Improving Pre-computation for Verification of Markov Decision Processes

Mohammadsadegh Mohagheghi
Department of Computer Science - Vali-e-asr Universityof Rafsanajn - Iran

Abstract: Probabilistic model checking of Markov Decision Processes is widely used in recent years. Numerical iterative techniques are used to compute the quantitative properties of the underlying models. Some graph-based methods can be used as pre-computation to improve the performance of these iterative algorithms. In this paper we compare the impact of forward vs. backward approaches for implementing these pre-computation algorithms, and we show that the backward reachability analysis can reduce the running time of these pre-computations by several orders of magnitude.

Keywords: Backward reachability analysis, Markov decision processes, Pre-computation, Probabilistic model checking,

I. INTRODUCTION

Probabilistic model checking is a formal method that is used for verification of qualitative and quantitative properties of systems with probabilistic behaviors [1], [2]. In this approach, a labeled transitions system is used to model the system [3]. Markov Decision Processes (MDPs), Discrete-time Markov Chains (DTMCs) and probabilistic timed automata (PTA) are examples of transition systems for modeling systems with probabilistic behaviors. Probabilistic Computation Tree Logic (PCTL) is a well-known formalism for specifying system properties that should be verified against the system model [3], [4]. A probabilistic model checker is software tool that automatically verifies the proposed properties (in PCTL) for the related model [1]. It reduces the problem of probabilistic model checking to the computation of the maximum or minimum reachability probabilities. PRISM [5] is the most prominent probabilistic model checker that is widely used in the recent years. In most cases, numerical computations are needed for calculating these reachability probabilities. Value iteration and policy iteration are two iterative numerical methods that are used in standard model checkers to compute the related reachability probabilities [1], [6]. Some graph based computations can be performed for accelerating iterative methods. For example a graph based pre-computation can detect those states for which the optimal reachability probability is exactly one or zero (which are shown by S^y and S^n respectively). These states can be disregarded in iterative computations and their values can be used for computation of the remaining states [1,7]. In some cases, these precomputation is time consuming but crucial for reward-based PCTL properties. For these class of properties the algorithm need to know all states in S^y while the expected values for other states may be infinity and the iterative method can not converge for these states[7].

In this paper we focus on maximum reachability probabilities. The value of these probabilities (and also minimum reachability probabilities) is usually needed in order to perform the verification of PCTL properties against MDPs [5], [8]. As the main contribution of our work, we use backward reachability techniques for accelerating pre-computation algorithms. Note that the running time of the best proposed method for pre-computation is in $O(n.\sqrt{n})$ where n is the size of the model [9]. We compare the forward and backward approaches for implementing the standard pre-computation algorithms and show that the backward technique reduces the time complexity of the standard algorithms for computing S^n and S^y sets.

We compare our backward technique with the forward on which is implemented in PRISM. Experimental results show a significant improvement in the performance of pre-computation when we use the backward technique. To the best of our knowledge, no previous work has compared the time complexity of forward and backward approaches for precomputation algorithms and the running time of them for real case studies.

The remainder of this paper is structured as follows: Section 2 reviews some related definitions. In section 3 we compare the forward and backward precomputation methods and in section 4, we present our experimental results and section 5 concludes the paper.

PRELIMINARIES

In this section we provide an overview of MDPs and some iterative algorithms for computing optimal reachability probabilities against MDPs. The notations and definitions are mainly from [2], [4], [5]. We use *Dist(S)* as the set of all discrete probability distributions over a finite set *S*, i.e., set of all functions $p : S$ -> [0,1] such that $\sum_{s\in S} p(s) = 1$.

Definition 1 (Markov Decision Process) A Markov Decision Process (MDP) is a tuple $M = (S, s_0, Act, \delta)$ where *S* is a finite set of states, $s_0 \in S$ is an initial state, *Act* is a finite set of actions and $\delta : S \times Act \rightarrow Dist(S)$ is a probabilistic transition function. The size of *M* which is shown as |*M*| is defined as the number of states of *M* plus the number of its transitions.

