress these different search strategies make. Next we will provide examples that show that DFS, BFS, PFS and BFPSS are incomparable. That is, for each pair of search strategies, we will construct a complete system such that the one strategy makes faster progress than the other. Obviously, RS is incomparable with the other four search strategies.

The first example shows that DFS can make faster progress than BFS.



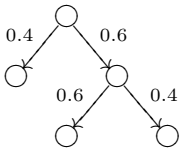
DFS	BFS
0	0
0.5	0
0.5	0.5
1.0	1.0

The second example shows the opposite, that is, BFS can make faster progress than DFS.



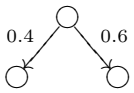
BFS	DFS
0	0
0.6	0.4
1.0	1.0

Our third example show that both DFS and BFS can make progress faster than PFS and BFPSS.



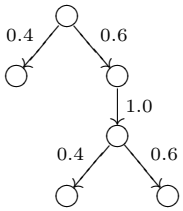
DFS	BFS	PFS	BFPSS
0.4	0.4	0	0
0.4	0.4	0.4	0.4
0.76	0.76	0.76	0.76
1.0	1.0	1.0	1.0

The fourth example shows the opposite, that is, both PFS and BFPSS can make faster progress than DFS and BFS.



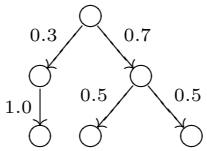
PFS	BFPSS	DFS	BFS
0.6	0.6	0.4	0.4
1.0	1.0	1.0	1.0

Next, we present an example of a system for which BFPSS makes faster progress than PFS.



BFPSS	PFS
0	0
0.4	0
0.4	0.4
0.76	0.76
1.0	1.0

Finally, we show that PFS can make progress faster than BFPSS.



PFS	BFPSS
0	0
0.35	0
0.7	0.3
0.7	0.65
1.0	1.0

8. Conclusion

To measure the amount of progress made by an explicit-state probabilistic model checker when checking a linear time property, we introduced the notion of a progress measure. Although our notion is based on a probability space that has been used by others to study probabilistic systems, as far as we know this notion is new. We have also shown how to compute the progress measure for invariants. Furthermore, we introduced several new search strategies and compared them with depth-first search and breadth-first search. Although these search strategies are not the main focus of the paper and most likely have already been introduced in different contexts, we believe that they show that our theoretical framework allows us to study new search strategies and compare them.

We have implemented our theoretical framework within JPF. The details can be found in [10]. We implemented a number of randomized algorithms in Java and used our extension of JPF to model-check them while keeping track of JPF's progress at the same time. Also these examples showed that the search strategies are incomparable, although our new search strategies made progress faster than depth-first search and breadth-first search for most examples.

Our framework only considers probabilistic choices. As a consequence, it is only applicable to implementations of sequential randomized algorithms. One direction of future work is to extend our framework so that it can handle concurrent randomized algorithms as well. In that case, we also have to deal with nondeterministic choices and, hence, we may have to consider schedulers.

Here, we focused on explicit-state probabilistic model-checkers. However, we believe that our developed theory can be adapted to symbolic probabilistic model-checkers. This is another direction for future research.

References

[1] C. Baier and J.-P. Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[2] C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, July 1995.

[3] C. Eisner and D. Peled. Comparing symbolic and explicit model checking of a software system. In *Proceedings of the 9th International SPIN Workshop on Model Checking of Software*, volume 2318 of *Lecture Notes in Computer Science*, pages 230–239. Springer-Verlag, Apr. 2002.

[4] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with PRISM: A hybrid approach. *International Journal on Software Tools for Technology Transfer*, 6(2):128–142, Sept. 2004.

[5] R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, Laboratory for Computer Science, Massachusetts Institute of Technology, June 1995.

[6] A. Sokolova, E. de Vink, and H. Woracek. A companion to coalgebraic weak bisimulation for action-type systems. Technical Report CSR-07-12, Eindhoven University of Technology, 2007.

[7] M. Y. Vardi. Automatic verification of probabilistic finite-state programs. In *Proceedings of the 26th IEEE Symposium on Foundations of Computer Science*, pages 327–338. IEEE, Oct. 1985.

[8] W. Visser, K. Havelund, G. Brat, S. Park, and F. Lerda. Model checking programs. *Automated Software Engineering*, 10(2):203–232, Apr. 2003.

[9] A. C. Zaanen. *Integration*. North-Holland, 1967.

[10] X. Zhang and F. van Breugel. Probabilistic model checking with Java PathFinder. Submitted to the *International Conference on Computer Aided Verification*. Available at `www.cse.yorku.ca/~franck/research/drafts/`, Jan. 2010.