Automated Learning of Probabilistic Assumptions for Compositional Reasoning

Marta Kwiatkowska

Oxford University Computing Laboratory

FASE'11, Saarbrücken, April 2011

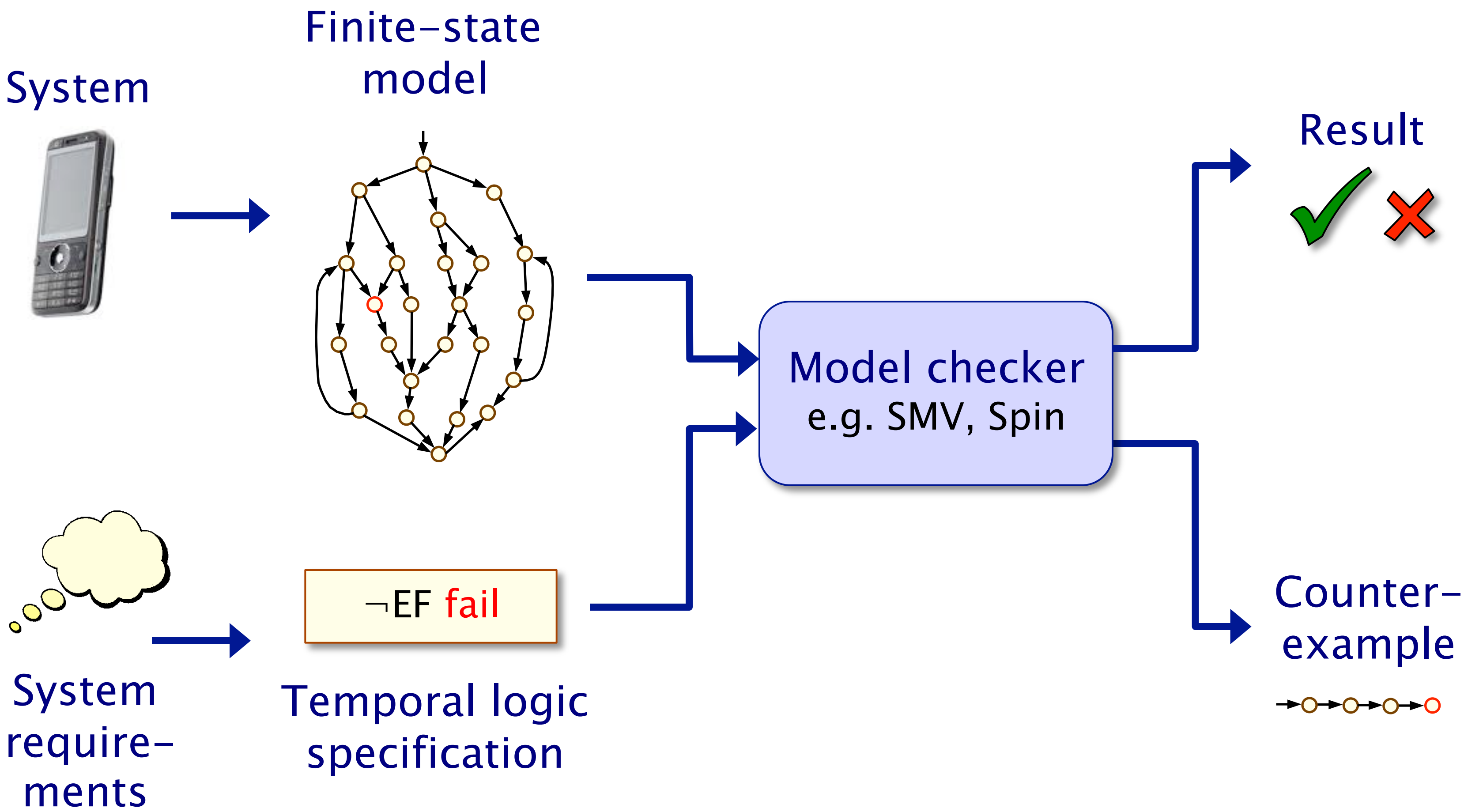
Joint work with: Lu Feng, Dave Parker, Gethin Norman, Hongyang Qu

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”

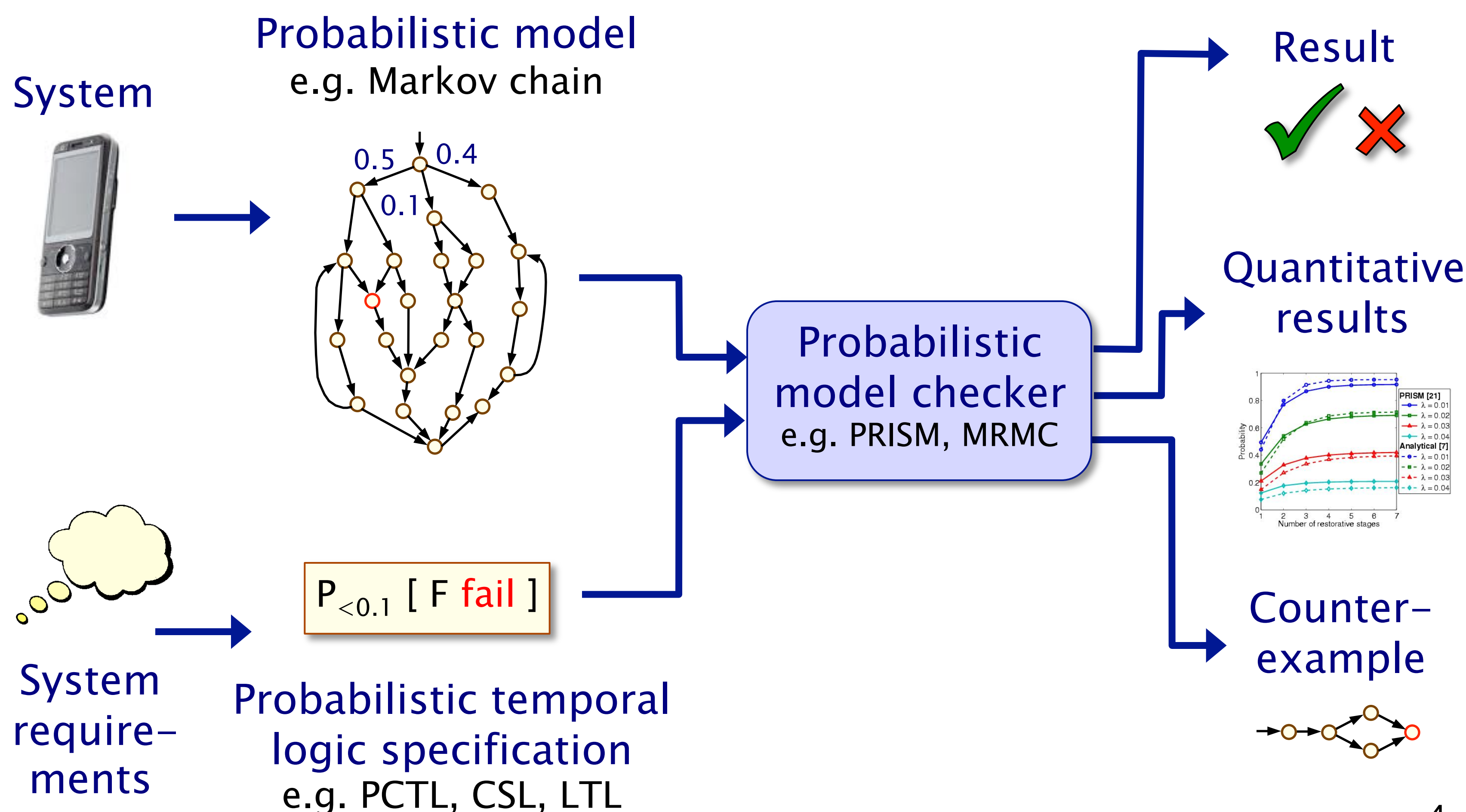
Model checking

Automated formal verification for finite-state models



Probabilistic model checking

Automatic verification of systems with probabilistic behaviour

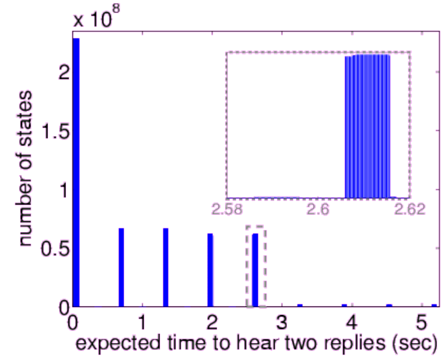


Probabilistic model checking

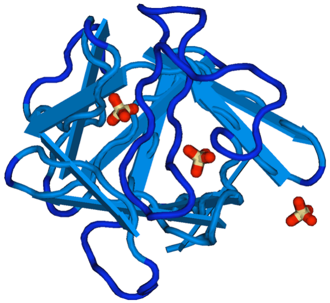
- First algorithms proposed in 1980s
  - [Vardi, Courcoubetis, Yannakakis, …]
  - algorithms [Hansson, Jonsson, de Alfaro] & first implementations
- 2000: tools ETMCC (MRMC) & PRISM released
  - PRISM: efficient extensions of symbolic model checking
  - ETMCC (now MRMC): model checking for continuous-time Markov chains [Baier, Hermanns, Haverkort, 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 π-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} 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

