Compositional Reverification of Probabilistic Safety Properties for Large-Scale Complex IT Systems

Radu Calinescu[1], Shinji Kikuchi[2] and Kenneth Johnson[1]

[1] Department of Computer Science
University of York, Deramore Lane, York YO10 , UK
{radu.calinescu,kenneth.johnson}@york.ac.uk
[2] Fujitsu Laboratories Limited
4-1-1 Kamikodanaka, Nakahara-ku, Kawasaki, Kanagawa 211-8588, Japan
skikuchi@jp.fujitsu.com

Abstract. Compositional verification has long been regarded as an effective technique for extending the use of symbolic model checking to large, component-based systems. This paper explores the effectiveness of the technique for large-scale complex IT systems (LSCITS). In particular, we investigate how compositional verification can be used to reverify LSCITS safety properties efficiently after the frequent changes that characterise these systems. We identify several *LSCITS change patterns*—including component failure, join and choice—and propose an approach that uses *assume-guarantee compositional verification* to reverify *probabilistic safety properties* compositionally in scenarios associated with these patterns. The application of this approach is illustrated using a case study from the area of cloud computing.

1 Introduction

A variant of symbolic model checking termed *compositional verification* has proved particularly effective in extending the applicability of formal verification to large, component-based systems [1, 2, 14, 21, 25, 27, 29]. This technique analyses the components of a system independently, and derives global system properties through verifying a composition of its component-level properties. The state-transition models verified in both steps of the technique are often orders of magnitude smaller than a monolithic model of the same system.

However, traditional compositional verification is less effective for a class of IT systems of growing practical importance, namely *large-scale complex IT systems* (LSCITS). LSCITS are affected by regular component failures, joins and departures, and by frequent modifications in environment and requirements [9, 28, 31]. This continual change has the effect of quickly invalidating the result of any compositional verification, which is based on a set of models that are accurate for only a short period of time.

Recent research has used (quantitative) model checking techniques at runtime, to ensure that IT systems continue to comply with their requirements after changes similar to those experienced by LSCITS [5, 6, 12, 17, 20, 24]. The approach proposed in this work involves monitoring the running system, and verifying an updated model of its behaviour whenever an environment or system change is identified. If the runtime verification confirms that the system continues to comply with its requirements, no further action is required. Otherwise, the verification results are used to guide a *self-adaptation* process through which the system adjusts its parameters to reinstate the compliance with its requirements. While the approach proved effective in applications ranging from dynamic power management [4, 11, 12] to quality-of-service optimisation in service-based systems [5, 6, 17], none of the systems in these applications was an LSCITS.

This paper presents the results of our work to integrate techniques from the areas of compositional verification and runtime model checking. We envisage that a successful integration of the two types of techniques will extend the benefits of our recent work on runtime model checking [5, 6, 8, 10, 12] to larger systems, and ultimately to certain classes of LSCITS.

The rest of the paper is organised as follows. In Section 2, we overview existing compositional verification techniques, focusing on the probabilistic assume-guarantee approach used in our work. In Section 3, we identify several *LSCITS change patterns*, and we explain how *assume-guarantee compositional verification* can be used reverify compliance with safety properties *incrementally* in scenarios associated with these patterns. A running example from the area of cloud computing is used to illustrate these results throughout this section. Finally, Section 4 summarises the contributions of our work and discusses a number of future research directions.

2 From monolithic to compositional verification

2.1 Running example

We will illustrate the concepts and verification techniques discussed in the paper using the three-tier software service whose deployment on cloud infrastructure is depicted in Figure 1. Several instances of each of the three components (or *functions*) of this service—Web, Application and Database—are run on different virtual machines (VMs) that are located on four physical servers.

Note that while the system in Figure 1 is not a large system, it can be easily scaled up to become one by using standard cloud infrastructure functionality to increase the number of servers, virtual machines and "function" instances, potentially by many orders of magnitude. Indeed, some of the discussion later in the paper assumes this to be the case. Likewise, running a scaled-up version of the service across multiple cloud data-centres (an increasingly common practice for some users of cloud infrastructure [16, 33]) can augment the system with LSCITS-specific characteristics.

2.2 Background

Model checking a component-based system involves verifying if the parallel composition of $n > 1$ interdependent models of the system components and environ-

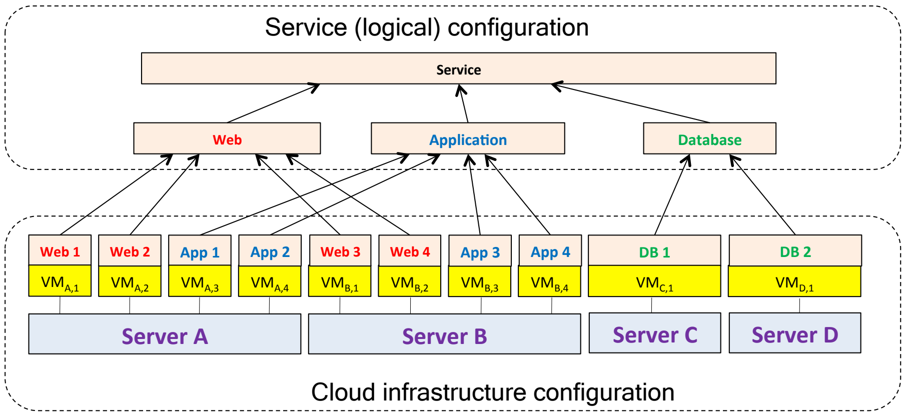


Fig. 1. Three-tier service deployed on cloud infrastructure

ment M_1, M_2, ..., M_n satisfies a set of requirements R:

$$M_1 \parallel M_2 \parallel \ldots \parallel M_n \models R. \tag{1}$$

Each model M_i, $1 \leq i \leq n$, comprises a finite set of states S_i and a *state transition relation* $T_i \subseteq S_i \times S_i$ with the property that for every state $s \in S_i$ there is at least a state $s' \in S_i$ such that $(s, s') \in T_i$. The states in S_i correspond to possible states of the modelled real-world component or environment element, and T_i encodes the possible transitions between these states. A *labelling function* $L_i : S_i \to AP_i$ is used to associate each state with a set of *atomic propositions* that are true in that state, and $s_i^0 \in S_i$ denotes the initial state of M_i. Formally,

$$M_i = (S_i, s_i^0, T_i, L_i), \quad 1 \leq i \leq n, \tag{2}$$

is termed a *Kripke structure* over the set of atomic propositions AP_i.

The requirements R are formulae defined over $\cup_{i=1}^{n} AP_i$ and expressed in extensions of propositional logic that support reasoning about the timing of events in the system. These *temporal logics* are used to specify desirable sequences of transitions between system states without referring to time explicitly. In particular, temporal logic formulae can specify *safety properties* (e.g., "the server failure state is *never* entered") and *liveness properties* (e.g., "the VM is *eventually* migrated to an operational server").

These concepts are illustrated in Figure 2, which depicts a model of a single physical server from our running example from Figure 1. We assume that the server has initially N_{DISK} disks, N_{CPU} CPUs and N_{MEM} memory blocks that are operational, but that each of these components can fail over time. To ensure that such component failures do not lead to failures of the VMs running on the server, the server is provided with a hardware failure detection mechanism. When multiple failures of components of the same type make the server "unsafe", this mechanism triggers the migration of the VMs to another physical server.

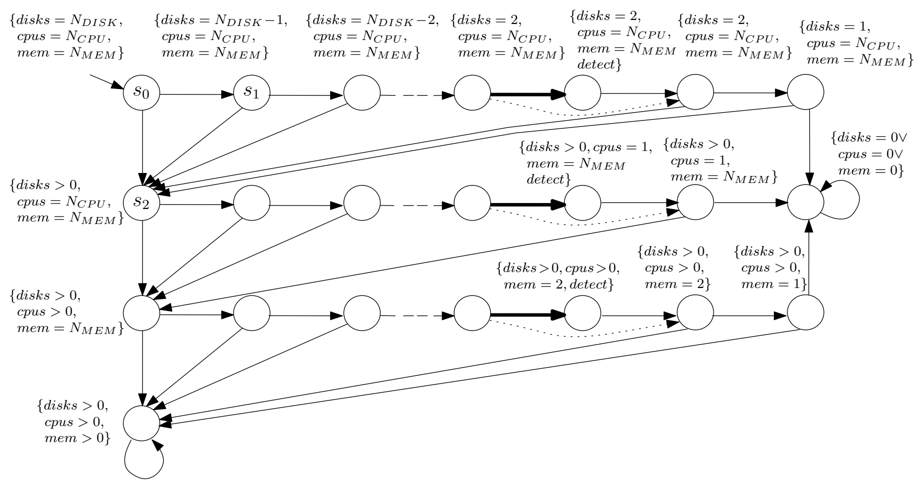

$\{disks = N_{DISK}, cpus = N_{CPU}, mem = N_{MEM}\}$
$\{disks = N_{DISK}-1, cpus = N_{CPU}, mem = N_{MEM}\}$
$\{disks = N_{DISK}-2, cpus = N_{CPU}, mem = N_{MEM}\}$
$\{disks = 2, cpus = N_{CPU}, mem = N_{MEM}\}$
$\{disks = 2, cpus = N_{CPU}, mem = N_{MEM}$ $detect\}$
$\{disks = 2, cpus = N_{CPU}, mem = N_{MEM}\}$
$\{disks = 1, cpus = N_{CPU}, mem = N_{MEM}\}$
s_0
s_1
$\{disks = 0\vee cpus = 0\vee mem = 0\}$
$\{disks > 0, cpus = 1, mem = N_{MEM}$ $detect\}$
$\{disks > 0, cpus = 1, mem = N_{MEM}\}$
$\{disks > 0, cpus = N_{CPU}, mem = N_{MEM}\}$
s_2
$\{disks>0, cpus>0, mem = 2, detect\}$
$\{disks > 0, cpus > 0, mem = 2\}$
$\{disks > 0, cpus > 0, mem = 1\}$
$\{disks > 0, cpus > 0, mem = N_{MEM}\}$
$\{disks > 0, cpus > 0, mem > 0\}$

Fig. 2. State transition model of a physical server. The states labelled with the atomic proposition *detect* are reached if multiple component failures render the server "unsafe" *and* the failure detection mechanism operates correctly. For our running example, the server is deemed unsafe when it is left with only two disks or one CPU or two memory blocks that are operational.

The model in Figure 2 supports the verification of safety properties such as "it is never the case that the failure of all server components of the same type (i.e., all disks or all CPUs or all memory blocks) is not detected" over a fixed time period (e.g., one year). The state transitions of this model correspond to:

– individual components failures (e.g., the transition (s_0, s_1) corresponds to the failure of the first disk within the analysed time period);
– individual components being operational at the end of the considered time period (e.g., the transition (s_0, s_2) is taken if the first disk is operational throughout the considered time period);
– the failure detection mechanism operating correctly (i.e., the three transitions depicted using thick lines in the transition graph from Figure 2);
– if applicable, the incorrect operation of the failure detection mechanism (i.e., the three transitions represented with dotted lines).

To keep the model small, the first two types of state transitions are included for a component only when the failure or correct operation of that component has an impact on the safety properties that we are interested in. For instance, state s_2 is reached if at least one of the disks remains operational throughout the considered time period. Therefore, the model does not include any transitions leaving s_2 or a state reachable from s_2 and modelling the failure or correct operation of a disk; and all these states are labelled with the atomic proposition "$disks > 0$". Choosing the right *level of abstraction* for the model in this way is essential in order to reduce the size of its state space.

The safety property "it is never the case that the failure of all disks or all CPUs or all memory blocks is not *detect*-ed" can be expressed formally using the G (globally) and U (until) linear-time temporal logic (LTL) operators:

$$\mathrm{G}[\neg(\neg \mathit{detect}\ \mathrm{U}\ \mathit{disk}=0 \vee \mathit{cpu}=0 \vee \mathit{mem}=0)]. \tag{3}$$

This property is satisfied by the server model if and only if the state transitions represented with dotted lines in Figure 2 are not present. We make this observation by examining every single *path* (i.e., sequence of transitions) from the initial state s_0 to the state labelled with the atomic proposition "$\mathit{disk}=0 \vee \mathit{cpu}=0 \vee \mathit{mem}=0$", and noting that it includes a state labelled "*detect*" if and only if the dotted-line transitions are not part of the model.

Various modelling formalisms support the verification of reliability, performance and cost-related properties by additionally annotating the model transitions and/or states with probabilities, transition rates and costs/rewards, respectively. For instance, annotating the state transitions from our server model in Figure 2 with probabilities allows the verification of *probabilistic safety properties* such as "the probability that the failure of all server components of the same type is detected is at least 0.999". This property can be expressed formally in probabilistic computation tree logic (PCTL) as

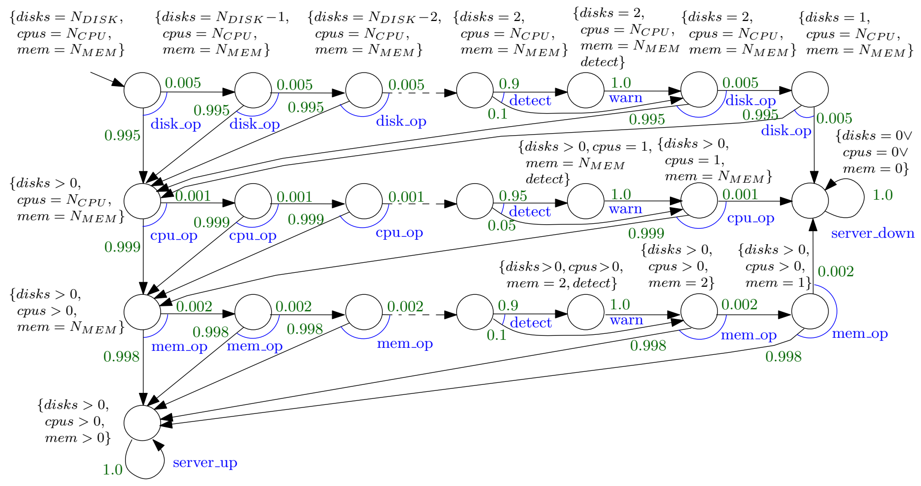
$$\mathrm{P}_{\geq 0.999}[\mathrm{G}[\neg(\neg \mathit{detect}\ \mathrm{U}\ \mathit{disk}=0 \vee \mathit{cpu}=0 \vee \mathit{mem}=0)]], \tag{4}$$

where P is the probabilistic PCTL operator.

