Quantitative Multi-Objective Verification for Probabilistic Systems

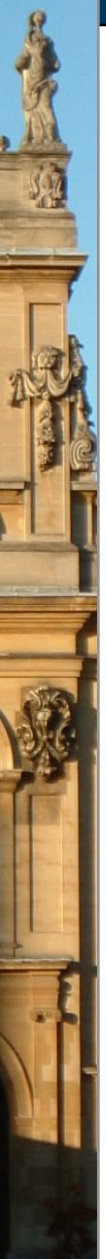
Vojtěch Forejt[1], Marta Kwiatkowska[1],
Gethin Norman[2], David Parker[1]
and Hongyang Qu[1]

[1] University of Oxford
[2] University of Glasgow

TACAS'11, Saarbrücken, March 2011

Overview

- Verification of probabilistic systems
  - probabilistic automata (or Markov decision processes)
  - quantitative verification of temporal logic specifications
- Multi-objective quantitative verification
  - formalise trade-offs between several different objectives
  - probabilistic ω-regular properties & expected reward (or cost)
  - verification, achievability & numerical queries
  - flexible property specification, efficient model checking
- Controller synthesis
  - synthesis of optimal adversaries/schedulers for MDPs (or PAs)
- Compositional probabilistic verification
  - assume-guarantee framework for probabilistic automata

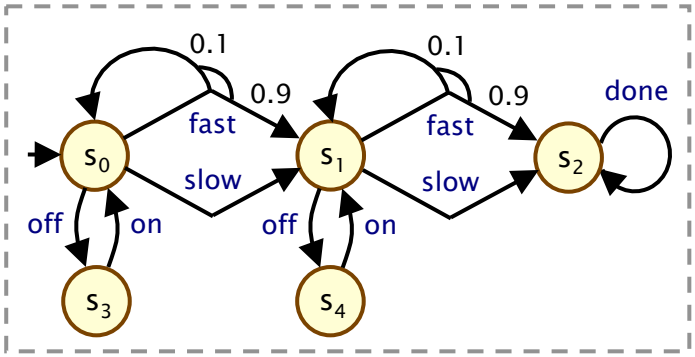


Overview

- Verification of probabilistic systems
  - probabilistic automata (or Markov decision processes)
  - quantitative verification of temporal logic specifications
- Multi-objective quantitative verification
  - formalise trade-offs between several different objectives
  - probabilistic ω-regular properties & expected total reward
  - verification, achievability & numerical queries
  - flexible property specification, efficient model checking
- Controller synthesis
  - synthesis of optimal adversaries/schedulers for PAs/MDPs
- Compositional probabilistic verification
  - assume-guarantee framework for probabilistic automata

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

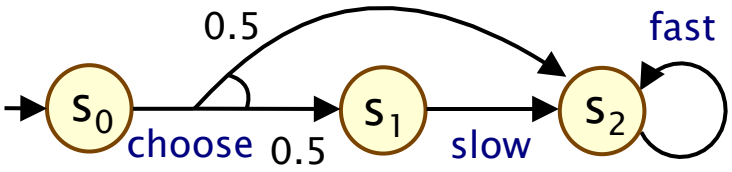


- Augment with (action-based) reward structures ρ
  - $\rho : \alpha_\rho \to \mathbb{R}_{>0}$ (where $\alpha_\rho \subseteq \alpha_M$)
  - (to model time, energy consumption, …)
- Parallel composition: $M_1 \,||\, M_2$
  - CSP style – synchronise over common actions [Segala]