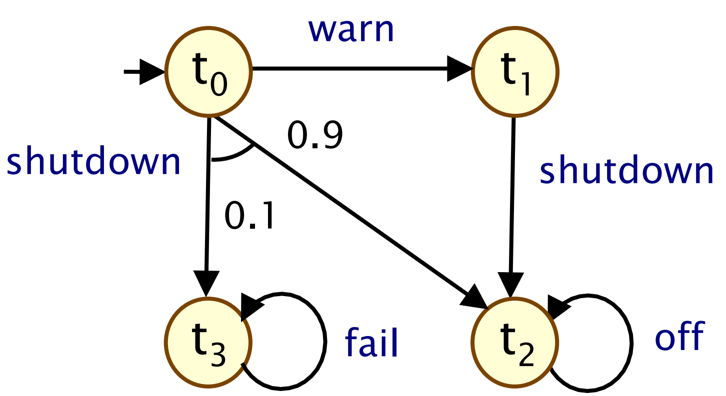
- Markov decision processes (MDPs) ← this talk
- Probabilistic automata (PAs) [Segala]
  - discrete states + probability + nondeterminism
  - for: concurrency, control, under-specification, abstraction

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



- Parallel composition: $M_1 \,||\, M_2$
  - CSP style – synchronise over common actions

Probabilistic model checking for PAs

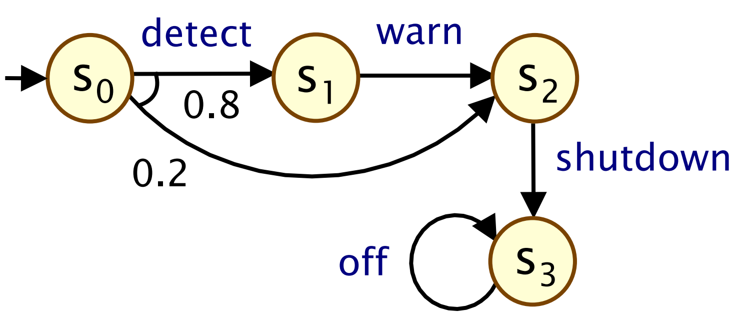
- To reason formally about PAs, we use adversaries
  - an adversary σ resolves nondeterminism in a PA M
  - also called "scheduler", "strategy", "policy", …
  - makes a (possibly randomised) choice, based on history
  - induces probability measure Pr_M^σ over (infinite) paths

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

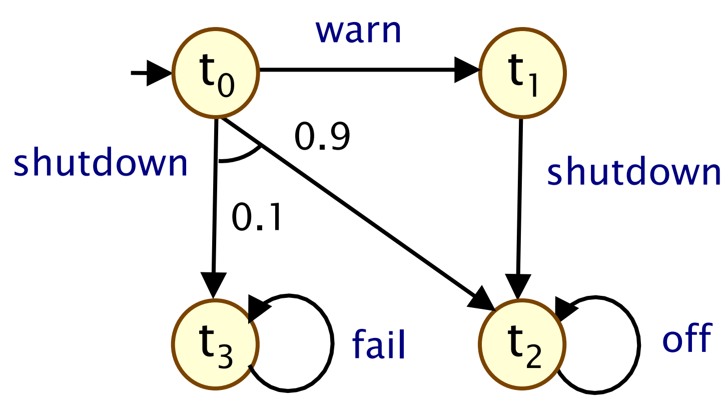
Running example

- 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)

PA M_1 ("sensor")

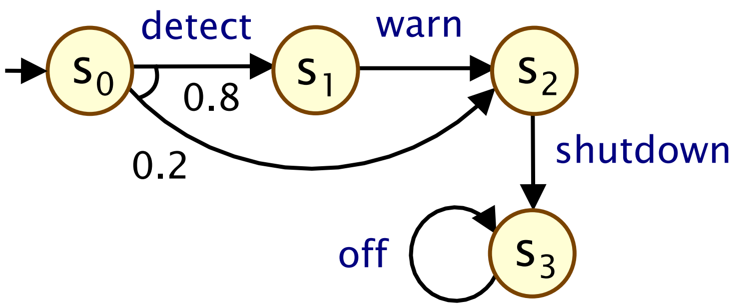


PA M_2 ("device")

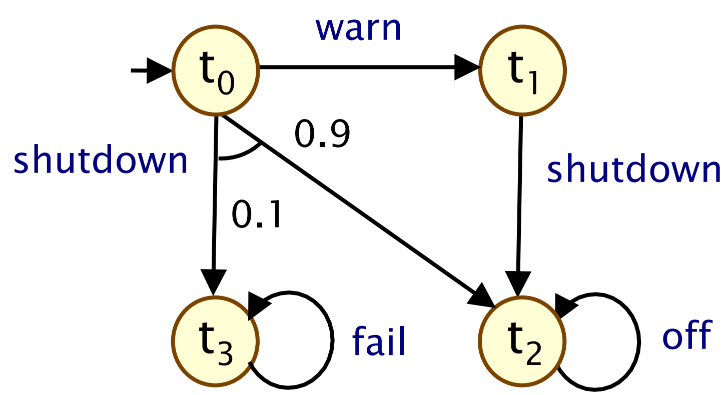


Running example

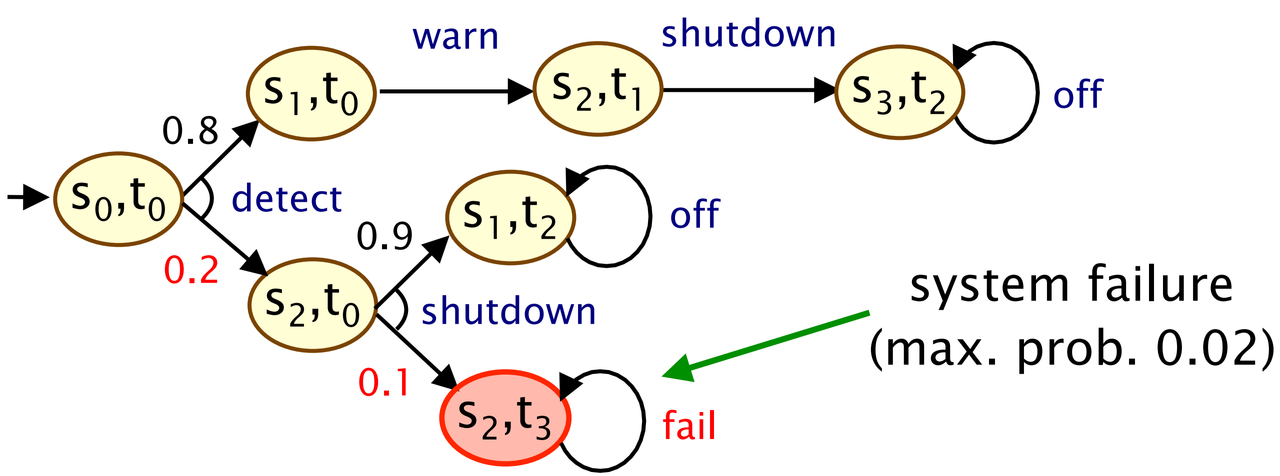
PA M_1 (“sensor”)



PA M_2 (“device”)

