Model Checking Failure-Prone Open Systems Using Probabilistic Automata

Yue Ben(✉) and A. Prasad Sistla

University of IIlinois, Chicago, USA
benyue06@gmail.com

Abstract. We consider finite-state Hierarchical Probabilistic Automata (HPA) to model failure-prone open systems. In an HPA, its states are stratified into a fixed number of levels. A k-HPA is an HPA with $k+1$ levels and it can be used to model open systems where up to k failures can occur. In this paper, we present a new forward algorithm that checks universality of a 1-HPA. This algorithm runs much faster than an earlier backward algorithm. We present the implementation and experimental results for verifying abstracted failure-prone web applications. We also prove a theoretical result showing that the problem of checking emptiness and universality for 2-HPA is undecidable answering an open question.

Keywords: Model checking · Verification tool · Failure-prone open systems · Emptiness and decision problems · Hierarchical Probabilistic Automata (HPA)

1 Introduction

Open concurrent systems, such as *web applications*, are usually deployed on multiple processors/servers. These systems take user inputs (thus called *open* systems), and service their requests. Failures may occur in such systems in the form of server/processor crashes. Once a processor fails, the other functioning processors take over the remaining tasks from the failed processor and continue their execution. In this paper, we model the failures probabilistically and assume that the probability of failure of a processor may depend on its current state as well as the input (inputs are usually submitted as a *form*) being processed. In such systems, it is critically important to verify that the system satisfies a correctness property with a minimum probability on all input sequences.

We employ *Open Probabilistic Transition Systems* (OPTSes) to model such failure-prone open concurrent systems. In general, an OPTS takes inputs from the environment and makes transitions to different states according to a probability distribution which may depend on the input as well as the current state. We consider systems abstracted as finite-state OPTSes. We consider the correctness property specified by a deterministic automaton on the computations of the system. The problem of checking that a system, specified by an OPTS $\mathbb{T}$, satisfies a specification given by an automaton $\mathcal{A}$ with a probability greater

B. Finkbeiner et al. (Eds.): ATVA 2015, LNCS 9364, pp. 148–165, 2015.
DOI: 10.1007/978-3-319-24953-7_11

than a given value x, reduces to the problem of checking if the Probabilistic Automaton (PA) $\mathbb{T} \times \mathcal{A}$, accepts all input sequences with probability greater than x. Therefore, checking correctness reduces to checking universality of the language accepted by a PA [1,8,9,11].

When we use the probabilistic behavior of the OPTS only for modeling processor failures, as described above, we get hierarchical OPTSes. A hierarchical OPTS is one in which its states are stratified into $k+1$ levels, say $0, 1, \ldots, k$, (for some $k \geq 0$) such that, from any state, on an input, at most one transition goes to a state at the same level and all other transitions go to higher levels, and the initial state is at level 0. Such an OPTS is called a k-OPTS. Intuitively, it captures the situation when up to k processors can fail probabilistically in some arbitrary sequence. The level of a state denotes the number of failures that occurred before reaching the state. Now checking correctness of a k-OPTS reduces to checking universality of the language accepted by an HPA (Hierarchical Probabilistic Automata). HPA introduced in [4] are a subclass of Probabilistic automata (PA). In that work it has been shown that the problem of checking the emptiness of $\mathsf{L}_{>x}(\mathcal{A})$ ($\mathsf{L}_{\geq x}(\mathcal{A})$) of an HPA $\mathcal{A}$ for a rational $x \in (0,1)$ is undecidable. (Here $\mathsf{L}_{>x}(\mathcal{A})$ is the set of all strings accepted by $\mathcal{A}$ from the initial state with probability $> x$).

It has been shown in [6], that checking emptiness and universality problems for 1-HPA is decidable in exponential time by presenting an algorithm, called *backward* algorithm. In this paper, we give an alternative algorithm, called *forward* algorithm, for checking emptiness and universality of 1-HPA. This algorithm employs the given threshold value x critically in its logic. It is shown to have a better complexity (although it is also exponential time in the worst case), and runs much faster in practice as shown by our experimental results.

We also show that the emptiness problem for 2-HPA is undecidable. The undecidability result for HPA given in [4], proves the result for 6-HPA. Our result closes the gap between the levels of HPA for which decidability and undecidability results are proved. Our result employs a new elegant construction of a 2-HPA, to check equality of two counter values of a counter machine (more specifically, to recognize the context free language $\{a^n b^n \mid n \geq 0\}$), replacing the Freivalds' construction employed in [4].

We have implemented a tool that takes a PA, checks if it is an HPA. If it is an HPA it computes the minimum k such that it is a k-HPA and classifies its states into $k+1$ levels; this algorithm runs in time $O(k \cdot m)$ where m is the size of the PA. If $k = 1$, it also takes a threshold value x and checks for non-emptiness of $\mathsf{L}_{>x}(\mathcal{A})$ using the forward and backward algorithm. We employed it to check the correctness of two abstracted web applications under a single failure. The experimental results confirm that the forward algorithm is orders of magnitude faster than the backward algorithm for these examples.

In summary, the main contributions of the paper are as follows.

– A new faster algorithm for checking non-emptiness of the language accepted by a 1-HPA with respect to a given rational threshold $x \in (0,1)$.

- Undecidability result for checking non-emptiness for 2-HPA, thus closing the gap in the known decidability results.
- A tool that takes a PA and checks if it is a 1-HPA and then checks for the non-emptiness of its language with respect to a given threshold value.
- A tool that takes the following as input: the specification of a single session of a web server as deterministic automaton, and a failure specification, and a correctness specification and checks for its correctness, under the assumption of at most one failure, of a system running two servers.

There has been much work done on verifying finite-state concurrent probabilistic programs [3,10]. All of them assume closed systems and model the system as a Markov Decision Process (MDP) and verify its correctness in an appropriate logic. To the best of our knowledge, ours is the first work that checks for correctness of a class of open probabilistic systems using probabilistic automata. The decidability problem for different classes of PA under different threshold conditions has been extensively studied [2,5,7]. Our algorithm for non-emptiness of 1-HPA is faster in practice than the one proposed in [6]. Our undecidability result for 2-HPA improves the result of [4] and closes the gap between known decidability/undecidability results for HPA.

The rest of the paper is organized as follows. Section 2 has basic definitions and notation used in the paper. In Sect. 3 we present our enhanced backward algorithm and the forward algorithm for deciding the non-emptiness of 1-HPA. In Sect. 4 we present the undecidability result for 2-HPA. Section 5 presents the model for OPTS. Section 6 presents implementation and application of the algorithms. Finally, Sect. 7 contains concluding remarks.

2 Preliminaries

We assume reader is familiar with finite-state automata, regular languages, and ω-regular languages. The closed unit interval is denoted by $[0,1]$ and the open unit interval by $(0,1)$. The power-set of a set X will be denoted by 2^X.

Sequences. Given a finite set S, $|S|$ denotes the cardinality of S. Given a finite sequence $\kappa = s_0 s_1 \ldots$ over S, $|\kappa|$ will denote the length of the sequence, $\kappa[i]$ will denote the element s_i of the sequence and $Pref(\kappa)$ denotes the set of prefixes of κ. As usual S^* will denote the set of all finite sequences/strings/words over S, S^+ will denote the set of all finite non-empty sequences/strings/words over S, and S^ω will denote the set of all infinite sequences/strings/words over S. A *language* over S is a subset of S^*.

Probabilistic Automata (PA). Informally, a PA is like a finite-state deterministic automaton except that the transition function from a state on a given input is described as a probability distribution which determines the probability of the next state.

Definition 1. A *finite-state probabilistic automaton (*PFA*)* over a finite alphabet Σ is a tuple $\mathcal{A} = (Q, q_0, \delta, \mathsf{Acc})$ where Q is a finite set of *states*, $q_0 \in Q$ is the

initial state, $\delta : Q \times \Sigma \times Q \to [0,1]$ is the *transition relation* such that for all $q, q' \in Q$ and $a \in \Sigma$, $\delta(q,a,q')$ is a rational number and $\sum_{q' \in Q} \delta(q,a,q') = 1$, and $\mathsf{Acc} \subseteq Q$ is an *acceptance condition*, also called set of final/accepting states.