Running example

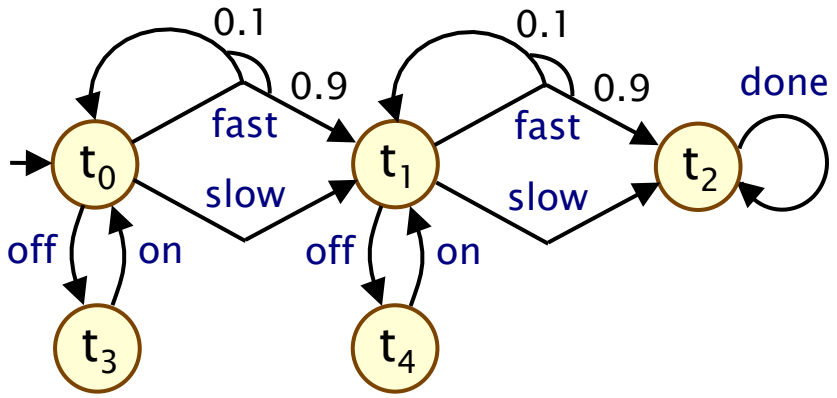
- Two components, each a probabilistic automaton:
  - M_m: a machine executing 2 tasks
  - M_c: a possible controller for the machine

PA M_c ("controller")



α_{M_c} = {choose,slow, fast, on, off}

PA M_m ("machine")



α_{M_m} = {slow, fast, on, off, done}

- Example reward structures:
  - ρ_{time}={fast↦1,slow↦3,on↦5}
  - ρ_{pow}={fast↦20,slow↦10,on↦2}

Property specifications for PAs

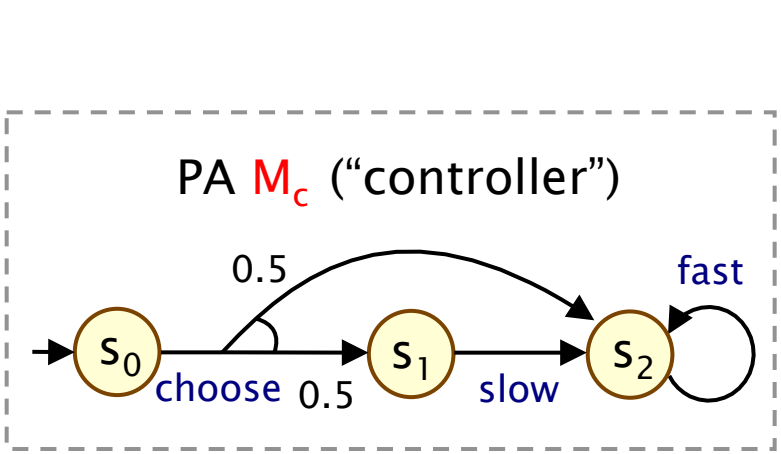
- To reason formally about PAs, we use adversaries
  - also called "schedulers", "strategies", "policies", …
  - an adversary σ resolves nondeterminism in a PA M
  - makes a (possibly randomised) choice, based on history
  - induces probability measure Pr_M^σ over (infinite) paths in M
- Probabilistic (linear-time) properties
  - we focus on action-based, ω-regular (e.g. LTL) properties
  - e.g. $\Diamond done \wedge \Box \neg off$ – "eventually finish, without switching off"
  - $Pr_M^\sigma(\phi)$ = probability of ϕ being true under adversary σ
- Reward-based properties
  - we focus on expected total reward properties
  - $ExpTot_M^\sigma(\rho)$ = expected sum of rewards ρ over paths wrt. Pr_M^σ
  - e.g. "expected total time/energy/cost/… to complete"