For every state $s \in S$ of an MDP *M* one or more actions of *Act* are defined as enabled actions. We define this set as $Act(s) = \{\alpha \in Act \mid \delta(s,\alpha) \text{ is defined}\}$. For $s \in S$ and $\alpha \in Act(s)$ we use $Post(s,\alpha)$ for the set of α successors of *s*, $Post(s)$ for all possible successors of *s* and *Pre(s)* for possible predecessors of *s* [1], [2]:

$$Post(s,\alpha) = \{s' \in S \mid \delta(s,\alpha,s') > 0\},$$

$$Post(s) = \cup_{\alpha\in Act(s)} Post(s,\alpha),$$

$$pre(s) = \{s' \in S \mid s \in Post(s')\}$$

There are two steps to define a transition from a state s. First, one enabled action $\alpha \in Act(s)$ is selected non-deterministically. Secondly, according to the probability distribution $\delta(s,\alpha)$ a successor state s' is chosen randomly. We use $\delta(s,\alpha)(s')$ to show the probability of a transition from *s* to s' by the action $\alpha \in Act(s)$. A discrete-time Markov chain (DTMC) is an MDP for which every state has exactly one enabled action. DTMCs are used to model systems with fully probabilistic behaviors [1], [2], [7]. A path in an MDP model shows a possible interaction between the model and the environment. It is defined as a non-empty (finite or infinite) sequence of states and actions of the form $s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \ldots$ where for every $i \geq 0$ we have $s_i \in S$ and $\alpha_i \in Act(s_i)$ and $s_{i+1} \in Post(s_i, \alpha_i)$. For a state $s \in S$ the set of all possible paths of the model that start from s are shown by $Path_s$. In a similar way, $Fpath_s$ is used for the set of all finite paths. We use π to show a path and $\pi(i)$ to show the (i+1)-th state of π, i.e., $\pi(i) = s(i)$. To resolve non-deterministic choices of an MDP *M* the notion of adversary (sometimes called policy or scheduler) is usually used [1], [8].

Definition 2 (Deterministic Adversary) A deterministic adversary of an MDP *M* is a function $\sigma : FPath \rightarrow Act$ that for every finite path $\pi = s_0 \xrightarrow{a0} s_1 \xrightarrow{a1} \ldots \xrightarrow{a_{i-1}} s_i$ selects an enabled action $a_i \in Act(s_i)$. An adversary is called memory-less if it depends only on the last state of the path. We use Adv_M to show the set of all deterministic adversaries of *M*.

De_nition 3 (Quotient DTMC) The quotient DTMC for an MDP $M = (S, s_0, Act, \delta)$ and a deterministic finite-memory adversary σ is the finite state DTMC $M^{\sigma} = (S, s_0, P)$ where S and s_0 are the same as in M and $P : S \times S \to [0,1]$ is a transition probability matrix and is defined as $P(s,s') = \delta(s, \sigma(s))(s')$ [5]. We use $Path_s^{\sigma}$ for the set of all (infinite) paths of the quotient DTMC M^{σ} starting in a state s.

- **Reachability Probabilities**

A main class of properties that are proposed in PCTL and are needed to be verified against MDPs includes reachability probabilities, i.e., the maximal or minimal probability of reaching a state in some target set $F \subset S$ when starting in a state $s \in S$:

$$P_s^{\min}(F) = \inf_{\sigma \in Adv_M} p_s^{\sigma}(F) \quad ,$$

$$P_s^{\max}(F) = \sup_{\sigma \in Adv_M} p_s^{\sigma}(F)$$

where the supremum and infimum range over all adversaries of M and $p_s^{\sigma}(F) = \Pr ob_s^{\sigma}(\{\pi \in Path_s^{\sigma} \mid \exists i.\pi(i) \in F\})$. The probability space $prob_s^{\sigma}$ is defined over the paths $path_s^{\sigma}$. More details about the definition of this probability space can be found in [1], [3]. Numerical computations are used to calculate these reachability probabilities. There are two general approaches to perform these computations:

- Linear programming (LP)
- Iterative methods

The first approach can obtain exact values of reachability probabilities but is unable to scale on large systems [1], [3]. Iterative methods on the other hand are used for practical systems but do not guarantee for convergence [10]. Value and policy iteration are two standard iterative methods for computing reachability probabilities. In practice most of probabilistic model checkers such as PRISM [5], LIQUOUR [4] and PAT[11] use iterative methods.