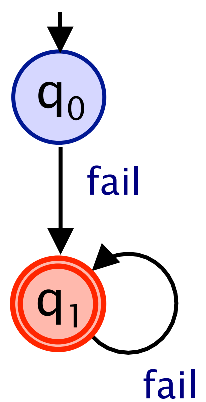


Parallel composition: $M_1 \| M_2$

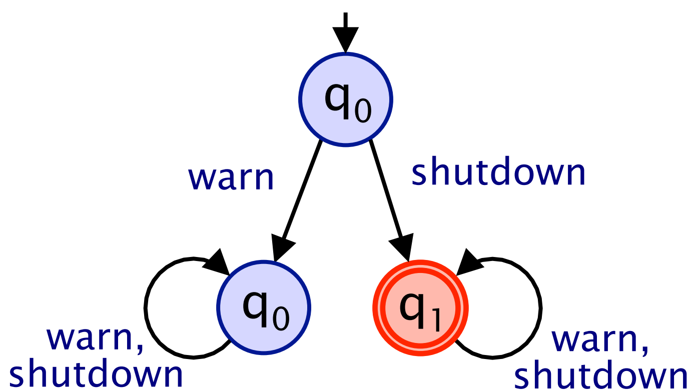


Safety properties

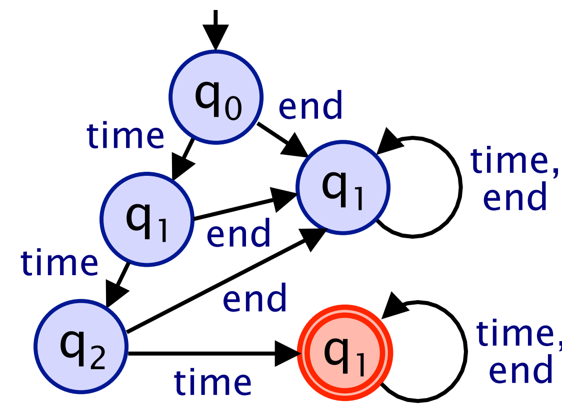
- Safety property: language of infinite words (over actions)
  - characterised by a set of “bad prefixes” (or “finite violations”)
  - i.e. finite words of which any extension violates the property
- Regular safety property
  - bad prefixes are represented by a regular language
  - property A represented by an *error automaton* A_{err}, a deterministic finite automaton (DFA) storing bad prefixes



“a fail action never occurs”



“warn occurs before shutdown”



“at most 2 time steps pass before termination”

Probabilistic safety properties

- 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"
  - $M \vDash P_{\geq p}[A] \;\Leftrightarrow\; Pr_M^{\sigma}(A) \geq p$ for all $\sigma \in Adv_M \;\Leftrightarrow\; Pr_M^{min}(A) \geq p$
  - or "the (max.) probability of violating A must be at most 1−p"

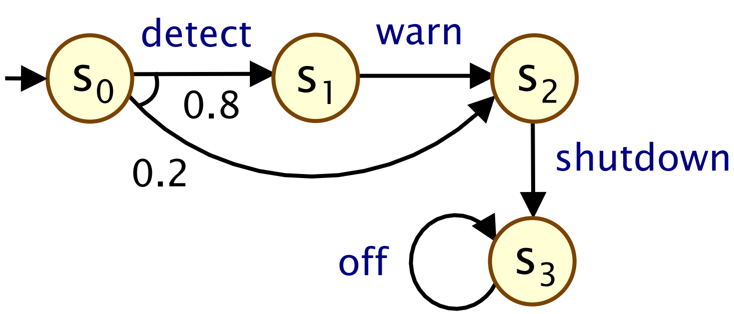
- Examples:
  - "*warn* occurs before *shutdown* with probability at least 0.8"
  - "the probability of a failure occurring is at most 0.02"
  - "probability of terminating within k time-steps is at least 0.75"

- Model checking:
  - construct (synchronous) PA-DFA product $M \otimes A_{err}$
  - compute probability of reaching "accept" in product PA

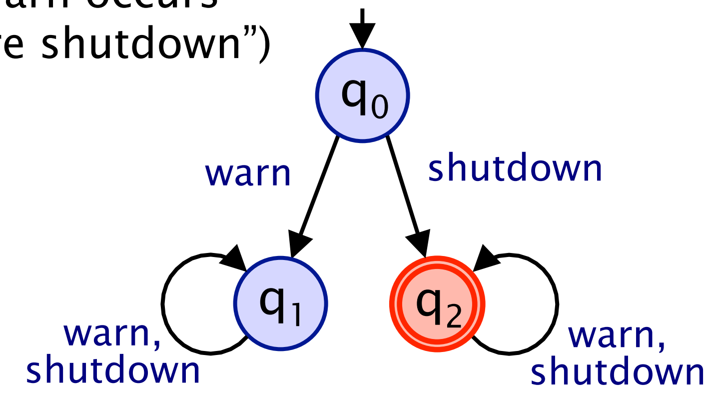
Running example

- Does probabilistic safety property $P_{\geq 0.8}$ [A] hold in M_1?

PA M_1 ("sensor")



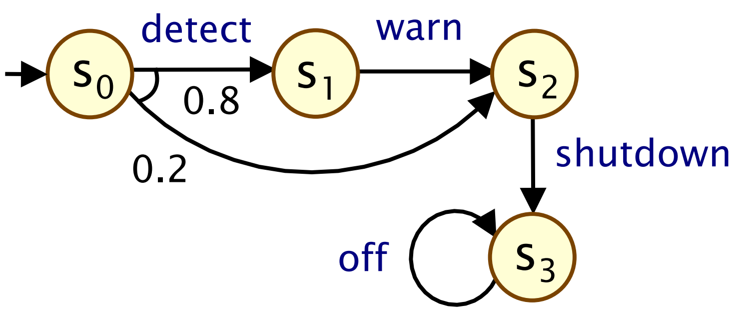
A ("warn occurs before shutdown")



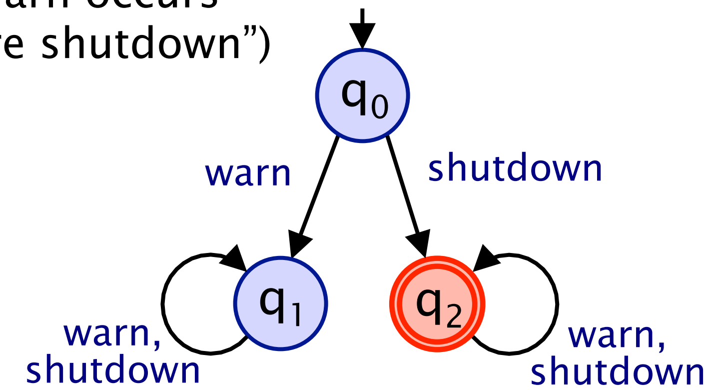
Running example

- Does probabilistic safety property $P_{\geq 0.8}$ [A] hold in M_1?

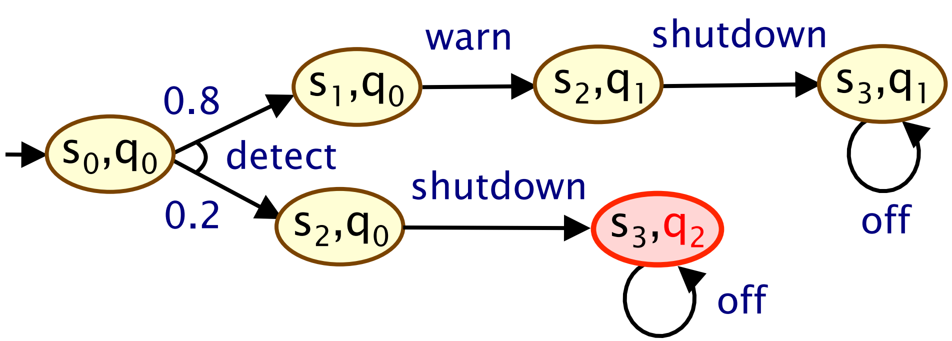
PA M_1 ("sensor")



A ("warn occurs before shutdown")



