Automated Learning of Probabilistic Assumptions for Compositional Reasoning

Marta Kwiatkowska
Oxford University Computing Laboratory

FASE’11, Saarbrücken, April 2011

Joint work with: Lu Feng, Dave Parker, Gethin Norman, Hongyang Qu
Probabilistic verification

- **Probabilistic verification**
  - formal verification of systems exhibiting stochastic behaviour

- **Why probability?**
  - unreliability (e.g. component failures)
  - uncertainty (e.g. message losses/delays over wireless)
  - randomisation (e.g. in protocols such as Bluetooth, ZigBee)

- **Quantitative properties**
  - reliability, performance, quality of service, ...
  - “the probability of an airbag failing to deploy within 0.02s”
  - “the expected time for a network protocol to send a packet”
  - “the expected power usage of a sensor network over 1 hour”
Model checking

Automated formal verification for finite-state models

System → Finite-state model → Model checker (e.g., SMV, Spin) → Result

System requirements → Temporal logic specification → ¬EF fail → Counter-example
Probabilistic model checking

Automatic verification of systems with probabilistic behaviour

System

Probabilistic model
e.g. Markov chain

System requirements

Probabilistic temporal logic specification
e.g. PCTL, CSL, LTL

Result

Quantitative results

Counter-example

P < 0.1 [ F fail ]

Probabilistic model checker
e.g. PRISM, MRMC
Probabilistic model checking

• First algorithms proposed in 1980s
  – [Vardi, Courcoubetis, Yannakakis, …]
  – algorithms [Hansson, Jonsson, de Alfaro] & first implementations

• 2000: tools ETMCC (MRMC) & PRISM released
  – PRISM: efficient extensions of symbolic model checking
  – ETMCC (now MRMC): model checking for continuous–time Markov chains [Baier, Hermanns, Havorkort, Katoen, …]

• Selected advances in probabilistic model checking:
  – compositional verification [Segala, Lynch, Stoelinga, Vaandrager, …]
  – probabilistic counterexample generation [Han/Katoen, Leue, …]
  – abstraction (and CEGAR) for probabilistic models
    · [Larsen, Hermanns, Wolf, Kwiatkowska, … ]
  – and much more…
Probabilistic model checking in action

- **Bluetooth device discovery protocol**
  - frequency hopping, randomised delays
  - low-level model in PRISM, based on detailed Bluetooth reference documentation
  - numerical solution of 32 Markov chains, each approximately 3 billion states
  - analysed performance, identified worst-case scenarios

- **Fibroblast Growth Factor (FGF) pathway**
  - complex biological cell signalling pathway, key roles e.g. in healing, not yet fully understood
  - model checking (PRISM) & simulation (stochastic $\pi$-calculus), in collaboration with Biosciences at Birmingham
  - “in-silico” experiments: systematic removal of components
  - behavioural predictions later validated by lab experiments
Probabilistic model checking

• What’s involved
  – specifying, constructing probabilistic models
  – graph-based analysis: reachability + qualitative verification
  – numerical solution, e.g. linear equations/linear programming

• The state of the art
  – fast/efficient techniques for a range of probabilistic models
  – (mostly Markov chains, Markov decision processes)
  – feasible for models of up to \(10^7\) states \((10^{10} \text{ with symbolic})\)
  – tool support exists and is widely used, e.g. PRISM, MRMC
  – successfully applied to many application domains:
    • distributed randomised algorithms, communication protocols, security protocols, biological systems, quantum cryptography, …
Probabilistic model checking

• Some observations
  – probabilistic model checking typically more expensive than the non-probabilistic case: need to build and solve model
  – most useful kinds results are quantitative (e.g. probability values/bounds) – study trends, find anomalies, …
  – successfully used by non-experts for many application domains, but full automation and good tool support essential

• Some key challenges
  – scalability and efficiency: larger models, verified faster
  – more realistic models (real-time behaviour, continuous dynamics, stochastic hybrid systems) and languages
  – beyond model checking: parametric methods, synthesis, …

• This talk: scalability/efficiency via compositional reasoning
Overview

• Probabilistic model checking
  – probabilistic models: probabilistic automata
  – property specifications: probabilistic safety properties
  – multi-objective model checking

• Compositional probabilistic verification
  – assume-guarantee reasoning
  – assume-guarantee for probabilistic systems
  – implementation & results

• Automated generation of assumptions
  – L* and its application to compositional verification
  – generating probabilistic assumptions
  – implementation, results & recent progress

• Conclusions
Probabilistic models

• **Discrete–time Markov chains (DTMCs)**
  – discrete states + **probability**
  – for: randomisation, component failures, unreliable media

