Incremental Runtime Verification of Probabilistic Systems

Vojtěch Forejt[1], Marta Kwiatkowska[1], David Parker[2], Hongyang Qu[1], and Mateusz Ujma[1]

[1] Department of Computer Science, University of Oxford, Oxford, UK
[2] School of Computer Science, University of Birmingham, Birmingham, UK

Abstract. Probabilistic verification techniques have been proposed for runtime analysis of adaptive software systems, with the verification results being used to steer the system so that it satisfies certain Quality-of-Service requirements. Since systems evolve over time, and verification results are required promptly, efficiency is an essential issue. To address this, we present incremental verification techniques, which exploit the results of previous analyses. We target systems modelled as Markov decision processes, developing incremental methods for constructing models from high-level system descriptions and for numerical solution using policy iteration based on strongly connected components. A prototype implementation, based on the PRISM model checker, demonstrates performance improvements on a range of case studies.

1 Introduction

Probabilistic systems are prevalent in our daily life: physical devices may fail, communication media are lossy and protocols use randomisation. Formally verifying that such systems behave correctly and reliably requires quantitative approaches, such as *probabilistic model checking*, which can assure, e.g., "the web service successfully delivers a response within 5ms with probability at least 0.99".

It has recently been proposed to use these techniques for *runtime* verification of *adaptive* systems [2], where quantitative verification is used to steer a system such that it satisfies formally specified Quality-of-Service (QoS) requirements. The framework of [2] comprises a computer system exhibiting probabilistic behaviour, a monitoring module that observes its behaviour and a reconfiguration component, which issues it instructions. Requirements to be fulfilled are verified against a high-level model of its behaviour, which is parameterised using data from the monitoring module. The results of verification are then forwarded to the reconfiguration module, which directs the system accordingly.

As a real world example of the applicability of these techniques (which we will return to later), consider a network containing a dynamic set of devices in which joining devices establish a local IP address using the Zeroconf protocol. A suitable QoS requirement for the network would be to minimise the probability of nodes choosing conflicting IP addresses. Parameters influencing this probability

include the number of hosts in the network and the number of probes (query messages) that are broadcast before claiming a given IP address.

In this paper, our aim is to optimise the performance of runtime verification for probabilistic systems. Since the systems being verified change dynamically and the results of verification are needed promptly to steer the system, efficiency is essential. We consider *incremental* verification techniques, which exploit the results of previous analyses following a small change to the system being verified.

We target systems modelled as Markov decision processes (MDPs), a widely used model for systems exhibiting both probabilistic and nondeterministic behaviour. We present incremental techniques for the two main phases of probabilistic verification: *model construction*, which exhaustively constructs an MDP from a high-level model description, and *quantitative verification*, which applies numerical techniques to determine the correctness of a system requirement, formally specified in temporal logic. For the former, we propose a technique that infers all states that have to be visited in the incremental step. For the latter, we use policy iteration, optimised using a decomposition of the system into strongly connected components, and performed incrementally by re-using policies between verification runs. We have implemented our techniques in a prototype extension of the PRISM model checker [6], and illustrate the benefits of our approach on a set of benchmark models.

An extended version of this paper with additional details is available as [5].

Related work. Various techniques have been developed that use model checking at runtime; see [1] for a discussion and further references. There is also increasing interest in incremental model checking techniques. Of particular relevance here are those for probabilistic systems. Wongpiromsarn et al. [8] studied incremental model construction for increasing numbers of system components. In contrast, we focus on changes within a fixed set of components. Filieri et al. [4] presented efficient incremental verification for the simpler model of discrete-time Markov chains using parametric techniques, but their method is subject to an exponential blow up when applied to MDPs and does not handle structural model changes. Kwiatkowska et al. [7] proposed incremental methods for MDPs based on a decomposition into strongly connected components. We consider model changes at the modelling language description level, which [7] does not, and also permit changes in model structure, rather than just transition probabilities.

2 Incremental Model Construction

We first consider incremental techniques for model construction. In this paper, we work with systems specified in the PRISM modelling language, a textual formalism based on guarded commands. Our incremental techniques are designed to operate after relatively small runtime changes to the structure of the MDP. At the level of the modelling language description, we assume that these changes are made by altering *parameters*: constants from the model description whose value is not determined until runtime. We only consider changes in parameters that occur in *guards* of commands, which is a common scenario in practice.

```
4:   const int N; // number of abstract hosts
5:   const int K; // number of probes to send
     ...
20:  module host0
         ...
26:      // send probe
27:      [send] l=2 & x=2 & probes<K → (x'=0) & (probes'=probes + 1);
28:      // sent K probes and waited 2 seconds
29:      [] l=2 & x=2 & probes=K → (l'=3) & (probes'=0) & (coll'=0) & (x'=0);
         ...
33:  endmodule
```

Fig. 1. Fragments of a PRISM model of the Zeroconf protocol.