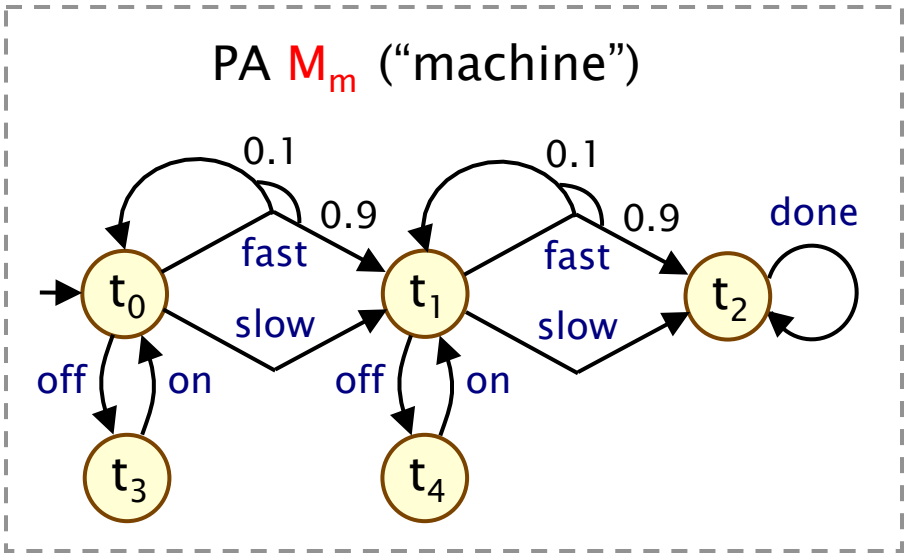
Probabilistic model checking for PAs

- Probabilistic model checking (e.g. using LTL, …)
  - usually quantify over all adversaries $\sigma \in \mathrm{Adv}_M$
  - e.g. $M \models P_{\geq p}[\phi] \Leftrightarrow \Pr_M^{\sigma}(\phi) \geq p$ for all $\sigma \in \mathrm{Adv}_M$
  - corresponds to best-/worst-case behaviour analysis
- Or, in a more *quantitative* fashion, just compute:
  - e.g. $\Pr_M^{\min}(\phi) = \inf \{ \Pr_M^{\sigma}(\phi) \mid \sigma \in \mathrm{Adv}_M \}$
  - similarly for $\Pr_M^{\max}(\phi)$, $\mathrm{ExpTot}_M^{\min}(\rho)$, $\mathrm{ExpTot}_M^{\max}(\rho)$
- Reduces to: graph-based analysis + linear program
  - for case of LTL ϕ, on (synchronous) PA-automaton product
  - only need to consider deterministic (pure) adversaries
  - efficient: complexity is polynomial in $|M|$ (but 2EXP in $|\phi|$)
    - in practice, for scalability, often approximate (e.g. value iter.)
  - tools available: PRISM, Liquor, ProbDiVinE, RAPTURE, PASS, …

Running example

- Two components, each a probabilistic automaton:
  - M_m: a machine executing 2 tasks
  - M_c: a possible controller for the machine