• **Markov decision processes (MDPs)**

• **Probabilistic automata (PAs)** [Segala]
  – discrete states + probability + **nondeterminism**
  – for: concurrency, control, under-specification, abstraction

• **Continuous–time Markov chains (CTMCs)**

• **Probabilistic timed automata (PTAs)**
  – and many other variants…
  – add notions of **real–time** behaviour to the above models
Probabilistic automata (PAs)

- Model nondeterministic as well as probabilistic behaviour
  - very similar to Markov decision processes (MDPs)

- A probabilistic automaton is a tuple \( M = (S, s_{\text{init}}, \alpha_M, \delta_M) \):
  - \( S \) is the state space
  - \( s_{\text{init}} \in S \) is the initial state
  - \( \alpha_M \) is the action alphabet
  - \( \delta_M \subseteq S \times \alpha_M \times \text{Dist}(S) \) is the transition probability relation
  - \( \text{Dist}(S) \) is set of all probability distributions over set \( S \)

- Parallel composition: \( M_1 \parallel M_2 \)
  - CSP style – synchronise over common actions
Probabilistic model checking for PAs

• To reason formally about PAs, we use adversaries
  – an adversary $\sigma$ resolves nondeterminism in a PA $M$
  – also called “scheduler”, “strategy”, “policy”, ...
  – makes a (possibly randomised) choice, based on history
  – induces probability measure $Pr_{M,\sigma}$ over (infinite) paths

• Property specifications (linear–time)
  – specify some measurable property $\phi$ of paths (e.g. in LTL)
  – $Pr_{M,\sigma}(\phi)$ gives probability of $\phi$ under adversary $\sigma$
  – best-/worst–case analysis: quantify over all adversaries
  – e.g. $M \models P_{\geq p}[\Box(req \rightarrow \Diamond ack)] \iff Pr_{M,\sigma}(\Box(req \rightarrow \Diamond ack)) \geq p$ for all $\sigma$
  – or just compute e.g. $Pr_{M,\min}(\phi) = \inf \{ Pr_{M,\sigma}(\phi) \mid \sigma \in \text{Adv}_M \}$
  – efficient algorithms and tools exist
  – (but scalability is always an issue)
Running example

- Two components, each a probabilistic automaton:
  - $M_1$: sensor – detects fault and sends warn/shutdown signals
  - $M_2$: device to be shut down (may fail if no warning sent)
Running example

Parallel composition: $M_1 \ || \ M_2$
Safety properties

- Safety property: language of infinite words (over actions)
  - characterised by a set of “bad prefixes” (or “finite violations”)
  - i.e. finite words of which any extension violates the property

- Regular safety property
  - bad prefixes are represented by a regular language
  - property $A$ represented by an error automaton $A_{err}$, a deterministic finite automaton (DFA) storing bad prefixes

```
\begin{align*}
\text{q}_0 & \quad \text{fail} \\
\text{q}_1 & \quad \text{q}_0 \quad \text{warn, shutdown} \\
\text{\textcolor{red}{\text{q}}}_1 & \quad \textcolor{red}{\text{\text{q}}}_1 \quad \text{warn, shutdown} \\
\text{\textcolor{red}{\text{q}}}_2 & \quad \text{end} \quad \text{time} \quad \text{end} \\
\end{align*}
```

- “a fail action never occurs”
- “warn occurs before shutdown”
- “at most 2 time steps pass before termination”
A probabilistic safety property $P_{\geq p}[A]$ comprises

- a regular safety property $A$
- a rational probability bound $p$
- “the (minimum) probability of satisfying $A$ must be at least $p$”

$M \models P_{\geq p}[A] \iff \Pr_M^{\sigma}(A) \geq p$ for all $\sigma \in \text{Adv}_M \iff \Pr_M^{\min}(A) \geq p$
- or “the (max.) probability of violating $A$ must be at most $1-p$”

Examples:

- “$\text{warn}$ occurs before $\text{shutdown}$ with probability at least 0.8”
- “the probability of a failure occurring is at most 0.02”
- “probability of terminating within $k$ time-steps is at least 0.75”

Model checking:

- construct (synchronous) PA–DFA product $M \otimes A_{\text{err}}$
- compute probability of reaching “accept” in product PA
• Does probabilistic safety property $P_{\geq 0.8} [A]$ hold in $M_1$?
• Does probabilistic safety property $P_{\geq 0.8} [A]$ hold in $M_1$?

