Automated Learning of Probabilistic Assumptions for Compositional Reasoning

Lu Feng, Marta Kwiatkowska, and David Parker

Oxford University Computing Laboratory, Parks Road, Oxford, OX1 3QD, UK

Abstract. Probabilistic verification techniques have been applied to the formal modelling and analysis of a wide range of systems, from communication protocols such as Bluetooth, to nanoscale computing devices, to biological cellular processes. In order to tackle the inherent challenge of scalability, compositional approaches to verification are sorely needed. An example is assume-guarantee reasoning, where each component of a system is analysed independently, using assumptions about the other components that it interacts with. We discuss recent developments in the area of automated compositional verification techniques for probabilistic systems. In particular, we describe techniques to automatically generate probabilistic assumptions that can be used as the basis for compositional reasoning. We do so using algorithmic learning techniques, which have already proved to be successful for the generation of assumptions for compositional verification of non-probabilistic systems. We also present recent improvements and extensions to this work and survey some of the promising potential directions for further research in this area.

1 Introduction

Formal verification is an approach to establishing mathematically rigorous guarantees about the correctness of real-life systems. Particularly successful are fully-automated techniques such as *model checking* [14], which is based on the systematic construction and analysis of a finite-state model capturing the possible states of the system and the transitions between states that can occur over time. Desired properties of the system are formally specified, typically using temporal logic, and checked against the constructed model.

Many real-life systems, however, exhibit *stochastic* behaviour, which analysis techniques must also take into account. For example, components of a system may be prone to failures or messages transmitted between devices may be subjected to delays or be lost. Furthermore, randomisation is a popular tool, for example as a symmetry breaker in wireless communication protocols, or in probabilistic security protocols for anonymity or contract-signing.

Probabilistic verification is a set of techniques for formal modelling and analysis of such systems. *Probabilistic model checking*, for instance, involves the construction of a finite-state model augmented with probabilistic information, such as a Markov chain or probabilistic automaton. This is then checked against properties specified in probabilistic extensions of temporal logic, such as PCTL [25,5].

This permits *quantitative* notions of correctness to be checked, e.g. "the probability of an airbag failing to deploy within 0.02 seconds is at most 0.0001". It also provides other classes of properties, such as performance or reliability, e.g. "the expected time for a successful transmission of a data packet".

As with any formal verification technique, one of the principal challenges for probabilistic model checking is *scalability*: the complexity of real-life systems quickly leads to models that are orders of magnitude larger than current verification techniques can support. A promising direction to combat this problem is to adopt *compositional* reasoning methods, which split the work required to verify a system into smaller sub-tasks, based on a decomposition of the system model. A popular approach is the *assume-guarantee* paradigm, in which individual system components are verified under *assumptions* about their environment. Once it has been verified that the other system components do indeed satisfy these assumptions, proof rules can be used to combine individual verification results, establishing correctness properties of the overall system.

In this paper, we discuss some recent developments in the area of automated compositional verification techniques for probabilistic systems, focusing on *assume-guarantee* verification techniques [29,21] for *probabilistic automata* [35,36], a natural model for compositional reasoning. When aiming to develop fully automated verification techniques, an important question that arises is how to devise suitable assumptions about system components. In the context of non-probabilistic systems, a breakthrough in the practical applicability of assume-guarantee verification came about through the adoption of algorithmic *learning* techniques to generate assumptions [15,32]. In recent work [20], we showed how learning could also be successfully used to generate the *probabilistic assumptions* needed for the compositional verification of probabilistic systems. We give an overview of this approach and discuss the relationship with the non-probabilistic case. We also present some recent improvements and extensions to the work and discuss directions for future research.

Paper structure. The rest of the paper is structured as follows. Section 2 gives a summary of probabilistic model checking and probabilistic assume-guarantee reasoning. Section 3 describes learning-based generation of assumptions for non-probabilistic systems, and Section 4 discusses how this can be adapted to the probabilistic case. Section 5 presents experimental results for some further recent developments. Section 6 concludes and identifies areas of future work.

2 Probabilistic Verification

2.1 Modelling and Verification of Probabilistic Systems

We begin by giving a brief overview of automated verification techniques for probabilistic systems, in particular probabilistic model checking. Then, in the next section, we discuss some of the challenges in adding compositional reasoning to these techniques and describe one particular approach to this: a probabilistic assume-guarantee framework.

Probabilistic models. There are a variety of different types of models in common use for probabilistic verification. The simplest are discrete-time Markov chains (DTMCs), whose behaviour is entirely *probabilistic*: every transition in the model is associated with a probability indicating the likelihood of that transition occurring. In many situations, though, it is also important to model *nondeterministic* behaviour. In particular, and crucially for the work described here, this provides a way to capture the behaviour of several parallel components.

Probabilistic automata (PAs) [35,36], and the closely related model of Markov decision processes (MDPs), are common formalisms for modelling systems that exhibit both probabilistic and nondeterministic behaviour. We focus here on PAs, which subsume MDPs and are particularly well suited for compositional reasoning about probabilistic systems [35]. In each state of a PA, several possible actions can be taken and the choice between them is assumed to be resolved in a nondeterministic fashion. Once one of these actions has been selected, the subsequent behaviour is specified by the probability of making a transition to each other state, as for a discrete-time Markov chain. The actions also serve another role: synchronisation. PAs can be composed in parallel, to model the concurrent behaviour of multiple probabilistic processes. In this case, synchronisation between components occurs by taking actions with the same label simultaneously.

Another important aspect that distinguishes probabilistic models is the notion of *time* used. For DTMCs, PAs and MDPs, time is assumed to proceed in discrete steps. In some cases, a more fine-grained model of time may be needed. One possibility is to use continuous-time Markov chains (CTMCs), an extension of DTMCs in which real-valued delays occur between each transition, modelled by exponential distributions. Other, more complex models, which incorporate probabilistic, nondeterministic and real-time behaviour, include probabilistic timed automata (PTAs), continuous-time Markov decision processes (CTMDPs) and interactive Markov chains (IMCs). In this paper, we focus entirely on PAs and thus a discrete model of time. Note, though, that several of the techniques for verifying PTAs reduce the problem to one of analysing a finite state PA; thus, the techniques described in this paper are still potentially applicable.