Notation: By default we mean PFA when we talk about PA in this paper. The transition function δ of PA $\mathcal{A}$ on input $a \in \Sigma$ can be seen as a square matrix δ_a of order $|Q|$ with the rows labeled by "current" state, columns labeled by "next state" and the entry $\delta_a(q,q')$ equal to $\delta(q,a,q')$. Given a word $u = a_0 a_1 \dots a_n \in \Sigma^+$, δ_u is the matrix product $\delta_{a_0}\delta_{a_1}\dots\delta_{a_n}$. For the empty word $\epsilon \in \Sigma^*$ we take δ_ϵ to be the identity matrix. Finally for any $Q_0 \subseteq Q$, we say that $\delta_u(q,Q_0) = \sum_{q' \in Q_0} \delta_u(q,q')$. Given a state $q \in Q$ and a word $u \in \Sigma^+$, $\mathsf{post}(q,u) = \{q' \mid \delta_u(q,q') > 0\}$ and $\mathsf{pre}(q,u) = \{q' \mid \delta_u(q',q) > 0\}$. For a set $C \subseteq Q$, $\mathsf{post}(C,u) = \cup_{q \in C}\mathsf{post}(q,u)$ and $\mathsf{pre}(C,u) = \cup_{q \in C}\mathsf{pre}(q,u)$.

Intuitively, a PA $\mathcal{A}$ starts in the initial state q_0, and if after reading $a_0, a_1, \dots, a_i$ it results in state q, then it moves to state q' with probability $\delta_{a_{i+1}}(q,q')$ on symbol a_{i+1}. A *run* of $\mathcal{A}$ starting in a state $q \in Q$ on an input $\kappa \in \Sigma^*$ is a sequence $\rho \in Q^*$ such that $|\rho| = 1 + |\kappa|$, $\rho[0] = q_0$ and for each $i \geq 0$, $\delta_{\kappa[i]}(\rho[i], \rho[i+1]) > 0$. We say that ρ is an *accepting* run if $\rho[|\rho|-1] \in \mathsf{Acc}$, i.e., the last state on ρ is an accepting state.

Given a word $\kappa \in \Sigma^*$, the PA $\mathcal{A}$ can be thought of as a Markov chain. The set of states of this Markov chain is the set $\{(q,v) \mid q \in Q, v \in \mathit{Pref}(\kappa)\}$, and the probability of transitioning from (q,v) to (q',u) is $\delta_a(q,q')$ if $u = va$ for some $a \in \Sigma$ and 0 otherwise.

PA **Languages.** Consider a PA $\mathcal{A} = (Q, q_0, \delta, \mathsf{Acc})$ over an alphabet Σ. Given a rational threshold $x \in [0,1]$ and $\rhd \in \{\geq, >\}$, the language $\mathsf{L}_{\rhd x}(\mathcal{A}) = \{u \in \Sigma^* \mid \delta_u(q_0, \mathsf{Acc}) \rhd x\}$ is the set of finite words accepted by $\mathcal{A}$ from q_0 with probability $\rhd x$.

2.1 Hierarchical Probabilistic Automata

Intuitively, a hierarchical probabilistic automaton is a PA such that the set of its states can be stratified into (totally) ordered levels. From a state q, on each letter a, the machine can transit with non-zero probability to at most one state in the same level as q, and all other probabilistic successors belong to higher levels.

Definition 2. For an integer $k > 0$, a *k-hierarchical probabilistic automaton* (HPA) is a probabilistic automaton $\mathcal{A} = (Q, q_0, \delta, \mathsf{Acc})$ over alphabet Σ such that Q can be partitioned into $k+1$ sets $Q_0, Q_1, \dots, Q_k$ satisfying the following properties:

- $q_0 \in Q_0$;
- for every i, $0 \leq i \leq k$ and every $q \in Q_i$ and $a \in \Sigma$, $|\mathsf{post}(q,a) \cap Q_i| \leq 1$; and,
- for every i, $0 < i \leq k$, $q \in Q_i$ and $a \in \Sigma$, $\mathsf{post}(q,a) \cap Q_j = \emptyset$ for every $j < i$.

For any k-HPA $\mathcal{A}$, as given above, for $0 \leq i \leq k$, we call elements of Q_i as level i states of $\mathcal{A}$. For the rest of the paper, by HPA we mean 1-HPA, unless otherwise stated.

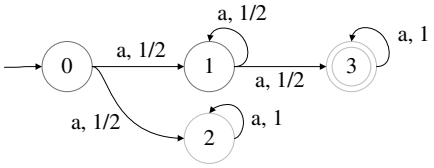

Fig. 1. 1-HPA example with initial state 0 and final state 3, and $Q_0 = \{0, 1\}$, $Q_1 = \{2, 3\}$.

Notation: Let $\mathcal{A} = (Q, q_0, \delta, Acc)$ be a 1-HPA over an input alphabet Σ, $\mathcal{A}$ has n states (i.e., $|Q| = n$) and m transitions where m is the cardinality of the set $\{(q, a, q') \mid \delta(q, a, q') > 0\}$. Let $s = |\Sigma|$. Q_0 and Q_1 are the level 0 and level 1 states respectively. It's easy to see that given any state $q \in Q_0$ and any word $\kappa \in \Sigma^*$, $\mathcal{A}$ has at most one run ρ on κ where all states in ρ belong to Q_0.

Witness Sets. A set $W \subseteq Q$ is said to be a *witness set* if W has at most one Q_0 state. Note that every subset of Q_1 is a witness set. For a witness set W such that $W \cap Q_0 \neq \emptyset$, we let q_W denote the unique state in $W \cap Q_0$. A word $\kappa \in \Sigma^*$ is *definitely accepted* from a witness set W iff for every $q \in W$ with $q \in Q_i$ (for $i \in \{0, 1\}$) there is an accepting run ρ on κ starting from q such that for every j, $\rho[j] \in Q_i$ and $\delta_{\kappa[j]}(\rho[j], \rho[j+1]) = 1$. In other words, κ is definitely accepted from a witness set W iff κ is accepted from every state q in W by a run where the HPA stays in the same level as q, and all transitions in the run are taken with probability 1. We say that a witness set W is *good* if there is at least one word that is definitely accepted from W.

Notation: Let $\mathcal{X}$ be the collection of all witness sets U such that $U \cap Q_1$ is a good witness set. Let $\mathcal{Y}$ be the collection of all good witness sets U such that $U \cap Q_0 = 1$.

3 Decision Algorithms and Robustness

For a 1-HPA $\mathcal{A}$, the problem of determining if $\mathsf{L}_{>x}(\mathcal{A})$ is non-empty (i.e., non-emptiness problem) has been shown to be in **EXPTIME** in [6]. The above paper gave an algorithm, called *backward algorithm*, for the non-emptiness problem. In this section, we present a new algorithm called *forward algorithm* and an improved version of the backward algorithm. We show that the forward algorithm has better complexity than the backward algorithm. Both algorithms are based on the calculation of good witness sets for the HPA, which is covered in Sect. 6.1.

Let $x \in [0, 1]$ be a rational threshold of size at most r[1]. It has been shown in [6], that for an HPA $\mathcal{A}$, $\mathsf{L}_{>x}(\mathcal{A}) \neq \emptyset$ iff there is a finite word u and a good non-empty witness set W, such that $|u| \leq 4rn8^n$ and $\delta_u(q_0, W) > x$. Now, let $L = 4rn8^n$.

[1] A rational number s has size r iff $s = \frac{m}{n}$ where m, n are integers, and the binary representation of m and n has at most r bits.