For simplicity, we do not consider parameters that affect transition probabilities values. Such changes could be handled using the techniques described in [7].

We work with an explicit-state implementation. Building an MDP from a PRISM model requires a systematic state-space exploration, the most costly parts of which are the evaluation of all commands in each state, and subsequent creation of new states found. The basic idea of our incremental method is to infer the subset of states needing to be rebuilt, reducing the number of commands to be re-evaluated. Full details of the algorithm are in [5]; here we give an informal description using an example.

Figure 1 shows a fragment of a PRISM model for the previously mentioned Zeroconf protocol example. We assume that parameter N is fixed and K varies. We consider a scenario where we have already built a model $\mathcal{M}_1$ for $K{=}k_1$ and need to construct a new model $\mathcal{M}_2$ for $K{=}k_2$. We start by identifying guards that contain K: we find them in lines 27 and 29; for convenience, call them g_1, g_2 (in the example, these commands have probability 1, but this is not a limitation of our approach). In each guard, there is a variable compared to the parameter K, in this case *probes* in both guards. The key observation is that, to build model M_2, we do not need to re-evaluate commands in all states: it is sufficient to examine states from $\mathcal{M}_1$ that satisfied g_1, g_2 for $K{=}k_1$ but no longer do for $K{=}k_2$, and states that now satisfy g_1, g_2 for $K{=}k_2$. To find such states, we need to compute bounds on the values of *probes* for $K{=}k_1$ and $K{=}k_2$.

For the majority of PRISM models (whose guards involve just linear arithmetic), we can accomplish this using an SMT solver. In fact, for many common classes of expressions (such as this example) we can extract the bounds directly. In our example, for $K{=}k_1$, we obtain $probes \in [0, k_1)$ for g_1 and $probes \in [k_1, k_1]$ for g_2. For $K{=}k_2$, we get $probes \in [0, k_2)$ for g_1 and $probes \in [k_2, k_2]$ for g_2. Taking the intersection identifies states in $\mathcal{M}_1$ that satisfy g_1 for both $K{=}k_1$ and $K{=}k_2$, giving $probes \in [0, k_1) \cap [0, k_2)$. The union, i.e. $probes \in [0, k_1) \cup [0, k_2)$, gives all states of $\mathcal{M}_1$ that may satisfy g_1. The states that need to be re-evaluated in the context of g_1 are then found by performing a state space exploration from states with variable $probes \in ([0, k_1) \cup [0, k_2)) \setminus ([0, k_1) \cap [0, k_2))$. The same process is subsequently repeated for guard g_2. The efficiency of performing these steps can be improved considerably by keeping the state space of $\mathcal{M}_1$ sorted, with respect to variable *probes*, and using binary search when looking for the states satisfying a given bound. Finally, we remove from model $\mathcal{M}_2$ any states that are no longer reachable from its initial state using standard reachability algorithms.

3 Incremental Quantitative Verification

Next, we consider incremental techniques for quantitative verification of MDPs, the key part of which is the numerical computation of either the *minimum* or *maximum* probability of reaching a set of target states, over all possible *adversaries* of the MDP (an adversary represents one way of resolving all nondeterminism in the model). Common methods for computing these probabilities are *value iteration*, which is an approximate iterative numerical solution method, and *policy iteration*, which analyses a sequence of adversaries with increasing/decreasing probabilities.

Previous incremental verification techniques for MDPs [7] were based on the use of value iteration, applied to a decomposition of the the model into its strongly connected components (SCCs) [3]. These methods are not directly applicable to the scenarios we consider in this paper since, unlike [7], we permit structural changes to be made to the MDP. Instead, we propose an SCC-based version of policy iteration. Like [7], we first decompose the MDP into its SCCs and determine their topological ordering; next, we solve each SCC separately, working through them backwards according to the topological ordering. Here, however, we use policy iteration to compute the probabilities for each SCC.

For incremental verification, the key benefit from using policy iteration comes when we select the initial adversary used to start the computation. For this, rather than taking the usual approach of selecting an arbitrary adversary, we adapt the optimal adversary from the previous run of verification. Let $\mathcal{M}_1$ be the previous MDP and $\mathcal{M}_2$ be the new one. An adversary for an MDP resolves the nondeterminism in each of its states. To construct the initial adversary for solving $\mathcal{M}_2$, we identify all the states that are present in both $\mathcal{M}_1$ and $\mathcal{M}_2$, and which have the same nondeterministic choices in both models; we then re-use the choices made by the old adversary for $\mathcal{M}_1$ in the one for $\mathcal{M}_2$.