Finally, component interactions are modelled by annotating the state transitions of the models $M_i = (S_i, s_i^0, T_i, L_i)$ from (2) with *actions* from an *action alphabet* α_i, $1 \leq i \leq n$. When a transition $(s, s') \in T_i$ is annotated with action $a \in \alpha_i$, it can be taken when model M_i is in state s only at the same time with an a-annotated transition in every other model $M_j \neq M_i$ whose action alphabet also includes a, $1 \leq j \leq n$.

Figure 3 shows a variant of the server model in which transitions are annotated with both probabilities and actions. The former support the verification of the probabilistic safety property (4). The latter enable the modelling of the interaction between the server and the other components of the system in Figure 1, e.g., through the parallel composition of the server model M_{server} with the model $M_{web+app}$ of the two Web and two App(lication) instances running on Server A or on Server B. This model (shown in Figure 4) comprises state transitions annotated with the actions "server_down", "warn" and "server_up" that also belong to the action alphabet for the server model. The $M_{web+app}$ state transitions corresponding to the actions shared between the two models are not annotated with probabilities like all the other $M_{web+app}$ state transitions, as these probabilities depend on the server behaviour and are unknown until the two models are composed.

The way in which we established the safety property (3) by examining every path starting at the initial state s_0 of the model in Figure 2 is applicable only to models that are relatively small or have a particularly regular structure. Advanced model checking techniques including *symbolic model checking* and *partial*



{disks = N_DISK, cpus = N_CPU, mem = N_MEM}
{disks = N_DISK − 1, cpus = N_CPU, mem = N_MEM}
{disks = N_DISK − 2, cpus = N_CPU, mem = N_MEM}
{disks = 2, cpus = N_CPU, mem = N_MEM}
{disks = 2, cpus = N_CPU, mem = N_MEM detect}
{disks = 2, cpus = N_CPU, mem = N_MEM}
{disks = 1, cpus = N_CPU, mem = N_MEM}
0.005
0.995
disk_op
0.9
detect
0.1
1.0
warn
{disks = 0∨ cpus = 0∨ mem = 0}
server_down
{disks > 0, cpus = N_CPU, mem = N_MEM}
{disks > 0, cpus = 1, mem = N_MEM detect}
{disks > 0, cpus = 1, mem = N_MEM}
0.001
0.999
cpu_op
0.95
0.05
{disks > 0, cpus > 0, mem = N_MEM}
{disks>0, cpus>0, mem = 2, detect}
{disks > 0, cpus > 0, mem = 2}
{disks > 0, cpus > 0, mem = 1}
0.002
0.998
mem_op
{disks > 0, cpus > 0, mem > 0}
server_up

Fig. 3. Model M_{server} for the running example: annotating the state transitions of the server model with probabilities and actions enables the verification of probabilistic safety properties and the modelling of component interactions, respectively.

order reduction overcome this limitation by avoiding the exhaustive enumeration and analysis of all such paths through the model [13].

2.3 Compositional verification

Even though symbolic model checking extends the applicability of formal verification to some very large models, this is still insufficient for many models associated with today's IT systems. A complete model of our service from Figure 1, for instance, requires the parallel composition of:

– Four instances of the server model M_{server} from Figure 3 (one for each of Servers A, B, C and D). These model instances—denoted M_{server_A}, M_{server_B}, M_{server_C} and M_{server_D}—are obtained from the model M_{server} in Figure 3 by subscripting all its actions and atomic proposition parameters with $_\mathsf{A}$, $_\mathsf{B}$, $_\mathsf{C}$ and $_\mathsf{D}$, respectively (e.g., server_down$_\mathsf{A}$ or $disk_\mathsf{B}$).
– Two instances of the "Web-Application" model $M_{web+app}$ from Figure 4 (corresponding to the web and application instances deployed on Server A and Server B, and denoted $M_{web+app_\mathsf{A}}$ and $M_{web+app_\mathsf{B}}$).
– Two instances of the "Database" model M_{db} from Figure 5 (M_{db_C} and M_{db_D}, corresponding to the Database instances on Servers C and D, respectively).
– The three-tier architecture model $M_{service}$ from Figure 6.

We implemented this composition as a monolithic model

$$M = M_{server_\mathsf{A}} \parallel M_{server_\mathsf{B}} \parallel M_{server_\mathsf{C}} \parallel M_{server_\mathsf{D}} \parallel M_{web+app_\mathsf{A}} \parallel M_{web+app_\mathsf{B}} \parallel M_{db_\mathsf{C}} \parallel M_{db_\mathsf{D}} \parallel M_{service} \tag{5}$$

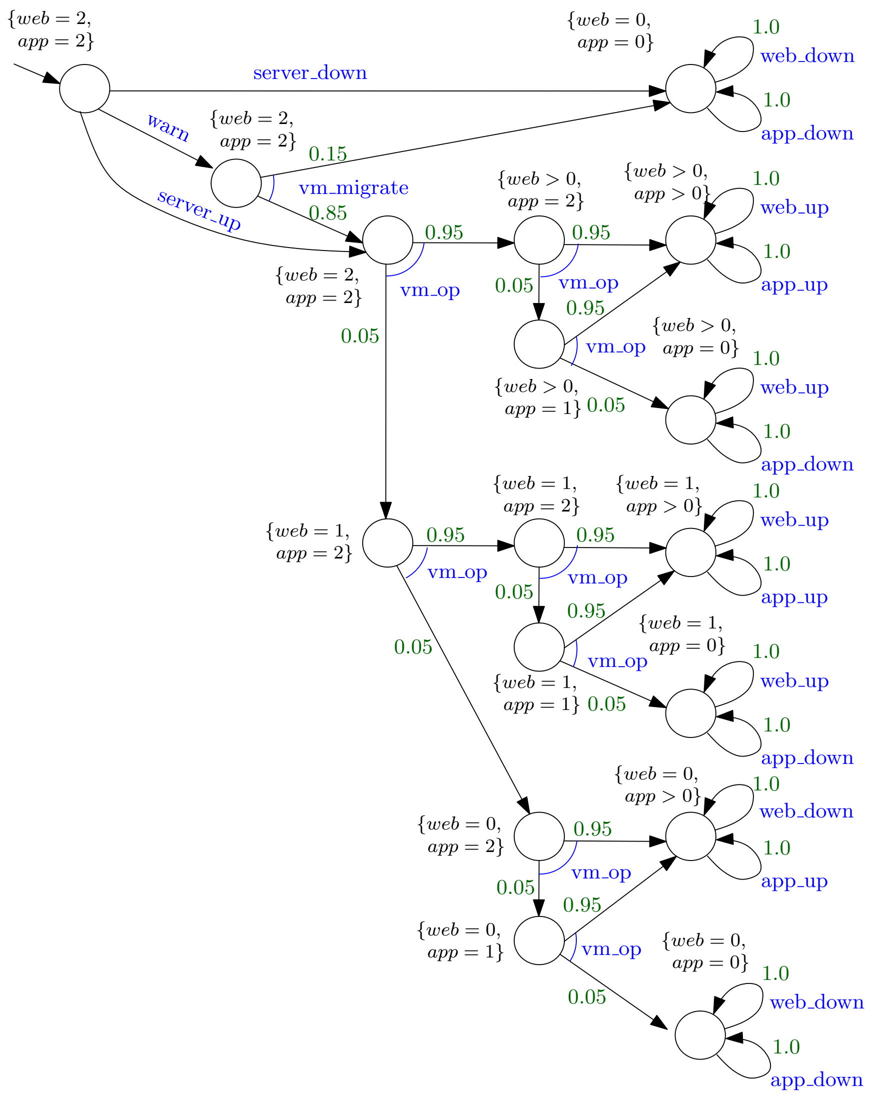


Fig. 4. Model $M_{web+app}$ for the running example: the two Web and the two App(lication) instances on Server A or B are down at the end of the analysed time period if the server fails, the VM migration triggered by a warning is unsuccessful, or the VMs running them fail.

for the probabilistic symbolic model checker PRISM [22], and the tool ran out of memory when attempting to verify if the resulting 176,381,406,182,650-state model satisfied the property “the probability that the service fails within a one-year time interval is under 0.0005”.

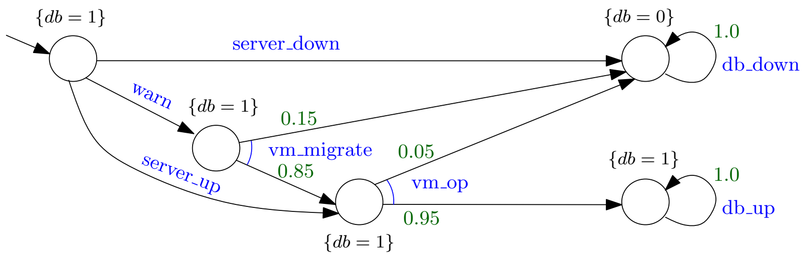


Fig. 5. Model M_{db} for the running example: the Database instance on Server C or D is down at the end of the analysed time period if the server fails, the VM migration triggered by a warning is unsuccessful, or the VM running it fails.

This *state explosion* is avoided by compositional verification, a collection of techniques that increase the size of the (component-based) systems that can be model checked significantly. In its original form proposed in the seminal work of Pnueli [29], compositional verification involves establishing that the parallel composition of two models $M_1 \parallel M_2$ satisfies a global property $\mathcal{G}$ through verifying two premises independently. The first premise is that M_2 satisfies $\mathcal{G}$ when it is part of a system that satisfies an *assumption* (i.e., property) $\mathcal{A}$. The second premise is that $\mathcal{A}$ is satisfied by the remainder of the system (i.e., by M_1) under all circumstances. This can be expressed formally as a proof tree by using Pnueli's generalisation [29] of the Hoare triple notation [23]:

$$\frac{\langle true\rangle M_1 \langle\mathcal{A}\rangle,\ \langle\mathcal{A}\rangle M_2 \langle\mathcal{G}\rangle}{\langle true\rangle M_1 \parallel M_2 \langle\mathcal{G}\rangle}. \tag{6}$$

The technique is termed *assume-guarantee reasoning*, to distinguish it from other compositional verification approaches that have emerged more recently.

Given the importance of extending the applicability of model checking to larger systems, assume-guarantee reasoning has received significant attention from the research community [3, 14, 15, 21]. In particular, assume-guarantee reasoning has been extended to probabilistic systems [27], enabling the compositional verification of probabilistic safety properties for parallel model compositions such as model (5) from our running example. The models used in this extension of the technique are *probabilistic automata* (PAs) [30] of the form

$$M_i = (S_i, s_i^0, \alpha_i, \delta_i, L_i). \tag{7}$$

As before, S_i, $s_i^0 \in S_i$, α_i and L_i represent a finite set of states, the initial state, the action alphabet and an atomic-proposition labelling function, respectively. However, the state transition relation T_i from the definition of the Kripke model in (2) is replaced by a *probabilistic state transition relation* $\delta_i \subseteq S_i \times (\alpha_i \cup \{\tau\}) \times \mathit{Dist}(S_i)$, where $\mathit{Dist}(S_i)$ denotes the set of all discrete probability distributions over the state set S_i. The possible transitions from a generic state $s \in S_i$ to another state in S_i are given by the set $\delta_i(s) = \{(s, a, d) \mid (s, a, d) \in \delta_i\}$. When the system is in state s, an element