3.1 Forward Algorithm

The forward algorithm is based on a quantity $\mathsf{val}(C, x, u)$ defined for a set $C \subseteq Q_1$ of states in HPA $\mathcal{A}$, a threshold $x \in (0,1)$, and a finite word u. The intuition of val is, starting from q_0, after the input string u, the probability that the automaton is in some state in C is $\delta_u(q_0, C)$; this means an additional probability of $(x - \delta_u(q_0, C))$ is needed to cross the threshold x; this additional probability can only come from the probability remaining at level 0, which is $\delta_u(q_0, Q_0)$. Thus, $\mathsf{val}(C, x, u)$, defined as the ratio of the above additional probability to that of $\delta_u(q_0, Q_0)$, is the fraction of $\delta_u(q_0, Q_0)$ that still needs to move to C so that the probability of reaching the accepting states at the end exceeds the threshold x. Formally, we have the following definition.

$$\mathsf{val}(C, x, u) = \begin{cases} \frac{x - \delta_u(q_0, C)}{\delta_u(q_0, Q_0)} & \text{if } \delta_u(q_0, Q_0) \neq 0 \\ +\infty & \text{if } \delta_u(q_0, C) < x, \delta_u(q_0, Q_0) = 0 \\ 0 & \text{if } \delta_u(q_0, C) = x, \delta_u(q_0, Q_0) = 0 \\ -\infty & \text{if } \delta_u(q_0, C) > x, \delta_u(q_0, Q_0) = 0 \end{cases}. \tag{1}$$

From the definition of the function val, and using algebraic simplifications, the following properties are easily proved.

Lemma 1. *For $u, v \in \Sigma^+$, if $C, D \subseteq Q_1$, $q, q' \in Q_0$ be such that $C = \mathsf{pre}(D, v) \cap Q_1, \{q\} = \mathsf{post}(q_0, u), \{q'\} = \mathsf{post}(q, v)$, $x' = \delta_v(q, D), z' = \delta_v(q, q'), y' = 1 - z'$ and $y', z' > 0$, then the following hold:*

1. $\mathsf{val}(D, x, uv) = \frac{\mathsf{val}(C, x, u) - x'}{z'}$, *and*
2. *If $C = D$ (i.e., $\mathsf{post}(C, v) = C$) and $q' = q$, then for any integer $p \geq 0$, $\mathsf{val}(C, x, uv^p) = \frac{x'}{y'} + (\frac{1}{z'})^p (\mathsf{val}(C, x, u) - \frac{x'}{y'})$.*

The val values play an important role in deciding whether a word κ is accepted by an HPA. It has been shown in [6] that κ is accepted with probability $> x$ iff κ can be divided into strings u, κ', i.e., $\kappa = u\kappa'$, and there is a witness W such that κ' is definitely accepted from W and one of the following conditions:

- $W \subseteq Q_1$, $\mathsf{val}(W, x, u) < 0$
- $W \cap Q_0 \neq \emptyset$ and $0 \leq \mathsf{val}(W \cap Q_1, x, u) < 1$.

Now, to check the existence of a string κ satisfying the above property, we define another quantity $\mathsf{minval}(W, i)$ for each $W \in \mathcal{X}$ and $i \geq 0$ as follows. Intuitively, this is the minimum of the values given by val over all strings of length at most i. Formally, we have the following definition where $\min\{\emptyset\} = +\infty$.

$$\mathsf{minval}(W, i) = \min\{\mathsf{val}(W \cap Q_1, x, u) || u| \leq i\}.$$

The following lemma is easily proved using the observations above.

Lemma 2. $\mathsf{L}_{>x}(\mathcal{A}) \neq \emptyset$ *iff for some $i \geq 0$, either ($\exists W \in \mathcal{X} : \mathsf{minval}(W, i) < 0$) or ($\exists W \in \mathcal{Y} : 0 \leq \mathsf{minval}(W, i) < 1$).*

For any $C \subset Q, u \in \Sigma^+$, recall that $\mathsf{pre}(C,u) = \cup_{q \in C} \mathsf{pre}(q,u)$ (see Sect. 2). Now, for any $W \in \mathcal{X}$, $a \in \Sigma$, and $q \in Q_0$, let $W_{a,q} = (\mathsf{pre}(W,a) \cap Q_1) \cup \{q\}$. It should be easy to see that $W_{a,q} \in \mathcal{X}$. To calculate the values $\mathsf{minval}(W,i)$ for $W \in \mathcal{X}$ and $i \geq 0$, we present the following incremental algorithm.

$$\mathsf{minval}(W,0) = \begin{cases} x & if (q_0 \in W); \\ +\infty & else. \end{cases}$$

For $i > 0$, and $W \cap Q_0 \neq \emptyset$,

$$\begin{aligned} \mathsf{minval}(W,i) = \; & \min\{\mathsf{minval}(W,i-1), \\ & \min\{\frac{\mathsf{minval}(W_{a,q},i-1) - \delta_a(q, W \cap Q_1)}{\delta_a(q,q_W)} \mid \\ & a \in \Sigma, q \in Q_0, \delta_a(q,q_W) > 0\}\}. \end{aligned} \tag{2}$$

Further more, for $i > 0$ and $W \cap Q_0 = \emptyset$, $\mathsf{minval}(W,i)$ is updated as follows: if $\exists a \in \Sigma, q \in Q_0$ such that $\mathsf{minval}(W_{a,q}, i-1) < \delta_a(q, W \cap Q_1)$ then $\mathsf{minval}(W,i) = -\infty$; otherwise, $\mathsf{minval}(W,i) = \mathsf{minval}(W,i-1)$. By induction on i, it can be shown the above algorithm computes $\mathsf{minval}(\cdot,\cdot)$ correctly, i.e., the computed values agree with the definition of $\mathsf{minval}(\cdot,\cdot)$ given earlier. Clearly, $\mathsf{minval}(W,i)$ is monotonically non-increasing with increasing values of i.

Let $w = |\mathcal{X}|$. It should be easy to see that $w \leq 2^n \leq L$. The algorithm computes the values of $\mathsf{minval}(W,i)$, for each witness set $W \in \mathcal{X}$, in increasing values of $i = 0, \ldots$ until one of the following conditions is satisfied: (a) $(\exists W \in \mathcal{X} : \mathsf{minval}(W,i) < 0)$ or $(\exists W \in \mathcal{Y} : 0 \leq \mathsf{minval}(W,i) < 1)$; (b) $i > 0$ and $\forall W \in \mathcal{X}$, $\mathsf{minval}(W,i) = \mathsf{minval}(W,i-1)$; (c) $i = w$. Observe that once condition (b) holds, the values of $\mathsf{minval}(W,i)$ do not change from that point onwards, i.e., they reach a fixed point. The algorithm terminates for the smallest integer i such that either of the conditions (a) or (b) hold. If at termination, condition (a) holds then it answers "yes"; if (a) does not hold but (b) holds then it answers "no"; if both (a) and (b) do not hold, but (c) holds then it will answer "yes". Now we have the following theorem.