Product PA $M_1 \otimes A_{\text{err}}$

$\Pr_{M_1} \min(A) = 1 - 0.2 = 0.8$

$\rightarrow M_1 \models P_{\geq 0.8} [A]$
Multi-objective PA model checking

- Study trade-off between several different objectives
  - existential queries: does there exist adversary $\sigma$ such that:
    - $\Pr_{M^\sigma}(\square (\text{queue\_size} < 10)) > 0.99 \land \Pr_{M^\sigma}(\Diamond \text{flat\_battery}) < 0.01$
  - useful for synthesising controllers

- Multi-objective PA model checking
  - [Etessami/Kwiatkowska/Vardi/Yannakakis, TACAS’07]
  - LTL formulae $\Phi_1,...,\Phi_k$ and probability bounds $\sim_1 p_1,...,\sim_k p_k$
  - check if $\exists \sigma \in \text{Adv}_M$ s.t. $\Pr_{M^\sigma}(\Phi_1) \sim_1 p_1 \land ... \land \Pr_{M^\sigma}(\Phi_k) \sim_k p_k$
  - construct product of automata for $M, \Phi_1,...,\Phi_k$
  - then solve linear programming (LP) problem
  - the resulting adversary $\sigma$ can obtained from LP solution
  - note: $\sigma$ may be randomised (unlike the single objective case)
Consider the two objectives ◊D and ◊E in the PA below
- i.e. the trade-off between the probabilities \( \Pr(◊D) \) and \( \Pr(◊E) \)
- an adversary resolves the choice between a/b/c
- increasing the probability of reaching one target decreases the probability of reaching the other
Multi-objective PA model checking

- Need to consider all randomised adversaries
  - for example, is there an adversary $\sigma$ such that:
  - $\Pr(\Diamond D) > 0.2 \land \Pr(\Diamond E) > 0.6$
Overview

• Probabilistic model checking
  – probabilistic automata
  – property specification + probabilistic safety properties
  – multi-objective model checking

• Compositional probabilistic verification
  – assume–guarantee reasoning
  – assume–guarantee for probabilistic systems
  – implementation & results

• Automated generation of assumptions
  – $L^*$ and its application to compositional verification
  – generating probabilistic assumptions
  – implementation & results

• Conclusions, current & future work
Compositional verification

• **Goal**: scalability through modular verification
  - e.g. decide if $M_1 || M_2 \models G$
  - by analysing $M_1$ and $M_2$ separately

• **Assume–guarantee (A/G) reasoning**
  - use assumption $A$ about the context of a component $M_2$
  - $\langle A \rangle M_2 \langle G \rangle$ – “whenever $M_2$ is part of a system satisfying $A$, then the system must also guarantee $G$”
  - example of asymmetric (non-circular) A/G rule:

$$
\begin{align*}
M_1 &\models A \\
\langle A \rangle M_2 \langle G \rangle &
\end{align*}
$$

\[ M_1 || M_2 \models G \]

[ Pasareanu/Giannakopoulou/et al. ]
AG rules for probabilistic systems

• How to formulate AG rules for probabilistic automata?

• Key questions:
  – 1. What form do assumptions $A$ take?
    • needs to be compositional
    • needs to be efficient to check
    • needs to allow compact assumptions

  – 2. How do we generate suitable assumptions?
    • preferably in a fully automated fashion

  – 3. Can we get “quantitative” results?
    • i.e. numerical values, rather than “yes”/”no”
A/G rules for probabilistic systems

• How to formulate A/G rules for probabilistic automata?

• Key questions:
  
  – 1. What form do assumptions A take?
    - needs to be compositional
    - needs to be efficient to check
    - needs to allow compact assumptions

▷ various compositional relations exist
  - e.g. strong/weak (probabilistic) (bi)simulation
  - but these are either too fine (difficult to get small assumptions) or expensive to check

▷ here, we use: probabilistic safety properties [TACAS’10]
  - less expressive, but compact and efficient
  - (see also generalisation to liveness/rewards [TACAS’11])
A/G rules for probabilistic systems

• How to formulate A/G rules for probabilistic automata?