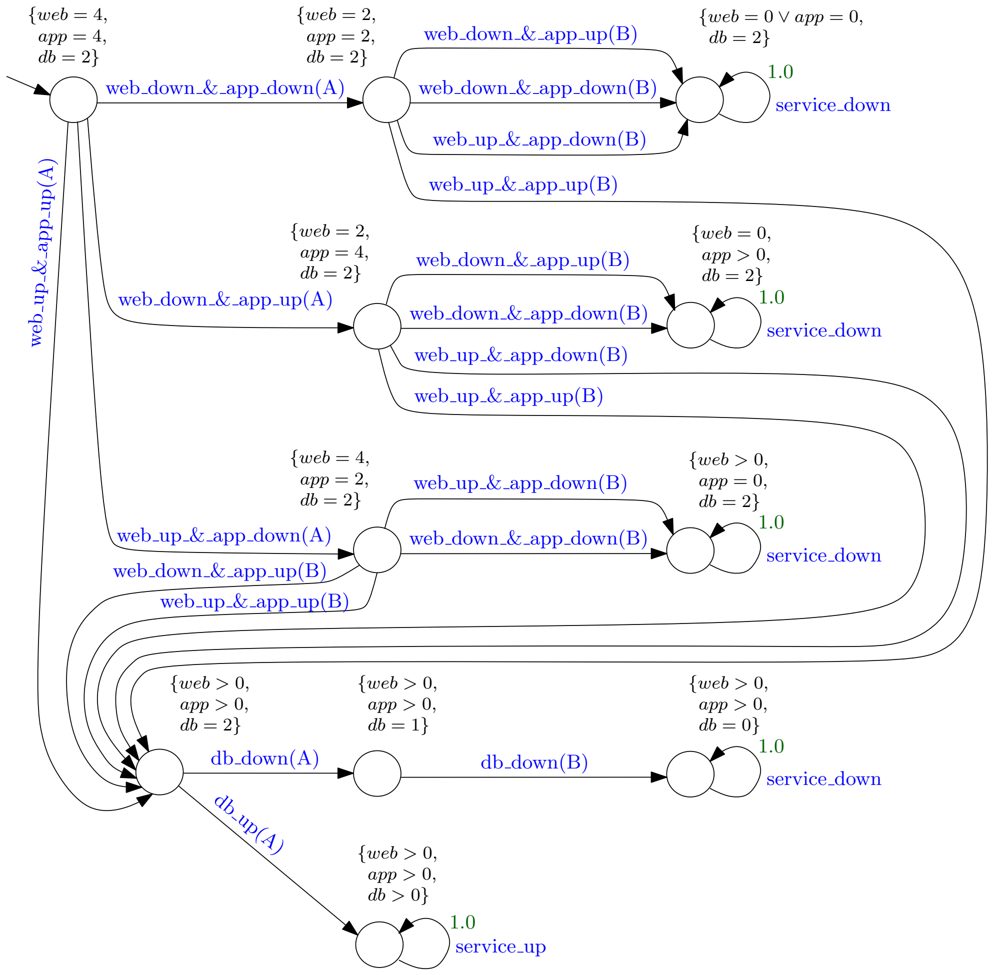
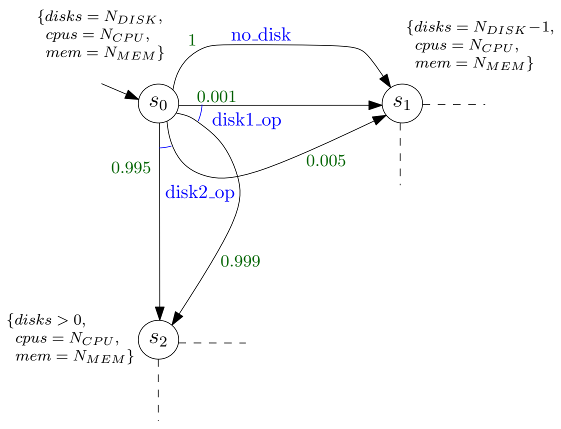

{web = 4, app = 4, db = 2}
{web = 2, app = 2, db = 2}
{web = 0 ∨ app = 0, db = 2}
web_down_&_app_down(A)
web_down_&_app_up(B)
web_down_&_app_down(B)
web_up_&_app_down(B)
web_up_&_app_up(B)
1.0
service_down
{web = 2, app = 4, db = 2}
{web = 0, app > 0, db = 2}
web_down_&_app_up(A)
{web = 4, app = 2, db = 2}
{web > 0, app = 0, db = 2}
web_up_&_app_down(A)
web_up_&_app_up(A)
{web > 0, app > 0, db = 2}
{web > 0, app > 0, db = 1}
{web > 0, app > 0, db = 0}
db_down(A)
db_down(B)
db_up(A)
{web > 0, app > 0, db > 0}
service_up

Fig. 6. Model $M_{service}$ for the running example: the service fails if all instances of any of the Web, Application and Database "functions" fail.

$(s, a, d) \in \delta_i(s)$ is chosen nondeterministically, and the next state s' is selected randomly according to the distribution $d \in Dist(S_i)$. This characteristic of probabilistic automata is particularly useful for modelling LSCITS components, as illustrated in Figure 7 for a physical server from our running example.

The analysis of PA properties requires the resolution of its nondeterministic choices by means of *adversaries*, i.e., functions that map any finite path ending in a generic state $s \in S_i$ to one of the discrete probability distributions in $\delta_i(s)$ or "decide" to remain in state s. Given the set of all adversaries Adv_i of a PA model M_i, we are typically interested in verifying a property related to the minimum and/or maximum probability of an event over all adversaries in Adv_i. For the



{disks = N_DISK, cpus = N_CPU, mem = N_MEM}
{disks = N_DISK−1, cpus = N_CPU, mem = N_MEM}
{disks > 0, cpus = N_CPU, mem = N_MEM}
s0
s1
s2
1
no_disk
0.001
disk1_op
0.005
0.995
disk2_op
0.999

Fig. 7. Fragment of the PA model of a server that may use disks of type "disk1", disks of type "disk2" or may not be equipped with the N_{DISK}'th disk. Accordingly, $\delta_1(s^0) = \{(s^0,$disk1_op$, [(s^0,0), (s^1,0.001), (s^2,0.999),\ldots]), (s^0,$ disk2_op$, [(s^0,0),(s^1,0.005),(s^2,0.995),\ldots]), (s^0,$ no_disk$, [(s^0,0),(s^1,1),(s^2,0),\ldots])\}$.

system in our running example, for instance, we want to establish that the PA version of the parallel composition M in (5) satisfies

$$\mathrm{P}^{min}_{\geq 0.9995}[\mathrm{F}\ \neg(web = 0 \vee app = 0 \vee db = 0)], \tag{8}$$

namely that the minimum probability that none of the three service functions fails, over all possible adversaries of M, is at least 0.9995.

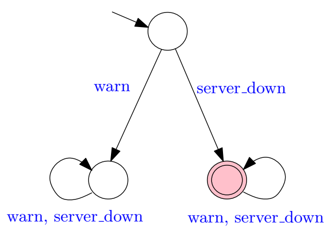
The core probabilistic assume-guarantee rule introduced in [27] is a probabilistic variant of (6):

$$\frac{\langle true\rangle M_1 \langle \mathcal{A}\rangle_{\geq p_1},\ \langle \mathcal{A}\rangle_{\geq p_1} M_2 \langle \mathcal{G}\rangle_{\geq p_2}}{\langle true\rangle M_1 \parallel M_2 \langle \mathcal{G}\rangle_{\geq p_2}}, \tag{9}$$

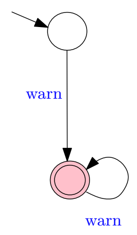
where, given a model M and a *probabilistic safety property* $\langle \mathcal{X}\rangle_{\geq p}$, $M \models \langle \mathcal{X}\rangle_{\geq p}$ holds iff the minimum probability that $\mathcal{X}$ is satisfied over all adversaries of M is at least p. A probabilistic safety property $\langle \mathcal{X}\rangle_{\geq p}$ is specified by means of:

– A *deterministic finite automaton* (DFA) $\mathcal{X}^{\mathrm{err}} = (Q, \alpha_{\mathcal{X}}, \delta_{\mathcal{X}}, q_0, F)$ with the state set Q, alphabet $\alpha_{\mathcal{X}}$, transition function $\delta_{\mathcal{X}} : Q \times \alpha_{\mathcal{X}} \to Q$, initial state q_0 and accepting states $F \subseteq Q$. The finite words accepted by $\mathcal{X}^{\mathrm{err}}$ specify the sequences of actions associated with prefixes of paths that do not satisfy $\mathcal{X}$.
– The rational probability bound p.

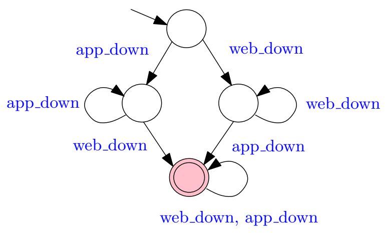
Consider, for instance, the server model M_{server} from Fig. 3, its action alphabet $\alpha_{server} = \{$disk_op, cpu_op, mem_op, detect, warn, server_up, server_down$\}$, and let $\langle \mathcal{A}_1\rangle_{\geq 0.999}$ be the probabilistic safety property from eq. (4). The DFA $\mathcal{A}_1^{err}$ and its regular language $L(\mathcal{A}_1^{err})$ of "bad prefixes" are shown in Fig 8(a).



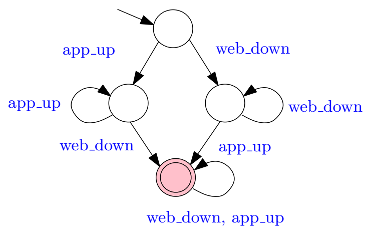
(a) $\mathcal{A}_1^{\mathrm{err}}$: $L(\mathcal{A}_1^{\mathrm{err}}) = \mathrm{server\_down}^+$



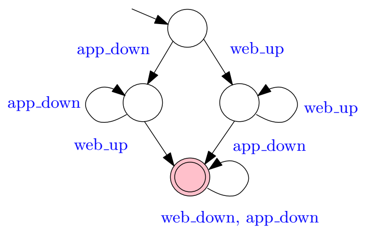
(b) $\mathcal{A}_2^{\mathrm{err}}$: $L(\mathcal{A}_2^{\mathrm{err}}) = \mathrm{warn}^+$



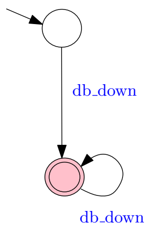
(c) $\mathcal{A}_3^{\mathrm{err}}$: $L(\mathcal{A}_3^{\mathrm{err}}) = (\mathrm{app\_down}^+\mathrm{web\_down} \mid \mathrm{web\_down}^+\mathrm{app\_down})(\mathrm{web\_down} \mid \mathrm{app\_down})^*$



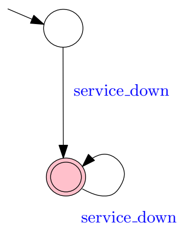
(d) $\mathcal{A}_4^{\mathrm{err}}$: $L(\mathcal{A}_4^{\mathrm{err}}) = (\mathrm{app\_up}^+\mathrm{web\_down} \mid \mathrm{web\_down}^+\mathrm{app\_up})(\mathrm{web\_down} \mid \mathrm{app\_up})^*$



(e) $\mathcal{A}_5^{\mathrm{err}}$: $L(\mathcal{A}_5^{\mathrm{err}}) = (\mathrm{app\_down}^+\mathrm{web\_up} \mid \mathrm{web\_up}^+\mathrm{app\_down})(\mathrm{web\_up} \mid \mathrm{app\_down})^*$



(f) $\mathcal{A}_6^{\mathrm{err}}$: $L(\mathcal{A}_6^{\mathrm{err}}) = \mathrm{db\_down}^+$



(g) $\mathcal{G}^{\mathrm{err}}$: $L(\mathcal{G}^{\mathrm{err}}) = \mathrm{service\_down}^+$

Fig. 8. Deterministic finite automata and regular expressions defining for the probabilistic safety properties from the running example.

Given the DFAs $\mathcal{A}^{\mathrm{err}}$ and $\mathcal{G}^{\mathrm{err}}$ for the assumed and guaranteed probabilistic safety properties in the proof rule (9), the verification of its two premises is carried out as follows [27]:

- To verify $\langle true \rangle M_1 \langle \mathcal{A} \rangle_{\geq p_1}$ quantitatively, the parallel composition of M_1 and $\mathcal{A}^{\mathrm{err}}$ is model checked to obtain $1-p_1$, the maximum probability of reaching the (undesirable) accepting states of $\mathcal{A}^{\mathrm{err}}$, over all adversaries of M_1.
- To verify $\langle \mathcal{A} \rangle_{\geq p_1} M_2 \langle \mathcal{G} \rangle_{\geq p_2}$, M_2 is composed with both $\mathcal{A}^{\mathrm{err}}$ and $\mathcal{G}^{\mathrm{err}}$. Because the satisfaction of $\mathcal{A}$ with probability p_1 and of $\mathcal{G}$ with probability p_2 must

be analysed together, a technique called *multi-objective model checking* [18] is then used. This technique produces $1 - p_2$, the maximum probability of reaching the (undesirable) accepting states of $\mathcal{G}^{\text{err}}$, under the assumption $\langle \mathcal{A} \rangle_{\geq p_1}$ and over all adversaries of M_2. These steps are described in detail in [27], and automated in the latest version of the probabilistic symbolic model checker PRISM [26].