Theorem 1 (Correctness of Forward Algorithm). *Forward algorithm will definitely terminate after at most w iterations providing the correct answer, i.e., it outputs "yes" iff $\mathsf{L}_{>x}(\mathcal{A}) \neq \emptyset$.*

Proof. Clearly, the algorithm terminates within at most w iterations and outputs an "yes" or "no" answer. It is enough if we prove that it outputs the correct answer. Assume that the algorithm outputs an "yes" answer. Here there are two cases. The first one is condition (a) is satisfied. Clearly, from Lemma 2, we see that $\mathsf{L}_{>x}(\mathcal{A}) \neq \emptyset$. The second case, is that condition (c) is satisfied, i.e., $i = w$ at termination and neither of conditions (a), (b) is satisfied. Let $F_0 = \{W \in \mathcal{X} \mid q_0 \in W\}$. For $j > 0$, let $F_j = \{W \in \mathcal{X} \mid \mathsf{minval}(W,j) < \mathsf{minval}(W,j-1)\}$. It is easily seen that for each $j > 0$, $F_j \neq \emptyset$, otherwise the fixed point condition (b) would have been satisfied before the w^{th} iteration. Also for each $W \in F_j$, $W \cap Q_0 \neq \emptyset$ and there exists $V \in F_{j-1}$ and $a \in \Sigma$ such

that $\mathsf{minval}(W,j) = \frac{\mathsf{minval}(V,j-1)-\delta_a(q_V,W\cap Q_1)}{\delta_a(q_V,W\cap Q_0)}$. Since $F_w \neq \emptyset$, the above property would imply that there exists $W_0 \in F_0$ and for each $0 < j \leq w$, there exists $W_j \in F_j$ and $a_j \in \Sigma$ such that $\mathsf{minval}(W_j,j) = \frac{\mathsf{minval}(W_{j-1},j-1)-\delta_{a_j}(q,W_j\cap Q_1)}{\delta_{a_j}(q,W_j\cap Q_0)}$ where $q = q_{W_{j-1}}$. Let $\kappa_j = (a_1,\ldots,a_j)$ for $j > 0$. Then, by simple induction on j, it is seen that $\mathsf{minval}(W_j,j) = \mathsf{val}(W_j \cap Q_1, x, \kappa_j)$.

Now, using Pigeon Hole principle, we see there exist integers $0 \leq j < k \leq w$, such that $W_j = W_k$. Let $V = W_j$ and q be the state q_{W_j}. Clearly $\mathsf{minval}(V,k) < \mathsf{minval}(V,j)$. Now, let $u = \kappa_j$ as given above, and $v \in \Sigma^*$ be the string $(a_{j+1}, a_{j+2}, \ldots, a_k)$. Let $x' = \delta_v(q, V \cap Q_1), z' = \delta_v(q,q)$ and $y' = 1 - z'$. From the earlier observation, $\mathsf{minval}(V,j) = \mathsf{val}(V \cap Q_1, x, u)$ and $\mathsf{minval}(V,k) = \mathsf{val}(V \cap Q_1, x, uv)$. Using property (1) of Lemma 1, it is seen that $\mathsf{minval}(V,j) - \mathsf{minval}(V,k) = \frac{y'}{z'}(\frac{x'}{y'} - \mathsf{minval}(V,j))$. Hence $\frac{x'}{y'} > \mathsf{minval}(V,j)$. Observe that $|u| = j, |v| = k-j$. Now, for an integer $p > 0$, let $n_p = j + p(k-j)$. By considering the string uv^p, and using part (2) of Lemma 1, we see that $\mathsf{minval}(V,n_p) = \rho + (\frac{1}{z'})^p(\mathsf{minval}(V,j) - \rho)$ where $\rho = \frac{x'}{y'}$. Since $(\mathsf{minval}(V,j) - \rho) < 0$ and $z' < 1$, we see that $\mathsf{minval}(V,n_p) < 0$ for sufficiently large $p > 0$. Applying Lemma 2, we see that $\mathsf{L}_{>x}(\mathcal{A}) \neq \emptyset$.

Now, assume the algorithm outputs a "no" answer on termination, which means condition (b) is satisfied but condition (a) is not. Assume the algorithm terminated after j iterations. Clearly, condition (a) is not satisfied for all $i \leq j$. Condition (b) implies for all $i \geq j$ and for all $W \in \mathcal{X}$, $\mathsf{minval}(W,i) = \mathsf{minval}(W,j)$. Hence for all $i \geq j$, condition (a) is not satisfied. By Lemma 2, we see that $\mathsf{L}_{>x}(\mathcal{A}) = \emptyset$.

3.2 Backward Algorithm

The basic backward algorithm has been introduced in [6]. Here we present an enhanced algorithm supporting convergence and early termination before completing L iterations, not only for cases when $\mathsf{L}_{>x}(\mathcal{A}) \neq \emptyset$, but also for some cases when $\mathsf{L}_{>x}(\mathcal{A}) = \emptyset$.

Let $\mathcal{X}'$ be the set of all $W \in \mathcal{X}$ such that $W \cap Q_0 \neq \emptyset$. Let $\mathcal{Y}'$ be the set of all good witness sets. A function $Prob(\cdot,\cdot)$ which maps each pair (W,i), where $W \in \mathcal{X}'$ and $i \geq 0$, to a probability value, is defined as follows: Here we take $\max\{\emptyset\} = 0$;

$$Prob(W,i) = \max\{\delta_u(q_W,V) | u \in \Sigma^*, V \in \mathcal{Y}', \mathsf{post}(W \cap Q_1, u) \subseteq V, |u| \leq i\}.$$

$Prob(W,i)$ denotes the maximum probability of reaching any set $V \in \mathcal{Y}'$ using an input of length at most i, from the state q_W. If $Prob(W,i) > x$, there is an input accepted from q_W with probability $> x$ (because from q_W using an input of length at most i, we can reach a good witness set V, with probability $> x$, and then we can use any sequence that is accepted from all states in V with probability 1; there exists at least one such sequence because $V \in \mathcal{X}'$). Clearly, $Prob(W,i)$ is monotonically non-decreasing with increasing values of i.

It has been shown in [6] that $\mathsf{L}_{>x}(\mathcal{A}) \neq \emptyset$ iff $Prob(\{q_0\}, L) > x$. The following inductive algorithm is proposed to compute $Prob(\cdot, \cdot)$.

$$\begin{aligned} Prob(W, 1) &= \max\{\delta_a(q_W, V) | a \in \Sigma, V \in \mathcal{Y}', \mathsf{post}(W \cap Q_1, a) \subseteq V\}; \\ Prob(W, i+1) &= \max(\{Prob(W, i)\} \cup \\ &\qquad \{\delta_a(q_W, q_V) Prob(V, i) + \delta_a(q_W, V \cap Q_1) \\ &\qquad\quad | a \in \Sigma, V \in \mathcal{X}', \mathsf{post}(W \cap Q_1, a) \subseteq V\}). \end{aligned} \tag{3}$$

The backward algorithm works as follows. For increasing values of $i = 1, \ldots, L$, it computes $Prob(W, i)$ for each $W \in \mathcal{X}'$. The algorithm terminates at the smallest $i > 1$ such that one of the following conditions is satisfied: (a) $Prob(\{q_0\}, i) > x$, (b) for each $W \in \mathcal{X}'$, $Prob(W, i) = Prob(W, i-1)$, (c) $i = L$ and $Prob(\{q_0\}, i) \leq x$. It is not difficult to see that if the convergence condition (b) is satisfied for a particular value i, then for all $j \geq i$, $Prob(W, j) = Prob(W, i)$, i.e., the values given by $Prob(.,.)$ do not change. On termination, if condition (a) is satisfied then the algorithm gives "yes" answer, otherwise it gives a "no" answer. We refer to condition (a) as positive termination, condition (b) and (c) as negative termination conditions.

The backward algorithm is in **EXPTIME** in the worst case. However, in case it converges and terminates early, it may take much less time than the theoretically worst time. The following theorem states the correctness of the algorithm and is proved by induction on i and using the result of [6].

Theorem 2 (Backward Termination). *The backward algorithm is correct, that is, it answers "yes", iff* $\mathsf{L}_{>x}(\mathcal{A}) \neq \emptyset$.

Consider the HPA $\mathcal{A}$ in Figure 1, for $x = \frac{1}{2}$. It is easy to see that $L_{>\frac{1}{2}}(\mathcal{A})$ is empty. The backward algorithm will run all the L iterations and output a "no" answer.