Probabilistic model checking. The typical approach to specifying properties to be verified on probabilistic systems is to use extensions of temporal logic. The basic idea is to add operators that place a bound on the probability of some event's occurrence. So, whereas in a non-probabilistic setting we might use an LTL formula such as $\Box\neg \mathit{fail}$ asserting that, along all executions of the model, *fail* never occurs, in the probabilistic case we might instead use $\langle\Box\neg \mathit{fail}\rangle_{\geqslant 0.98}$. Informally, this means the probability of *fail* never occurring is at least 0.98.

In fact, for models such as probabilistic automata, which exhibit both nondeterministic and probabilistic behaviour, formalising these properties requires care. This is because it is only possible to define the probability of an event's occurrence in the absence of any nondeterminism. The standard approach is to use the notion of *adversaries* (also called strategies, schedulers or policies), which represent one possible way of resolving all nondeterminism in a PA. For an adversary σ, we denote by $Pr_M^\sigma(G)$ the probability of event G when the nondeterminism in M is resolved by σ, and we say that a PA M satisfies a property $\langle G\rangle_{\geqslant p_G}$, denoted $M \models \langle G\rangle_{\geqslant p_G}$, if $Pr^{\sigma}_{M}(G) \geqslant p_G$ for all adversaries σ. Equivalently, M satisfies $\langle G\rangle_{\geqslant p_G}$ when $Pr^{\min}_{M}(G) \geqslant p_G$, where $Pr^{\min}_{M}(G)$ denotes the minimum probability, over all adversaries, of G.

A useful class of properties for PAs are *probabilistic safety properties.* These take the form $\langle G\rangle_{\geqslant p_G}$, as described above, where G is a *safety property* (a set of "good" model traces, defined by a set of "bad" prefixes, finite traces for which any extension is *not* a "good" trace). More precisely, we assume here that the "bad" traces form a regular language and, for efficiency, we represent this using a deterministic finite automaton (DFA), denoted G^{err}, rather than, say, temporal logic. Probabilistic safety properties can capture a wide range of useful properties of probabilistic automata, including for example:

– "event A always occurs before event B with probability at least 0.9"
– "the probability of a system failure occurring is at most 0.02"
– "the probability of terminating within k time-units is at least 0.75"

Many other classes of properties are also in common use for probabilistic verification. First, we can generalise $\langle G\rangle_{\geqslant p_G}$ to $\langle \phi\rangle_{\sim p}$ where $\sim \in \{<, \leqslant, \geqslant, >\}$ and ϕ is any ω-regular language (subsuming, for example, the temporal logic LTL). We can also consider *branching-time* probabilistic temporal logics such as PCTL or PCTL* [25,5], as opposed to the *linear-time* properties considered so far. Yet another possibility is to augment the PA with costs or rewards and consider, for example, the expected total cumulated reward or the expected long-run average reward. Finally, we mention *multi-objective* properties, which can be used to express trade-offs between multiple quantitative (e.g. probabilistic linear-time) properties across the set of adversaries of a PA. Multi-objective probabilistic model checking (see e.g. [18,21]) is a key ingredient in the implementation of the probabilistic assume-guarantee framework discussed in this paper.

Algorithmically, probabilistic model checking for PAs reduces in most cases to a combination of graph-based algorithms and numerical computation [16,1]. For the latter, either approximate iterative calculations (e.g. value iteration) can be used or a reduction to linear programming (LP). Prior to this, it is often necessary to construct a *product* PA for analysis. For example, for the probabilistic safety properties described above, we would construct the synchronous product of the PA to be verified and the DFA representing the safety property [29]. Tool support for verifying PAs (or MDPs) is also available: several probabilistic model checkers have been developed and are widely used. The most popular of these is PRISM [26]; others include LiQuor [13], RAPTURE [27] and ProbDiVinE [4].

Example 1. In Figure 1, we show a simple example (taken from [29]) comprising two components, each modelled as a PA. Component M_1 represents a controller that powers down devices. When it identifies a problem (modelled by the action *detect*), it sends two messages, represented by actions *warn* and *shutdown*, respectively. However, with probability 0.2 it will fail to issue the *warn* message first. The second component, M_2 represents a device to be powered down by M_1. It expects to receive two messages, *warn* and *shutdown.* If the first of these is absent, it will only shut down correctly 90% of the time.

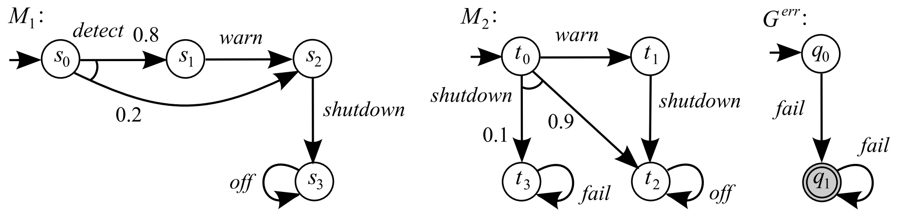

Fig. 1. Example, from [29]: two PAs M_1, M_2 and the DFA G^{err} for a safety property G; we have that $M_1 \,\|\, M_2$ satisfies probabilistic safety property $\langle G \rangle_{\geqslant 0.98}$.

We consider the parallel composition $M_1 \,\|\, M_2$ of the two devices and check a simple probabilistic safety property: "with probability 0.98, action *fail* never occurs". Formally, this is captured as $\langle G \rangle_{\geqslant 0.98}$, where G is the safety property "action *fail* never occurs", represented by the DFA G^{err} also shown in Figure 1 (this is over the alphabet $\{fail\}$, so any trace apart from the empty string is accepted as an error trace). It can be seem that the maximum probability of *fail* occurring in $M_1 \,\|\, M_2$ is $0.2 \cdot 0.1 = 0.02$. So, we have that $M_1 \,\|\, M_2 \models \langle G \rangle_{\geqslant 0.98}$.