II. PRE-COMPUTATION

This step is used to partition the state space *S* into three sets:

- $S^y = \{s \in S \mid P_s^{\max}(F) = 1\}$
- $S^n = \{s \in S \mid P_s^{\max}(F) = 0\}$
- $S^? = S - (S^y \cup S^n)$

Pre-computation can reduce the size of states for quantitative reachability computations and may also accelerate the model checking process. Identifying the S^y and S^n sets of states usually improves the precision of reachability probabilities. These three sets can also be defined for minimum reachability probabilities [1], [6]. The significance of these pre-computations depends on the properties and the numerical methods that are used for quantitative verification. The computation of S^n is essential when linear programming is used for computing reachability probabilities [6]. The computation of these two sets is also essential for the convergence of iterative methods in the case of optimal expected reward properties. In the remainder of this section, we review the standard algorithms for computing the S^n and S^y sets. According to the data structure that is used to represent the information of the model, we consider two approaches for implementing these algorithms: the forward approach that uses adjacency lists (an extension of sparse matrixes [12]) as the data structure, and the backward approach that also uses the inverse adjacency list. In the forward approach, the information of the transitions are sorted according to the source states and the $Post(s,\alpha)$ set can be computed easily (each member of $Post(s,\alpha)$ can be computed in $O(1)$) but in general, the method has to consider all transitions in order to compute the $\Pr e(s)$ set (see [2] for more details).

In the backward approach, we restore the information of transitions while they are sorted according to their destination states and we have an efficient implementation for computing the $Pre(s)$ set and each member of this set can be computed in $O(1)$. We analyze the time complexity and the memory overhead of these two approaches for implementing pre-computation algorithms.

The sparse matrix technique uses two levels of indexing to store the information of an MDP [12]. Using this technique, the forward approach needs $4\times(|S|+1)+4\times(|Act|+1)+12\times|trans|$ 4 (|S| + 1) bytes where |S| is the number of states, |Act| is the number of actions and |$trans$| is the number of transitions of the MDP.

Pre-computation for S^n

Algorithm 1 explains the standard method for computing the set S^n. Starting from the set F, it uses a BFS to compute those states that can reach the goal set. In the k-th iteration of the loop, R' is the set of states that can reach to the F set in k steps and can not reach to it in less than k steps. The algorithm continues until reaching the fixed point; the point that no other states can be added to the R' set. The remaining states of S cannot reach the goal set and are in the S^n set.

```
Algorithm1 : Pre – computation for $S^n$
**Input** : An MDP $M = (S, s_0, Act, \delta)$, target set $F \subset S$
**Output** : The set $S^n = \{s \in S \mid P_s^{\max}(F) = 0\}$
1. $R := F$;
2. $R' := F$;
3. **do**
4.     $R' := Pre(R') - R$;
5.     $R := R \cup R'$;
6. **while** $R' \neq \phi$;
7. **return** $S / R$;
```

The worst case time complexity of Algorithm 1 is in $O(|S|\times|M|)$ in the case of forward approach, and it is linear in the size of the model in the case of the backward approach. The maximum number of iterations of the while loop is |S|, and in each iteration, the algorithm needs to compute the $Pre(R')$ set. In the case of the forward approach, the algorithm should consider all of the transitions of the model to compute the $Pre(R')$ set where it takes $O(|M|)$ time. On the other hand, in the case of the backward approach, for each R' set, the time for computing the $Pre(R')$ is equal to the number of incoming transitions to the states of R'. In this case, each transition of M is considered at most once, because each state $s \in S$ is added to R' at most once. As a result, the worst case running time of Algorithm 1 is equal to the size of M.