3.3 Forward vs. Backward

Both forward and backward algorithms can be utilized to solve the decidability problem. Suppose the HPA $\mathcal{A}$ has n states, m transitions, $w = |\mathcal{X}|$ and the number of input symbols is s. After careful calculation, we see that in the worst case the forward algorithm runs in time $O((mn + L_f m^2)w)$, while backward algorithm runs in $O((L_b m + m + n)sw^2)$, where L_f and L_b denote the number of iterations before the respective algorithms terminate. Clearly $L_f \leq w$, while $L_b \leq L$. Since $w = O(2^n)$ and $L > w$ is also exponential in n, both these algorithms run in time exponential in n in the worst case. However, the above complexities show that the forward algorithm is quadratic in w, while that of the backward algorithm is cubic in w, considering the dependence of L_f, L_b on w. Thus, forward algorithm has better worst case complexity. Our experimental results, given in Sect. 6.3, show that the forward algorithm runs much faster than the backward algorithm. Also, the forward algorithm critically depends on x, i.e., the definition of $\mathsf{minval}(\cdot, \cdot)$ function depends on x, while the $Prob(\cdot, \cdot)$ used

in the backward algorithm is independent of x. Also, forward algorithms often terminates much faster than the backward algorithm, especially when $\mathsf{L}_{>x}(\mathcal{A}) = \emptyset$ and the latter terminates on condition (c), as is for the example in Fig. 1.

3.4 Robustness

So far we have given algorithms for the verification problem. When failure-prone systems are modeled as HPA, an equally important notion is *robustness*. As we will show later, we model the incorrectness of an open system under failures as a 1-HPA. In this case, we define the robustness of an HPA $\mathcal{A}$ as $(1-y)$ where y is the least upper bound of the set of values $\{x \,|\, \mathsf{L}_{>x}(\mathcal{A}) \neq \emptyset\}$. This value is the greatest lower bound of $\{z|\ z$ is the probability of rejection of some input string by $\mathcal{A}\}$. The value of y, and hence the robustness, can be found within some accuracy by using binary search repeatedly employing the forward algorithm for various values of x. Although the backward algorithm is less efficient for the verification problem, it can be used to compute the exact value of the robustness in some cases. Suppose the backward algorithm reaches a fixed point after k iterations, then $y = Prob(\{q_0\}, k)$ where q_0 is the initial state of $\mathcal{A}$. In this case, robustness has the exact value $(1 - y)$.

4 Undecidability for 2-HPA

In this section, we consider 2-HPA and show that the problem of checking $\mathsf{L}_{>x}(\mathcal{A}) \neq \emptyset$ is undecidable for the case when $\mathcal{A}$ is a 2-HPA. This result closes the gap between the decidability and undecidability results for HPA.

Theorem 3. *Given a* 2-HPA $\mathcal{A}$*, a rational threshold* $x \in [0, 1]$ *the problem of determining if* $\mathsf{L}_{>x}(\mathcal{A}) \neq \emptyset$ *is undecidable.*

Proof. We use the approach given in the [4], with major critical modifications. There the result was proved by reducing the halting problem of deterministic 2-counter machines to the non-emptiness problem of HPA with strict acceptance thresholds. The broad ideas behind that construction are as follows. Let T be deterministic 2-counter machine with control states Q and a special halting state q_h. It is assumed, without loss of generality, that each transition of T changes at most one counter and the initial counter values are 0. Recall that a configuration of such a machine is of the form (q, a^{i+1}, b^{j+1}), where $q \in Q$ is the current control state, and a^i (b^j) is the unary representation of the value stored in the first counter (second counter, respectively). The input alphabet Σ of the constructed HPA $\mathcal{A}_T$ will consist of the set Q as well as the 5 symbols- ",", "(", ")", a and b. The HPA $\mathcal{A}_T$ will have the following property: if $\rho = \sigma_1\sigma_2\cdots\sigma_n$ is a halting computation of T then $\mathcal{B}$ will accept the word ρ with probability $> \frac{1}{2}$; if $\rho = \sigma_1\sigma_2\cdots$ is a non-halting computation of T then $\mathcal{A}_T$ will accept every prefix of ρ with probability $< \frac{1}{2}$; and if ρ is an encoding of an invalid computation (i.e., if ρ is not of the right format or has incorrect transitions) and no prefix of ρ is a valid halting computation of T then $\mathcal{A}_T$ will accept ρ with probability

$< \frac{1}{2}$. Given this property we will be able to conclude that T halts iff $\mathsf{L}_{>\frac{1}{2}}(\mathcal{A}_T)$ is non-empty, thus proving the theorem.

In order to construct an HPA $\mathcal{A}_T$ with the above properties, $\mathcal{A}_T$ must be able to check if there is a finite prefix α of input ρ that encodes a valid halting computation of T. This requires checking the following properties. (1) α is of the right format, i.e., it is a sequence of tuples of the form (q, a^i, b^j). (2) The first configuration is the initial configuration. (3) Successive configurations in the sequence follow because of a valid transition of T. (4) In the last configuration, T is in the halting state q_h.

Checking properties (1), (2) and (4) can be easily accomplished using only finite memory. On the other hand checking (3) requires checking that the counters are updated correctly which cannot be done deterministically using finite memory. Our major changes to the proof are for step (3). Our construction for step (3) is as follows. The main part of the 2-HPA $\mathcal{A}_T$, corresponding to step (3), is shown in Fig. 2. In this automaton, The level 0 states keep track of which part of each configuration the input sequence is in. The set of states of $\mathcal{A}_T$ is a super set of Q. After the input q of an input configuration (q, a^i, b^j), $\mathcal{A}_T$ will be in state q. On the following input symbol comma, $\mathcal{A}_T$ from state q, probabilistically decides to do: (a) check that the first set of counters (i.e., those denoted by a symbols) in the current and next configurations match; (b) check that the second set of counters (i.e., those denoted by b symbols) in the current and next configurations match; (c) continue and go to state q' at level 0. From state q', $\mathcal{A}_T$ has transitions as shown on inputs a, b, and at the end of the current configuration, on the input string of the form "$)(u$" (where $u \in Q$), it will go to state u with probability 1. From any state, if no transition on an input symbol is shown, it is assumed that the automaton stays in the same state on that input. In every state, whenever the input symbol is a state component of a configuration of T which is a halting state, $\mathcal{A}_T$ goes to the accepting state with probability 1. To check the second counter values in the current and next configurations, a similar mechanism with additional level 1 states s_1, s_2, s_3, etc., is employed.

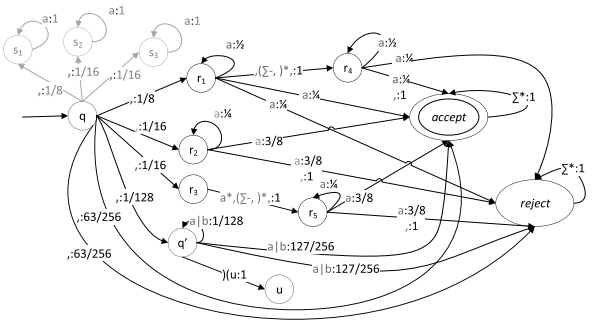

Fig. 2. Partial HPA used in the Undecidability Proof. Inputs and associated probabilities are shown on each transition separated by ":" symbol.

Now, we analyze probability of acceptance and rejection as follows. Suppose that the first counter values in the current and next configurations have values i, j respectively. Then, let p_{accept}, p_{reject} be the probabilities of $\mathcal{A}_T$ going to accept and reject states, respectively, after going to r_1, r_2 or r_3 at the beginning of the current configuration. Their values are shown below in Table 1.

Table 1. Probability analysis for Undecidability Proof.