• Key questions:
  - 2. How do we generate suitable assumptions?
    - preferably in a fully automated fashion
      ▶ algorithmic learning (based on L* algorithm)
        adapt techniques for (non-probabilistic) assumptions
  - 3. Can we get “quantitative” results?
    - i.e. numerical values, rather than “yes”/”no”
      ▶ yes: generate lower/upper bounds on probabilities
Probabilistic assume guarantee

- Assume–guarantee triples $\langle A \rangle \geq p_A \ M \langle G \rangle \geq p_G$ where:
  - $M$ is a probabilistic automaton
  - $P \geq p_A[A]$ and $P \geq p_G[G]$ are probabilistic safety properties

- Informally:
  - “whenever $M$ is part of a system satisfying $A$ with probability at least $p_A$, then the system is guaranteed to satisfy $G$ with probability at least $p_G$”

- Formally:
  - $\forall \sigma \in \text{Adv}_M, ( \Pr_{M,\sigma}(A) \geq p_A \rightarrow \Pr_{M,\sigma}(G) \geq p_G )$
  - where $M'$ is $M$ with its alphabet extended to include $\alpha_A$
  - reduces to multi–objective model checking on $M'$
  - look for adversary satisfying assumption but not guarantee
  - i.e. can check $\langle A \rangle \geq p_A \ M \langle G \rangle \geq p_G$ efficiently via LP problem
An assume–guarantee rule

- The following asymmetric proof rule holds
  - (asymmetric = uses one assumption about one component)

\[
\begin{align*}
M_1 & \models P_{\geq p_A} [A] \\
\langle A \rangle_{\geq p_A} M_2 \langle G \rangle_{\geq p_G} & \\
M_1 \parallel M_2 & \models P_{\geq p_G} [G]
\end{align*}
\]

(ASYM)

- So, verifying \( M_1 \parallel M_2 \models P_{\geq p_G} [G] \) requires:
  - premise 1: \( M_1 \models P_{\geq p_A} [A] \) (standard model checking)
  - premise 2: \( \langle A \rangle_{\geq p_A} M_2 \langle G \rangle_{\geq p_G} \) (multi-objective model checking)

- Potentially much cheaper if \(|A|\) much smaller than \( |M_1| \)
Running example

- Does probabilistic safety property $P_{\geq 0.98} [G]$ hold in $M_1 || M_2$?

PA $M_1$ ("sensor")

- $s_0$ to $s_1$: detect
- $s_1$ to $s_2$: warn
- $s_2$ to $s_3$: shutdown
- $s_3$: off

PA $M_2$ ("device")

- $t_0$: warn
- $t_0$: shutdown
- $t_1$: shutdown
- $t_2$: off
- $t_3$: fail

$G$ ("a fail action never occurs")

- $q_0$: fail
- $q_1$: fail
Running example

• Does probabilistic safety property \( P_{\geq 0.98} [G] \) hold in \( M_1 \parallel M_2 \)?

PA \( M_1 \) ("sensor")

\[ s_0 \xrightarrow{0.8} s_1 \xrightarrow{0.2} s_3 \xrightarrow{\text{shut}} s_0 \]

PA \( M_2 \) ("device")

\[ t_0 \xrightarrow{\text{warn}} t_1 \xrightarrow{0.9} t_2 \xrightarrow{\text{shut}} t_2 \]

• Use A/G with assumption \( P_{\geq 0.8} [A] \) about \( M_1 \)

\[ M_1 \models P_{\geq 0.8} [A] \]

\[ \langle A \rangle_{\geq 0.8} M_2 \langle G \rangle_{\geq 0.98} \]

\[ M_1 \parallel M_2 \models P_{\geq 0.98} [G] \]
Running example

- Premise 1: Does $M_1 \models P_{\geq 0.8} [A]$ hold? Yes (earlier example)

Product PA $M_1 \otimes A_{err}$

$\Pr_{M_1} \min(A) = 1 - 0.2 = 0.8$

$\rightarrow M_1 \models P_{\geq 0.8} [A]$
Running example

- Premise 2: Does $\langle A \rangle \geq 0.8$ $M_2 \langle G \rangle \geq 0.98$ hold? Yes...

There is no adversary of $M_2$ satisfying $Pr_{M^\sigma}(A) \geq 0.8$ but not $Pr_{M^\sigma}(G) \geq 0.98$
Running example

- Premise 2: Does $\langle A \rangle \geq 0.8 \ M_2 \ \langle G \rangle \geq 0.98$ hold? Yes...