The backward approach uses an inverse adjacency list (that stores the information of transitions according to the destination states) to have an efficient access to the members of the $Pre(R')$ set. This data structure needs $4\times|S+1|+4\times|trans|$ extra bytes. It is usually less than 50 percent of the space for storing the information of the MDP. In the experimental results we compare the running time of the forward and backward approaches for several case studies.

Pre-computation for S^y

Pre-computation for S^y set is shown in Algorithm 2 [1, 5].

```
Algorithm2 : Pre-computation for $S^y$
**Input** : An MDP $M=(S,s_0,Act,\delta)$, target set $F\subset S$
**Output** : The set $S^y=\{s\in S \mid P_s^{\max}(F)=1\}$
1. $R := S$;
2. do
3.   $R' := R$;
4.   $R := F$;
5.   do
6.     $R'' := R$;
7.     $R := R'' \cup \{s\in S \mid \exists\alpha\in Act(s).(\forall s'\in S.(\delta(s,\alpha)(s')\in s>0 \rightarrow s'\in R'))\wedge$
8.         $(\exists s'\in R''.\delta(s,\alpha)(s')>0)\}$;
9.   while  $R\neq R''$;
10. while  $R\neq R'$;
11. return S / R;
```

It uses a double fixed point iterative method with a nested *while* loop for computing the set of S^y states [1]. Starting from *S*, the outer loop successively removes those states $s\in S$, for which $P_s^{\max}(F)$ is less than one. It induces a sequence of R_i sets, where $S=R_0\supset R_1 ...\supset R_i\supset R_{i+1}=S^y$. The inner loop starts from F and iteratively adds those states s' to R' for which $P_{s'}^{\max}(F)$ is one. For the remaining states $(S - R')$, we are sure that they do not belong to S^y. If the algorithm only uses the forward approach, it should consider all states and transitions of the model for computing the *R* set in the inner loop (line 7). In the worst case, the number of iterations of the outer loop is equal to $|S|$. It means that the worst case time complexity of Algorithm 4 is in $O(|S|^2\times|M|)$, if we use the forward approach for representing the model. On the other hand, we use the inverse adjacency list as the data structure for the backward approach in order to accelerate the computations of Algorithm 3. Line 7 of this algorithm computes the set of states $s\in S$ that have at least one action $\alpha\in Act(s)$ for which the following conditions hold:

- $\forall s'\in S.(s'\in Post(s,\alpha)\rightarrow s'\in R')$
- $\exists s'\in R''\cap Post(s,\alpha)$

Instead of considering every state $s\in S$, the algorithm can consider the source state of those transitions $(s,\alpha,s',p)\in M$ for which $Post(s,\alpha)\subseteq R'$ (first condition) and $s'\in R''$(second condition) hold. To do so, for each $s'\in R''$, the algorithm should have an efficient access to the set of transitions of the form

$(s,\alpha,s',p)\in M$. It can be achieved by an inverse adjacency list. In addition, for each state $s\in S$, the algorithm can mark those actions $\alpha\in Act(s)$ that $Post(s,\alpha)\subseteq S-R'$ holds. In this way, the first condition holds for every unmarked action. The reason is that for every $s'\in S-R'$ we are sure that $P^{\max}(s')<1$, which means the maximum reachability probability of s is less than one if it uses the action α .