	P_{accept}	P_{reject}	Sum
Through r_1	$\frac{1}{16}(1 + 2^{-(i+j)})$	$\frac{1}{16}(1 - 2^{-(i+j)})$	$\frac{1}{8}$
Through r_2	$\frac{1}{32}(1 - 2^{-2i})$	$\frac{1}{32}(1 + 2^{-2i})$	$\frac{1}{16}$
Through r_3	$\frac{1}{32}(1 - 2^{-2j})$	$\frac{1}{32}(1 + 2^{-2j})$	$\frac{1}{16}$

From this, we see that

$$p_{reject} - p_{accept} = \frac{1}{16}(2^{-2i} + 2^{-2j} - 2 \cdot 2^{-(i+j)}) = \frac{1}{16}(2^{-i} - 2^{-j})^2 \geq 0.$$

If $i = j$, then $p_{accept} - p_{reject} = 0$. Now consider the case when $i \neq j$. Assume with out loss of generality that $i > j$. In this case,

$$p_{reject} - p_{accept} \geq \frac{1}{16}(2^{-j} - 2^{-j-1})^2 \geq \frac{1}{16}2^{-2(j+1)} = \frac{1}{64} \cdot 2^{-2j}.$$

Now the probability that $\mathcal{A}_T$ is at a level 0 state (i.e., such as q) is $\frac{1}{128} \cdot (\frac{1}{128})^{(i+i')}$ where i, i' are the values of the first and second counters in the current configuration, which is $\leq (\frac{1}{64})^2 \cdot 2^{-(2 \cdot i + 7i')}$. This value is less than $\frac{1}{64} \cdot 2^{-2j}$ when $i > j$, and is less than $\frac{1}{64} \cdot 2^{-2i}$ when $j > i$. (The same reasoning is used to establish this property when we consider the second counter values using the values i', j'). This shows that $p_{reject} - p_{accept}$ is greater the probability that $\mathcal{A}_T$ is at a level 0 state plus the probabilities that $\mathcal{A}_T$ is in any of the intermediate level 1 states which go to the accepting state if the following configuration has a halt state in an illegal computation. From this, we see that even if we get an illegal computation that ends with a halting state, that computation will be accepted with probability $< \frac{1}{2}$. On the other hand a legal halting computation will be accepted with probability $> \frac{1}{2}$.

Here we are assuming the counter values in successive configurations are required to be equal. However, if a counter values in the next configurations need to be one less (or one more) than that in the current configuration, then the above mechanism is easily modified to achieve this.

5 Modeling Failure-Prone Open Systems

Many concurrent systems, such as web servers, run on multiple processors. They take inputs from the environment (e.g. users), and consume inputs by going

through a sequence of states. However, they are subjected to processor failures. When such failures occur, exception handling is executed and usually the load on the failed processor, i.e., the processes running on it, are transferred to the remaining processors. Thus after occurrence of a failure, the computation of the system changes. We model the failures with probability distributions, and model the behavior as *Open Probabilistic Transition Systems* (OPTS).

An Open Probabilistic Transition System $\mathbb{T}$ is 6-tuple $(S, \Sigma, \eta, s_0, \mathcal{P}, \phi)$ where S is a set of states, Σ is an input alphabet, $\eta : S \times \Sigma \times S \rightarrow [0,1]$ is the *transition relation* so that for all $s, s' \in S$ and $a \in \Sigma$, $\eta(s,a,s')$ is a rational number and $\sum_{s' \in Q} \eta(s,a,s') = 1$, and $s_0 \in S$ is the initial state, $\mathcal{P}$ is a set of atomic propositions and $\phi : S \rightarrow 2^{\mathcal{P}}$ is a function assigning a set of atomic propositions to each state. We assume that $S, \mathcal{P}$ are finite sets. Observe that S, η, s_0 and Σ are similar to the corresponding components of a PA. However, $\mathbb{T}$ has additional information given by $\mathcal{P}$ and ϕ that label the states with atomic propositions. For a given OPTS $\mathbb{T}$ as given above, we can define a probability space $(S^\omega, \mathbb{E}, \psi)$ where S^ω is the set of infinite sequences of states, $\mathbb{E}$ is the set of measurable subsets of S^ω which is the σ-algebra generated by cylinders of the form uS^ω where $u \in S^*$, and ψ is a probability function defined on it. As pointed out earlier, we use OPTSes to model failure-prone open concurrent systems. An input to $\mathbb{T}$ is an infinite sequence $\beta \in \Sigma^\omega$. A computation σ of $\mathbb{T}$ on input β is an infinite sequence of states $(s_0, \ldots, s_i, \ldots)$ starting from the initial state s_0 such that $\eta(s_i, \beta[i], s_{i+1}) > 0$ for all $i \geq 0$. We let $\mathcal{C}(\mathbb{T}, \beta)$ denote the set of computations of $\mathbb{T}$ on β. For a computation σ as given above, we let $\phi(\sigma)$ to be the sequence $(\phi(s_0), \phi(s_1), \ldots)$.

We consider the problem of verifying OPTSes against correctness property specified by deterministic safety automata. The inputs to the safety automaton are elements of $2^{\mathcal{P}}$. Formally a safety specification $\mathcal{B}$ for a OPTS $\mathbb{T}$ as given above is a deterministic finite-state automaton $\mathcal{B} = (R, \delta_1, r_0, F_1)$ where R is a finite set of automaton states, $\delta_1 : R \times 2^{\mathcal{P}} \rightarrow R$ is the next state function such that $\delta_1(r_{error}, c) = r_{error}$ for all $c \in 2^{\mathcal{P}}$, $r_0 \in R$ is the initial state and $F_1 = R - \{r_{error}\}$ where r_{error} is unique state in R called the *error* state. As usual, we define a run ρ of $\mathcal{B}$ on an input $t = (t_0, \ldots) \in (2^{\mathcal{P}})^\omega$ to be an infinite sequence of states $(r_0, r_1, \ldots)$ starting from the initial state r_0 such that $r_{i+1} \in \delta_1(r_i, t_i)$. We say that the above run is an accepting run if $r_i \in F_1$ for all $i \geq 0$, i.e., r_{error} does not appear on the run.

Now, we define the probability of satisfaction of the property $\mathcal{B}$ by the OPTS $\mathbb{T}$ on an input sequence $\beta \in \Sigma^\omega$, denoted by $Pr(\mathbb{T}, \beta, \mathcal{B})$, to be the probability given by $\psi(D)$ where D is the set of all computations σ of $\mathbb{T}$ on the input β such that $\phi(\sigma)$ is accepted by $\mathcal{B}$. Note that $1 - Pr(\mathbb{T}, \beta, \mathcal{B})$ denotes the probability that $\mathbb{T}$ does not satisfy the property $\mathcal{B}$ on the input β. The *verification problem* for OPTSes is — given $\mathbb{T}, \mathcal{B}$ as above and a rational $x \in [0,1]$ – determine if $Pr(\mathbb{T}, \beta, \mathcal{B}) \geq x$ for all $\beta \in \Sigma^\omega$.

Now, we transform the above verification problem to checking emptiness problem for PAs. Given $\mathbb{T}, \mathcal{B}, x$ as specified above, we construct a PA, $\mathcal{A}(\mathbb{T}, \mathcal{B})$, over the set of input symbols Σ, which is a product of $\mathbb{T}, \mathcal{B}$, and is defined as

follows: $\mathcal{A}(\mathbb{T},\mathcal{B}) = (Q, q_0, \delta, F)$ where $Q = S \times R$, $q_0 = (s_0, r_0)$, $F = S \times \{r_{error}\}$ and δ is defined as follows: for any $s, s' \in S$ and $r, r' \in R$, $a \in \Sigma$, $\delta((s,r),a,(s',r')) = \eta(s,a,s')$ if $r' = \delta_1(r, \phi(s))$, and is 0 otherwise. The following theorem can be easily shown.

Theorem 4. *For any OPTS* $\mathbb{T} = (S, \Sigma, \eta, s_0, \mathcal{P}, \phi)$ *and safety automaton* $\mathcal{B} = (R, \delta_1, r_0, F)$ *over* $2^{\mathcal{P}}$ *and rational* $x \in [0,1]$*, for all* $\beta \in \Sigma^\omega$*,* $\mathbb{T}$ *satisfies the specification* $\mathcal{B}$ *with probability* $\geq x$ *(i.e.,* $Pr(\mathbb{T},\beta,\mathcal{B}) \geq x$*) iff for all* $u \in \Sigma^*$ *the* PA $\mathcal{A}(\mathbb{T},\mathcal{B})$ *accepts* u *with probability* $\leq 1 - x$ *iff* $\mathsf{L}_{>(1-x)}(\mathcal{A}(\mathbb{T},\mathcal{B})) = \emptyset$.