Product PA $M_1 \otimes A_{err}$



$Pr_{M_1}{}^{min}(A)$

$= 1 - 0.2 = 0.8$

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

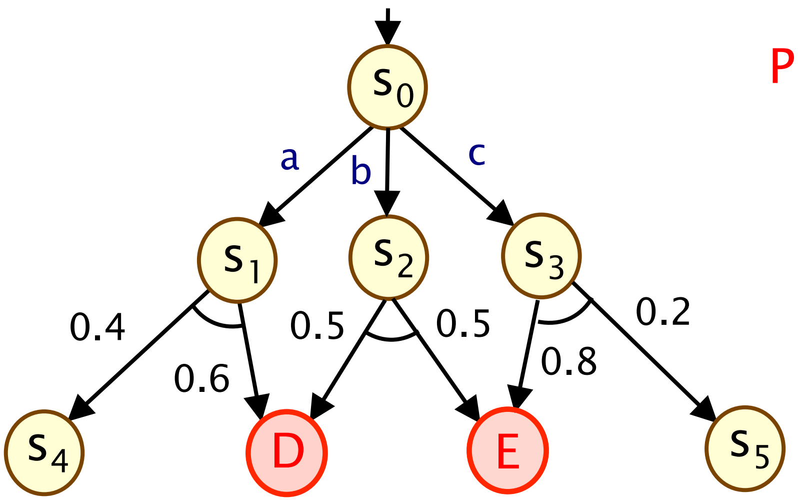
Multi-objective PA model checking

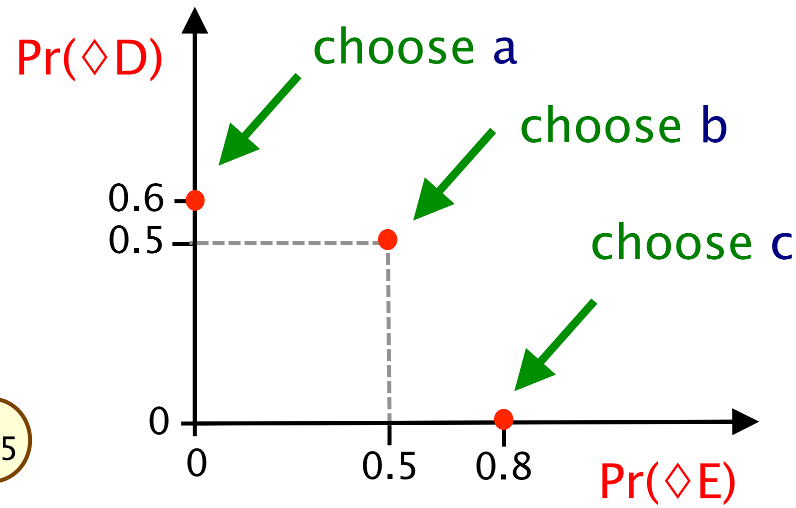
- Study trade-off between several different objectives
  - existential queries: does there exist adversary σ such that:
  - $Pr_M^{\sigma}(\square(queue\_size<10)) > 0.99 \wedge Pr_M^{\sigma}(\Diamond flat\_battery) < 0.01$
  - useful for synthesising controllers

- Multi-objective PA model checking
  - [Etessami/Kwiatkowska/Vardi/Yannakakis, TACAS'07]
  - LTL formulae $\Phi_1,\ldots,\Phi_k$ and probability bounds $\sim_1 p_1,\ldots,\sim_k p_k$
  - check if $\exists\ \sigma \in Adv_M$ s.t. $Pr_M^{\sigma}(\phi_1) \sim_1 p_1 \wedge \ldots \wedge Pr_M^{\sigma}(\phi_k) \sim_k p_k$
  - construct product of automata for M, $\Phi_1,\ldots,\Phi_k$
  - then solve linear programming (LP) problem
  - the resulting adversary σ can obtained from LP solution
  - note: σ may be randomised (unlike the single objective case)

Multi-objective PA model checking

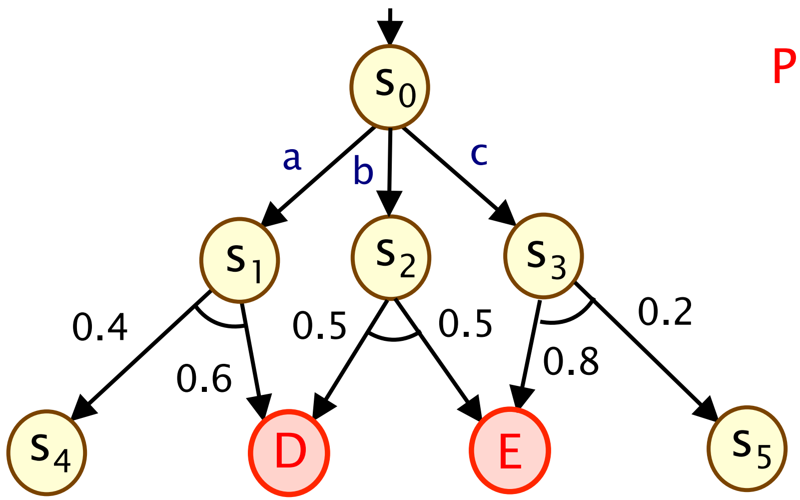
- Consider the two objectives ◊D and ◊E in the PA below
  - i.e. the trade-off between the probabilities Pr(◊D) and Pr(◊E)
  - an adversary resolves the choice between a/b/c
  - increasing the probability of reaching one target decreases the probability of reaching the other

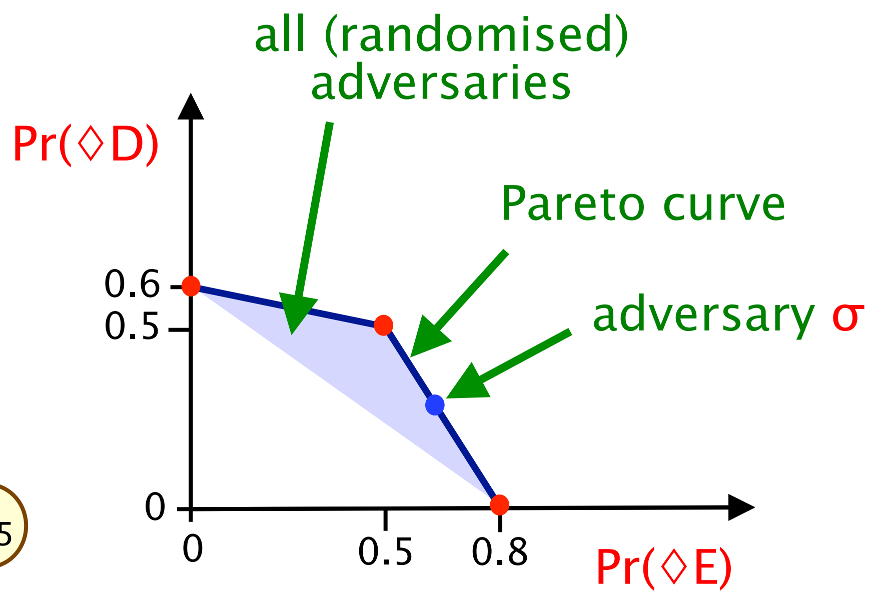




Multi-objective PA model checking

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





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:

$$\frac{M_1 \models A \qquad \langle A \rangle\ M_2\ \langle G \rangle}{M_1\ ||\ M_2 \models G}$$

[Pasareanu/Giannakopoulou/et al.]

AG rules for probabilistic systems

- How to formulate AG rules for probabilistic automata?

$$\frac{M_1 \models A \qquad \langle A \rangle\, M_2\, \langle G \rangle}{M_1 \,||\, M_2 \models G}$$

- 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?

$$\frac{M_1 \models A \qquad \langle A\rangle\, M_2\, \langle G\rangle}{M_1 \,||\, M_2 \models G}$$

- 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?

$$\frac{M_1 \models A \qquad \langle A \rangle\, M_2\, \langle G \rangle}{M_1 \,||\, M_2 \models G}$$

- 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:
  - $\forall \sigma \in Adv_{M'}\ (\ Pr_{M'}^{\sigma}(A) \geq p_A \rightarrow Pr_{M'}^{\sigma}(G) \geq p_G\ )$
  - where M' is M with its alphabet extended to include α_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)

$$\frac{M_1 \models P_{\geq p_A}[A] \qquad \langle A \rangle_{\geq p_A} \, M_2 \, \langle G \rangle_{\geq p_G}}{M_1 \,||\, M_2 \models P_{\geq p_G}[G]} \quad \text{(ASYM)}$$