Algorithm 3 shows the details of the backward approach for computing the S^y set. Although, it uses the same idea of Algorithm 2, it also uses (lines 8 and 16) the inverse adjacency list in order to have an efficient implementation of the backward approach. For each state $s'\in S$, the inverse adjacency list points to all pairs of states and actions of the form (s,α) that $s'\in Post(s,\alpha)$ holds.

```
Algorithm 3: Backward Pre-computation for $S^y$
Input : An MDP $M=(S,s_0,Act,\delta)$, target set $F\subset S$
Output : The set $S^y=\{s\in S \mid P_s^{\max}(F)=1\}$
1. $R := S$;
2. do
3.    $R' := R$;
4.    $R := F$;
5.    $Q := F$;
6.    while Q is not empty do
7.       remove a state t from Q;
8.       foreach transition (s, $\alpha$, t, p) of M do
9.          if $\alpha$ is not marked and $s\notin R$ then
10.            Add s to $R$;
11.            Add s to Q;
12.         end if;
13.      end for;
14.   end while;
15.   foreach $t\in R'-R$ do
16.      foreach transition (s, $\alpha$, t, p) of $M$ do
17.         mark a;
18.      endfor;
19.   endfor;
20. while $R'\neq R$;
21. return $R$;
```

The outer loop of Algorithm 3 is the same as the outer loop of Algorithm 2. The inner *while* loop (lines 6 to 14) computes the *R* sets and works like the inner loop of Algorithm 2. It uses *Q* to consider those states that are added to the *R* set. Using *Q* and the inverse adjacency list, it considers (in line 8) all states $s\in S$ for which the second condition holds. The for loop (lines 15-19) marks those actions α that lead to at least one state in *S* - *R* and cannot be used in the next iterations of the computation. Note that for each state $t \in S - R$ (and hence $t \in R' - R$), we are sure that $P^{\max}(t) < 1$ (it also holds after each termination of the inner while loop of Algorithm 2).

The maximal number of iterations of the outer *while* loop is equal to the number of states of *S*. In each iteration, the algorithm should consider the set of transitions that lead to any state $s \in R$, that it takes O(|M|) time. The overall running time of the for loop is in O(|M|), because the condition $t \in R' - R$ holds at most once for each $t \in S$. Thus, the worst case time complexity of Algorithm 4 is in $O(|S| \times |M|)$. Note that Algorithm 4 is another representation of Algorithm 3 that is proposed for clarifying the backward approach for computing the S^y set. The main contribution of this algorithm is to use the inverse adjacency list in order to access each transitions of *M* (lines 8 and 16) in *O*(1). In practice, the running times of the forward and backward algorithms for computing the S^y and S^n sets depend on the number of iterations of their loops. In the experimental results, we compare the number of iterations of these algorithms in both the forward and the backward approaches.

III. Experimental Results

We have implemented our methods within the PRISM model checker, and we used its sparse engine for this implementation. The current version of PRISM (4.4) supports value iteration, policy iteration, Gauss-Seidel, and modified policy iteration for the explicit engine and only value iteration for the sparse and MTBDD engines [12] as MDP solution methods. For better comparison, we implemented Gauss-Seidel, policy iteration and modified policy iteration for the sparse engine. The current version of PRISM has implemented pre-computation algorithms for BDD-based and sparse data-structures [12].

Table 1. Case studies and the size of S, S^y and S^n sets

Model	Parameter(s)	|S|	$\|S^y\|$	$\|S^n\|$	$\|S^?\|/\|S\|$
consensus(N,K)	6 , 15	274,548	87,278	1,418	67.69%
consensus(N,K)	6 , 45	814,548	255,038	1,418	68.69%
consensus(N,K)	8 , 5	399,488	169,666	4,690	56.36%
consensus(N,K)	8 , 15	1,157,248	476,546	4,690	58.42%
csma (N,K)	3 , 6	14,222,529	10,120,379	169,206	27.65 %
csma (N,K)	4 , 4	133,301,572	514,457	5,360,396	95.59%
Leader (N)	7	2,095,783	2,095,783	0	0%
Leader (N)	8	18,674,484	18,674,484	0	0%
Wlan collide(n,ttmx,col,k)	5 , 25 , 15 , 15	12,275,400	1,189,754	10,829,381	2.09 %
Wlan collide(n,ttmx,col,k)	6 , 25 , 10 , 10	20,011,416	4,609,402	15,204,714	0.98 %
Wlan(n, ttmx, k)	6, 1500, 5	9,651,878	1,117,396	8,482,095	0.54 %
Firewire(dd,dl)	2500, 36	1,850,010	773,582	132,303	51.03 %
Firewire(dd,dl)	5000, 36	3,790,010	1,616,082	132,303	53.87%
Firewire(dd,dl)	2500, 128	2,957,080	786,726	270,901	64.23%
Firewire(dd,dl)	2500, 128	6,047,080	1,629,226	270,901	68.58%
Zeroconf(N,K)	15 , 10	3,001,911	197,004	957,807	61.53 %
Zeroconf(N,K)	20 , 14	4,427,159	171,851	1,160,964	69.89 %
Zeroconf(N,K)	20 , 18	5,477,150	128,427	1,240,173	75.01 %