We say that an OPTS $\mathbb{T}$ is a k-OPTS ($k > 0$) if its set of states S can be partitioned into $k+1$ sets $S_0, S_1, \ldots, S_k$ satisfying the following conditions: (a) $s_0 \in S_0$; (b) for every integer $i \in [0,k]$, every $s \in S_i$, and every $a \in \Sigma$, there is at most one state $s' \in S_i$ such that $\eta(s,a,s') > 0$; (c) for every i, j such that $0 \leq j < i \leq k$, for every $s \in S_i, s' \in S_j$, $\eta(s,a,s') = 0$. k-OPTSes can be used to model web applications with at most k processor failures. Obviously, if $\mathbb{T}$ is a k-OPTS then $\mathcal{A}(\mathbb{T},\mathcal{B})$ is a k-HPA. If $\mathbb{T}$ is a 1-OPTS, using the above theorem, we can verify that $Pr(\mathbb{T},\beta,\mathcal{B}) \geq x$ for all $\beta \in \Sigma^\omega$ by checking $\mathsf{L}_{>1-x}(\mathcal{A}(\mathbb{T},\mathcal{B})) = \emptyset$. The latter property can be checked using the algorithms given in Sect. 3. We can define the *robustness* of $\mathbb{T}$ with respect to the correctness specification $\mathcal{B}$ to be the greatest lower bound of the set $\{Pr(\mathbb{T},\beta,\mathcal{B}) \mid \beta \in \Sigma^\omega\}$. This value can be computed as the robustness of $\mathcal{A}(\mathbb{T},\mathcal{B})$ which can be computed as presented in the previous section. Finally, the result of Sect. 4 also shows that the problem of checking if a given k-OPTS with $k \geq 2$ satisfies a safety specification with probability $> x$ on all inputs, for some rational threshold x, is undecidable.

6 Implementation and Experiment

In this section, we will present the implementation of various HPA related algorithms in our HPA tool[2]. We will also describe how we abstract 1-HPA models from web applications assuming at most one failure. We will present experimental results showing the effectiveness of our verification algorithms on the abstracted web applications.

6.1 Implementation of the Verification Algorithms

In our verification process, we first validate whether a given PA $\mathcal{A}$ is an HPA. If so, we compute the smallest integer k such that $\mathcal{A}$ is a k-HPA, and also classify the states of $\mathcal{A}$ into different levels. Then we obtain good witness sets for 1-HPA. Finally, we run the forward and backward verification algorithms on good witness sets for 1-HPA.

[2] HPA tool website: https://github.com/benyue/HPA.

Detect HPA **and Assign Levels.** In this section, we describe the algorithm for checking if a given PA is an HPA, and for classifying its states into different levels.

Given a PA $\mathcal{A} = (Q, q_0, \delta, \mathsf{Acc})$ over input alphabet Σ, we first construct a directed graph $G_{\mathcal{A}} = (Q, E)$ with Q as its set of nodes and $E = \{(q, q') \mid \exists a{\in}\Sigma, \delta(q, a, q') > 0\}$. Without loss of generality, we assume that all nodes in Q are reachable from the initial node q_0. A *strongly connected component* (SCC) of $G_{\mathcal{A}}$ is a maximal subset of Q such that there is a path in $G_{\mathcal{A}}$ between every pair of nodes in it; the *component graph* of $G_{\mathcal{A}}$ is the directed graph $F_{\mathcal{A}} = (\mathcal{C}, \mathbb{E})$ where $\mathcal{C}$ is the set of SCCs of $G_{\mathcal{A}}$, and $(C, D){\in}\mathbb{E}$ iff $C \neq D$ and $\exists q{\in}C, q'{\in}D$ such that $(q, q'){\in}E$. It is known that $F_{\mathcal{A}}$ is acyclic. Let $n = |Q|$ and m be the number of triples (q, a, q') such that $\delta(q, a, q') > 0$. Using standard graph algorithms, $G_{\mathcal{A}}$ and $F_{\mathcal{A}}$ can be constructed in time $O(n+m)$. We refer to $C_0{\in}\mathcal{C}$ where $q_0{\in}C_0$ as the *initial* SCC. Under reachability assumption, every node in $F_{\mathcal{A}}$ is reachable from C_0.

A SCC node $C{\in}\mathcal{C}$ is said to be *conflicting* iff $\exists a{\in}\Sigma$ and $\exists q, q_1, q_2{\in}C$ such that $q_1 \neq q_2$ and $\delta(q, a, q_1) > 0$ and $\delta(q, a, q_2) > 0$. All nodes in each $C{\in}\mathcal{C}$ are on the same level. It is not difficult to see that $\mathcal{A}$ is an HPA iff there are no conflicting nodes in $\mathcal{C}$, i.e., there are no conflicting SCCs in $G_{\mathcal{A}}$. This algorithm has time complexity $O(n + m)$.

For an HPA $\mathcal{A}$, let $Min\_level(\mathcal{A})$ be the minimum k such that $\mathcal{A}$ is k-HPA. Now, we present an algorithm to compute $Min\_level(\mathcal{A})$. For any $Q'{\subseteq}Q$ and any $q{\in}Q$, we say that q *is deterministic with respect to* Q', if for each $a{\in}\Sigma$, there is at most one state $q'{\in}Q'$ such that $\delta(q, a, q') > 0$. We say that an SCC $C{\in}\mathcal{C}$ is *deterministic* with respect to Q', if every state in C is deterministic with respect to Q'. For any sub-graph H of $F_{\mathcal{A}}$, let $States(H)$ be the union of all SCCs C such that C is a node in H and $TD(H)$ be the set of all nodes C in H such that all nodes in H, that are reachable from C (including C), are deterministic with respect to $States(H)$. Note that all terminal nodes of H are in $TD(H)$. Let $Level\_seq(\mathcal{A})$ be the unique maximum length sequence $(H_0, H_1, \ldots, H_\ell)$ of non-empty subgraphs of $F_{\mathcal{A}}$ such that $H_0 = F_{\mathcal{A}}$, for each $i, 0 \leq i < \ell$, H_{i+1} is the subgraph of H_i obtained by deleting all nodes in $TD(H_i)$. Since $Level\_seq(\mathcal{A})$ is the maximum length sequence, it is easy to see that every node in H_ℓ is a deterministic SCC with respect to $States(H_\ell)$. Now, we have the following theorem.

Theorem 5. *$\mathcal{A}$ is an $\ell-$*HPA *iff $G_{\mathcal{A}}$ has no conflicting SCCs. Further more, if $G_{\mathcal{A}}$ has no conflicting SCCs and $Level\_seq(\mathcal{A}) = (H_0, \ldots, H_\ell)$ then $Min\_level(\mathcal{A}) = \ell$ and $\mathcal{A}$ is a ℓ-*HPA *with $Q_i = States(H_{\ell-i})$ being the set of states at level i, for $0 \leq i \leq \ell$.*

It is easy to see that using standard graph algorithms, we can check whether $\mathcal{A}$ is an HPA in time $O(n + m)$; if it is an HPA, we compute $Level\_seq(\mathcal{A})$, $Min\_level(\mathcal{A})$ and partition the states of $\mathcal{A}$ into different levels in time $O(Min\_level(\mathcal{A}) \cdot (n + m))$.