2.2 Compositional Reasoning for Probabilistic Systems

Verification techniques for probabilistic systems can be expensive, both in terms of their time and space requirements, making scalability an important challenge. A promising way to address this is to perform verification in a *compositional* manner, decomposing the problem into sub-tasks which analyse each component separately. However, devising compositional analysis techniques for probabilistic systems requires considerable care, particularly for models such as probabilistic automata that exhibit both probabilistic and nondeterministic behaviour [35].

Assume-guarantee reasoning. In the case of *non-probabilistic* models, a popular approach to compositional verification is the use of *assume-guarantee* reasoning. This is based on checking assume-guarantee *triples* of the form $\langle A \rangle\, M_i \,\langle G \rangle$, with the meaning "whenever component M_i is part of a system satisfying the *assumption* A, then the system is *guaranteed* to satisfy property G". Proof rules can then be established that combine properties of individual components to show correctness properties of the overall system.

To give a concrete example, we adopt the framework used in [15,32]. Components M_i and assumption A are labelled transition systems and G is a safety property. Then, $\langle A \rangle\, M_i \,\langle G \rangle$ has the meaning that $A \,\|\, M_i \models G$. We also say that a component M_j satisfies an assumption A if all traces of M_j are included in A, denoted $M_j \sqsubseteq A$. For components M_1 and M_2, the following proof rule holds:

$$\frac{\begin{array}{c} M_1 \sqsubseteq A \\ \langle A \rangle\, M_2 \,\langle G \rangle \end{array}}{M_1 \,\|\, M_2 \models G}$$

Thus, verifying property G on the combined system $M_1 \,\|\, M_2$ reduces to two separate (and hopefully simpler) checks, one on M_1 and one on $A \,\|\, M_2$.

Assume-guarantee for probabilistic systems. There are several ways that we could attempt to adapt the above assume-guarantee framework to probabilistic automata. A key step is to formalise the notion of an assumption A about a (probabilistic) component M_i. The two most important requirements are: (i) that it is *compositional*, allowing proof rules such as the one above to be constructed; and (ii) that the premises of the proof rule can be checked *efficiently*.

Unfortunately, the most natural probabilistic extension of the trace inclusion preorder $\sqsubseteq$ used above, namely *trace distribution inclusion* [35] is *not* compositional [34]. One way to address this limitation is to restrict the ways in which components can be composed in parallel; examples include the switched probabilistic I/O automata model of [12] and the synchronous parallel composition of probabilistic Reactive Modules used in [2]. Another is to characterise variants of the trace distribution inclusion preorder that *are* compositional, e.g. [35,30]. However, none of these preorders can be checked efficiently, limiting their applicability to automated compositional verification.

Other candidates for ways to formalise the relationship between a component PA and its assumption include the notions of (strong and weak) probabilistic simulation and bisimulation [35,36]. There are a number of variants, most of which are compositional. Unfortunately, for the weak variants, efficient methods to check the relations are not yet known and, for the strong variants, although relatively efficient algorithms exist, the relations are usually too fine to yield suitably small assumptions. Other work in this area includes [17], which uses a notion of probabilistic contracts to reason compositionally about systems with both stochastic and nondeterministic behaviour. Automating the techniques in an implementation has not yet been attempted.

In this paper, we focus on the probabilistic assume-guarantee framework of [29]. This makes no restrictions on the PAs that can be analysed, nor the way that they are composed in parallel; instead it opts to use a less expressive form for assumptions, namely probabilistic safety properties. An assume-guarantee triple now takes the form $\langle A\rangle_{\geqslant p_A}\, M_i\, \langle G\rangle_{\geqslant p_G}$ where M_i is a PA and $\langle A\rangle_{\geqslant p_A}, \langle G\rangle_{\geqslant p_G}$ are two probabilistic safety properties; the former is a *probabilistic assumption*, the latter a property to be checked. This is interpreted as follows: $\langle A\rangle_{\geqslant p_A}\, M_i\, \langle G\rangle_{\geqslant p_G}$ is true if, for all adversaries σ of M_i such that $Pr^{\sigma}_{M_i}(A) \geqslant p_A$ holds, $Pr^{\sigma}_{M_i}(G) \geqslant p_G$ also holds. Crucially, verifying whether this is true reduces to a multi-objective model checking problem [18,29], which can be carried out efficiently by solving an LP problem. The proof rule used earlier now becomes:

$$\frac{\begin{array}{c} M_1 \models \langle A\rangle_{\geqslant p_A} \\ \langle A\rangle_{\geqslant p_A}\, M_2\, \langle G\rangle_{\geqslant p_G} \end{array}}{M_1 \,\|\, M_2 \models \langle G\rangle_{\geqslant p_G}} \quad (\text{ASym})$$

thus reducing the problem of checking property $\langle G\rangle_{\geqslant p_G}$ on $M_1 \,\|\, M_2$ to two smaller sub-problems: (i) verifying a probabilistic safety property on M_1; and (ii) checking an assume-guarantee triple for M_2.

In [29], this proof rule, along with several others, are proved and then used to perform compositional verification on a set of large case studies. This includes

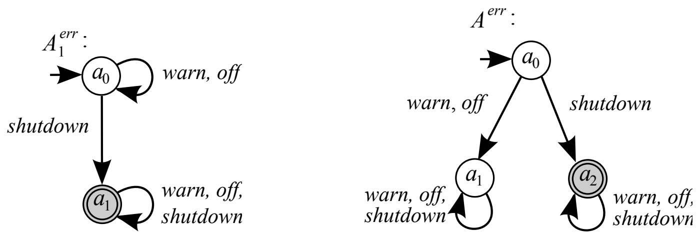

Fig. 2. DFAs for two learning-generated assumptions to verify the system from Figure 1 compositionally with rule (ASym); A (right) is a valid assumption, A_1 (left) is not.

cases where non-compositional verification is either slower or infeasible. In recent work [21], this framework is extended to permit the use of more expressive assumptions (and properties): *quantitative multi-objective properties*, essentially Boolean combinations of probabilistic safety, ω-regular and expected total reward properties. An example of an expressible assumption is "with probability 1, component M_i always eventually sends a message and the expected time to do so is at most 10 time-units." Again, the framework is implemented through multi-objective model checking and applied to several case studies.