To verify that model M from eq. (5) satisfies the probabilistic safety property (8), we used the probabilistic assume-guarantee proof tree

$$\dfrac{\begin{array}{c}\dfrac{\begin{array}{c}\langle true\rangle M_{server_{\mathsf{A}}} \langle \mathcal{A}1_{\mathsf{A}}, \mathcal{A}2_{\mathsf{A}}\rangle_{\geq p_1,p_2} \\ \langle \mathcal{A}1_{\mathsf{A}}, \mathcal{A}2_{\mathsf{A}}\rangle_{\geq p_1,p_2} M_{web+app_{\mathsf{A}}} \langle \mathcal{A}3_{\mathsf{A}}, \mathcal{A}4_{\mathsf{A}}, \mathcal{A}5_{\mathsf{A}}\rangle_{\geq p_3,p_4,p_5}\end{array}}{\langle true\rangle M_{server_{\mathsf{A}}} \parallel M_{web+app_{\mathsf{A}}} \langle \mathcal{A}3_{\mathsf{A}}, \mathcal{A}4_{\mathsf{A}}, \mathcal{A}5_{\mathsf{A}}\rangle_{\geq p_3,p_4,p_5}}\ ,\ (*) \\[2ex] \dfrac{\begin{array}{c}\langle true\rangle M_{server_{\mathsf{B}}} \langle \mathcal{A}1_{\mathsf{B}}, \mathcal{A}2_{\mathsf{B}}\rangle_{\geq p_1,p_2} \\ \langle \mathcal{A}1_{\mathsf{B}}, \mathcal{A}2_{\mathsf{B}}\rangle_{\geq p_1,p_2} M_{web+app_{\mathsf{B}}} \langle \mathcal{A}3_{\mathsf{B}}, \mathcal{A}4_{\mathsf{B}}, \mathcal{A}5_{\mathsf{B}}\rangle_{\geq p_3,p_4,p_5}\end{array}}{\langle true\rangle M_{server_{\mathsf{B}}} \parallel M_{web+app_{\mathsf{B}}} \langle \mathcal{A}3_{\mathsf{B}}, \mathcal{A}4_{\mathsf{B}}, \mathcal{A}5_{\mathsf{B}}\rangle_{\geq p_3,p_4,p_5}}\ ,\ (*) \\[2ex] \dfrac{\begin{array}{c}\langle true\rangle M_{server_{\mathsf{C}}} \langle \mathcal{A}1_{\mathsf{C}}, \mathcal{A}2_{\mathsf{C}}\rangle_{\geq p_1,p_2} \\ \langle \mathcal{A}1_{\mathsf{C}}, \mathcal{A}2_{\mathsf{C}}\rangle_{\geq p_1,p_2} M_{db_{\mathsf{C}}} \langle \mathcal{A}6_{\mathsf{C}}\rangle_{\geq p_6}\end{array}}{\langle true\rangle M_{server_{\mathsf{C}}} \parallel M_{db_{\mathsf{C}}} \langle \mathcal{A}6_{\mathsf{C}}\rangle_{\geq p_6}}\ ,\ (*) \\[2ex] \dfrac{\begin{array}{c}\langle true\rangle M_{server_{\mathsf{D}}} \langle \mathcal{A}1_{\mathsf{D}}, \mathcal{A}2_{\mathsf{D}}\rangle_{\geq p_1,p_2} \\ \langle \mathcal{A}1_{\mathsf{D}}, \mathcal{A}2_{\mathsf{D}}\rangle_{\geq p_1,p_2} M_{db_{\mathsf{D}}} \langle \mathcal{A}6_{\mathsf{D}}\rangle_{\geq p_6}\end{array}}{\langle true\rangle M_{server_{\mathsf{D}}} \parallel M_{db_{\mathsf{D}}} \langle \mathcal{A}6_{\mathsf{D}}\rangle_{\geq p_6}}\ ,\ (*) \\[2ex] \langle \mathcal{A}3_{\mathsf{A}},\mathcal{A}4_{\mathsf{A}},\mathcal{A}5_{\mathsf{A}},\mathcal{A}3_{\mathsf{B}},\mathcal{A}4_{\mathsf{B}},\mathcal{A}5_{\mathsf{B}},\mathcal{A}6_{\mathsf{C}},\mathcal{A}6_{\mathsf{D}}\rangle_{\geq p_3,p_4,p_5,p_3,p_4,p_5,p_6,p_6} M_{service} \langle \mathcal{G}\rangle_{\geq p_7}\end{array}}{\langle true\rangle M \langle \mathcal{G}\rangle_{\geq p_7}}\ (\#) \tag{10}$$

Notice that this proof tree represents a bottom-up reflection of the structure of the real-world system from Figure 1, where:

– the probabilistic safety properties $\langle \mathcal{A}1_{\mathsf{A}}\rangle_{\geq p_1}$ to $\langle \mathcal{A}1_{\mathsf{D}}\rangle_{\geq p_1}$, $\langle \mathcal{A}2_{\mathsf{A}}\rangle_{\geq p_2}$ to $\langle \mathcal{A}2_{\mathsf{D}}\rangle_{\geq p_2}$, etc. are defined by the DFAs in Figure 8 (with the appropiate subscript—A, B, C or D—applied to their action names);
– the probabilities p_1 to p_7 were obtained using the probabilistic model checker PRISM as described earlier;
– $(*)$ denotes the application of the ASym-Mult probabilistic assume-guarantee proof rule from [27];
– $(\#)$ marks the application of the new assume-guarantee proof rule that we introduce in Appendix A.

We executed the verification steps for all premises in (10) on a Macbook Pro laptop with 2.66 GHz Intel Core 2 Duo processor and 8GB of memory, using the hardware failure probabilities reported in [32, 34] ($p_{disk\_fail} = 0.0231$, $p_{cpu\_fail} = 0.0018$ and $p_{mem\_fail} = 0.0231$ for a one-year period of operation).[3]

[3] The component failure probabilities in Figs. 3 and 7 were used only for illustration.

Table 1. Experimental results for the probabilistic assume-guarantee proof tree (10). The probabilities associated with the assumed and guaranteed properties in (10) were calculated for a one-year time interval, based on the hardware component failure probabilities reported in [32, 34].

Verified model	Number of states	Result
$M_{server_{\mathsf{A-D}}}$	570	$p_1 = 0.999998$ $p_2 = 0.999544$
$M_{web+app_{\mathsf{A-B}}}$	54	$p_3 = 0.999946$ $p_4 = 0.997452$ $p_5 = 0.997452$
$M_{db_{\mathsf{C-D}}}$	13	$p_6 = 0.949954$
$M_{service}$	1035	$p_7 = 0.997482$

The results of the verification and the size of the models verified are shown in Table 1. As indicated by these results, the size of the state space for the verified models ranged between 13 and 1035, which explains why each of the verification steps completed in under one second. We anticipate that safety proporties for systems comprising much larger numbers of servers, VMs per server, and function instances per service could be verified using the approach, and we are planning to confirm this experimentally in the future.

3 Reverification of safety properties for LSCITS

We showed in the previous section how compositional verification can be used to verify safety properties of a class of systems that can potentially be very large. However, size is not the only defining characteristic of LSCITS. LSCITS can be seen as *coalitions of systems* whose components join and leave continually, and within which frequent component selection and failure represent the norm rather than an exception [9, 28, 31].

In this section, we present techniques for the calculation of the minimal sequence of assume-guarantee premises that need to be reverified in response to several of these key *patterns of LSCITS change*. To describe these techniques, we will use the following additional notation:

- $\mathcal{M}$, the set of PA models (7);
- $\mathcal{P}$, the set of probabilistic safety properties;
- DFA, the set of deterministic finite automata;
- $dfa : \mathcal{P} \to DFA$, the function that maps each probabilistic safety property to its defining deterministic finite automaton;
- $prob : \mathcal{P} \to [0, 1]$, the function that maps each probabilistic safety property to its associated probability (i.e., $\forall \langle \mathcal{X} \rangle_{\geq p} \in \mathcal{P} \bullet prob(\langle \mathcal{X} \rangle_{\geq p}) = p$);
- $mc : 2^{\mathcal{P}} \times \mathcal{M} \times DFA \to [0, 1]$, the quantitative model checking function that, given a set of assumptions $A \in 2^{\mathcal{P}}$, a model $M \in \mathcal{M}$ and a deterministic finite automaton $\mathcal{G}^{\text{err}} \in DFA$, ensures that $\langle \mathcal{G} \rangle_{\geq p}$, i.e., $M \models \langle \mathcal{G} \rangle_{\geq mc(A,M,\mathcal{G}^{\text{err}})}$ under the assumptions A;

– $\mathcal{V} \subset 2^{\mathcal{P}} \times \mathcal{M} \times 2^{\mathcal{P}}$, the set of all *verification steps* that can appear as premises in a probabilistic assume-guarantee proof tree; $(A, M, G) \in \mathcal{V}$ iff A and G are finite sets of assumed and guaranteed probabilistic safety properties for the PA model M, respectively.

Note that $\langle true \rangle$ is a special element of $\mathcal{P}$; when it is used as an assumption for a model $M \in \mathcal{M}$, $dfa(\langle true \rangle)$ is the one-state DFA that has the same alphabet α_M as M and does not accepts any word, i.e., $dfa(\langle true \rangle) = (\{q_0\}, \alpha_M, \{a \in \alpha_M \bullet (q_0, a) \mapsto q_0\}, q_0, \{\})$ and $prob(\langle true \rangle) = 1$. In the definition of the transition function for $dfa(\langle true \rangle)$, we used the set comprehension notation $\{a \in \alpha_M \bullet (q_0, a) \mapsto q_0\}$ to build the set of mappings "$(q_0, a) \mapsto q_0$" for all possible values $a \in \alpha_M$. This notation, including its generalised form $\{declaration \mid predicate \bullet expression\}$ will be used again in this section as a concise way of specifying sets such as $\{x \in \mathbb{N} \mid 5 \leq x \leq 20 \bullet \sqrt{x}\}$, the set comprising the square root of all natural numbers between 5 and 20.

We are interested in the finite sequences of verification steps $(v_1, v_2, \ldots, v_n) \in \mathrm{seq}\mathcal{V}$ that correspond to probabilistic assume-guarantee proof trees, which we term *compositional verification tasks*. A sequence $(v_1, v_2, \ldots, v_n)$, where $v_i = (A_i, M_i, G_i) \in \mathcal{V}$ for all $1 \leq i \leq n$, is a compositional verification task iff the set of assumed properties for each of its verification steps comprises only the special property $\langle true \rangle$ and properties guaranteed by preceeding verification steps: $A_i \subseteq \{\langle true \rangle\} \cup G_1 \cup G_2 \cup G_{i-1}$, for $1 \leq i \leq n$.

Using the notation introduced above, the compositional verification task (9) from our running example can be specified as a nine-element sequence of verification steps $(v_1, v_2, \ldots, v_9)$, where

$$\begin{aligned}
v_1 &= (\{\langle true \rangle\}, M_{server_\mathsf{A}}, \{\langle \mathcal{A}_{1_\mathsf{A}} \rangle_{\geq p_1}, \langle \mathcal{A}_{2_\mathsf{A}} \rangle_{\geq p_2}\}) \\
v_2 &= (\{\langle true \rangle\}, M_{server_\mathsf{B}}, \{\langle \mathcal{A}_{1_\mathsf{B}} \rangle_{\geq p_1}, \langle \mathcal{A}_{2_\mathsf{B}} \rangle_{\geq p_2}\}) \\
v_3 &= (\{\langle true \rangle\}, M_{server_\mathsf{C}}, \{\langle \mathcal{A}_{1_\mathsf{C}} \rangle_{\geq p_1}, \langle \mathcal{A}_{2_\mathsf{C}} \rangle_{\geq p_2}\}) \\
v_4 &= (\{\langle true \rangle\}, M_{server_\mathsf{D}}, \{\langle \mathcal{A}_{1_\mathsf{D}} \rangle_{\geq p_1}, \langle \mathcal{A}_{2_\mathsf{D}} \rangle_{\geq p_2}\}) \\
v_5 &= (\{\langle \mathcal{A}_{1_\mathsf{A}} \rangle_{\geq p_1}, \langle \mathcal{A}_{2_\mathsf{A}} \rangle_{\geq p_2}\}, M_{web+app_\mathsf{A}}, \{\langle \mathcal{A}_{3_\mathsf{A}} \rangle_{\geq p_3}, \langle \mathcal{A}_{4_\mathsf{A}} \rangle_{\geq p_4}, \langle \mathcal{A}_{5_\mathsf{A}} \rangle_{\geq p_5}\}) \\
v_6 &= (\{\langle \mathcal{A}_{1_\mathsf{B}} \rangle_{\geq p_1}, \langle \mathcal{A}_{2_\mathsf{B}} \rangle_{\geq p_2}\}, M_{web+app_\mathsf{B}}, \{\langle \mathcal{A}_{3_\mathsf{B}} \rangle_{\geq p_3}, \langle \mathcal{A}_{4_\mathsf{B}} \rangle_{\geq p_4}, \langle \mathcal{A}_{5_\mathsf{B}} \rangle_{\geq p_5}\}) \\
v_7 &= (\{\langle \mathcal{A}_{1_\mathsf{C}} \rangle_{\geq p_1}, \langle \mathcal{A}_{2_\mathsf{C}} \rangle_{\geq p_2}\}, M_{db_\mathsf{C}}, \{\langle \mathcal{A}_{6_\mathsf{C}} \rangle_{\geq p_{\mathcal{A}_6}}\}) \\
v_8 &= (\{\langle \mathcal{A}_{1_\mathsf{D}} \rangle_{\geq p_1}, \langle \mathcal{A}_{2_\mathsf{D}} \rangle_{\geq p_2}\}, M_{db_\mathsf{D}}, \{\langle \mathcal{A}_{6_\mathsf{D}} \rangle_{\geq p_{\mathcal{A}_6}}\}) \\
v_9 &= (\{\langle \mathcal{A}_{3_\mathsf{A}} \rangle_{\geq p_3}, \langle \mathcal{A}_{4_\mathsf{A}} \rangle_{\geq p_4}, \langle \mathcal{A}_{5_\mathsf{A}} \rangle_{\geq p_5}, \langle \mathcal{A}_{3_\mathsf{B}} \rangle_{\geq p_3}, \langle \mathcal{A}_{4_\mathsf{B}} \rangle_{\geq p_4}, \\
&\qquad \langle \mathcal{A}_{5_\mathsf{B}} \rangle_{\geq p_5}, \langle \mathcal{A}_{6_\mathsf{C}} \rangle_{\geq p_6}, \langle \mathcal{A}_{6_\mathsf{D}} \rangle_{\geq p_6}\}, M_{service}, \{\langle \mathcal{G} \rangle_{\geq p_7}\})
\end{aligned} \tag{11}$$

3.1 Reverification of a sequence of verification steps

Consider a compositional verification task $cv = (v_1, v_2, \ldots, v_n) \in \mathrm{seq}\mathcal{V}$ that was completed successfully as described in the Section 2.3. The rest of this section describes a technique for the derivation of the minimal sequence of verification steps $\Delta cv \in \mathrm{seq}\mathcal{V}$ that need to be carried out to reverify the safety properties associated with cv after different types of changes in the verified system.