4 Experiments

We implemented our techniques in an extension of PRISM [6], using its explicit-state model checking engine, and evaluated them on 4 existing benchmarks: *zeroconf*, *mer*, *consensus* and *firewire*. Details of these examples and additional results are in [5]. We ran a series of verification instances, varying a particular model parameter. Table 1 shows the parameter ranges, the sizes of the resulting models, and the total time required to perform model construction and verification for all models, in both a non-incremental and incremental fashion. The overhead on memory usage is negligible for both algorithms, and thus omitted.

For incremental model construction, we obtained speed-ups in all cases, up to a 10-fold improvement for the *mer* example. The key factor for performance is the number of states added between each verification run. For incremental verification due to space restrictions we do not show results for original algorithms, which we outperform for each case study. Incremental policy iteration is quicker than SCC-based policy iteration in some, but not all, cases. The best results are those for the *zeroconf* example, with a 2-fold speedup. The performance of this phase is mostly influenced by the structure of the state space.

Model			Time (s)			
Name [parameters]	Parameter values	States [10^3]	Original model constr.	Incremental model constr.	SCC-based policy iteration	Incremental policy iteration
zeroconf $[N, K]$	10,1-5	32-496	15.9	12.3	10.6	8.5
	10,10-20	3002-5812	859.7	320.2	1859.1	1329.2
	60000,1-5	32-496	16.2	11.6	49.7	50.9
	60000,10-20	3002-5812	853.9	313.7	9333.9	4218.4
mer $[N]$	1-100	8-592	429.5	44.3	70.5	65.5
	200-300	1183-1774	2352.4	192.5	400.6	369.7
	400-500	2364-2955	4375.7	358.3	695.9	683.3
consensus $[N, K]$	2,1-40	1-5	0.8	0.4	33.6	23.2
	2,80-120	10-15	2.3	0.9	1235.8	900.1
	4,1-20	12-20	15.5	4.8	1029.1	666.5
firewire [*deadline*]	1000-1050	369-398	62.3	10.3	38.5	37.7
	2000-2050	970-1000	160.5	25.7	99.7	97
	3000-3050	1571-1601	265.7	42.4	174	181.6

Table 1. Performance comparison for incremental techniques.

5 Conclusions

We have described ongoing work to develop incremental verification techniques for Markov decision processes, aimed at improving the efficiency of runtime methods for systems with probabilistic behaviour. Future directions include evaluating presented techniques on a deployed adaptive system and improving system reconfiguration using policies obtained from model checking.

Acknowledgements. The authors are part supported by ERC Advanced Grant VERIWARE, EU FP7 project CONNECT and EPSRC project EP/F001096/1.

References

1. Calinescu, R.: When the requirements for adaptation and high integrity meet. In: Proc. 8th Workshop on Assurances for Self-Adaptive systems. pp. 1–4 (2011)
2. Calinescu, R., Grunske, L., Kwiatkowska, M., Mirandola, R., Tamburrelli, G.: Dynamic QoS management and optimisation in service-based systems. IEEE Transactions on Software Engineering 37(3), 387–409 (2011)
3. Ciesinski, F., Baier, C., Größer, M., Klein, J.: Reduction techniques for model checking Markov decision processes. In: Proc. QEST'08. pp. 45–54. IEEE (2008)
4. Filieri, A., Ghezzi, C., Tamburrelli, G.: Run-time efficient probabilistic model checking. In: Proc. ICSE'11. pp. 341–350. ACM, New York, NY, USA (2011)
5. Forejt, V., Kwiatkowska, M., Parker, D., Qu, H., Ujma, M.: Incremental runtime verification of probabilistic systems. Tech. Rep. RR-12-05, Department of Computer Science, University of Oxford (2012)
6. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: Proc. CAV'11. LNCS, vol. 6806, pp. 585–591. Springer (2011)
7. Kwiatkowska, M., Parker, D., Qu, H.: Incremental quantitative verification for Markov decision processes. In: Proc. DSN-PDS'11. pp. 359–370. IEEE (2011)
8. Wongpiromsarn, T., Ulusoy, A., Belta, C., Frazzoli, E., Rus, D.: Incremental temporal logic synthesis of control policies for robots interacting with dynamic agents. In: Proc. IROS'12 (2012), to appear