Example 2. Recall the PAs M_1, M_2 and probabilistic safety property $\langle G\rangle_{\geqslant 0.98}$ from Example 1, for which we stated that $M_1 \parallel M_2 \models \langle G\rangle_{\geqslant 0.98}$. This property can also be checked in a compositional manner using the rule (ASym). We use an assumption about the behaviour of M_1 which can be stated informally as "with probability at least 0.8, *warn* occurs before *shutdown*". This takes the form of a probabilistic safety property $\langle A\rangle_{\geqslant 0.8}$ where the DFA for A is shown on the right-hand side of Figure 2. To perform compositional verification, we perform two separate checks, $M_1 \models \langle A\rangle_{\geqslant 0.8}$ and $\langle A\rangle_{\geqslant 0.8}\, M_2\, \langle G\rangle_{\geqslant 0.98}$, both of which hold.

3 Learning Assumptions for Compositional Verification

The ideas behind assume-guarantee verification for non-probabilistic systems have a long history [28]. Making these techniques work in practice, however, is a challenge. In particular, deciding how to break down a system into its components and devising suitable assumptions about the behaviour of those components initially proved difficult to automate. A breakthrough in this area came with the observation of [15] that *learning* techniques, such as Angluin's L* algorithm [3], could be used to automate the process of generating assumptions. In this section, we give a short description of L* and its application to automatic compositional verification of non-probabilistic systems.

The L* algorithm. L* [3] is a learning algorithm for generating a *minimal* DFA that accepts an unknown regular language $\mathcal{L}$. It uses the *active* learning model, interacting with a *teacher* to which it can pose *queries*. There are two kinds of these: *membership queries*, asking whether a particular word t is contained in $\mathcal{L}$; and *equivalence queries*, asking whether a conjectured automaton

A accepts exactly $\mathcal{L}$. Initially, L* poses a series of membership queries to the teacher, maintaining results in an *observation table*. After some time (more precisely, when the table is *closed* and *consistent*; see [3] for details), L* generates a conjecture A and submits this as an equivalence query. If the answer to this query is "no", the teacher must provide a counterexample c, in either positive form ($c \in \mathcal{L} \backslash \mathcal{L}(A)$) or negative form ($c \in \mathcal{L}(A) \backslash \mathcal{L}$). The algorithm then resumes submission of membership queries until the next conjecture can be generated.

The total number of queries required by L* is polynomial in the size of the minimal DFA required to represent the language $\mathcal{L}$ being learnt. Furthermore, the number of equivalence queries is bounded by the number of states in the automaton since at most one new state is added at each iteration.

Learning assumptions with L*. In [15], it was shown how L* could be adapted to the problem of automatically generating assumptions for assume-guarantee verification of labelled transition systems. This is done by phrasing the problem in a language-theoretic setting: the assumption A to be generated is a labelled transition system, whose set of (finite) traces forms a regular language.

The approach works by using the notion of the *weakest assumption* [22] as the target language $\mathcal{L}$. Intuitively, assuming the non-probabilistic proof rule from Section 2.2, the weakest assumption is the set of all possible traces of a process that, when put in parallel with M_2, do not violate the property G. Thus, the membership query used by the teacher checks, for a trace t, whether $t \,\|\, M_2 \models G$, where t denotes a transition system comprising the single trace t. Both this and the required equivalence queries can be executed automatically by a model checker. An important observation, however, is that, in practice, this is not usually the language actually learnt. The algorithm works in such a way that conjectured assumptions generated as L* progresses may be sufficient to either prove or refute the property G, allowing the procedure to terminate early.

A variety of subsequent improvements and extensions to the basic technique of [15] were proposed (see e.g. [32] for details) and the approach was successfully applied to several large case studies. It tends to perform particularly well when the size of a generated assumption and the size of its alphabet remain small. The work has sparked a significant amount of interest in the area in recent years. There have been several attempts to improve performance, including symbolic implementations [31] and optimisations to the use of L* [9]. Others have also devised alternative learning-based methods, for example by reformulating the assumption generation problem as one of computing the smallest finite automaton separating two regular languages [23,11], or using the CDNF learning algorithm to generate implicit representations of assumptions [10].

4 Learning Probabilistic Assumptions

As mentioned in Section 2.2, there are several different possibilities for the type of assumption used to perform probabilistic assume-guarantee verification. We focus on the use of probabilistic safety properties, as in [29]. Although these have limited expressivity, an advantage is that the generation of such assumptions can be automated by adapting the L*-based techniques developed for non-probabilistic compositional verification [15,32].

This approach was proposed in [20] and shown to be applicable on several large case studies. In this section, we give a high-level overview of the approach and describe the key underlying ideas; for the technical details, the reader is referred to [20]. The basic setting is as outlined in Section 2.2: we consider the assume-guarantee proof rule (ASym) from [29], applied to check that the parallel composition of two PAs M_1 and M_2 satisfies a property $\langle G\rangle_{\geqslant p_G}$. To do this, we need an assumption $\langle A\rangle_{\geqslant p_A}$, which will be generated automatically.

Probabilistic and non-probabilistic assumptions. The first key point to make is that, although we are required to learn a *probabilistic assumption*, i.e. a probabilistic safety property $\langle A\rangle_{\geqslant p_A}$, we can essentially reduce this task to the problem of learning a *non-probabilistic* assumption, i.e. the corresponding safety property A. The reasoning behind this is as follows.