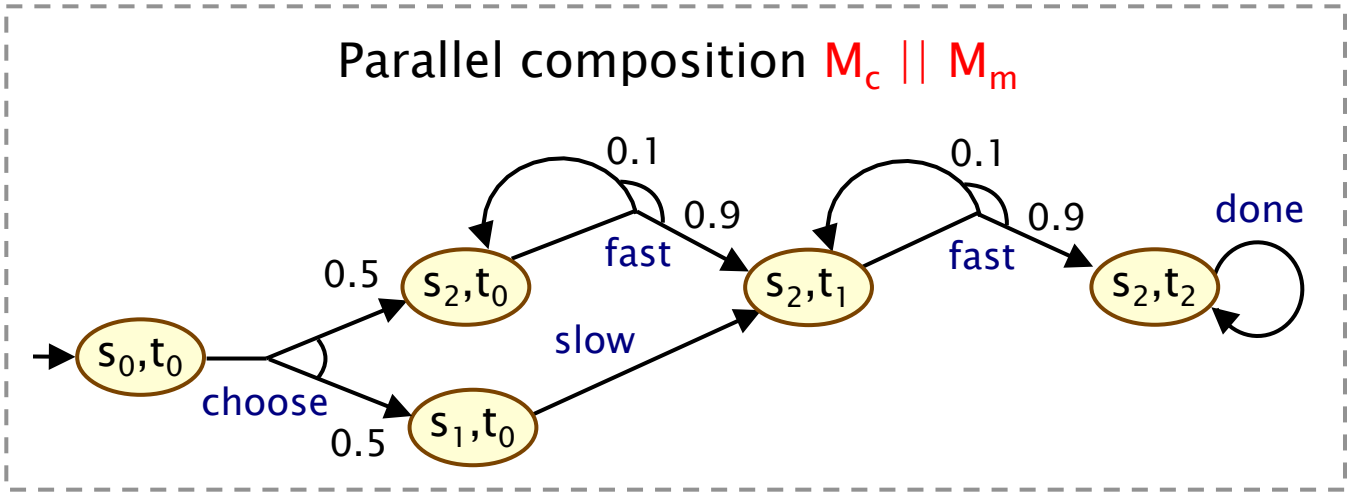




- Example properties for $M = M_c \,||\, M_m$
  - $Pr_M^{min}(\Diamond \text{done})$ – "minimum probability of termination"
  - $ExpTot_M^{max}(\rho_{time})$ – "maximum expected execution time"

Running example

- Two components, each a probabilistic automaton:
  - M_m: a machine executing 2 tasks
  - M_c: a possible controller for the machine



- Example properties for $M = M_c \,||\, M_m$
  - $Pr_M^{min}(\Diamond \text{done}) = 1$ so $M \models P_{\geq 1}[\Diamond \text{done}]$
  - $ExpTot_M^{max}(\rho_{time}) = 3.1666\ldots$ so $M \models R_{<3.2}[\rho_{time}]$

Overview

- Verification of probabilistic systems
  - probabilistic automata (or Markov decision processes)
  - quantitative verification of temporal logic specifications
- Multi-objective quantitative verification
  - formalise trade-offs between several different objectives
  - probabilistic ω-regular properties & expected total reward
  - verification, achievability & numerical queries
  - flexible property specification, efficient model checking
- Controller synthesis
  - synthesis of optimal adversaries/schedulers for PAs/MDPs
- Compositional probabilistic verification
  - assume-guarantee framework for probabilistic automata

Quantitative multi-objective properties

- Analyse trade-offs between multiple quantitative objectives
  - e.g. “expected send time vs. expected power consumption”
- Quantitative multi-objective (qmo) properties Ψ
  - boolean combinations of probabilistic/reward predicates, i.e.
  - $\Psi ::= \text{true} \mid \Psi \wedge \Psi \mid \Psi \vee \Psi \mid \neg \Psi \mid [\phi]_{\sim p} \mid [\rho]_{\sim r}$
    - where ϕ is an ω-regular (e.g. LTL) property, $p \in [0,1]$, ρ is a reward structure, $r \in \mathbb{R}_{\geq 0}$ and $\sim \in \{<, \leq, \geq, >\}$
  - example: $[\Diamond \text{done}]_{\geq 1} \wedge [\rho_{\text{pow}}]_{\leq 30}$
- Satisfaction with respect to both PA M and adversary σ
  - $M,\sigma \models [\phi]_{\sim p} \Leftrightarrow \Pr_M^\sigma(\phi) \sim p$
  - $M,\sigma \models [\rho]_{\sim r} \Leftrightarrow \text{ExpTot}_M^\sigma(\rho) \sim r$
  - obvious semantics for logical connectives…
    - e.g. $M,\sigma \models \Psi_1 \vee \Psi_2 \Leftrightarrow M,\sigma \models \Psi_1$ or $M,\sigma \models \Psi_2$

Quantitative multi-objective queries

- 3 types of queries for a qmo-property Ψ:

- Verification queries: $M \models_{\forall} \Psi$
  - is $M,\sigma \models \Psi$ satisfied *for all* adversaries σ of M?
  - also: $M \models_{\forall \text{fair}} \Psi$ – satisfaction for all *fair* adversaries
- Achievability queries: $M \models_{\exists} \Psi$
  - does *there exist* an adversary σ of M such that $M,\sigma \models \Psi$?
- Numerical queries:
  - *min/max* probability/reward subject to constraint Ψ ?
  - e.g. $\Pr_M^{\max}(\phi \mid \Psi) = \sup \{ \Pr_M^{\sigma}(\phi) \mid \sigma \in \text{Adv}_M \text{ and } M,\sigma \models \Psi \}$