There is no adversary of $M_2 \otimes A_{err} \otimes G_{err}$ satisfying

$Pr_M(\diamond a_2) \leq 0.2$

and

$Pr_M(\diamond q_1) > 0.02$
Other assume-guarantee rules

- **Multiple assumptions:**

  \[
  M_1 \models P \geq p_1 [A_1] \land \ldots \land P \geq p_k [A_k] \\
  \langle A_1, \ldots, A_k \rangle \geq p_1, \ldots, p_k \quad M_2 \langle G \rangle \geq p_G \\
  \phantom{\text{M}_1 \mid \mid M_2 \models P \geq p_G [G]} \text{(ASYM-MULT)} \\
  \text{M}_1 \mid \mid M_2 \models P \geq p_G [G]
  \]

- **Circular rule:**

  \[
  M_2 \models P \geq p_2 [A_2] \\
  \langle A_2 \rangle \geq p_2 \quad M_1 \langle A_1 \rangle \geq p_1 \\
  \langle A_1 \rangle \geq p_1 \quad M_2 \langle G \rangle \geq p_G \\
  \phantom{\text{M}_1 \mid \mid M_2 \models P \geq p_G [G]} \text{(CIRC)} \\
  \text{M}_1 \mid \mid M_2 \models P \geq p_G [G]
  \]

- **Multiple components (chain):**

  \[
  M_1 \models P \geq p_1 [A_1] \\
  \langle A_1 \rangle \geq p_1 \quad M_2 \langle A_2 \rangle \geq p_2 \\
  \ldots \\
  \langle A_n \rangle \geq p_n \quad M_n \langle G \rangle \geq p_G \\
  \phantom{\text{M}_1 \mid \mid \ldots \mid \mid M_n \models P \geq p_G [G]} \text{(ASYM-N)} \\
  \text{M}_1 \mid \mid \ldots \mid \mid M_n \models P \geq p_G [G]
  \]

- **Asynchronous components:**

  \[
  \langle A_1 \rangle \geq p_1 \quad M_1 \langle G_1 \rangle \geq q_1 \\
  \langle A_2 \rangle \geq p_2 \quad M_2 \langle G_2 \rangle \geq q_2 \\
  \langle A_1, A_2 \rangle \geq p_1 p_2 \quad M_1 \mid \mid M_2 \langle G_1 \lor G_2 \rangle \geq (q_1 + q_2 - q_1 q_2) \\
  \phantom{\text{M}_1 \mid \mid \ldots \mid \mid M_n \models P \geq p_G [G]} \text{(ASYNC)}
  \]
Implementation + Case studies

- **Implemented using:**
  - extension of PRISM model checker
  - added support for multi-objective model checking
  - built-in support for assume-guarantee in progress

- **Two large case studies**
  - randomised consensus algorithm (Aspnes & Herlihy)
    - minimum probability consensus reached by round R
  - Zeroconf network protocol
    - maximum probability network configures incorrectly
    - minimum probability network configured by time T
## Experimental results

<table>
<thead>
<tr>
<th>Case study [parameters]</th>
<th>Non–compositional</th>
<th>Compositional</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>States</td>
<td>Time (s)</td>
</tr>
<tr>
<td><strong>Randomised consensus (3 processes) [R,k]</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3, 2</td>
<td>1,418,545</td>
<td>18,971</td>
</tr>
<tr>
<td>3, 20</td>
<td>39,827,233</td>
<td>time–out</td>
</tr>
<tr>
<td>4, 2</td>
<td>150,487,585</td>
<td>78,955</td>
</tr>
<tr>
<td>4, 20</td>
<td>2,028,200,209</td>
<td>mem–out</td>
</tr>
<tr>
<td><strong>ZeroConf [K]</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>313,541</td>
<td>103.9</td>
</tr>
<tr>
<td>6</td>
<td>811,290</td>
<td>275.2</td>
</tr>
<tr>
<td>8</td>
<td>1,892,952</td>
<td>592.2</td>
</tr>
<tr>
<td><strong>ZeroConf time–bounded [K, T]</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2, 10</td>
<td>65,567</td>
<td>46.3</td>
</tr>
<tr>
<td>2, 14</td>
<td>106,177</td>
<td>63.1</td>
</tr>
<tr>
<td>4, 10</td>
<td>976,247</td>
<td>88.2</td>
</tr>
<tr>
<td>4, 14</td>
<td>2,288,771</td>
<td>128.3</td>
</tr>
</tbody>
</table>
## Experimental results

<table>
<thead>
<tr>
<th>Case study [parameters]</th>
<th>Non–compositional</th>
<th>Compositional</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>States</td>
<td>Time (s)</td>
</tr>
<tr>
<td>Randomised consensus (3 processes) [R,K]</td>
<td>3, 2</td>
<td>1,418,545</td>
</tr>
<tr>
<td></td>
<td>3, 20</td>
<td>39,827,233</td>
</tr>
<tr>
<td></td>
<td>4, 2</td>
<td>150,487,585</td>
</tr>
<tr>
<td></td>
<td>4, 20</td>
<td>2,028,200,209</td>
</tr>
<tr>
<td>ZeroConf [K]</td>
<td>4</td>
<td>313,541</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>811,290</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>1,892,952</td>
</tr>
<tr>
<td>ZeroConf time–bounded [K, T]</td>
<td>2, 10</td>
<td>65,567</td>
</tr>
<tr>
<td></td>
<td>2, 14</td>
<td>106,177</td>
</tr>
<tr>
<td></td>
<td>4, 10</td>
<td>976,247</td>
</tr>
<tr>
<td></td>
<td>4, 14</td>
<td>2,288,771</td>
</tr>
</tbody>
</table>