We used sparse data-structure for implementing the backward pre-compuation methods. We used six standard case studies of PRISM, including *wlan*, *zeroconf*, *firewire*, *csma*, *consensus* and *leader*, as have been used in previous related works [1], [4], [6], [8]. For more scalability of *consensus* and *csma* examples, we applied symmetric reduction [4] (which is supported by PRISM) before other computations.

Except for the *leader* example, used only for pre-computation comparison, other case studies are used for comparing the improved pre-computation and reachability probability computation methods. The experiences were executed on a corei7 2.8GHz processor with 8 GB of Ram running Ubuntu 14. Table 1 describes more details of these case studies. The first two columns show the model names and parameters. The third column shows the total number of states of each model, and the 4th and 5th columns show the number of states in S^y and S^n sets. The last column shows the ratio of states in $S^?$.

Running times of the backward and forward approaches for pre-computation

We first compare the running time of pre-computation for these case studies. Table 2 shows the running time of the forward and backward approaches for computing the S^y and S^n sets. The second and the 5th columns show the running time of the forward approaches for computing the S^n and the S^y sets and the 4th and the 7th columns show the running time for the backward approach for computing these sets. All times are in seconds. The results show that the backward approach for pre-computations performs faster than the forward approach by several orders of magnitude. The 3-th and 6th and 8th columns of the table show the number of iterations of the forward and backward approaches for these pre-comptations. For the S^n sets, the forward method usually needs hundreds or thousands of iterations, and in each iteration, it uses a large number of states of the model for its computation.

Table 2. Performance comparison of the forward and backward pre-computation approaches

Model	Forward S^n time	Forward S^n iterations	Backward S^n time	Forward S^y time	Forward S^y iterations	Backward S^y time	Backward S^y iterations
consensus(6,15)	1.1	817	0.01	196	66704	1.65	184
consensus(6,45)	2.7	2437	0.02	1293	564494	14.2	544
consensus(8,5)	2.5	369	0.05	230	16006	1.7	86
consensus(8,15)	5	1089	0.02	1297	119806	13.9	246
csma (3, 6)	1629	227	0.47	19543	1590	5.5	8
csma (4, 4)	58.9	172	0.11	378	818	0.5	6
Leader (7)	10.6	76	0.1	9.4	77	0.23	1
Leader (8)	61.2	90	0.16	54.5	91	1.5	1
Wlan_collide(5)	30.9	1740	0.01	47.2	3080	0.13	2
Wlan_collide(6)	79.6	3126	0.04	156.2	6152	0.17	2
Wlan(6, 1500, 5)	18.3	399	0.02	42.4	798	0.1	2
Firewire(2500,36)	1.48	376	0.02	2.15	2167	0.04	1
Firewire(5000,36)	1.61	376	0.03	4.16	4705	0.06	1
Firewire(2500,128)	1.86	468	0.05	1.9	2064	0.05	1
Firewire(5000,128)	1.76	468	0.05	3.36	4589	0.04	1
Zeroconf(15, 10)	3.87	140	0.07	72.6	2357	1.86	30
Zeroconf(20, 14)	5.96	156	0.1	106.48	2665	2.45	27
Zeroconf(20, 18)	7.5	172	0.13	117.42	2768	2.78	24

The backward approach for the S^n set performs a BFS and terminates after the first iteration. It considers each state at most once. For the S^y sets, the number of iterations of the forward approach is computed as the total number of iterations of the inner while loop. For the backward approach, it is defined as the number of iterations of the outer while loop of Algorithm 4. In both cases, the running time of each iteration is linear in the size of the model. For most cases, the forward approach for computing the S^y set needs thousands of iterations, while the backward approach usually terminates after a few number of iterations.