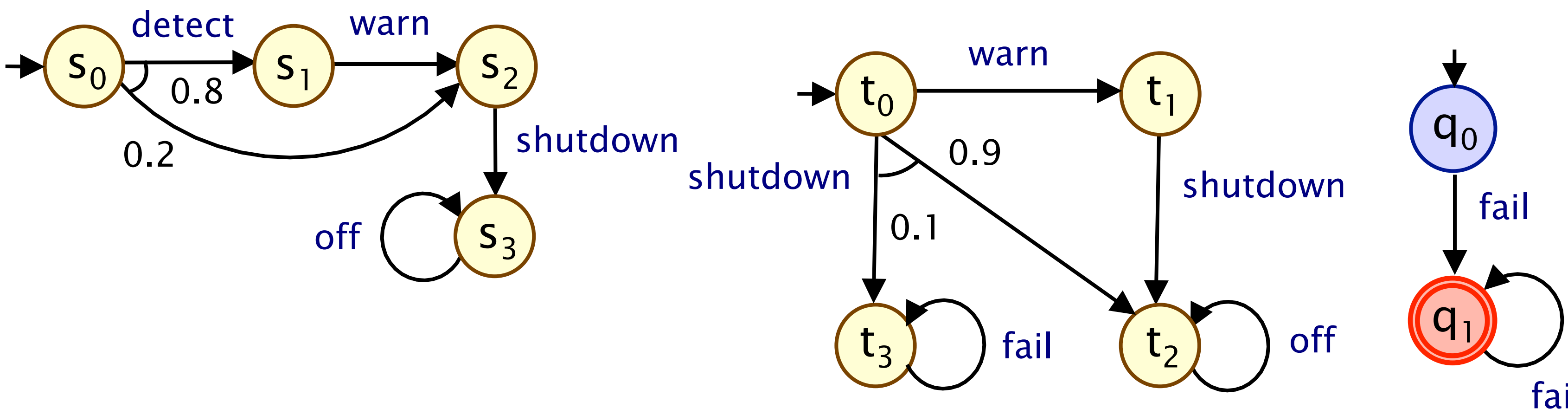
- So, verifying $M_1 \,||\, M_2 \models P_{\geq p_G}[G]$ requires:
  - premise 1: $M_1 \models P_{\geq p_A}[A]$ (standard model checking)
  - premise 2: $\langle A \rangle_{\geq p_A} \, M_2 \, \langle G \rangle_{\geq p_G}$ (multi-objective model checking)

- Potentially much cheaper if $|A|$ much smaller than $|M_1|$

Running example

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

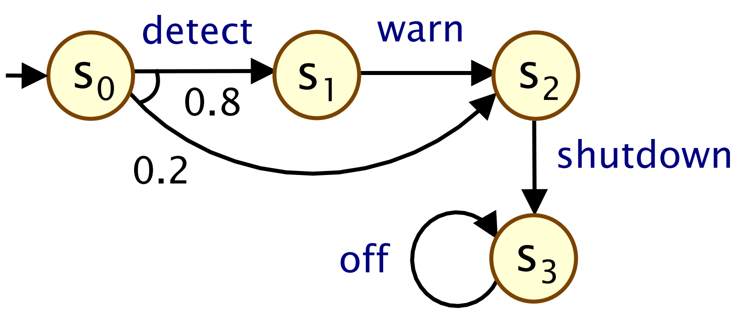
PA M_1 ("sensor") PA M_2 ("device") G ("a fail action never occurs")



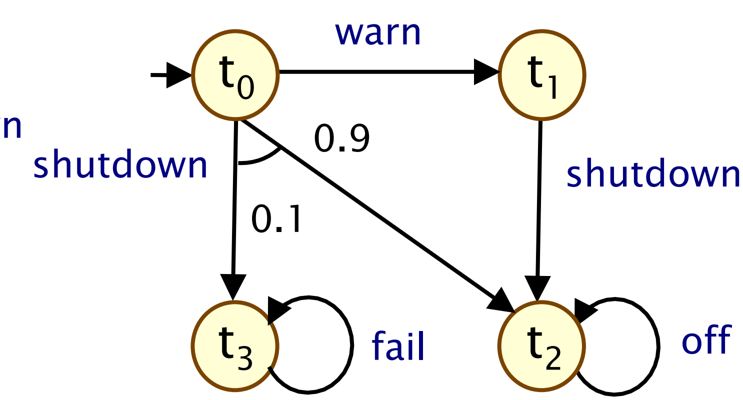
Running example

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

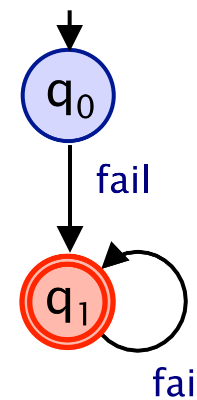
PA M_1 ("sensor")



PA M_2 ("device")



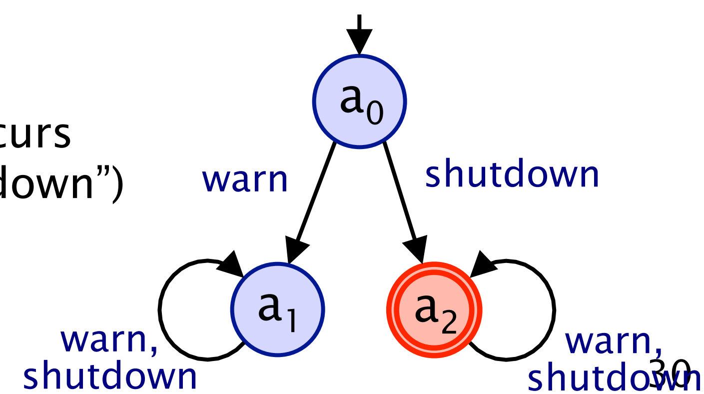
G ("a fail action never occurs")



- Use A/G with assumption $P_{\geq 0.8}$ [A] about M_1

$$\frac{M_1 \models P_{\geq 0.8}\,[A] \qquad \langle A \rangle_{\geq 0.8}\, M_2\, \langle G \rangle_{\geq 0.98}}{M_1 \,||\, M_2 \models P_{\geq 0.98}\,[G]}$$

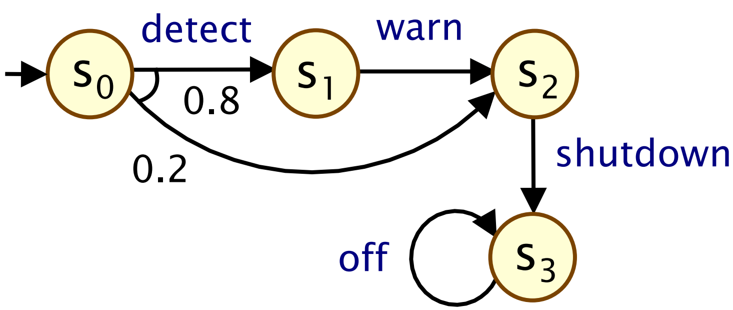
A ("warn occurs before shutdown")



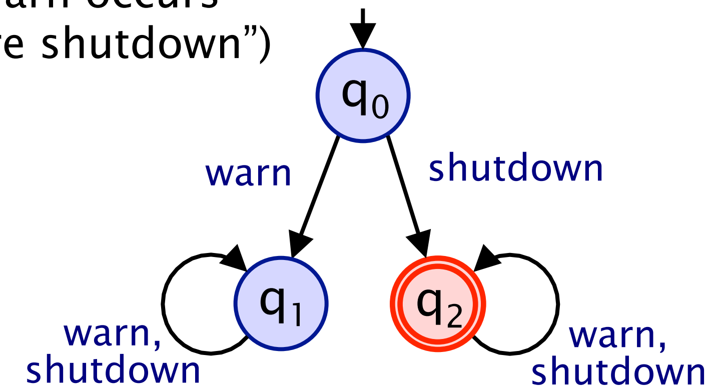
Running example

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

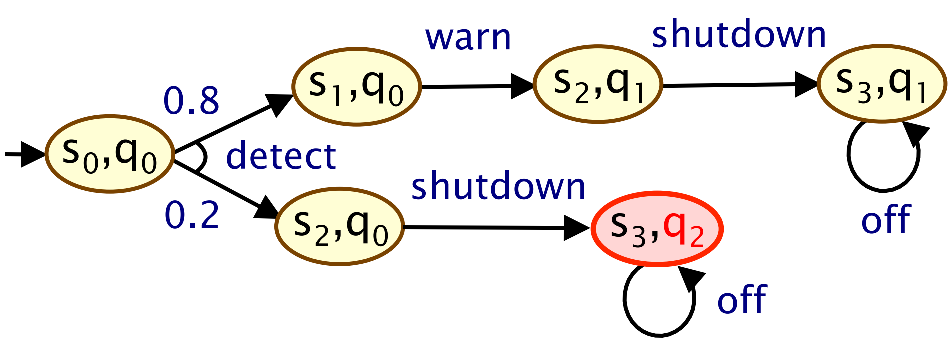
PA M_1 ("sensor")



A ("warn occurs before shutdown")