We need the probabilistic assumption $\langle A\rangle_{\geqslant p_A}$ to be such that both premises of the proof rule (ASym) hold: (i) $M_1 \models \langle A\rangle_{\geqslant p_A}$; and (ii) $\langle A\rangle_{\geqslant p_A} M_2 \langle G\rangle_{\geqslant p_G}$. However, if (i) holds for a particular value of p_A, then it also holds for any lower value of p_A. Conversely, if (ii) holds for some p_A, it then must hold for any higher value. Thus, given a safety property A, we can determine an appropriate probability bound p_A (if one exists) by finding the lowest value of p_A (if any) such that (ii) holds and checking it against (i). Alternatively, we can find the highest value of p_A such that (i) holds (this is just $Pr^{\min}_{M_1}(A)$ in fact) and seeing if this suffices for (ii). A benefit of the latter is that, even if $M_1 \,\|\, M_2 \models \langle G\rangle_{\geqslant p_G}$ cannot be shown to be true with this particular assumption, we still obtain a *lower bound* on $Pr^{\min}_{M_1\|M_2}(G)$. Furthermore, with an additional simple check, an *upper bound* can also be generated (see [20] for details).

Adapting the L* algorithm. Next, we describe how we adapt the L*-based approach of [15,32] for generating non-probabilistic assumptions to our setting. The underlying idea beind the use of L* in [15,32] is the notion of *weakest assumption*: this will always exist and, if the property G being verified is true, will permit a compositional verification. It is used as the target language for L* and forms the basis of the membership and equivalence queries. In practice, however, this language is often not the one that is finally generated since intermediate conjectured assumptions may suffice, either to show that the property is true or that it is false.

An important difference in the probabilistic assume-guarantee framework of [29] is that it is *incomplete*, meaning that, even if the property $\langle G\rangle_{\geqslant p_G}$ holds, there may be no probabilistic assumption $\langle A\rangle_{\geqslant p_A}$ for which the rule (ASym) can be used to prove the property correct. So, there can be no equivalent notion of weakest assumption to be used as a target language. However, we can adopt a similar approach whereby we use L* to generate a sequence of conjectured assumptions A and, for each one, potentially show that $\langle G\rangle_{\geqslant p_G}$ is either true or false. Furthermore, as described above, each assumption A yields a lower and

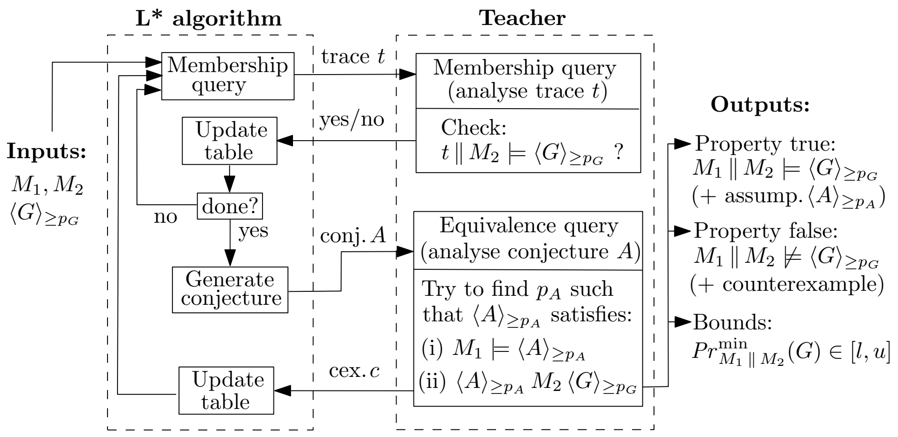
L* algorithm
Teacher
Inputs:
M_1, M_2
$\langle G \rangle_{\geq p_G}$
Membership query
trace t
Membership query (analyse trace t)
yes/no
Update table
Check: $t \parallel M_2 \models \langle G \rangle_{\geq p_G}$?
done?
no
yes
conj. A
Generate conjecture
Equivalence query (analyse conjecture A)
Try to find p_A such that $\langle A \rangle_{\geq p_A}$ satisfies:
(i) $M_1 \models \langle A \rangle_{\geq p_A}$
(ii) $\langle A \rangle_{\geq p_A} M_2 \langle G \rangle_{\geq p_G}$
cex. c
Update table
Outputs:
Property true: $M_1 \parallel M_2 \models \langle G \rangle_{\geq p_G}$ (+ assump. $\langle A \rangle_{\geq p_A}$)
Property false: $M_1 \parallel M_2 \not\models \langle G \rangle_{\geq p_G}$ (+ counterexample)
Bounds: $Pr^{\min}_{M_1 \parallel M_2}(G) \in [l, u]$

Fig. 3. Overview of probabilistic assumption generation [20], using an adaption of L*: generates assumption $\langle A \rangle_{\geqslant p_A}$ for verification of $M_1 \parallel M_2 \models \langle G \rangle_{\geqslant p_G}$ using rule (ASym).

an upper bound on $Pr^{\min}_{M_1 \| M_2}(G)$. We can retain these values as the learning algorithm progresses, keeping the highest lower bound and lowest upper bound discovered to report back to the user.[1] This means that, even if the algorithm terminates early (without concluding that $\langle G \rangle_{\geqslant p_G}$ is true or false), it produces useful *quantitative* information about the property of interest.

In order to produce conjectures, L* needs answers to membership queries about whether certain traces t should be in the language being generated. For this, we use the following check: $t \parallel M_2 \models \langle G \rangle_{\geqslant p_G}$, which can be seen as an analogue of the corresponding one for the non-probabilistic case. Intuitively, the idea is that if, under a single possible behaviour t of M_1, M_2 satisfies the property, then t should be included in the assumption A. There may be situations where this scheme leads to an assumption that cannot be used to verify the property. This is because it is possible that there are multiple traces which do not violate property $\langle G \rangle_{\geqslant p_G}$ individually but, when combined, cause $\langle G \rangle_{\geqslant p_G}$ to be false. In practice, though, this approach seems to work well in most cases.