- Faster than conventional model checking in a number of cases
## Experimental results

<table>
<thead>
<tr>
<th>Case study [parameters]</th>
<th>Non–compositional</th>
<th>Compositional</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>States</td>
<td>Time (s)</td>
</tr>
<tr>
<td>Randomised consensus (3 processes) [R,K]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3, 2</td>
<td>1,418,545</td>
<td>18,971</td>
</tr>
<tr>
<td>3, 20</td>
<td>39,827,233</td>
<td><strong>time-out</strong></td>
</tr>
<tr>
<td>4, 2</td>
<td>150,487,585</td>
<td>78,955</td>
</tr>
<tr>
<td>4, 20</td>
<td>2,028,200,209</td>
<td><strong>mem-out</strong></td>
</tr>
<tr>
<td>ZeroConf [K]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>313,541</td>
<td>103.9</td>
</tr>
<tr>
<td>6</td>
<td>811,290</td>
<td>275.2</td>
</tr>
<tr>
<td>8</td>
<td>1,892,952</td>
<td>592.2</td>
</tr>
<tr>
<td>ZeroConf time–bounded [K, T]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2, 10</td>
<td>65,567</td>
<td>46.3</td>
</tr>
<tr>
<td>2, 14</td>
<td>106,177</td>
<td>63.1</td>
</tr>
<tr>
<td>4, 10</td>
<td>976,247</td>
<td>88.2</td>
</tr>
<tr>
<td>4, 14</td>
<td>2,288,771</td>
<td>128.3</td>
</tr>
</tbody>
</table>

- Verified instances where conventional model checking is infeasible.
Overview

• Probabilistic model checking
  – probabilistic automata
  – property specification + probabilistic safety properties
  – multi-objective model checking

• Compositional probabilistic verification
  – assume-guarantee reasoning
  – assume-guarantee for probabilistic systems
  – implementation & results

• **Automated generation of assumptions**
  – L* and its application to compositional verification
  – generating probabilistic assumptions
  – implementation & results

• Conclusions, current & future work
Generating assumptions

• Can model check $M_1 \parallel M_2$ compositionally
  – but this relies on the existence of a suitable assumption $P_{\geq p_A} [A]$

1. Does such an assumption always exist?

2. When it does exist, can we generate it automatically?

Our approach: use algorithmic learning techniques
  – inspired by non-probabilistic AG work of [Pasareanu et al.]
  – uses $L^*$ algorithm to learn finite automata for assumptions
  – we use a modified version of $L^*$
  – to learn probabilistic assumptions for rule (ASYM) [QEST’10]
The L* learning algorithm

- The L* algorithm [[Angluin]]
  - learns an unknown regular language \( L \), as a (minimal) DFA

- Based on “active” learning
  - relies on existence of a “teacher” to guide the learning
  - answers two type of queries: “membership” and “equivalence”
    - membership: “is trace (word) \( t \) in the target language \( L \)?”
      - stores results of membership queries in observation table
      - based on these, generates conjectures \( A \) for the automata
    - equivalence: “does automata \( A \) accept the target language \( L \)?”
      - if not, teacher must return counterexample \( c \)
      - \( (c \) is a word in the symmetric difference of \( L \) and \( L(A) \))
The L* learning algorithm

L*
- Membership query
- Update table
  - done?
    - no
    - yes
      - Generate conjecture
        - yes
          - equivalence query (analyse conjecture A)
        - no
          - counterexample c
    - yes
      - equivalence query (analyse conjecture A)

Teacher
- Membership query (analyse trace t)
- equivalence query (analyse conjecture A)
L* for assume–guarantee

• **Breakthrough in automated compositional verification**
  – use of L* to learn assumptions for A/G reasoning
  – [Pasareanu/Giannakopoulou/et al.]
  – uses notion of “weakest assumption” about a component that suffices for compositional verification (always exists)
  – weakest assumption is the target regular language

• **Fully automated L* learning loop**
  – model checker plays role of teacher, returns counterexamples
  – in practice, can usually stop early: either with a simpler (stronger) assumption or by refuting the property