We start by introducing a *reverify* function that takes as parameters:

1. a sequence of verification steps $vs \in \mathrm{seq}\mathcal{V}$; and

2. a set of guaranteed property *changes* of the form $(g, g') \in \mathcal{P} \times \mathcal{P}$ (where g and g' are related properties before and after a system change, respectively)

and produces the minimum sequence of verification steps that need to be carried out in order to reestablish the probabilistic safety properties from vs. We define the function

$$\mathit{reverify} : \mathrm{seq}\mathcal{V} \times 2^{\mathcal{P}\times\mathcal{P}} \to \mathrm{seq}\mathcal{V} \tag{12}$$

recursively on the size of the sequence of verification steps vs:

$$\mathit{reverify}((), \mathit{changes}) = ()$$

$$\begin{aligned}&\mathit{reverify}((A, M, G) ^\frown vs, \mathit{changes}) = \\ &= \begin{cases} \mathit{reverify}(vs, \mathit{changes}), & \text{if } A \cap \{(g, g') \in \mathit{changes} \bullet g\} = \emptyset \\ (A', M, G') ^\frown \mathit{reverify}(vs, \mathit{changes}'), & \text{otherwise} \end{cases}\end{aligned} \tag{13}$$

where:

(i) $A' = \{a \in A \mid \neg(\exists (g, g') \in \mathit{changes} \bullet a = g)\} \cup \{(g, g') \in \mathit{changes} \mid g \in A \bullet g'\}$ is obtained by updating all the assumptions from A that changed;

(ii) $G' = \{x \in \mathcal{P} \mid (\exists\, g \in G \bullet \mathit{dfa}(x) = \mathit{dfa}(g)) \wedge \mathit{prob}(x) = \mathit{mc}(A', M, \mathit{dfa}(g))\}$ is the new set of probabilistic safety properties guaranteed by the model M given the changed assumed property set A';

(iii) $\mathit{changes}' = \mathit{changes} \cup \{(g, g') \in G \times G' \mid \mathit{dfa}(g) = \mathit{dfa}(g') \wedge \mathit{prob}(g') < \mathit{prob}(g)\}$ represents the new set of guaranteed property changes, which is obtained by extending the old *changes* set with all pairs from $G \times G'$ that correspond to a decrease in a safety probability bound.

Throughout this section we assume that the goal of the reverification is to establish whether the analysed system continues to satisfy given probabilistic safety properties after changes. If the aim is instead to find the new probability bounds for all safety properties, then $\mathit{prob}(g') < \mathit{prob}(g)$ should be replaced with $\mathit{prob}(g') \neq \mathit{prob}(g)$ in the calculation of $\mathit{changes}'$ above.

The cost of executing the *reverify* function has two components:

1. the cost of running the verification steps (A', M, G') from (13);
2. the cost of performing the set intersection from (13) and the calculations from steps (i)–(iii) described above.

For each use of *reverify* in handling one of the LSCITS change patterns covered later in this section, we will prove that *reverify* yields the minimum sequence of verification steps required to reverify the analysed probabilistic safety properties. Therefore, we focus here only on the second cost component. To evaluate this cost component, we consider the execution of $\mathit{reverify}(cv, \mathit{changes})$ for a generic compositional verification task cv and a generic property change set *changes*. Without loss of generality, we assume that cv comprises $n > 0$ verification steps, and that these n verification steps and the *changes* set taken together contain $m > 0$ assumed and guaranteed probabilistic safety properties. Under these assumptions, the set intersection $A \cap \{(g, g') \in \mathit{changes} \bullet g\}$ from (13) requires at most $O(m^2)$ time. Likewise, the two set comprehensions from step (i) take at most $O(m^2)$ time even for the most basic implementation of set membership queries. Building the set G' in step (ii) requires $O(m)$ time (in addition to the cost of executing $mc(A', M, \mathit{dfa}(g))$, but this is part of the first cost component). Finally, the cost of updating *changes* to *changes*$'$ in step (iii) requires again at most $O(m^2)$ time for the examination of the elements in $G \times G'$. Due to the recursive *reverify* invocations, the operations analysed above are performed n times, so the overall time complexity for the operations covered by the second cost component is $O(nm^2)$. Note that, even for large values of n and m, this represents a modest overhead compared to the first cost component, which corresponds to executing the model checking operations $mc(A', M, \mathit{dfa}(g))$ from (13). Since we will prove that the minimal set of such model checking operations is executed in each scenario in which *reverify* is used in the remainder of the section, we conclude that *reverify* is cost effective for the scenarios in which it is used.

Having introduced and analysed the generic *reverify* function in (12)–(13), we are ready to calculate the minimum sequences of verification steps required after different types of LSCITS changes.

3.2 LSCITS component failure (or "departure")

Suppose that the system component associated with model M_i from the verification step v_i of the compositional verification task $cv = (v_1, v_2, \ldots, v_n)$ failed (or left the system), where $1 \leq i \leq n$. In this scenario, appropriately modified variants of some or all of the verification steps $v_{i+1}, v_{i+2}, \ldots, v_n$ need to be redone. The theorem below provides a method for the derivation of these verification steps.

Theorem 1: The minimal sequence of verification steps that needs to be carried out to reverify a compositional verification task $cv = (v_1, v_2, \ldots, v_n)$ after the failure of the component associated with its i-th verification step is

$$\Delta cv = \mathit{reverify}((v_{i+1}, v_{i+2}, \ldots, v_n), \{g \in G_i \bullet (g, \langle \mathit{true} \rangle)\}). \quad (14)$$

The proof of this theorem is included in Appendix A.

Returning to our running example, suppose that the database function on server D is removed from the system because the service workload no longer justifies maintaining two instances of the database. Since the verification step associated with this component in (11) is v_8, the sequence of verification steps that need to be redone is given by

$$\Delta cv = \mathit{reverify}((v_9), \{(\langle \mathcal{A}_{6_\mathrm{D}} \rangle_{\geq p_6}, \langle \mathit{true} \rangle)\}). \quad (15)$$

According to the *reverify* definition in (13), this is

$$\begin{aligned}
\Delta cv &= \mathit{reverify}((\{\langle\mathcal{A}_{3_\mathsf{A}}\rangle_{\geq p_3}, \langle\mathcal{A}_{4_\mathsf{A}}\rangle_{\geq p_4}, \langle\mathcal{A}_{5_\mathsf{A}}\rangle_{\geq p_5}, \langle\mathcal{A}_{3_\mathsf{B}}\rangle_{\geq p_3}, \langle\mathcal{A}_{4_\mathsf{B}}\rangle_{\geq p_4},\\
&\qquad\qquad \langle\mathcal{A}_{5_\mathsf{B}}\rangle_{\geq p_5}, \langle\mathcal{A}_{6_\mathsf{C}}\rangle_{\geq p_6}, \langle\mathcal{A}_{6_\mathsf{D}}\rangle_{\geq p_6}\}, M_{service}, \{\langle\mathcal{G}\rangle_{\geq p_\mathcal{G}}\}),\\
&\qquad \{(\langle\mathcal{A}_{6_\mathsf{D}}\rangle_{\geq p_6}, \langle true\rangle)\})\\
&= (\{\langle\mathcal{A}_{3_\mathsf{A}}\rangle_{\geq p_3}, \langle\mathcal{A}_{4_\mathsf{A}}\rangle_{\geq p_4}, \langle\mathcal{A}_{5_\mathsf{A}}\rangle_{\geq p_5}, \langle\mathcal{A}_{3_\mathsf{B}}\rangle_{\geq p_3}, \langle\mathcal{A}_{4_\mathsf{B}}\rangle_{\geq p_4}, \langle\mathcal{A}_{5_\mathsf{B}}\rangle_{\geq p_5},\\
&\qquad \langle\mathcal{A}_{6_\mathsf{C}}\rangle_{\geq p_6}, \langle true\rangle\}, M_{service}, \{\langle\mathcal{G}\rangle_{\geq p'_7}\}) \frown \mathit{reverify}((), \mathit{changes}')\\
&= (\{\langle\mathcal{A}_{3_\mathsf{A}}\rangle_{\geq p_3}, \langle\mathcal{A}_{4_\mathsf{A}}\rangle_{\geq p_4}, \langle\mathcal{A}_{5_\mathsf{A}}\rangle_{\geq p_5}, \langle\mathcal{A}_{3_\mathsf{B}}\rangle_{\geq p_3}, \langle\mathcal{A}_{4_\mathsf{B}}\rangle_{\geq p_4}, \langle\mathcal{A}_{5_\mathsf{B}}\rangle_{\geq p_5},\\
&\qquad \langle\mathcal{A}_{6_\mathsf{C}}\rangle_{\geq p_6}\}, M_{service}, \{\langle\mathcal{G}\rangle_{\geq p'_7}\}) \frown ()\\
&= (\{\langle\mathcal{A}_{3_\mathsf{A}}\rangle_{\geq p_3}, \langle\mathcal{A}_{4_\mathsf{A}}\rangle_{\geq p_4}, \langle\mathcal{A}_{5_\mathsf{A}}\rangle_{\geq p_5}, \langle\mathcal{A}_{3_\mathsf{B}}\rangle_{\geq p_3}, \langle\mathcal{A}_{4_\mathsf{B}}\rangle_{\geq p_4}, \langle\mathcal{A}_{5_\mathsf{B}}\rangle_{\geq p_5},\\
&\qquad \langle\mathcal{A}_{6_\mathsf{C}}\rangle_{\geq p_6}\}, M_{service}, \{\langle\mathcal{G}\rangle_{\geq p'_7}\})
\end{aligned} \tag{16}$$

where the probability bounds p_1 to p_7 are those in Table 1,

$$\begin{aligned}
p'_7 = mc(&\{\langle\mathcal{A}_{3_\mathsf{A}}\rangle_{\geq p_3}, \langle\mathcal{A}_{4_\mathsf{A}}\rangle_{\geq p_4}, \langle\mathcal{A}_{5_\mathsf{A}}\rangle_{\geq p_5}, \langle\mathcal{A}_{3_\mathsf{B}}\rangle_{\geq p_3}, \langle\mathcal{A}_{4_\mathsf{B}}\rangle_{\geq p_4}, \langle\mathcal{A}_{5_\mathsf{B}}\rangle_{\geq p_5},\\
&\langle\mathcal{A}_{6_\mathsf{C}}\rangle_{\geq p_6}\}, M_{service}, dfa(\langle\mathcal{G}\rangle_{\geq p_7}))
\end{aligned}$$

and

$$\mathit{changes}' = \begin{cases} \{(\langle\mathcal{A}_{6_\mathsf{D}}\rangle_{\geq p_6}, \langle true\rangle), (\langle\mathcal{G}\rangle_{\geq p_7}, \langle\mathcal{G}\rangle_{\geq p'_7})\}, & \text{if } p'_7 < p_7\\ \{(\langle\mathcal{A}_{6_\mathsf{D}}\rangle_{\geq p_6}, \langle true\rangle)\}, & \text{otherwise}\end{cases}$$

Redoing the only verification step in (16) yields $p'_7 = 0.949943$. Since $p'_7 < p_7 = 0.997482$ (cf. Table 1), $\mathit{changes}' = \{(\langle\mathcal{A}_{6_\mathsf{D}}\rangle_{\geq p_6}, \langle true\rangle), (\langle\mathcal{G}\rangle_{\geq p_7}, \langle\mathcal{G}\rangle_{\geq p'_7})\}$ (although this updated set of changes is not used in the recursive invocation of *reverify*, which is applied to the empty sequence of verification stepss).

3.3 LSCITS component change

Assume that the system component associated with model M_i, $1 \leq i \leq n$, from the verification step v_i of compositional verification task $cv = (v_1, v_2, \ldots, v_n)$ changed. The theorem below specifies the minimum sequence of verification steps that need redone to re-establish the properties corresponding to cv.

Theorem 2: The minimal sequence of verification steps that needs to be carried out to reverify a compositional verification task $cv = (v_1, v_2, \ldots, v_n)$ after a change in the component associated with its i-th verification step is

$$\begin{aligned}
\Delta cv = (A_i, M'_i, G'_i)&\frown\\
\mathit{reverify}(&(v_{i+1}, v_{i+2}, \ldots, v_n),\\
&\{(g, g') \in G_i \times G'_i \mid dfa(g) = dfa(g') \wedge prob(g') < prob(g)\}),
\end{aligned} \tag{17}$$

where M'_i represents the updated model for the changed system component and $G'_i = \{x \in \mathcal{P} \mid (\exists\, g \in G_i \bullet dfa(g) = dfa(x)) \wedge prob(x) = mc(A_i, M'_i, dfa(x))\}$.

Proof The proof is similar to that of Theorem 1.

□

To illustrate the application of the result in Theorem 2, suppose that the service functions running on Server A from our running example are redeployed on a different type of server (perhaps located in a different data centre). Suppose that the new server has $N'_{DISK} = 4$ disks instead of $N_{DISK} = 3$ disks for the server from our original scenario, but that the N'_{DISK} new disks are less reliable, i.e., $p'_{disk\_fail} = 0.0250$ compared to $p_{disk\_fail} = 0.0231$ previously. According to (17), the sequence of verification steps that need to be redone is

$$\begin{aligned}\Delta cv = (\{\langle true\rangle\}, M'_{server_{\mathsf{A}}}, \{\langle \mathcal{A}_{1_{\mathsf{A}}}\rangle_{\geq p'_1}, \langle \mathcal{A}_{2_{\mathsf{A}}}\rangle_{\geq p'_2}\})^\frown \\ \mathit{reverify}((v_2, v_3, \ldots, v_9), \mathit{changes}),\end{aligned}$$

where $M'_{server_{\mathsf{A}}}$ is the updated model for Server A,

$$p'_1 = mc(\{\langle true\rangle\}, M'_{server_{\mathsf{A}}}, \mathit{dfa}(\langle \mathcal{A}_{1_{\mathsf{A}}}\rangle_{\geq p_1})),$$
$$p'_2 = mc(\{\langle true\rangle\}, M'_{server_{\mathsf{A}}}, \mathit{dfa}(\langle \mathcal{A}_{2_{\mathsf{A}}}\rangle_{\geq p_2})),$$

the probabilities p_1 and p_2 are those in Table 1, v_2 to v_9 are defined in (11), and

$$\mathit{changes} = \{i \in \mathbb{N} \mid 1 \leq i \leq 2 \wedge p'_i < p_i \bullet (\langle \mathcal{A}_{i_{\mathsf{A}}}\rangle_{\geq p_i}, \langle \mathcal{A}_{i_{\mathsf{A}}}\rangle_{\geq p'_i})\}.$$

Executing the first verification step in Δcv yields

$p'_1 = 1 - 5.85\text{E-}8$ (which is larger than $p_1 = 0.999998$)
$p'_2 = 0.999984$ (which is larger than $p_2 = 0.999544$)

hence $\mathit{changes} = \{\}$ and, since $\mathit{reverify}((v_2, v_3, \ldots, v_9), \{\}) = ()$, no further verification step needs to be carried out.