A final aspect of L* that needs discussion is *counterexamples*. In [15,32], when the response to an equivalence query is “no”, a counterexample in the form of a trace is returned to L*. In our case, there are two differences in this respect. Firstly, a counterexample may constitute multiple traces; this is because the results of the model checking queries executed by the teacher yield *probabilistic counterexamples* [24], which comprise multiple paths (again, see [20] for precise details). Secondly, situations may arise where no such counterexample can be generated (recall that there is no guarantee that an assumption can eventually be created). In this case, since no trace can be returned to L*, we terminate the learning algorithm, returning the current tightest bounds on $Pr^{\min}_{M_1 \| M_2}(G)$ that have been computed so far.

[1] Note that the sequence of assumptions generated is *not* monotonic, e.g. it does *not* yield a sequence of increasing lower bounds on $Pr^{\min}_{M_1 \| M_2}(G)$.

The learning loop. We summarise in Figure 3 the overall structure of the L*-based algorithm for generating probabilistic assumptions. The left-hand side shows the basic L* algorithm. This interacts, through queries, with the teacher, shown on the right-hand side. The teacher responds to queries as described above. Notice that the equivalence query, which analyses a particular conjectured assumption A, has four possible outcomes: the first two are when A can be used to show either that $M_1 \| M_2 \models \langle G \rangle_{\geqslant p_G}$ or $M_1 \| M_2 \not\models \langle G \rangle_{\geqslant p_G}$; the third is when a counterexample (comprising one or more traces) is passed back to L*; and the fourth is when no counterexample can be generated so the algorithm returns the tightest bounds computed so far for $Pr^{\min}_{M_1 \| M_2}(G)$.

Example 3. Figure 2 shows the two successive conjunction assumptions generated by the learning loop, when applied to the PAs and property of Example 1. The first conjecture A_1 does not permit (ASym) to be applied but a counterexample can be found. L* then generates the conjecture A, which we know from Example 2 *does* allow compositional verification that $M_1 \| M_2 \models \langle G \rangle_{\geqslant p_G}$.

5 Experimental Results

The approach outlined in the previous section was successfully used in [20] to automatically generate probabilistic assumptions for several large case studies. Furthermore, these assumptions were much smaller than the corresponding components that they represented, leading to gains in performance. In this section, we present some recent extensions and improvements to that work.

5.1 A New Case Study: Mars Exploration Rovers

First, we present an application of the techniques to a new case study, based on a module from the flight software for JPL's Mars Exploration Rovers (MER). Compositional verification of a non-probabilistic model of this system was performed previously in [33]. The module studied is a *resource arbiter*, which controls access to various shared resources between a set of user threads, each of which performs a different application on the rover.

The arbiter also enforces priorities between resources, granting and rescinding access rights to users as required. For example, it is considered that communication is more important than driving; so, if a communication request is received while the rover is driving, the arbiter will rescind permission to use the drive motors in order to grant permission for use of the rover's antennas.

Our model adds the possibility of faulty behaviour. When the arbiter sends a *rescind* message to a user thread, there is a small chance of the message being lost in transmission. We also add information about the likelihood of a user thread requesting a given type of resource in each cycle of the system's execution. We are interested in a mutual exclusion property which checks that permission for communication and driving is not granted simultaneously by the arbiter. More precisely, we verify a probabilistic safety property relating to "the minimum probability that mutual exclusion is not violated within k cycles of

Case study [parameters]		Component sizes		Compositional (L*)				Compositional (NL*)				Non-comp.
		$\lvert M_2 \otimes G^{err} \rvert$	$\lvert M_1 \rvert$	$\lvert A^{err} \rvert$	MQ	EQ	Time	$\lvert A^{err} \rvert$	MQ	EQ	Time	Time
client-server1 [N]	3	81	16	5	99	3	**6.8**	6	223	4	7.7	0.02
	5	613	36	7	465	5	**21.6**	8	884	5	26.1	0.04
	7	4,733	64	9	1,295	7	484.6	10	1,975	5	**405.9**	0.08
client-serverN [N]	3	229	16	5	99	3	**6.6**	6	192	3	7.4	0.04
	4	1,121	25	6	236	4	**26.1**	7	507	4	33.1	0.12
	5	5,397	36	7	465	5	**191.1**	8	957	5	201.9	0.28
consensus [N R K]	2 3 20	391	3,217	6	149	5	24.2	7	161	3	**14.9**	108.1
	2 4 4	573	431,649	12	2,117	8	413.2	12	1,372	5	**103.4**	2.59
	3 3 20	8,843	38,193	11	471	6	438.9	15	1,231	5	**411.3**	>24h
sensor network [N]	1	42	72	3	17	2	**3.5**	4	31	2	3.9	0.03
	2	42	1,184	3	17	2	**3.7**	4	31	2	4.0	0.25
	3	42	10,662	3	17	2	**4.6**	4	31	2	4.8	2.01
mer [N R]	2 2	961	85	4	113	3	**9.0**	7	1,107	5	28.9	0.09
	2 5	5,776	427,363	4	113	3	**31.8**	7	1,257	5	154.4	1.96
	3 2	16,759	171	4	173	3	**210.5**	–	–	–	mem-out	0.42

Fig. 4. Performance comparison of the L*- and NL*-based methods for rule (ASym).

system execution". The model comprises N user threads $U_1, \ldots, U_N$ and the arbiter ARB which controls R shared resources (in the full system model, N is 11 and R is 15). We perform compositional verification using the rule (ASym), decomposing the system into two parts: $M_1 = U_1 \| U_2 \| \cdots \| U_N$ and $M_2 = ARB$.

5.2 A Comparison of Learning Methods: L* versus NL*

Next, we investigate the use of an alternative learning algorithm to generate probabilistic assumptions. Whereas L* learns a minimal DFA for a regular language, the algorithm NL* [7] learns a minimal *residual finite-state automaton* (RFSA). RFSAs are a subclass of nondeterministic finite automata. For the same regular language $\mathcal{L}$, the minimal RFSA that accepts $\mathcal{L}$ can be exponentially more succinct than the corresponding minimal DFA. In fact, for the purposes of probabilistic model checking, the RFSA needs to be determinised anyway [1]. However, the hope is that the smaller size of the RFSA may lead to a faster learning procedure. NL* works in a similar fashion to L*, making it straightforward to substitute into the learning loop shown in Figure 3.