Product PA $M_1 \otimes A_{err}$



$Pr_{M_1}^{min}(A)$

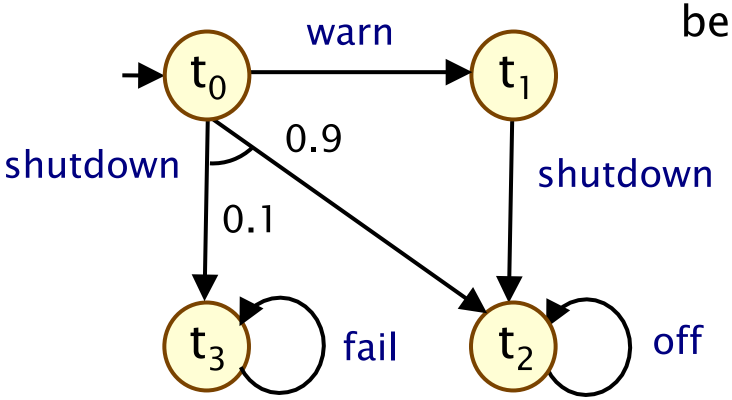
$= 1 - 0.2 = 0.8$

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

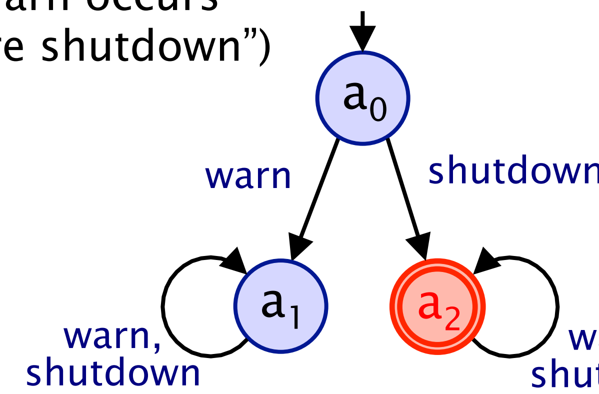
Running example

- Premise 2: Does $\langle A \rangle_{\geq 0.8}\ M_2\ \langle G \rangle_{\geq 0.98}$ hold? Yes…

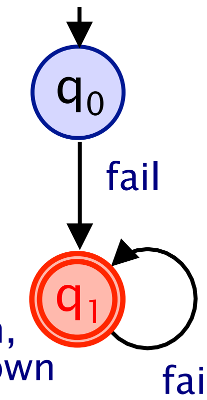
PA M_2 ("device")



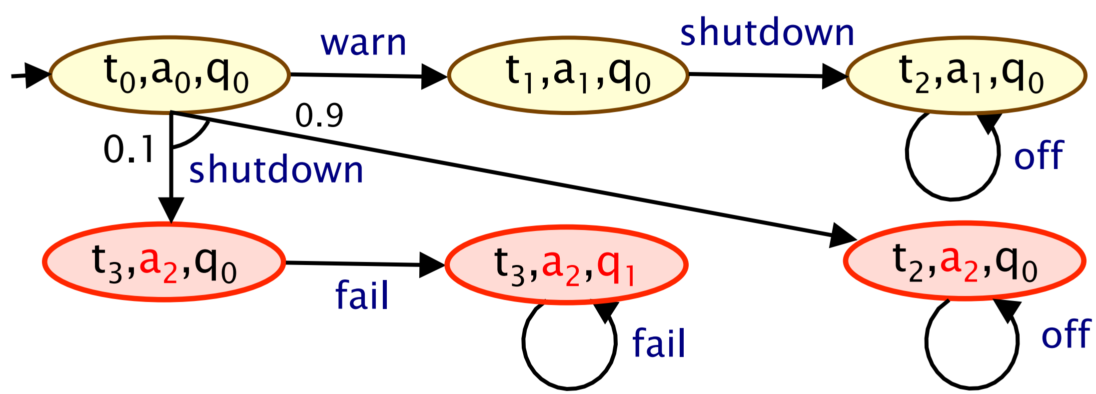
A ("warn occurs before shutdown")



G ("a fail action never occurs")



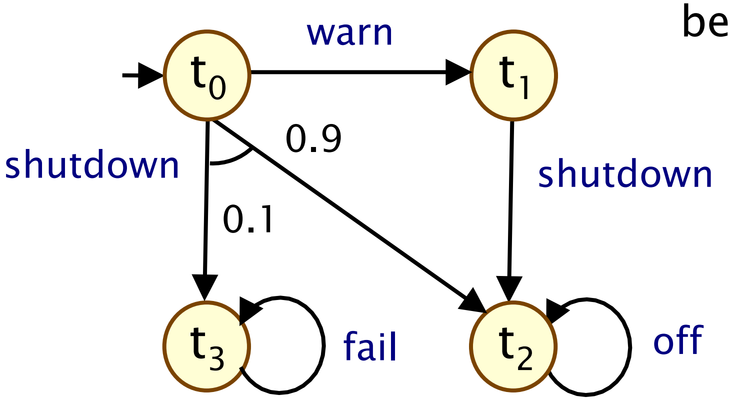
There is *no* adversary of M_2 satisfying $Pr_M^{\sigma}(A) \geq 0.8$ but not $Pr_M^{\sigma}(G) \geq 0.98$



Running example

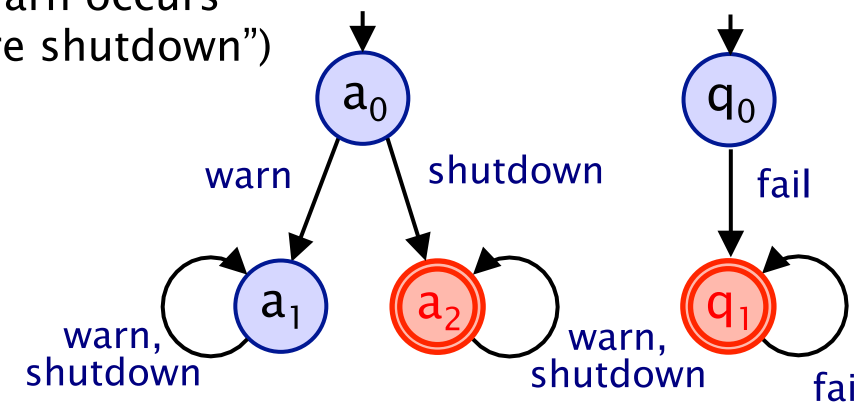
- Premise 2: Does $\langle A \rangle_{\geq 0.8}\ M_2\ \langle G \rangle_{\geq 0.98}$ hold? Yes…

PA M_2 ("device")



A ("warn occurs before shutdown")

G ("a fail action never occurs")

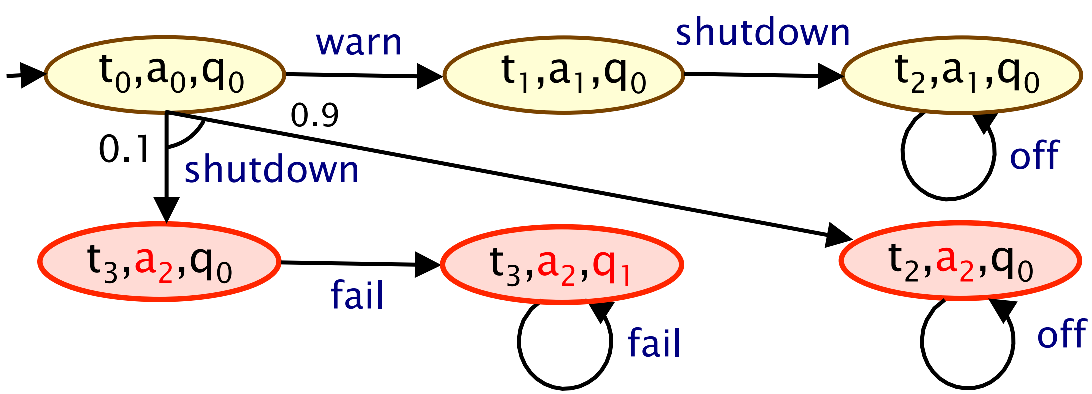


There is *no* adversary of $M_2 \otimes A_{err} \otimes G_{err}$ satisfying

$Pr_M^{\sigma}(\Diamond a_2) \leq 0.2$

and

$Pr_M^{\sigma}(\Diamond q_1) > 0.02$



Other assume-guarantee rules

- Multiple assumptions:

$$\frac{M_1 \models P_{\geq p_1}[A_1] \wedge \ldots \wedge P_{\geq p_k}[A_k] \qquad \langle A_1,\ldots,A_k\rangle_{\geq p_{1,\ldots,}p_k}\ M_2\ \langle G\rangle_{\geq p_G}}{M_1 \,||\, M_2 \models P_{\geq p_G}[G]} \quad \text{(ASYM-MULT)}$$