- Examples…
  - $M \models_{\forall} [\Diamond \text{done}]_{>0.99} \vee [\Diamond \Box \text{off}]_{\geq 1}$
  - $M \models_{\exists} [\rho_{\text{time}}]_{\leq 5} \wedge [\rho_{\text{pow}}]_{\leq 30} \wedge [\Diamond \text{done}]_{\geq 1}$
  - $\text{ExpTot}_M^{\min}(\rho_{\text{time}} \mid [\rho_{\text{pow}}]_{\leq 30} \wedge [\Diamond \text{done}]_{\geq 1})$

Model checking qmo-properties

- Just consider numerical queries
  - verification query = (negated) achievability query
  - achievability query = numerical query with dummy objective
- Just consider conjunctions of probability/reward predicates
  - convert to disjunctive normal form, check separately

- So let's assume a query of the form $\mathrm{ExpTot}_M^{\max}(\rho_0 \mid \Psi)$
  - where $\Psi = ([\phi_1]_{\sim p_1} \wedge \ldots \wedge [\phi_n]_{\sim p_n}) \wedge ([\rho_1]_{\sim r_1} \wedge \ldots \wedge [\rho_m]_{\sim r_m})$

- First, we impose the following assumption:
  - $\sup \{ \mathrm{ExpTot}_M^{\sigma}(\rho) \mid M,\sigma \models \wedge_i [\phi_i]_{\sim p_i} \} < \infty$
    for all $\rho \in \{\rho_0, \rho_1, \ldots, \rho_m\}$ that are being maximised
  - verifiable during model checking

Model checking qmo-properties

- Algorithm summary for $\mathrm{ExpTot}_M^{\max}(\rho_0 \mid \Psi)$
  - convert probabilistic predicates to $[\phi_i]_{\triangleright p_i}$ where $\triangleright \in \{\geq, >\}$
  - build product M' of PA M and determ. Rabin automata for ϕ_i
  - check assumption, remove actions yielding infinite rewards
  - convert predicates $[\phi_i]_{\triangleright p_i}$ to reward predicates $[\lambda_i]_{\triangleright p_i}$
  - build and solve (dual) linear programming problem
  - yielding randomised (memoryless) adversary σ' of M'
  - convert to randomised (finite-memory) adversary σ of M
  - similar approach to [Etessami et al., TACAS'07]

- Complexity: polynomial in |M| (but 2EXP in property)
  - i.e. same as for standard (single-objective) verification
  - in practice, slightly less scalable since can't use value iteration

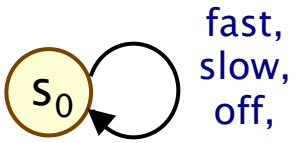
Overview

- Verification of probabilistic systems
  - probabilistic automata (or Markov decision processes)
  - quantitative verification of temporal logic specifications
- Multi-objective quantitative verification
  - formalise trade-offs between several different objectives
  - probabilistic ω-regular properties & expected total reward
  - verification, achievability & numerical queries
  - flexible property specification, efficient model checking
- Controller synthesis
  - synthesis of optimal adversaries/schedulers for PAs/MDPs
- Compositional probabilistic verification
  - assume-guarantee framework for probabilistic automata