We added NL* to our existing implementation from [20], which is based on an extension of PRISM [26] and the `libalf` [6] learning library. We then compared the performance of the two learning algorithms on a set of five case studies. The first four are taken from [20]: client-server benchmark models from [32] incorporating failures in one or all clients (*client-server1* and *client-serverN*), Aspnes & Herlihy's randomised consensus algorithm (*consensus*) and a sensor network exhibiting message losses (*sensor network*). The fifth example is the MER model from above (*mer*). Experiments were run on a 1.86GHz PC with 2GB RAM and we imposed a time-out of 24 hours.

Figure 4 compares the performance of the L*-based and NL*-based methods to generate probabilistic assumptions for the rule (ASym). The "Component sizes" columns give the state space of the two components, M_1 and M_2, in each model; for M_2, this also includes the automaton for the safety property G being checked. For each method, we report the size of the learnt assumption

Case study		Component sizes		(ASYM)		(ASYM-N)		Non-comp.
[parameters]		$\lvert M_2 \otimes G^{err} \rvert$	$\lvert M_1 \rvert$	$\lvert A^{err} \rvert$	Time (s)	$\lvert A^{err} \rvert$	Time (s)	Time (s)
client-serverN [N]	6	25,801	49	–	mem-out	8	40.9	0.7
	7	123,053	64	–	mem-out	24	164.7	1.7
mer [N R]	3 5	224,974	1,949,541	–	mem-out	4	29.8	48.2
	4 5	7,949,992	6,485,603	–	mem-out	4	122.9	mem-out
	5 5	265,559,722	17,579,131	–	mem-out	4	3,903.4	mem-out

Fig. 5. Performance comparison of the rule (ASYM) and the rule (ASYM-N).

A^{err} (DFA or RFSA), the number of membership queries (MQ) and equivalence queries (EQ) needed, and the total time (in seconds) for learning. We also give the time for non-compositional verification using PRISM.

With the exception of one model, both algorithms successfully generated a correct (and small) assumption in all cases. The results show that the L*-based method is faster than NL* in most cases. However, on several of the larger models, NL* has better performance due to a smaller number of equivalence queries. NL* needs more membership queries, but these are less costly. We do not compare the execution time of our prototype tool with the (highly-optimised) PRISM in detail. But, it is worth noting that, for two of the *consensus* models, compositional verification is actually faster than non-compositional verification.

5.3 Learning Multiple Assumptions: Rule (ASYM-N)

Lastly, in this section, we consider an extension of the probabilistic assumption generation scheme of Section 4, adapting it to the proof rule (ASYM-N) of [29]. This is motivated by the observation that, for several of the case studies in the previous section, scalability is limited because one of the two components comprises several sub-components. As the number of sub-components increases, model checking becomes infeasible due to the size of the state space. The (ASYM) proof rule allows decomposition of the system into more than 2 components:

$$\frac{\begin{array}{c}\langle \mathtt{true} \rangle\, M_1\, \langle A_1 \rangle_{\geqslant p_1} \\ \langle A_1 \rangle_{\geqslant p_1}\, M_2\, \langle A_2 \rangle_{\geqslant p_2} \\ \dots \\ \langle A_{n-1} \rangle_{\geqslant p_{n-1}}\, M_n\, \langle G \rangle_{\geqslant p_G}\end{array}}{\langle \mathtt{true} \rangle\, M_1 \,\|\, \dots \,\|\, M_n\, \langle G \rangle_{\geqslant p_G}} \quad \text{(ASYM-N)}$$

For the earlier MER case study, for instance, we can now decompose the system into $N+1$ components: the N user threads $U_1, U_2, \dots, U_N$ and the arbiter ARB.

Adapting our probabilistic assumption generation process to (ASYM-N) works by learning assumptions for the rule in a recursive fashion, with each step requiring a separate instantiation of the learning algorithm for (ASYM), as is done for the non-probabilistic version of a similar rule in [32]. Experimental results are presented in Figure 5 for two of the case studies from Section 5.2. The results demonstrate that, using the rule (ASYM-N), we can successfully learn small assumptions and perform compositional verification in several cases where the rule (ASYM) runs out of memory. Furthermore, in two instances, (ASYM-N) permits verification of models which cannot be checked in a non-compositional fashion.

6 Conclusions and Future Work

We have discussed recent progress in the development of automated compositional verification techniques for probabilistic systems, focusing on the assume-guarantee framework of [29,21] for probabilistic automata. We also described how the verification process can be automated further using learning-based generation of the assumptions needed to apply assume-guarantee proof rules and described some recent improvements and extensions to this work.

There are a variety of possible directions for future research in this area. One is to extend our techniques for learning probabilistic assumptions to the assume-guarantee framework in [21], which additionally includes ω-regular and expected reward properties. Here, the ω-regular language learning algorithms of [19,8] may provide a useful starting point. There are also possibilities to enhance the underlying compositional verification framework. This includes developing efficient techniques to work with richer classes of probabilistic assumption and extending the approach to handle more expressive types of probabilistic models, such as those that incorporate continuous-time behaviour.