3.4 LSCITS component joining

Suppose that a new component with model M_{new} joins the system. Re-establishing the probabilistic safety properties of the system requires updating any component models that depend on M_{new}, and carrying out verification steps for M_{new}, these updated models, and any other models whose verification steps include assumed properties that have changed. The minimal sequence of verification steps that need to be carried out is given by the theorem below.

Theorem 3: Let M_{new} be the model of a new component that joins a system for which a composition verification task $cv = (v_1, v_2, \ldots, v_n)$ was completed successfully before this operation. Also, let $M_{i_1}, M_{i_2}, \ldots, M_{i_m}$, $m > 0$, be the models of the components that depend on M_{new}, $1 \leq i_1 < i_2 < \ldots < i_m \leq n$, and assume that their updated versions reflecting the presence of the new component are $M'_{i_1}, M'_{i_2}, \ldots, M'_{i_m}$. Under these circumstances, the minimal sequence of verification steps that needs to be carried out to reverify cv is

$$\begin{aligned}\Delta cv = {} & (A_{new}, M_{new}, G_{new})^\frown \\ & \mathit{reverify}(((A'_{i_1}, M'_{i_1}, G_{i_1}), v_{i_1+1}, v_{i_1+2}, \ldots, v_{i_2-1}, \\ & \qquad (A'_{i_2}, M'_{i_2}, G_{i_2}), v_{i_2+1}, v_{i_2+2}, \ldots, v_{i_3-1}, \\ & \qquad \ldots \\ & \qquad (A'_{i_m}, M'_{i_m}, G_{i_m}), v_{i_m+1}, v_{i_m+2}, \ldots, v_n), \\ & \quad \{(g, g_{new}) \in \mathcal{P} \times G_{new} \mid \mathit{dfa}(g) = \mathit{dfa}(g_{new}) \wedge \mathit{prob}(g) = 0\}),\end{aligned} \tag{18}$$

where $A_{new} \subseteq \cup_{j=1}^{i_1-1} G_j \cup \{\langle true\rangle\}$ is the set of assumed properties for the verification of the new system component, and $A'_{i_j} \subseteq A_{i_j} \cup \{a \in \mathcal{P} \mid (\exists\, g \in G_{new} \bullet dfa(a) = dfa(g)) \wedge prob(a) = 0\}$ represents the new set of assumed properties for the model M'_{i_j}, $1 \leq j \leq m$. Note that $A'_{i_j} \setminus A_{i_j} \neq \{\}$ for all $1 \leq j \leq m$ since M'_{i_j} depends on M_{new}.

Proof We note first that the minimal sequence of verification steps must include the verification step for the new component, i.e., $(A_{new}, M_{new}, G_{new})$. Moreover, this step can appear at the beginning of the sequence since its assumed property set, A_{new}, consists of properties already established by the previously executed compositional verification task cv. We also note that the assumed property sets for the verification tasks v_1 to v_{i_1-1} are unchanged after the new component joined the system. Accordingly, the use of a sequence of verification steps that start at the i_1-th component as the first argument for the *reverify* invocation from (18) is correct. The rest of the proof shows that this invocation of *reverify* yields the sequence of verification steps required to re-establish the probabilistic safety properties in cv for the system components associated with the models M'_{i_1}, M_{i_1+1}, M_{i_1+2}, $\ldots$, M_{i_2-1}, M'_{i_2}, M_{i_2+1}, M_{i_2+2}, $\ldots$, M_{i_m-1}, M'_{i_m}, M_{i_m+1}, M_{i_m+2}, $\ldots$, M_n after the execution of the verification step $(A_{new}, M_{new}, G_{new})$. This part of the proof is similar to the proof of Theorem 1, and therefore not included in the paper.

$\square$

Returning to our running example, suppose that the service is augmented with a third database instance running on an additional server (Server E). The first verification step from $(v_1, v_2, \ldots, v_9)$ that is affected by this change is v_9, whose model needs to be updated to $M'_{service}$. The new verification step for the component that joined is

$$v_{new} = (A_{new}, M_{new}, G_{new}) = (\{\langle \mathcal{A}_{1_\mathsf{E}}\rangle_{\geq p_1}, \langle \mathcal{A}_{2_\mathsf{E}}\rangle_{\geq p_2}\}, M_{db_\mathsf{E}}, \{\langle \mathcal{A}_{6_\mathsf{E}}\rangle_{\geq p_6}\}),$$

so, according to Theorem 3,

$$\begin{aligned} \Delta cv = {} & (\{\langle \mathcal{A}_{1_\mathsf{E}}\rangle_{\geq p_1}, \langle \mathcal{A}_{2_\mathsf{E}}\rangle_{\geq p_2}\}, M_{db_\mathsf{E}}, \{\langle \mathcal{A}_{6_\mathsf{E}}\rangle_{\geq p_6}\})^\frown \\ & reverify(((A'_9, M'_{service}, \{\langle \mathcal{G}\rangle_{\geq p'_7}\})), \{(\langle \mathcal{A}_{6_\mathsf{E}}\rangle_{\geq 0}\}, \langle \mathcal{A}_{6_\mathsf{E}}\rangle_{\geq p_6}\})\}) \end{aligned}$$

with $A'_9 = \{\langle \mathcal{A}_{3_\mathsf{A}}\rangle_{\geq p_3}, \langle \mathcal{A}_{4_\mathsf{A}}\rangle_{\geq p_4}, \langle \mathcal{A}_{5_\mathsf{A}}\rangle_{\geq p_5}, \langle \mathcal{A}_{3_\mathsf{B}}\rangle_{\geq p_3}, \langle \mathcal{A}_{4_\mathsf{B}}\rangle_{\geq p_4}, \langle \mathcal{A}_{5_\mathsf{B}}\rangle_{\geq p_5}, \langle \mathcal{A}_{6_\mathsf{C}}\rangle_{\geq p_6}, \langle \mathcal{A}_{6_\mathsf{D}}\rangle_{\geq p_6}, \langle \mathcal{A}_{6_\mathsf{E}}\rangle_{\geq 0}\}$. As a result,

$$\begin{aligned} \Delta cv = {} & (\{\langle \mathcal{A}_{1_\mathsf{E}}\rangle_{\geq p_1}, \langle \mathcal{A}_{2_\mathsf{E}}\rangle_{\geq p_2}\}, M_{db_\mathsf{E}}, \{\langle \mathcal{A}_{6_\mathsf{E}}\rangle_{\geq p_6}\})^\frown \\ & (\{\langle \mathcal{A}_{3_\mathsf{A}}\rangle_{\geq p_3}, \langle \mathcal{A}_{4_\mathsf{A}}\rangle_{\geq p_4}, \langle \mathcal{A}_{5_\mathsf{A}}\rangle_{\geq p_5}, \langle \mathcal{A}_{3_\mathsf{B}}\rangle_{\geq p_3}, \langle \mathcal{A}_{4_\mathsf{B}}\rangle_{\geq p_4}, \\ & \quad \langle \mathcal{A}_{5_\mathsf{B}}\rangle_{\geq p_5}, \langle \mathcal{A}_{6_\mathsf{C}}\rangle_{\geq p_6}, \langle \mathcal{A}_{6_\mathsf{D}}\rangle_{\geq p_6}, \langle \mathcal{A}_{6_\mathsf{E}}\rangle_{\geq p_6}\}, M'_{service}, \{\langle \mathcal{G}\rangle_{\geq p'_7}\}) \end{aligned}$$

Carrying out the two verification steps yields $p_6 = 0.949954$ (as for the other database instances) and $p'_7 = 0.999861$.

3.5 LSCITS component choice

Assume that the functionality of the i-th system component, $1 \leq i \leq n$, can be provided by $m > 1$ new concrete implementations of this component, each characterised by different performance, reliability and cost. Let M_i^1, M_i^2, $\ldots M_i^m$ be the models associated with these functionally equivalent component implementations. Assume that the implementation that helps the system satisfy its requirements with minimum cost needs to be identified.

Theorem 4: The minimal sequence of verification steps that needs to be carried out to select the least expensive i-th component in the scenario described above is

$$\begin{aligned}
\mathit{\Delta cv} = {} & (A_i, M_i^1, G_i^1)^\frown \\
& \quad \mathit{reverify}((v_{i+1}, v_{i+2}, \ldots, v_n), \\
& \qquad \{(g, g') \in G_i \times G_i^1 \mid \mathit{dfa}(g) = \mathit{dfa}(g') \wedge \mathit{prob}(g') < \mathit{prob}(g)\})^\frown \\
& (A_i, M_i^2, G_i^2)^\frown \\
& \quad \mathit{reverify}((v_{i+1}, v_{i+2}, \ldots, v_n), \\
& \qquad \{(g, g') \in G_i \times G_i^2 \mid \mathit{dfa}(g) = \mathit{dfa}(g') \wedge \mathit{prob}(g') < \mathit{prob}(g)\})^\frown \\
& \ldots^\frown \\
& (A_i, M_i^m, G_i^m)^\frown \\
& \quad \mathit{reverify}((v_{i+1}, v_{i+2}, \ldots, v_n), \\
& \qquad \{(g, g') \in G_i \times G_i^m \mid \mathit{dfa}(g) = \mathit{dfa}(g') \wedge \mathit{prob}(g') < \mathit{prob}(g)\}),
\end{aligned} \tag{19}$$

where $G_i^j = \{x \in \mathcal{P} \mid (\exists\, g \in G_i \bullet \mathit{dfa}(g) = \mathit{dfa}(x)) \wedge \mathit{prob}(x) = \mathit{mc}(A_i, M_i^j, \mathit{dfa}(x))\}$ for $1 \leq i \leq m$.

Proof Selecting the least expensive component requires the independent examination of the effect of changing M_i with each of the models M_i^1, M_i^2, $\ldots M_i^m$, in order to identify the options that satisfy the requirements of the system. Therefore, the minimal sequence of verification steps is obtained by concatenating the minimal sequences of verification steps from Theorem 2 for models M_i^1 to M_i^m, as shown in (19).

$\square$

Returning again to our running example, suppose that the version of the virtualisation middleware installed on Server A from Figure 1 can be selected from three options. Assume that these options are associated with different levels of functionality/configurability, and with different levels of reliability, reflected in the probability p_{VM_fail} that a VM fails to operate correctly during a given time period:

1. The latest stable version of the virtualisation software, which is characterised by $p_{VM\_fail} = 0.05$ for a one-year time period. As shown by the probabilities annotating the state transitions associated with *vm_op* actions in Figure 4, this is the option used by model $M_{web+app_{\mathsf{A}}}$ from our case study.

2. The latest beta version of the virtualisation middleware, which provides the richest functionality and configurability, but which is also the least reliable, with $p_{VM\_fail} = 0.1$ for a one-year time period.
3. A highly reliable old version of the middleware that is characterised by $p_{VM\_fail} = 0.01$ over one year, but which lacks some of the monitoring capabilities of the other two options.