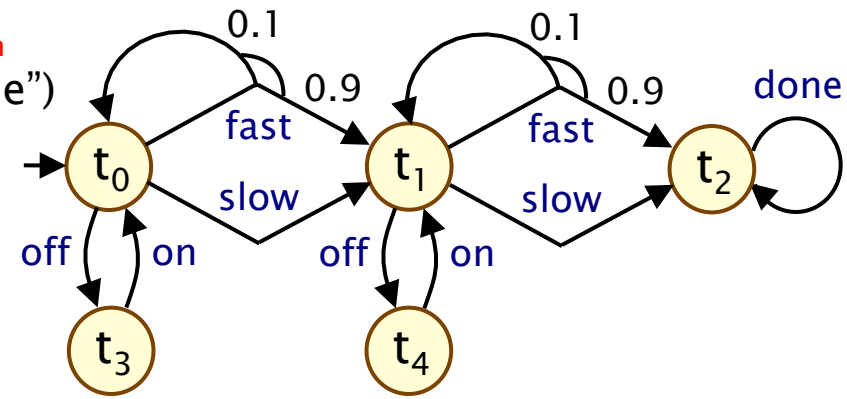
Controller synthesis

- Achievability and numerical queries are directly applicable to the problem of controller synthesis
- Running example, using numerical query:
  - $\text{ExpTot}_M^{\min}(\rho_{\text{time}} \mid [\rho_{\text{pow}}]_{\leq 30} \wedge [\Diamond \text{done}]_{\geq 1})$ on $M = M_d \,||\, M_m$
  - i.e. "minimise expected completion time, subject to upper bound on expected energy consumption"

PA M_d ("dummy controller")



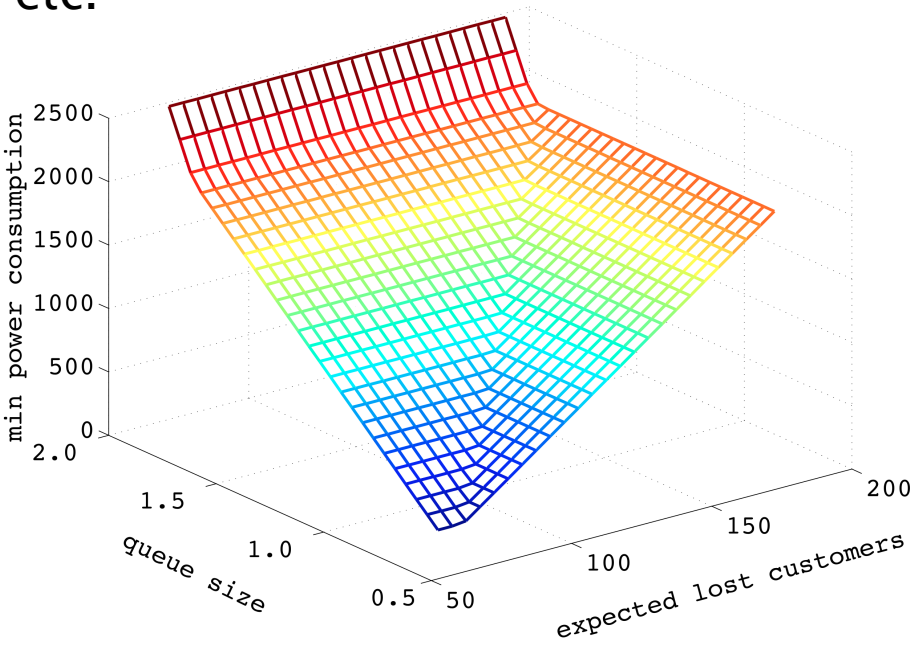
PA M_m ("machine")



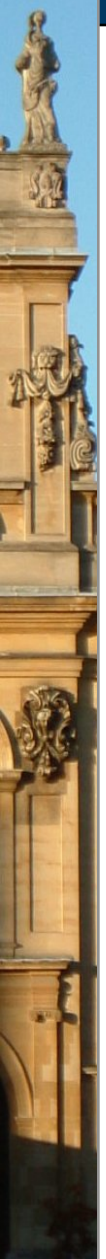
- Result: minimum expected time: $49/11 = 4.4545\ldots$
  - controller: job 1 fast/slow with prob. 5/6 and 1/6; job 2 slow

Case study: Dynamic power management

- Synthesis of dynamic power management schemes
  - for an IBM TravelStar VP disk drive
  - 5 different power modes: active, idle, idlelp, stby, sleep
  - power manager controller bases decisions on current power mode, disk request queue, etc.
- Build controllers that
  - minimise energy consumption, subject to constraints on e.g.
  - probability that a request waits more than K steps
  - expected number of lost disk requests



- See: http://www.prismmodelchecker.org/files/tacas11/



Overview

- Verification of probabilistic systems
  - probabilistic automata (or Markov decision processes)
  - quantitative verification of temporal logic specifications
- Multi-objective quantitative verification
  - formalise trade-offs between several different objectives
  - probabilistic ω-regular properties & expected total reward
  - verification, achievability & numerical queries
  - flexible property specification, efficient model checking
- Controller synthesis
  - synthesis of optimal adversaries/schedulers for PAs/MDPs
- Compositional probabilistic verification
  - assume-guarantee framework for probabilistic automata