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

References

1. de Alfaro, L.: Formal Verification of Probabilistic Systems. Ph.D. thesis, Stanford University (1997)
2. de Alfaro, L., Henzinger, T., Jhala, R.: Compositional methods for probabilistic systems. In: Proc. CONCUR'01. LNCS, vol. 2154. Springer (2001)
3. Angluin, D.: Learning regular sets from queries and counterexamples. Information and Computation 75(2), 87–106 (1987)
4. Barnat, J., Brim, L., Cerna, I., Ceska, M., Tumova, J.: ProbDiVinE-MC: Multi-core LTL model checker for probabilistic systems. In: Proc. QEST'08 (2008)
5. Bianco, A., de Alfaro, L.: Model checking of probabilistic and nondeterministic systems. In: Proc. FSTTCS'95. LNCS, vol. 1026, pp. 499–513. Springer (1995)
6. Bollig, B., Katoen, J.P., Kern, C., Leucker, M., Neider, D., Piegdon, D.: libalf: The automata learning framework. In: Proc. CAV'10. LNCS, vol. 6174, pp. 360–364. Springer (2010)
7. Bollig, B., Habermehl, P., Kern, C., Leucker, M.: Angluin-style learning of NFA. In: Proc. IJCAI'09. pp. 1004–1009. Morgan Kaufmann Publishers Inc. (2009)
8. Chaki, S., Gurfinkel, A.: Automated assume-guarantee reasoning for omega-regular systems and specifications. In: Proc. NFM'10. pp. 57–66 (2010)
9. Chaki, S., Strichman, O.: Optimized L*-based assume-guarantee reasoning. In: Proc. TACAS'07. pp. 276–291 (2007)
10. Chen, Y.F., Clarke, E., Farzan, A., Tsai, M.H., Tsay, Y.K., Wang, B.Y.: Automated assume-guarantee reasoning through implicit learning. In: Proc. CAV'10. LNCS, vol. 6174, pp. 511–526. Springer (2010)
11. Chen, Y.F., Farzan, A., Clarke, E., Tsay, Y.K., Wang, B.Y.: Learning minimal separating DFAs for compositional verification. In: Proc. TACAS'09 (2009)
12. Cheung, L., Lynch, N., Segala, R., Vaandrager, F.: Switched probabilistic I/O automata. In: Proc. ICTAC'04. LNCS, vol. 3407. Springer (2004)

13. Ciesinski, F., Baier, C.: Liquor: A tool for qualitative and quantitative linear time analysis of reactive systems. In: Proc. QEST'06. pp. 131–132. IEEE CS Press (2006)
14. Clarke, E., Grumberg, O., Peled, D.: Model Checking. The MIT Press (2000)
15. Cobleigh, J., Giannakopoulou, D., Pasareanu, C.: Learning assumptions for compositional verification. In: Proc. TACAS'03. LNCS, vol. 2619. Springer (2003)
16. Courcoubetis, C., Yannakakis, M.: Markov decision processes and regular events. IEEE Transactions on Automatic Control 43(10), 1399–1418 (1998)
17. Delahaye, B., Caillaud, B., Legay, A.: Probabilistic contracts: A compositional reasoning methodology for the design of stochastic systems. In: Proc. ACSD'10. pp. 223–232. IEEE CS Press (2010)
18. Etessami, K., Kwiatkowska, M., Vardi, M., Yannakakis, M.: Multi-objective model checking of Markov decision processes. LMCS 4(4), 1–21 (2008)
19. Farzan, A., Chen, Y.F., Clarke, E., Tsay, Y.K., Wang, B.Y.: Extending automated compositional verification to the full class of omega-regular languages. In: Proc. TACAS'08. LNCS, vol. 4963, pp. 2–17. Springer (2008)
20. Feng, L., Kwiatkowska, M., Parker, D.: Compositional verification of probabilistic systems using learning. In: Proc. 7th International Conference on Quantitative Evaluation of Systems (QEST'10). pp. 133–142. IEEE CS Press (2010)
21. Forejt, V., Kwiatkowska, M., Norman, G., Parker, D., Qu, H.: Quantitative multi-objective verification for probabilistic systems. In: Proc. TACAS'11 (2011)
22. Giannakopoulou, D., Pasareanu, C., Barringer, H.: Component verification with automatically generated assumptions. ASE 12(3), 297–320 (2005)
23. Gupta, A., McMillan, K., Fu, Z.: Automated assumption generation for compositional verification. Formal Methods in System Design 32(3), 285–301 (2008)
24. Han, T., Katoen, J.P., Damman, B.: Counterexample generation in probabilistic model checking. IEEE Transactions on Software Engineering 35(2), 241–257 (2009)
25. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Aspects of Computing 6(5), 512–535 (1994)
26. Hinton, A., Kwiatkowska, M., Norman, G., Parker, D.: PRISM: A tool for automatic verification of probabilistic systems. In: Proc. TACAS'06 (2006)
27. Jeannet, B., D'Argenio, P., Larsen, K.: Rapture: A tool for verifying Markov decision processes. In: Proc. CONCUR'02 Tools Day. pp. 84–98 (2002)
28. Jones, C.: Tentative steps towards a development method for interfering programs. ACM Transactions on Programming Languages and Systems 5(4), 596–619 (1983)
29. Kwiatkowska, M., Norman, G., Parker, D., Qu, H.: Assume-guarantee verification for probabilistic systems. In: Proc. TACAS'10. LNCS, vol. 6105, pp. 23–37 (2010)
30. Lynch, N., Segala, R., Vaandrager, F.: Observing branching structure through probabilistic contexts. SIAM Journal on Computing 37(4) (2007)
31. Nam, W., Madhusudan, P., Alur, R.: Automatic symbolic compositional verification by learning assumptions. FMSD 32(3), 207–234 (2008)
32. Pasareanu, C., Giannakopoulou, D., Bobaru, M., Cobleigh, J., Barringer, H.: Learning to divide and conquer: Applying the L* algorithm to automate assume-guarantee reasoning. Formal Methods in System Design 32(3), 175–205 (2008)
33. Pasareanu, C., Giannakopoulou, D.: Towards a compositional SPIN. In: Proc. SPIN'06. pp. 234–251 (2006)
34. Segala, R.: A compositional trace-based semantics for probabilistic automata. In: Proc. CONCUR'95. LNCS, vol. 962, pp. 234–248. Springer (1995)
35. Segala, R.: Modelling and Verification of Randomized Distributed Real Time Systems. Ph.D. thesis, Massachusetts Institute of Technology (1995)
36. Segala, R., Lynch, N.: Probabilistic simulations for probabilistic processes. Nordic Journal of Computing 2(2), 250–273 (1995)