• **Successfully applied to several large case studies**
  – does particularly well when assumption/alphabet are small
  – much recent interest in learning for verification...
Probabilistic assumption generation

- **Goal:** automate A/G rule (ASYM)
  - generate probabilistic assumption $P \geq p_A[A]$
  - for checking property $P \geq p_G[G]$ on $M_1 \parallel M_2$

- **Reduce problem to generation of non-probabilistic assumption $A$**
  - then (if possible) find lowest $p_A$ such that premises 1 & 2 hold
  - in fact, for fixed $A$, we can generate lower and upper bounds on $\Pr_{M_1 \parallel M_2}^{\min}(G)$, which may suffice to verify/refute $P \geq p_G[G]$

- **Use adapted L* to learn non-probabilistic assumption $A$**
  - note: there is no “weakest assumption” (AG rule is incomplete)
  - but can generate sequence of conjectures for $A$ in similar style
  - “teacher” based on a probabilistic model checker (PRISM), feedback is from probabilistic counterexamples [Han/Katoen]
  - three outcomes of loop: “true”, “false”, lower/upper bounds
Probabilistic assumption generation

**IN:**
- Membership query
- Update table
- Generate conjecture
- Update table

**L*:**
- trace t
- yes/no
- done?
- conj. A
- cex. c

**Teacher:**
- Membership query (analyse trace t)
- Check:
  - t || M₂ ⊨ P≥PG [G]?
  - Equivalence query (analyse conjecture A)
    - Try to find p_A such that:
    - (i) M₁ ⊨ P≥p_A [A]
    - (ii) ⟨A⟩≥p_A M₂ ⟨G⟩≥p_G

**OUT:**
- "true"
  - M₁ || M₂ ⊨ P≥PG [G]
  - bounds
  - \( \Pr_{M₁ || M₂}^{\min} (G) \in [lo, up] \)
- "false"
  - M₁ || M₂ ⊬ P≥PG [G]
Implementation + Case studies

• **Implemented using:**
  – extension of PRISM model checker
  – libalf learning library [Bollig et al.]

• **Several case studies**
  – client–server (A/G model checking benchmark + failures)
    • minimum probability mutual exclusion not violated
  – randomised consensus algorithm [Aspnes & Herlihy]
    • minimum probability consensus reached by round R
  – sensor network [QEST’10]
    • minimum probability of processor error occurring
  – Mars Exploration Rovers (MER) [NASA]
    • minimum probability mutual exclusion not violated in k cycles
Experimental results (learning)

<table>
<thead>
<tr>
<th>Case study [parameters]</th>
<th>Component sizes</th>
<th>Compositional</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$</td>
<td>M_2 \otimes G_{err}</td>
</tr>
<tr>
<td><strong>Client-server (N failures) [N]</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>229</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>1,121</td>
<td>25</td>
</tr>
<tr>
<td>5</td>
<td>5,397</td>
<td>36</td>
</tr>
<tr>
<td><strong>Randomised consensus [N,R,K]</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2, 3, 20</td>
<td>391</td>
<td>3,217</td>
</tr>
<tr>
<td>2, 4, 4</td>
<td>573</td>
<td>431,649</td>
</tr>
<tr>
<td>3, 3, 20</td>
<td>8,843</td>
<td>38,193</td>
</tr>
<tr>
<td><strong>Sensor network [N]</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>42</td>
<td>1,184</td>
</tr>
<tr>
<td>3</td>
<td>42</td>
<td>10,662</td>
</tr>
<tr>
<td><strong>MER [N R]</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2, 5</td>
<td>5,776</td>
<td>427,363</td>
</tr>
<tr>
<td>3, 2</td>
<td>16,759</td>
<td>171</td>
</tr>
</tbody>
</table>
## Experimental results (learning)

<table>
<thead>
<tr>
<th>Case study [parameters]</th>
<th>Component sizes</th>
<th>Compositional</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$</td>
<td>M_2 \otimes G_{\text{err}}</td>
</tr>
<tr>
<td><strong>Client-server (N failures) [N]</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>229</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>1,121</td>
<td>25</td>
</tr>
<tr>
<td>5</td>
<td>5,397</td>
<td>36</td>
</tr>
<tr>
<td><strong>Randomised consensus [N,R,K]</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2, 3, 20</td>
<td>391</td>
<td>3,217</td>
</tr>
<tr>
<td>2, 4, 4</td>
<td>573</td>
<td>431,649</td>
</tr>
<tr>
<td>3, 3, 20</td>
<td>8,843</td>
<td>38,193</td>
</tr>
<tr>
<td><strong>Sensor network [N]</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>42</td>
<td>1,184</td>
</tr>
<tr>
<td>3</td>
<td>42</td>
<td>10,662</td>
</tr>
<tr>
<td><strong>MER [N R]</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2, 5</td>
<td>5,776</td>
<td>427,363</td>
</tr>
<tr>
<td>3, 2</td>
<td>16,759</td>
<td>171</td>
</tr>
</tbody>
</table>