Compositional verification

- Goal: scalability through modular verification
  - e.g. decide if $M_1 || M_2 \models G$ using separate analysis of M_1, M_2
- Assume-guarantee (A/G) reasoning
  - use assumptions A about the context of a component M
  - $\langle A \rangle\ M\ \langle G \rangle$ – "whenever M is part of a system that satisfies A, then the system must also guarantee G"
  - example (asymmetric) A/G proof rule:

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

Probabilistic assume guarantee

- Assumptions Ψ_A and guarantees Ψ_G are qmo-properties
  - i.e. combinations of probabilistic ω-regular properties (incl. probabilistic safety, liveness), expected total reward properties
- Assume-guarantee triples $\langle \Psi_A \rangle$ M $\langle \Psi_G \rangle$ for a PA M
  - checking reduces to qmo verification query
- Extends our earlier A/G framework for PAs [TACAS'10]
  - where A and G are probabilistic safety properties
  - much richer class of properties for G and (crucially) A
- Adapt proof rules to incorporate (unconditional) fairness
  - with probability 1, M_1 and M_2 make transitions infinitely often

Probabilistic assume guarantee triple

- Assume-guarantee triple $\langle \Psi_A \rangle\ M\ \langle \Psi_G \rangle$
- Informally:
  - "when M is a component of a system satisfying Ψ_A, then the combined system (under fairness) is guaranteed to satisfy Ψ_G"
- Formally:

$$\langle \Psi_A \rangle\ M\ \langle \Psi_G \rangle$$
$$\Leftrightarrow$$
$$\forall \sigma \in Adv_{M'} (M', \sigma \models \Psi_A \rightarrow M', \sigma \models \Psi_G)$$
$$\Leftrightarrow$$
$$M' \models_\forall (\neg \Psi_A \vee \Psi_G)$$

qmo verification query

  - where $M' = M[\alpha_A]$, i.e. M with alphabet extended to that of Ψ_A

An assume-guarantee rule

- The following (asymmetric) proof rule holds

$$\frac{M_1 \models_{\forall} \Psi_A \qquad \langle \Psi_A \rangle\, M_2\, \langle \Psi_G \rangle}{M_1 \,||\, M_2 \models_{\forall \text{fair}} \Psi_G} \quad \text{(ASYM)}$$

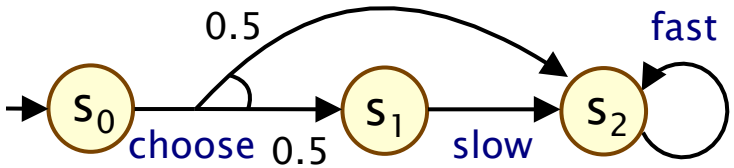
- So, verifying $M_1 \,||\, M_2 \models_{\forall \text{fair}} \Psi_G$ reduces to 2 sub-problems:
  - premise 1: $M_1 \models_{\forall} \Psi_A$ – standard model checking (usually)
  - premise 2: $\langle \Psi_A \rangle\, M_2\, \langle \Psi_G \rangle$ – multi-objective model checking

- Compositional verification can be much more efficient
  - for small assumption Ψ_A about large M_1