Obtain Good Witness Sets. Now assume $\mathcal{A} = (Q, q_0, \delta, \mathsf{Acc})$ is a 1-HPA. In Sect. 2.1, we defined the set $\mathcal{Y}$ as the set of good witness sets U such that $U \cap Q_0 \neq \emptyset$, and set $\mathcal{X}$ as the set of all witness sets U such that $U \cap Q_1$ is a good witness set. To compute these sets, we start from the set Acc of final states of $\mathcal{A}$, and traverse backward using each input symbol and its predecessors, and compute $\mathcal{X}$ and $\mathcal{Y}$ incrementally. Equivalently, we construct a standard nondeterministic automaton $\mathcal{A}^{-1}$ which is a reversal of $\mathcal{A}$, i.e., $\mathcal{A}^{-1}$ has the same set of states as $\mathcal{A}$; there is a transition from q to q' on input a iff $\delta(q', a, q) > 0$; the set of its initial states is Acc. Our algorithm for computing $\mathcal{X}$ and $\mathcal{Y}$, is similar (with some critical modifications) to that of determinizing $\mathcal{A}^{-1}$ using standard subset-construction. The complexity of this algorithm is $O(wsmn)$ where $n = |Q|$, m is the number of transitions, $w = |\mathcal{X}|$ and $s = |\Sigma|$.

Run the Verification Algorithms on the Witness Sets. Now we can run the forward and backward algorithms (Sect. 3) on $\mathcal{A}$ and its witness sets. The algorithms use the threshold value x obtained from environment. In implementing the two algorithms, we precompute and store some of the values that are repeatedly used in the algorithm, such as the sets $W_{a,q}$ in the forward algorithm, for each $W \in \mathcal{X}$, $q \in Q$ and $a \in \Sigma$.

6.2 Abstracting Models of Web Applications

We consider web applications implemented on multiple servers. Only the server-side code of such web applications is abstracted. We assume a server accepts and processes a single session at a time. Concurrent sessions are executed by different servers. Each server is abstracted as a labeled deterministic finite-state automaton whose inputs are user-submitted *forms*; these include initiating a new session and ending the current session. These automata each have a special input symbol T that denotes elapse of one unit time (of course, we use discrete time and assume synchronized clocks). The states of these automata are labeled with atomic propositions denoting their properties.

We assume all servers are modeled by identical finite-state machines, since they run identical server code. We consider a system with only two servers. Our tool takes the automaton A describing the server logic as an input. It takes two such automata (distinguishing all input symbols of the two automata excepting T), and composes them by synchronizing on the T symbol and capturing their parallel execution until a failure occurs. This composition results in a 1-OPTS as described in Sect. 5. We assume that these two automata do not interfere with each other. The tool then takes a failure model as input. Only one server can fail, and its failure probabilities for each (state, input) pair are specified in the failure model. After a failure, the current session of the failed server is taken over by the good server until completed. During this period, the good server is executing its own session as well as that of the failed processor. The order in which these should be processed is also specified as part of the failure model. Once a good server completes the session of the failed server and its own current session, it will continue to accept one new session at a time and process it.

The correctness property to be verified is specified as a safety automaton whose input symbols are $2^{\mathcal{P}}$ where $\mathcal{P}$ are the atomic propositions labeling the nodes of the server automaton; actually $\mathcal{P}$ includes two sets of propositions corresponding to the two sessions and it also includes input symbols of the server automata. In the composed OPTS we abstract away the T transitions as these are not the actual user inputs. Using the approach given in Sect. 5, we obtain a 1-HPA and check its emptiness using both the forward and backward algorithms.

6.3 Experiment

In the experiment section, we abstracted several 1-HPA from server-end web applications using the methodology stated in Sect. 6.2, and our incorrectness property states that session one takes strictly more time when run normally without failures. We then ran verification algorithms on them. Experimental results are presented in Table 2. Each row represents a different 1-HPA obtained using a different combination of process, failure definition and threshold probability. For the rows where the answer is empty, the threshold probability x is 0.9, and for those with non-empty answer, it was 0.01 for the example "Larger HPA" and 0.1 for the others. Empty answer indicates there is no input sequence on which the probability of violating the correctness is greater x. The column "BKD/FWD" explicitly shows the ratio of the time taken by the backward algorithm to that of the improved forward algorithm. The table shows that the forward algorithm is any where from 8 to 4500 times faster than the backward algorithm. This supports our analysis in Sect. 3.3.

Table 2. Backward vs Forward on Time Efficiency. Recall notations in Sect. 3.3.

Web Application	n	m	s	w	Result	(CPU Time in ms)			L_f	L_b
						Forward	Backward	BKD/FWD		
eBay Auction	19	248	13	60	empty	16	140	**8.75**	4	2
					non-empty	0	125	**N/A**	2	2
On-line Shopping 1	86	1472	17	342	empty	78	4009	**51.40**	16	15
					non-empty	47	4165	**88.62**	13	14
On-line Shopping 2	87	1489	17	2051	empty	328	338273	**1031.32**	14	10
					non-empty	140	321362	**2295.44**	5	8
Medium HPA	191	4209	22	3402	empty	391	491719	**1257.59**	8	10
					non-empty	172	502984	**2924.33**	5	8
Larger HPA	399	797	12	3874	empty	47	210734	**4483.70**	1	5
					non-empty	156	221641	**1420.78**	2	7

7 Conclusions

In this paper, we have given a new faster algorithm (with better complexity) for checking non-emptiness of the language accepted by a 1-HPA on finite strings with probability greater than a given value. We can trivially extend this algorithm to the case of automata over infinite strings under Büchi as well as Muller acceptance conditions. We simply have to use a different algorithm for computing good witness sets. These extended algorithms can be used to verify correctness of 1-OPTSes when the correctness is specified by a Büchi or Muller automata. This allows us to verify general properties including liveness properties. Also, the faster algorithm can be extended to the case when the threshold is a non-strict threshold. We have also proved the undecidability result for 2-HPA and closed the gap between the known decidability/undecidability results. Possible future work include extending our tool to support the case when different web sessions can interfere, and also support verification of LTL formulas.

Acknowledgements. This research is supported by the NSF grants CCF-1319754, CNS-1314485 and CNS-1035914.

References

1. Baier, C., Größer, M.: Recognizing ω-regular languages with probabilistic automata. In: 20th IEEE Symposium on Logic in Computer Science, pp. 137–146 (2005)
2. Baier, C., Größer, M., Bertrand, N.: Probabilistic ω-automata. J. ACM **59**(1), 1–52 (2012)
3. Baier, C., Haverkort, B.R., Hermanns, H., Katoen, J.-P.: Efficient computation of time-bounded reachability probabilities in uniform continuous-time markov decision processes. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 61–76. Springer, Heidelberg (2004)
4. Chadha, R., Sistla, A.P., Viswanathan, M.: Probabilistic Büchi automata with non-extremal acceptance thresholds. In: Jhala, R., Schmidt, D. (eds.) VMCAI 2011. LNCS, vol. 6538, pp. 103–117. Springer, Heidelberg (2011)
5. Chadha, R., Sistla, A.P., Viswanathan, M.: Power of randomization in automata on infinite strings. Log. Methods Comput. Sci. **7**(3), 1–22 (2011)
6. Chadha, R., Sistla, A.P., Viswanathan, M., Ben, Y.: Decidable and expressive classes of probabilistic automata. In: Pitts, A. (ed.) FOSSACS 2015. LNCS, vol. 9034, pp. 200–214. Springer, Heidelberg (2015)
7. Gimbert, H., Oualhadj, Y.: Probabilistic automata on finite words: decidable and undecidable problems. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6199, pp. 527–538. Springer, Heidelberg (2010)
8. Größer, M.: Reduction Methods for Probabilistic Model Checking. Ph.D. thesis, TU Dresden (2008)
9. Paz, A.: Introduction to Probabilistic Automata. Academic Press, Orlando (1971)
10. PRISM – Probabilistic Symbolic Model Checker. http://www.prismmodelchecker.org
11. Rabin, M.O.: Probabilistic automata. Inf. Control **6**(3), 230–245 (1963)