- Successfully learnt (small) assumptions in all cases
Experimental results (learning)

<table>
<thead>
<tr>
<th>Case study [parameters]</th>
<th>Component sizes</th>
<th>Compositional</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(</td>
<td>M_2 \otimes G_{err}</td>
<td>)</td>
</tr>
<tr>
<td>Client-server (N failures) [N]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>229</td>
<td>16</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>1,121</td>
<td>25</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>5,397</td>
<td>36</td>
<td>7</td>
</tr>
<tr>
<td>Randomised consensus [N,R,K]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2, 3, 20</td>
<td>391</td>
<td>3,217</td>
<td>6</td>
</tr>
<tr>
<td>2, 4, 4</td>
<td>573</td>
<td>431,649</td>
<td>12</td>
</tr>
<tr>
<td>3, 3, 20</td>
<td>8,843</td>
<td>38,193</td>
<td>11</td>
</tr>
<tr>
<td>Sensor network [N]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>42</td>
<td>1,184</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>42</td>
<td>10,662</td>
<td>3</td>
</tr>
<tr>
<td>MER [N R]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2, 5</td>
<td>5,776</td>
<td>427,363</td>
<td>4</td>
</tr>
<tr>
<td>3, 2</td>
<td>16,759</td>
<td>171</td>
<td>4</td>
</tr>
</tbody>
</table>

- In some cases, learning + compositional verification is faster (than non-compositional verification, using PRISM)
Recent developments

- An alternative learning algorithm: NL* [Bollig et al.]
  - learns residual finite-state automata (subclass of NFAs)
  - can be exponentially smaller than corresponding DFA
  - basic learning loop remains the same
  - we need to determinise NFA for model checking; but still get gains in some cases due to less equivalence queries (EQ)

<table>
<thead>
<tr>
<th>Case study [parameters]</th>
<th>Compositional (L*)</th>
<th>Compositional (NL*)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A&lt;sup&gt;err&lt;/sup&gt;</td>
<td>EQ</td>
</tr>
<tr>
<td>Client-server1</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>Client-serverN</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>Rand. cons. [N,R,K]</td>
<td>2, 4, 4</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>3, 3, 20</td>
<td>11</td>
</tr>
<tr>
<td>MER [N R]</td>
<td>2, 5</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>3, 2</td>
<td>4</td>
</tr>
</tbody>
</table>
Recent developments...

• **Learning multiple assumptions**
  – decompose into >2 components
  – using A/G rule (ASYM–N)
  – recursive application of learning loop
  – learn assumptions $P \geq p_1[A_1] \ldots P \geq p_n[A_n]$
  – much better scalability...

$$M_1 \models P \geq p_1[A_1]$$
$$\langle A_1 \rangle \geq p_1 M_2 \langle A_2 \rangle \geq p_2$$
$$\ldots$$
$$\langle A_n \rangle \geq p_n M_n \langle G \rangle \geq p_G$$

$$M_1 \parallel \ldots \parallel M_n \models P \geq p_G[G]$$

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time (s)</td>
<td>Time (s)</td>
<td>Time (s)</td>
</tr>
<tr>
<td>Client–serverN [N]</td>
<td>6</td>
<td>memout</td>
<td>40.9</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>memout</td>
<td>164.7</td>
</tr>
<tr>
<td>MER [N R]</td>
<td>3, 5</td>
<td>memout</td>
<td>29.8</td>
</tr>
<tr>
<td></td>
<td>4, 5</td>
<td>memout</td>
<td>122.9</td>
</tr>
<tr>
<td></td>
<td>5, 5</td>
<td>memout</td>
<td>3,903.4</td>
</tr>
</tbody>
</table>
Conclusions

• Probabilistic model checking
  – active research area, efficient tools, widely used
  – but scalability is still the biggest challenge

• Compositional probabilistic verification
  – assume–guarantee framework for probabilistic automata
  – reduction to (efficient) multi–objective model checking
  – verified safety/performance on several large case studies
  – cases where infeasible using non–compositional verification
  – full automation: learning–based generation of assumptions

• But this is only the beginning…