Multiple components (chain):

$$\frac{M_1 \models P_{\geq p_1}[A_1] \qquad \langle A_1\rangle_{\geq p_1}\ M_2\ \langle A_2\rangle_{\geq p_2} \qquad \ldots \qquad \langle A_n\rangle_{\geq p_n}\ M_n\ \langle G\rangle_{\geq p_G}}{M_1 \,||\, \ldots \,||\, M_n \models P_{\geq p_G}[G]} \quad \text{(ASYM-N)}$$

- Circular rule:

$$\frac{M_2 \models P_{\geq p_2}[A_2] \qquad \langle A_2\rangle_{\geq p_2}\ M_1\ \langle A_1\rangle_{\geq p_1} \qquad \langle A_1\rangle_{\geq p_1}\ M_2\ \langle G\rangle_{\geq p_G}}{M_1 \,||\, M_2 \models P_{\geq p_G}[G]} \quad \text{(CIRC)}$$

Asynchronous components:

$$\frac{\langle A_1\rangle_{\geq p_1}\ M_1\ \langle G_1\rangle_{\geq q_1} \qquad \langle A_2\rangle_{\geq p_2}\ M_2\ \langle G_2\rangle_{\geq q_2}}{\langle A_1,A_2\rangle_{\geq p_1 p_2}\ M_1 \,||\, M_2\ \langle G_1 \vee G_2\rangle_{\geq (q_1+q_2-q_1 q_2)}} \quad \text{(ASYNC)}$$

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

Case study [parameters]		Non-compositional		Compositional	
		States	Time (s)	LP size	Time (s)
Randomised consensus (3 processes) [R,K]	3, 2	1,418,545	18,971	40,542	29.6
	3, 20	39,827,233	time-out	40,542	125.3
	4, 2	150,487,585	78,955	141,168	376.1
	4, 20	2,028,200,209	mem-out	141,168	471.9
ZeroConf [K]	4	313,541	103.9	20,927	21.9
	6	811,290	275.2	40,258	54.8
	8	1,892,952	592.2	66,436	107.6
ZeroConf time-bounded [K, T]	2, 10	65,567	46.3	62,188	89.0
	2, 14	106,177	63.1	101,313	170.8
	4, 10	976,247	88.2	74,484	170.8
	4, 14	2,288,771	128.3	166,203	430.6

Experimental results

Case study [parameters]		Non-compositional		Compositional	
		States	Time (s)	LP size	Time (s)
Randomised consensus (3 processes) [R,K]	3, 2	1,418,545	18,971	40,542	29.6
	3, 20	39,827,233	time-out	40,542	125.3
	4, 2	150,487,585	78,955	141,168	376.1
	4, 20	2,028,200,209	mem-out	141,168	471.9
ZeroConf [K]	4	313,541	103.9	20,927	21.9
	6	811,290	275.2	40,258	54.8
	8	1,892,952	592.2	66,436	107.6
ZeroConf time-bounded [K, T]	2, 10	65,567	46.3	62,188	89.0
	2, 14	106,177	63.1	101,313	170.8
	4, 10	976,247	88.2	74,484	170.8
	4, 14	2,288,771	128.3	166,203	430.6

- Faster than conventional model checking in a number of cases

Experimental results

Case study [parameters]		Non-compositional		Compositional	
		States	Time (s)	LP size	Time (s)
Randomised consensus (3 processes) [R,K]	3, 2	1,418,545	18,971	40,542	29.6
	3, 20	39,827,233	time-out	40,542	125.3
	4, 2	150,487,585	78,955	141,168	376.1
	4, 20	2,028,200,209	mem-out	141,168	471.9
ZeroConf [K]	4	313,541	103.9	20,927	21.9
	6	811,290	275.2	40,258	54.8
	8	1,892,952	592.2	66,436	107.6
ZeroConf time-bounded [K, T]	2, 10	65,567	46.3	62,188	89.0
	2, 14	106,177	63.1	101,313	170.8
	4, 10	976,247	88.2	74,484	170.8
	4, 14	2,288,771	128.3	166,203	430.6

- 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 $M_1 || M_2$ compositionally
  - but this relies on the existence of a suitable assumption $P_{\geq p_A}[A]$

$$\frac{M_1 \models P_{\geq p_A}[A] \qquad \langle A \rangle_{\geq p_A} M_2 \langle G \rangle_{\geq p_G}}{M_1 \,||\, M_2 \models P_{\geq p_G}[G]}$$

- 1. Does such an assumption always exist?

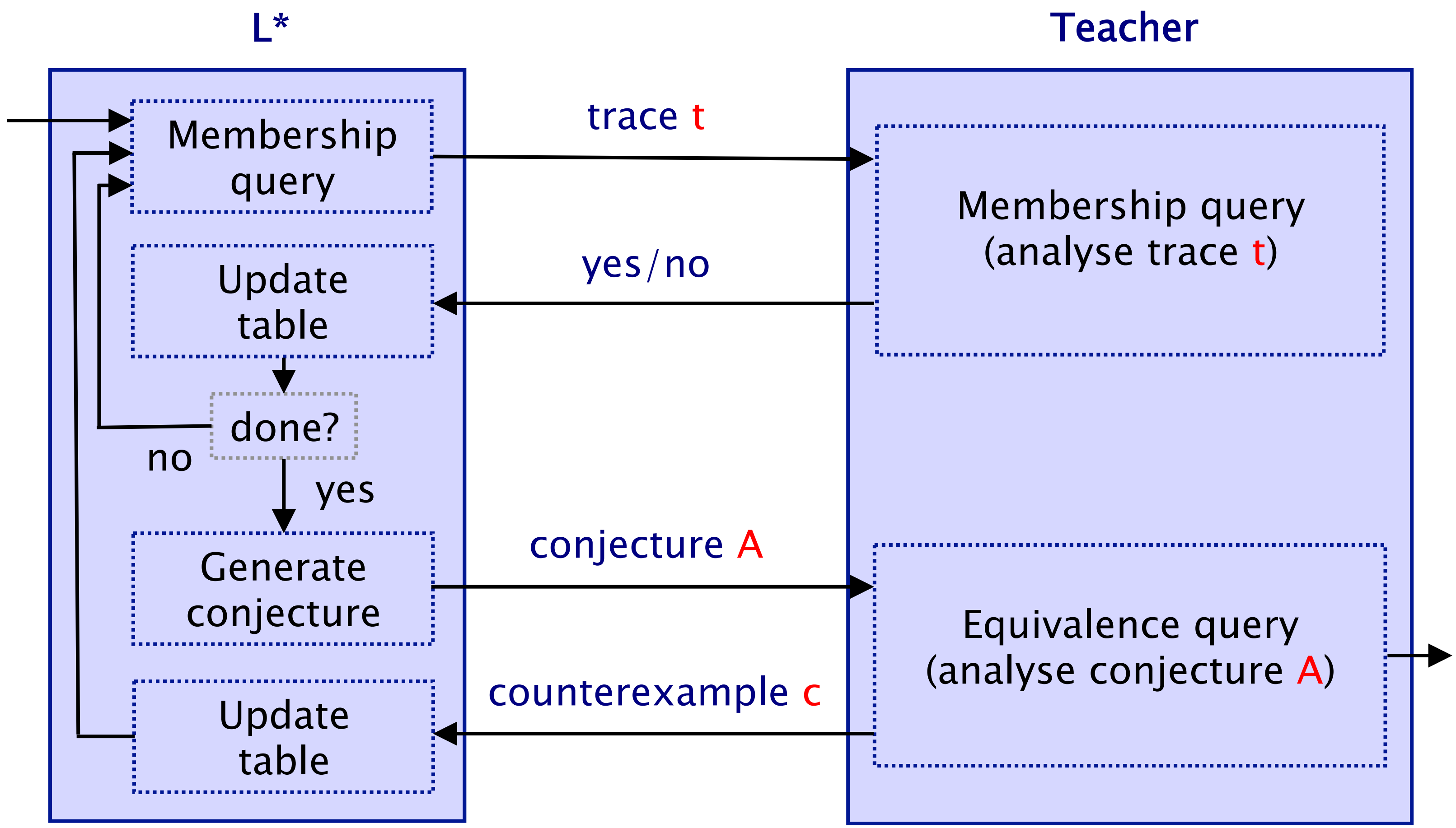
- 2. When it does exist, can we generate it automatically?