IV. Conclusion

In this paper, we proposed a backward approach for pre-computation algorithms that compute the S^n and S^y sets that are needed in probabilistic model checking. We also analyzed the memory consumption of the proposed methods. Experimental results for PRISM model checker show a considerable speed-up in the pre-computation algorithms. For future works, we plan to implement the backward methods in BDD-based data structure.

REFERENCES

[1] Forejt V, Kwiatkowska M, Norman G, Parker D. Automated verification techniques for probabilistic systems. InInternational School on Formal Methods for the Design of Computer, Communication and Software Systems 2011 Jun 13 (pp. 53-113). Springer, Berlin, Heidelberg.

[2] Baier, Christel, and Joost-Pieter Katoen. Principles of model checking. MIT press, 2008.

[3] Kwiatkowska, Marta, Gethin Norman, and David Parker. "Probabilistic Model Checking: Advances and Applications." In Formal System Verification, pp. 73-121. Springer, Cham, 2018.

[4] Ciesinski, Frank, Christel Baier, Marcus Größer, and Joachim Klein. "Reduction techniques for model checking Markov decision processes." In Quantitative Evaluation of Systems, 2008. QEST'08. Fifth International Conference on, pp. 45-54. IEEE, 2008.

[5] Klein, Joachim, Christel Baier, Philipp Chrszon, Marcus Daum, Clemens Dubslaff, Sascha Klüppelholz, Steffen Märcker, and David Müller. "Advances in probabilistic model checking with PRISM: variable reordering, quantiles and weak deterministic Büchi automata." International Journal on Software Tools for Technology Transfer (2017): 1-16.

[6] Kwiatkowska, Marta, David Parker, and Hongyang Qu. "Incremental quantitative verification for Markov decision processes." In Dependable Systems & Networks (DSN), 2011 IEEE/IFIP 41st International Conference on, pp. 359-370. IEEE, 2011.

[7] Nipkow, T. "Advances in probabilistic model checking." Software Safety and Security: Tools for Analysis and Verification 33, no. 126 (2012).

[8] Brázdil, Tomáš, Krishnendu Chatterjee, Martin Chmelik, Vojtěch Forejt, Jan Křetínský, Marta Kwiatkowska, David Parker, and Mateusz Ujma. "Verification of Markov decision processes using learning algorithms." In International Symposium on Automated Technology for Verification and Analysis, pp. 98-114. Springer, Cham, 2014.

[9] Chatterjee, Krishnendu, and Monika Henzinger. "Faster and dynamic algorithms for maximal end-component decomposition and related graph problems in probabilistic verification." In Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms, pp. 1318-1336. Society for Industrial and Applied Mathematics, 2011.

[10] Baier, Christel, Joachim Klein, Linda Leuschner, David Parker, and Sascha Wunderlich. "Ensuring the reliability of your model checker: Interval iteration for markov decision processes." In International Conference on Computer Aided Verification, pp. 160-180. Springer, Cham, 2017.

[11] Sun, Jun, Yang Liu, Jin Song Dong, and Jun Pang. "PAT: Towards flexible verification under fairness." In *International Conference on Computer Aided Verification*, pp. 709-714. Springer, Berlin, Heidelberg, 2009.

[12] Parker, David Anthony. "Implementation of symbolic model checking for probabilistic systems." PhD diss., University of Birmingham, 2003.

Biography: Mohammadsadegh Mohagheghi is a faculty member of department of Computer science in Vali-e-asr university of Rafsanjan - Iran. His research interest is Formal methods, software engineering and theory of computations.