The last two options mentioned above correspond to two new models $M^1_{web+app_\mathsf{A}}$ and $M^2_{web+app_\mathsf{A}}$ for verification step v_5 from our compositional verification task cv from eq. (11). According to Theorem 4, the minimal sequence of verification steps required to assess the suitability these two new options is

$$\begin{aligned}
\Delta cv = {} & (\{\langle\mathcal{A}_{1_\mathsf{A}}\rangle_{\geq p_1}, \langle\mathcal{A}_{2_\mathsf{A}}\rangle_{\geq p_2}\}, M^1_{web+app_\mathsf{A}}, \{\langle\mathcal{A}_{3_\mathsf{A}}\rangle_{\geq p^1_3}, \langle\mathcal{A}_{4_\mathsf{A}}\rangle_{\geq p^1_4}, \langle\mathcal{A}_{5_\mathsf{A}}\rangle_{\geq p^1_5}\})^\frown \\
& \mathit{reverify}((v_6, v_7, v_8, v_9), \\
& \qquad \{(g, g') \in G_5 \times G^1_5 \mid \mathit{dfa}(g) = \mathit{dfa}(g') \wedge \mathit{prob}(g') < \mathit{prob}(g)\})^\frown \\
& (\{\langle\mathcal{A}_{1_\mathsf{A}}\rangle_{\geq p_1}, \langle\mathcal{A}_{2_\mathsf{A}}\rangle_{\geq p_2}\}, M^2_{web+app_\mathsf{A}}, \{\langle\mathcal{A}_{3_\mathsf{A}}\rangle_{\geq p^2_3}, \langle\mathcal{A}_{4_\mathsf{A}}\rangle_{\geq p^2_4}, \langle\mathcal{A}_{5_\mathsf{A}}\rangle_{\geq p^2_5}\})^\frown \\
& \mathit{reverify}((v_6, v_7, v_8, v_9), \\
& \qquad \{(g, g') \in G_5 \times G^2_5 \mid \mathit{dfa}(g) = \mathit{dfa}(g') \wedge \mathit{prob}(g') < \mathit{prob}(g)\})
\end{aligned}$$

Carrying out the two verification steps shown explicitly above yields: $p^1_3 = 0.999852$, $p^1_4 = p^1_5 = 0.989953$, $p^2_3 = 0.999952$ and $p^2_4 = p^2_5 = 0.999852$. Since the probability bounds for the original compositional verification task were $p_3 = 0.999946$ and $p_4 = p_5 = 0.997452$, we have

$$\begin{aligned}
\Delta cv = {} & (\{\langle\mathcal{A}_{1_\mathsf{A}}\rangle_{\geq p_1}, \langle\mathcal{A}_{2_\mathsf{A}}\rangle_{\geq p_2}\}, M^1_{web+app_\mathsf{A}}, \{\langle\mathcal{A}_{3_\mathsf{A}}\rangle_{\geq p^1_3}, \langle\mathcal{A}_{4_\mathsf{A}}\rangle_{\geq p^1_4}, \langle\mathcal{A}_{5_\mathsf{A}}\rangle_{\geq p^1_5}\})^\frown \\
& \mathit{reverify}((v_6, v_7, v_8, v_9), \\
& \quad \{(\langle\mathcal{A}_{3_\mathsf{A}}\rangle_{\geq p_3}, \langle\mathcal{A}_{3_\mathsf{A}}\rangle_{\geq p^1_3}), (\langle\mathcal{A}_{4_\mathsf{A}}\rangle_{\geq p_4}, \langle\mathcal{A}_{4_\mathsf{A}}\rangle_{\geq p^1_4}), (\langle\mathcal{A}_{5_\mathsf{A}}\rangle_{\geq p_5}, \langle\mathcal{A}_{5_\mathsf{A}}\rangle_{\geq p^1_5})\})^\frown \\
& (\{\langle\mathcal{A}_{1_\mathsf{A}}\rangle_{\geq p_1}, \langle\mathcal{A}_{2_\mathsf{A}}\rangle_{\geq p_2}\}, M^2_{web+app_\mathsf{A}}, \{\langle\mathcal{A}_{3_\mathsf{A}}\rangle_{\geq p^2_3}, \langle\mathcal{A}_{4_\mathsf{A}}\rangle_{\geq p^2_4}, \langle\mathcal{A}_{5_\mathsf{A}}\rangle_{\geq p^2_5}\})^\frown \\
& \mathit{reverify}((v_6, v_7, v_8, v_9), \{\}) \\
= {} & (\{\langle\mathcal{A}_{1_\mathsf{A}}\rangle_{\geq p_1}, \langle\mathcal{A}_{2_\mathsf{A}}\rangle_{\geq p_2}\}, M^1_{web+app_\mathsf{A}}, \{\langle\mathcal{A}_{3_\mathsf{A}}\rangle_{\geq p^1_3}, \langle\mathcal{A}_{4_\mathsf{A}}\rangle_{\geq p^1_4}, \langle\mathcal{A}_{5_\mathsf{A}}\rangle_{\geq p^1_5}\})^\frown \\
& (\{\langle\mathcal{A}_{3_\mathsf{A}}\rangle_{\geq p^1_3}, \langle\mathcal{A}_{4_\mathsf{A}}\rangle_{\geq p^1_4}, \langle\mathcal{A}_{5_\mathsf{A}}\rangle_{\geq p^1_5}, \langle\mathcal{A}_{3_\mathsf{B}}\rangle_{\geq p_3}, \langle\mathcal{A}_{4_\mathsf{B}}\rangle_{\geq p_4}, \\
& \quad \langle\mathcal{A}_{5_\mathsf{B}}\rangle_{\geq p_5}, \langle\mathcal{A}_{6_\mathsf{C}}\rangle_{\geq p_6}, \langle\mathcal{A}_{6_\mathsf{D}}\rangle_{\geq p_6}\}, M_{service}, \{\langle\mathcal{G}\rangle_{\geq p^1_7}\})^\frown \\
& (\{\langle\mathcal{A}_{1_\mathsf{A}}\rangle_{\geq p_1}, \langle\mathcal{A}_{2_\mathsf{A}}\rangle_{\geq p_2}\}, M^2_{web+app_\mathsf{A}}, \{\langle\mathcal{A}_{3_\mathsf{A}}\rangle_{\geq p^2_3}, \langle\mathcal{A}_{4_\mathsf{A}}\rangle_{\geq p^2_4}, \langle\mathcal{A}_{5_\mathsf{A}}\rangle_{\geq p^2_5}\})^\frown \\
& () \\
= {} & (\{\langle\mathcal{A}_{1_\mathsf{A}}\rangle_{\geq p_1}, \langle\mathcal{A}_{2_\mathsf{A}}\rangle_{\geq p_2}\}, M^1_{web+app_\mathsf{A}}, \{\langle\mathcal{A}_{3_\mathsf{A}}\rangle_{\geq p^1_3}, \langle\mathcal{A}_{4_\mathsf{A}}\rangle_{\geq p^1_4}, \langle\mathcal{A}_{5_\mathsf{A}}\rangle_{\geq p^1_5}\})^\frown \\
& (\{\langle\mathcal{A}_{3_\mathsf{A}}\rangle_{\geq p^1_3}, \langle\mathcal{A}_{4_\mathsf{A}}\rangle_{\geq p^1_4}, \langle\mathcal{A}_{5_\mathsf{A}}\rangle_{\geq p^1_5}, \langle\mathcal{A}_{3_\mathsf{B}}\rangle_{\geq p_3}, \langle\mathcal{A}_{4_\mathsf{B}}\rangle_{\geq p_4}, \\
& \quad \langle\mathcal{A}_{5_\mathsf{B}}\rangle_{\geq p_5}, \langle\mathcal{A}_{6_\mathsf{C}}\rangle_{\geq p_6}, \langle\mathcal{A}_{6_\mathsf{D}}\rangle_{\geq p_6}\}, M_{service}, \{\langle\mathcal{G}\rangle_{\geq p^1_7}\})^\frown \\
& (\{\langle\mathcal{A}_{1_\mathsf{A}}\rangle_{\geq p_1}, \langle\mathcal{A}_{2_\mathsf{A}}\rangle_{\geq p_2}\}, M^2_{web+app_\mathsf{A}}, \{\langle\mathcal{A}_{3_\mathsf{A}}\rangle_{\geq p^2_3}, \langle\mathcal{A}_{4_\mathsf{A}}\rangle_{\geq p^2_4}, \langle\mathcal{A}_{5_\mathsf{A}}\rangle_{\geq p^2_5}\}).
\end{aligned}$$

The only remaining verification step to carry out is the one in the middle, which yields $p^1_7 = 0.997494$, a value that is slightly lower than the probability bound $p_7 = 0.997482$ provided by the original choice of a virtualisation middleware version for Server A.

4 Conclusion and future work

Large-scale complex IT systems (LSCITS) are notoriously difficult to verify formally. Their extremely large state spaces, continual changes and nondeterministic behaviour challenge not only the scalability of existing verification techniques, but also the validity of the traditional approach of performing the verification offline, typically at design time. While an increasing number of compositional verification techniques address the scalability challenge, less research has explored the effect that continual change has on the verification of LSCITS.

This paper overviewed assume-guarantee compositional verification in the context of a case study from the area of cloud computing, and presented a formalism for specifying several classes of change that are common to LSCITS. We showed how this formalism can be used to generate the sequence of verification steps that need to be (re-)done after each type of change, and illustrated the application of this approach for several scenarios from our case study.

Our future work will focus on extending the change specification formalism to other classes of LSCITS change (e.g., changes in requirements), and on validating it in additional case studies. In the longer term, we envisage the integration of the approach with online learning techniques supporting change detection [7, 17] and with techniques for learning the assumptions for its sequence of compositional verification steps [15, 19].

Finally, an important challenge for our compositional reverification approach is the availability of suitable models for the components of the analysed LSCITS. In the work presented in this paper, we assumed that such models were available for all LSCITS components, including those joining the system "on the fly". Clearly, this assumption does not hold in many real-world scenarios. Significant future research is therefore needed to devise techniques that can learn these models from observations of the running system, or at least automate their synthesis from domain-specific descriptions of the LSCITS components.

Acknowledgements

This work was partly supported by the UK Engineering and Physical Sciences Research Council grant EP/H042644/1.

References

1. de Alfaro, L., Henzinger, T.A.: Interface automata. SIGSOFT Softw. Eng. Notes 26(5), 109–120 (Sep 2001), `http://doi.acm.org/10.1145/503271.503226`
2. Berezin, S., Campos, S.V.A., Clarke, E.M.: Compositional reasoning in model checking. In: Revised Lectures from the International Symposium on Compositionality: The Significant Difference. pp. 81–102. COMPOS'97, Springer-Verlag, London, UK (1998), `http://dl.acm.org/citation.cfm?id=646738.701964`
3. Blundell, C., Giannakopoulou, D., Pasareanu, C.S.: Assume-guarantee testing. ACM SIGSOFT Software Engineering Notes 31(2) (2006)

4. Calinescu, R.: General-purpose autonomic computing. In: Denko, M., et al. (eds.) Autonomic Computing and Networking. pp. 3–30. Springer (2009)
5. Calinescu, R., Ghezzi, C., Kwiatkowska, M., Mirandola, R.: Self-adaptive software needs quantitative verification at runtime. Communications of the ACM (September 2012)
6. Calinescu, R., Grunske, L., Kwiatkowska, M., Mirandola, R., Tamburrelli, G.: Dynamic QoS management and optimization in service-based systems. IEEE Transactions on Software Engineering 37, 387–409 (2011)
7. Calinescu, R., Johnson, K., Rafiq, Y.: Using observation ageing to improve Markovian model learning in QoS engineering. In: Proceedings 2nd ACM/SPEC International Conference on Performance Engineering. pp. 505–510 (2011)
8. Calinescu, R., Kikuchi, S., Kwiatkowska, M.: Formal methods for the development and verification of autonomic IT systems. In: Cong-Vinh, P. (ed.) Formal and Practical Aspects of Autonomic Computing and Networking: Specification, Development and Verification. pp. 1–37. IGI Global (2012)
9. Calinescu, R., Kwiatkowska, M.: Software engineering techniques for the development of systems of systems. In: Foundations of Computer Software: Future Trends and Techniques for Development. LNCS, vol. 6028, pp. 59–82. Springer (2010)
10. Calinescu, R., Kikuchi, S.: Formal methods @ runtime. In: Foundations of Computer Software. Modeling, Development, and Verification of Adaptive Systems, Lecture Notes in Computer Science, vol. 6662, pp. 122–135. Springer (2011)
11. Calinescu, R., Kwiatkowska, M.: CADS*: Computer-aided development of self-* systems. In: Chechik, M., Wirsing, M. (eds.) Fundamental Approaches to Software Engineering (FASE 2009). Lecture Notes in Computer Science, vol. 5503, pp. 421–424. Springer (March 2009), `http://qav.comlab.ox.ac.uk/papers/fase09.pdf`
12. Calinescu, R., Kwiatkowska, M.Z.: Using quantitative analysis to implement autonomic IT systems. In: 31st International Conference on Software Engineering. pp. 100–110 (2009), `http://dx.doi.org/10.1109/ICSE.2009.5070512`
13. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press (1999)
14. Clarke, E., Long, D., McMillan, K.: Compositional model checking. In: Proc. 4th Intl. Symp. Logic in Computer Science. pp. 353–362 (1989), `http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=39190`
15. Cobleigh, J.M., Giannakopoulou, D., Păsăreanu, C.S.: Learning assumptions for compositional verification. In: Proceedings of the 9th international conference on Tools and algorithms for the construction and analysis of systems. pp. 331–346. TACAS'03, Springer-Verlag, Berlin, Heidelberg (2003), `http://dl.acm.org/citation.cfm?id=1765871.1765903`
16. Dikaiakos, M.D., Katsaros, D., Mehra, P., Pallis, G., Vakali, A.: Cloud computing: Distributed internet computing for it and scientific research. IEEE Internet Computing 13(5), 10–13 (September–October 2009)
17. Epifani, I., Ghezzi, C., Mirandola, R., Tamburrelli, G.: Model evolution by runtime adaptation. In: Proceedings of the 31st International Conference on Software Engineering. pp. 111–121. IEEE Computer Society (2009)
18. Etessami, K., Kwiatkowska, M., Vardi, M., Yannakakis, M.: Multi-objective model checking of Markov decision processes. In: Grumberg, O., Huth, M. (eds.) Proc. 13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'07). LNCS, vol. 4424, pp. 50–65. Springer (2007)
19. Feng, L., Kwiatkowska, M.Z., Parker, D.: Automated learning of probabilistic assumptions for compositional reasoning. In: Fundamental Approaches to Software Engineering - 14th International Conference, FASE 2011. pp. 2–17 (2011)