- 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* 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…

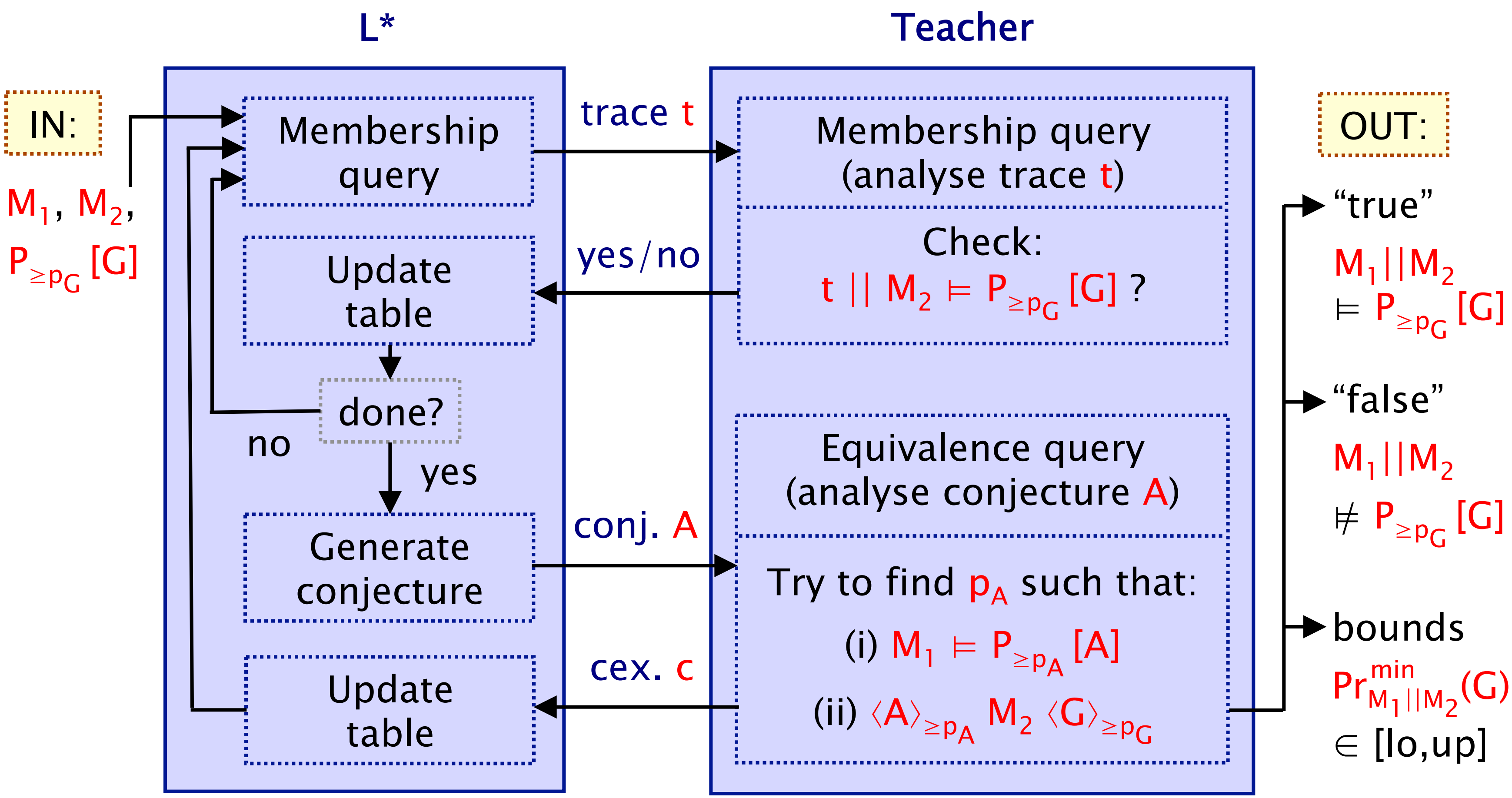
Probabilistic assumption generation

- Goal: automate A/G rule (ASYM)
  - generate probabilistic assumption $P_{\geq p_A}[A]$
  - for checking property $P_{\geq p_G}[G]$ on $M_1 \| M_2$

$$\frac{M_1 \models P_{\geq p_A}[A] \qquad \langle A \rangle_{\geq p_A} M_2 \langle G \rangle_{\geq p_G}}{M_1 \| M_2 \models P_{\geq p_G}[G]}$$

- 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 $Pr_{M_1\|M_2}^{min}(G)$, 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



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)

Case study [parameters]		Component sizes		Compositional	
		$\lvert M_2 \otimes G_{err} \rvert$	$\lvert M_1 \rvert$	$\lvert A^{err} \rvert$	Time (s)
Client-server (N failures) [N]	3	229	16	5	6.6
	4	1,121	25	6	26.1
	5	5,397	36	7	191.1
Randomised consensus [N,R,K]	2, 3, 20	391	3,217	6	24.2
	2, 4, 4	573	431,649	12	413.2
	3, 3, 20	8,843	38,193	11	438.9
Sensor network [N]	2	42	1,184	3	3.7
	3	42	10,662	3	4.6
MER [N R]	2, 5	5,776	427,363	4	31.8
	3, 2	16,759	171	4	210.5

Experimental results (learning)

Case study [parameters]		Component sizes		Compositional	
		$\lvert M_2 \otimes G_{err} \rvert$	$\lvert M_1 \rvert$	$\lvert A^{err} \rvert$	Time (s)
Client-server (N failures) [N]	3	229	16	5	6.6
	4	1,121	25	6	26.1
	5	5,397	36	7	191.1
Randomised consensus [N,R,K]	2, 3, 20	391	3,217	6	24.2
	2, 4, 4	573	431,649	12	413.2
	3, 3, 20	8,843	38,193	11	438.9
Sensor network [N]	2	42	1,184	3	3.7
	3	42	10,662	3	4.6
MER [N R]	2, 5	5,776	427,363	4	31.8
	3, 2	16,759	171	4	210.5

- Successfully learnt (small) assumptions in all cases

Experimental results (learning)

Case study [parameters]		Component sizes		Compositional	
		$\lvert M_2 \otimes G_{err} \rvert$	$\lvert M_1 \rvert$	$\lvert A^{err} \rvert$	Time (s)
Client-server (N failures) [N]	3	229	16	5	6.6
	4	1,121	25	6	26.1
	5	5,397	36	7	191.1
Randomised consensus [N,R,K]	2, 3, 20	391	3,217	6	24.2
	2, 4, 4	573	431,649	12	413.2
	3, 3, 20	8,843	38,193	11	438.9
Sensor network [N]	2	42	1,184	3	3.7
	3	42	10,662	3	4.6
MER [N R]	2, 5	5,776	427,363	4	31.8
	3, 2	16,759	171	4	210.5

- 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)

Case study [parameters]		Compositional (L*)			Compositional (NL*)		
		$\|A^{err}\|$	$\|EQ\|$	Time (s)	$\|A^{err}\|$	$\|EQ\|$	Time (s)
Client-server1	7	9	7	484.6	10	5	405.9
Client-serverN	5	7	5	191.1	8	5	201.9
Rand. cons. [N,R,K]	2, 4, 4	12	8	413.2	12	5	103.4
	3, 3, 20	11	6	438.9	15	5	411.3
MER [N R]	2, 5	4	3	31.8	7	5	154.4
	3, 2	4	3	210.5	-	-	memout

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…

$$\frac{\begin{array}{c} M_1 \models P_{\geq p_1}[A_1] \\ \langle A_1\rangle_{\geq p_1} M_2 \langle A_2\rangle_{\geq p_2} \\ \ldots \\ \langle A_n\rangle_{\geq p_n} M_n \langle G\rangle_{\geq p_G} \end{array}}{M_1 || \ldots || M_n \models P_{\geq p_G}[G]}$$

Case study [parameters]		(ASYM)	(ASYM-N)	Non-comp.
		Time (s)	Time (s)	Time (s)
Client-serverN [N]	6	memout	40.9	0.7
	7	memout	164.7	1.7
MER [N R]	3, 5	memout	29.8	48.2
	4, 5	memout	122.9	memout
	5, 5	memout	3,903.4	memout

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…