20. Filieri, A., Ghezzi, C., Tamburrelli, G.: A formal approach to adaptive software: continuous assurance of non-functional requirements. Formal Asp. Comput. 24(2), 163–186 (2012)
21. Grumberg, O., Long, D.E.: Model checking and modular verification. ACM Trans. Program. Lang. Syst. 16(3), 843–871 (May 1994), http://doi.acm.org/10.1145/177492.177725
22. Hinton, A., Kwiatkowska, M., Norman, G., Parker, D.: PRISM: A tool for automatic verification of probabilistic systems. In: Hermanns, H., Palsberg, J. (eds.) Proc. 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06). LNCS, vol. 3920, pp. 441–444. Springer (2006)
23. Hoare, C.A.R.: An axiomatic basis for computer programming. Commun. ACM 12(10), 576–580 (Oct 1969), http://doi.acm.org/10.1145/363235.363259
24. Inverardi, P., Patrizio, Tivoli, M.: Towards an assume-guarantee theory for adaptable systems. In: Proceedings of the Software Engineering for Adaptive and Self-Managing Systems Workshop (SEAMS). pp. 106–115 (2009)
25. Kesten, Y., Pnueli, A.: A compositional approach to ctl* verification. Theor. Comput. Sci. 331(2-3), 397–428 (Feb 2005), http://dx.doi.org/10.1016/j.tcs.2004.09.023
26. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) Proc. 23rd International Conference on Computer Aided Verification (CAV'11). LNCS, vol. 6806, pp. 585–591. Springer (2011)
27. Kwiatkowska, M., Norman, G., Parker, D., Qu, H.: Assume-guarantee verification for probabilistic systems. In: Esparza, J., Majumdar, R. (eds.) Proc. 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'10). LNCS, vol. 6105, pp. 23–37. Springer (2010), http://qav.cs.ox.ac.uk/bibitem.php?key=KNPQ10
28. Northrop, L., et al.: Ultra-large-scale systems - the software challenge of the future. Tech. rep., Software Engineering Institute, Carnegie Mellon University (June 2006)
29. Pnueli, A.: In transition from global to modular temporal reasoning about programs. In: Apt, K.R. (ed.) Logics and models of concurrent systems, pp. 123–144. Springer-Verlag New York, Inc., New York, NY, USA (1985), http://dl.acm.org/citation.cfm?id=101969.101977
30. Segala, R., Lynch, N.A.: Probabilistic simulations for probabilistic processes. Nord. J. Comput. 2(2), 250–273 (1995)
31. Sommerville, I., Cliff, D., Calinescu, R., Keen, J., Kelly, T., Kwiatkowska, M., McDermid, J., Paige, R.: Large-scale complex IT systems. Communications of the ACM 55(7), 71–77 (July 2012)
32. Thomas, K.: Solid state drives no better than others, survey says. http://www.pcworld.com/businesscenter/article/213442/solid_state_drives_no_better_than_others_survey_says.html
33. Tordsson, J., Montero, R.S., Moreno-Vozmediano, R., Llorente, I.M.: Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers. Future Generation Computer Systems 28(2), 358 – 367 (2012)
34. Vishwanath, K.V., Nagappan, N.: Characterizing cloud computing hardware reliability. In: Proceedings of the 1st ACM symposium on Cloud computing. pp. 193–204. SoCC '10, ACM, New York, NY, USA (2010), http://doi.acm.org/10.1145/1807128.1807161

Appendix A

A.1 Additional probabilistic assume-guarantee proof rule

The proposition below introduces the assume-guarantee proof rule (#) that we used in eq. (9). To prove the rule we use the following additional notation:

– $Pr_{M_i}^{\sigma_i}(A_i)$ represents the probability that model M_i satisfies the safety property A_i for a fixed adversary $\sigma_i \in Adv_i$.
– Given an adversary $\sigma \in Adv_{M_1 \| M_2 \| \ldots \| M_x}$, $\sigma \upharpoonright_{M_i} \in Adv_i$ denotes the projection of σ onto M_i, $1 \leq i \leq x$.

Proposition 1: If M_1, M_2, ..., M_k are probabilistic automata, and $\langle A_1 \rangle_{\geq p_1}$, $\langle A_2 \rangle_{\geq p_2}$, ..., $\langle A_k \rangle_{\geq p_k}$ are probabilistic safety properties such that $\alpha_{A_i} \subseteq \alpha_{M_i}$ for all $1 \leq i \leq k-1$ and $\alpha_{A_k} \subseteq \alpha_{M_k} \cup \alpha_{A_1} \cup \alpha_{A_2} \cup \ldots \cup \alpha_{A_{k-1}}$, then the following proof rule holds:

$$\frac{\begin{array}{c}\langle true \rangle M_1 \langle A_1 \rangle_{\geq p_1} \\ \langle true \rangle M_2 \langle A_2 \rangle_{\geq p_2} \\ \ldots \\ \langle true \rangle M_{k-1} \langle A_{k-1} \rangle_{\geq p_{k-1}} \\ \langle A_1, A_2, \ldots, A_{k-1} \rangle_{\geq p_1, p_2, \ldots, p_{k-1}} M_k \langle A_k \rangle_{\geq p_k}\end{array}}{\langle true \rangle M_1 \parallel M_2 \parallel \ldots \parallel M_k \langle A_k \rangle_{\geq p_k}} \tag{20}$$

Proof Starting from the hypothesis, we have:

$\forall\, i \in \{1, 2, \ldots, k-1\} \bullet \forall\, \sigma_i \in Adv_{M_i} \bullet Pr_{M_i}^{\sigma_i}(A_i) \geq p_i$
(according to the definition of $\langle true \rangle M_i \langle A_i \rangle_{\geq p_i}$)
$\Rightarrow$
$\forall\, i \in \{1, 2, \ldots, k-1\} \bullet \forall\, \sigma \in Adv_{M_1 \| M_2 \| \ldots \| M_{k-1}} \bullet Pr_{M_i}^{\sigma \upharpoonright_{M_i}}(A_i) \geq p_i$
(since $\sigma \upharpoonright_{M_i} \in Adv_{M_i}$)
$\Rightarrow$
$\forall\, i \in \{1, 2, \ldots, k-1\} \bullet \forall\, \sigma \in Adv_{M_1 \| M_2 \| \ldots \| M_{k-1}} \bullet Pr_{M_1 \| M_2 \| \ldots \| M_{k-1}}^{\sigma}(A_i) \geq p_i$
(by part (a) of Lemma 1 from [27], since $\alpha_{A_i} \subseteq \alpha_{M_i}$)
$\Rightarrow$
$\forall\, \sigma \in Adv_{M_1 \| M_2 \| \ldots \| M_{k-1}} \bullet Pr_{M_1 \| M_2 \| \ldots \| M_{k-1}}^{\sigma}(A_1) \geq p_1 \wedge \ldots \wedge$
$\wedge Pr_{M_1 \| M_2 \| \ldots \| M_{k-1}}^{\sigma}(A_{k-1}) \geq p_{k-1}$
(rewrite of the previous step)
$\Rightarrow$
$\langle true \rangle M_1 \| M_2 \| \ldots \| M_{k-1} \langle A_1, A_2, \ldots, A_{k-1} \rangle_{\geq p_1, p_2, \ldots, p_{k-1}}$
(by definition)
$\Rightarrow$
$\langle true \rangle M_1 \parallel M_2 \parallel \ldots \parallel M_k \langle A_k \rangle_{\geq p_k}$
(by applying the ASym-Mult rule from [27])

□

A.2. Proof of Theorem 1

Theorem 1: The minimal sequence of verification steps that needs to be carried out to reverify a compositional verification task $cv = (v_1, v_2, \ldots, v_n)$ after the failure of the component associated with its i-th verification step is

$$\Delta cv = \mathit{reverify}((v_{i+1}, v_{i+2}, \ldots, v_n), \{g \in G_i \bullet (g, \langle \mathit{true} \rangle)\}). \tag{21}$$

Proof We start by observing that, according to the recursive definition of *reverify* from (13), Δcv can be rewritten as:

$$\begin{aligned} \Delta cv &= \Delta cv_1 \frown \mathit{reverify}((v_{i+1}, v_{i+2}, \ldots, v_n), \mathit{changes}_0) \\ &= \Delta cv_2 \frown \mathit{reverify}((v_{i+2}, v_{i+3}, \ldots, v_n), \mathit{changes}_1) \\ &= \ldots \\ &= \Delta cv_{n-i+1} \frown \mathit{reverify}((), \mathit{changes}_{n-i+1}), \end{aligned} \tag{22}$$

where $\Delta cv_1 = ()$, $\mathit{changes}_1 = \{g \in G_i \bullet (g, \langle \mathit{true} \rangle)\}$ and, for $1 < j \leq n - i + 1$, Δc_j and $\mathit{changes}_j$ are obtained by carrying out the calculations defined by (13). We will prove the following intermediate results by induction on the value of j:

1. Δc_j is the minimal sequence of verification steps required to re-establish the probabilistic safety properties associated with the first $i + j - 1$ elements of cv;
2. $\mathit{changes}_j$ is the set of all changes in the probabilistic safety properties guaranteed by models M_1, M_2, $\ldots$, M_{i+j-1},

for $j = 1, 2, \ldots, n - i + 1$. The theorem will then follow immediately from the fact that Δcv_{n-i+1} is "the minimal sequence of verification steps required to re-establish the probabilistic safety properties associated with the first $i + (n - i + 1) - 1 = n$ elements of cv", since $\Delta cv = \Delta cv_{n-i+1} \frown \mathit{reverify}((), \mathit{changes}_{n-i+1}) = \Delta cv_{n-i+1} \frown () = \Delta cv_{n-i+1}$.

The base case, corresponding to $j = 1$, is straightforward:

1. $\Delta cv_1 = ()$ since the verification steps v_1, v_2, $\ldots$, v_{i-1} do not need to be redone (as their assumption sets A_1, A_2, $\ldots$, A_{i-1} do not contain any properties guaranteed by the failed component), and v_i does not need to be redone (because we already know that it corresponds to the failed component);
2. $\mathit{changes}_1 = \{g \in G_i \bullet (g, \langle \mathit{true} \rangle)\}$ since none of the properties guaranteed by M_1, M_2, $\ldots$, M_{i-1} has changed, and all properties in G_i, which were guaranteed by the failed component, need to be replaced with the property that does not offer any guarantees, i.e., $\langle \mathit{true} \rangle$.

Suppose now that Δc_j and $\mathit{changes}_j$ satisfy our two properties for a value of j such that $1 \leq j < n - i + 1$. We will prove that the two properties are also satisfied by Δc_{j+1} and $\mathit{changes}_{j+1}$.

According to the notation introduced in (22) and to the definition of *reverify* from (13),

$$\Delta c_{j+1} = \Delta c_j \frown \begin{cases} (), & \text{if } A_{i+j} \cap \mathit{changes}_j = \emptyset \\ (A'_{i+j}, M, G'_{i+j}), & \text{otherwise} \end{cases} \tag{23}$$

and

$$\mathit{changes}_{j+1} = \mathit{changes}_j \cup \begin{cases} \{\} & \text{if } A_{i+j} \cap \mathit{changes}_j = \emptyset \\ \{(g, g') \in G_{i+j} \times G'_{i+j} \mid \mathit{dfa}(g) = \mathit{dfa}(g') \wedge \\ \qquad \mathit{prob}(g') < \mathit{prob}(g)\} & \text{otherwise} \end{cases} \tag{24}$$

where

- $A'_{i+j} = \{a \in A_{i+j} \mid \neg(\exists (g, g') \in \mathit{changes}_j \bullet a = g)\} \cup \{(g, g') \in \mathit{changes}_j \mid g \in A_{i+j} \bullet g'\}$;
- $G'_{i+j} = \{x \in \mathcal{P} \mid (\exists\, g \in G_{i+j} \bullet \mathit{dfa}(x) = \mathit{dfa}(g)) \wedge \mathit{prob}(x) = \mathit{mc}(A'_{i+j}, M_{i+j}, \mathit{dfa}(g))\}$.

We analyse each of the two cases above in turn, recalling the fact that, according to the inductive hypothesis, $\mathit{changes}_j$ is the set of all changes in the probabilistic safety properties guaranteed by models M_1 to M_{i+j-1}:

- The case $A_{i+j} \cap \mathit{changes}_j = \emptyset$ corresponds to the scenario in which the assumptions for the verification step v_{i+j} are unchanged. Therefore, the minimal sequence of verification steps for models M_1 to M_{i+j} coincides in this case with the minimal sequence of verification steps for models M_1 to M_{i+j-1}, i.e., with Δcv_j (according to the inductive hypothesis). Since $\Delta cv_{j+1} = \Delta cv_j$, it follows that, in this case, Δcv_{j+1} satisfies the first required property. Finally, no probabilistic safety properties guaranteed by M_{i+j} changed, so $\mathit{changes}_{j+1} = \mathit{chages}_j$ satisfies the second required property.
- The second case (i.e., $A_{i+j} \cap \mathit{changes}_j \neq \emptyset$) corresponds to the scenario in which at least one of the assumptions for the verification step v_{i+j} changed, hence the model M_{i+j} needs to be reverified against the updated set of assumptions A'_{i+j} defined above, yielding the new guaranteed probabilistic safety properties in G'_{i+j}. The minimal set of verification steps to be redone for models M_1 to M_{i+j} consists of all the verification steps that need to be redone for models M_1 to M_{i+j-1} (i.e., Δcv_j) and the additional step $(A'_{i+j}, M_{i+j}, G'_{i+j})$. This is precisely Δcv_{j+1}, so the first required property is also satisfied in the second case. Finally, we note again that $\mathit{changes}_j$ the set of all changes in the probabilistic safety properties guaranteed by M_1 to M_{i+j-1}, and that the set $\{(g, g') \in G_{i+j} \times G'_{i+j} \mid \mathit{dfa}(g) = \mathit{dfa}(g') \wedge \mathit{prob}(g') < \mathit{prob}(g)\}$ contains precisely the changes to the probabilistic safety properties guaranteed by M_{i+j}. Therefore, the union of these two sets (i.e., $\mathit{changes}_{j+1}$) represents the set of all changes in the probabilistic safety properties guaranteed by models M_1 to M_{i+j}.

□