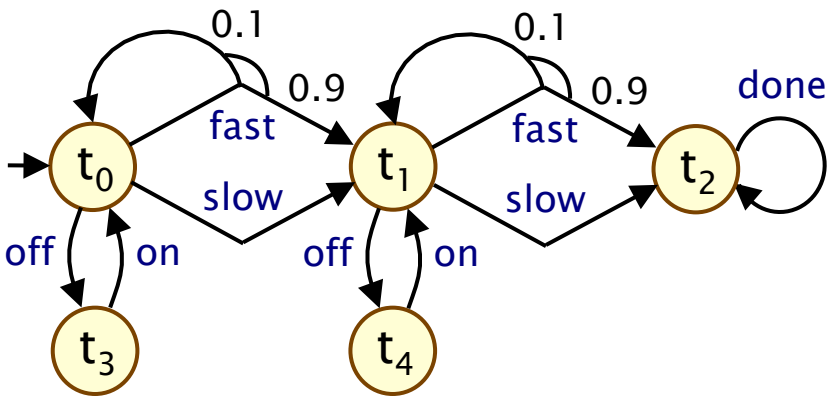
Running example

- Compositional probabilistic model checking:
  - verify that: $M_c \,||\, M_m \models_{\forall \text{fair}} [\rho_{\text{time}}]_{\leq 3.2}$
    ("expected completion time $\leq$ 3.2")
  - using rule (ASYM) with assumption: $\Psi_A = [\Box \neg \text{off}]_{\geq 1} \wedge [\rho_{\text{slow}}]_{\leq 0.5}$
    ("never switch off and expected num. of slow jobs $\leq$ 0.5")

PA M_c ("controller")



PA M_m ("machine")



Reward structures: $\rho_{\text{slow}} = \{\text{slow} \mapsto 1\}$ $\rho_{\text{time}} = \{\text{fast} \mapsto 1, \text{slow} \mapsto 3, \text{on} \mapsto 5\}$

Running example

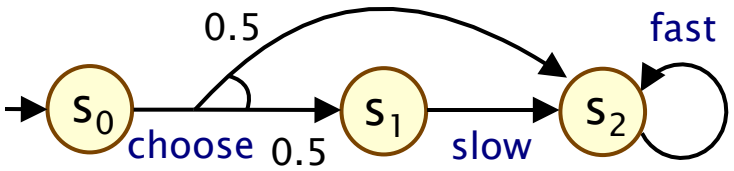
- Premise 1:
  - verify that: $M_c \models_\forall [\Box \neg \text{off}]_{\geq 1} \wedge [\rho_{\text{slow}}]_{\leq 0.5}$
  - yes, since $Pr_M^{\min}(\neg \text{off}) = 1$ and $\text{ExpTot}(\rho_{\text{slow}}) = 0.5$

$$\frac{M_c \models_\forall \langle \Psi_A \rangle \quad \langle \Psi_A \rangle\, M_m\, \langle [\rho_{\text{time}}]_{\leq 3.2} \rangle}{M_c \,||\, M_m \models_{\forall \text{fair}} [\rho_{\text{time}}]_{\leq 3.2}}$$

with assumption

$$\Psi_A = [\Box \neg \text{off}]_{\geq 1} \wedge [\rho_{\text{slow}}]_{\leq 0.5}$$

PA M_c ("controller")



$\rho_{\text{slow}} = \{\text{slow} \mapsto 1\}$

Running example

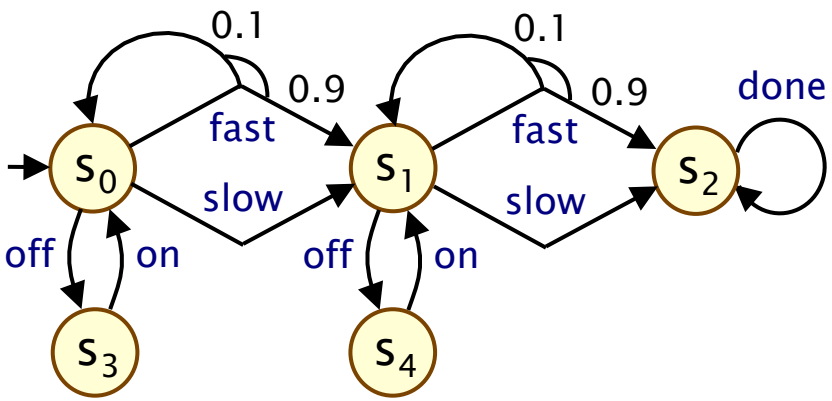
- Premise 2:
  - verify that: $M_m \models_\forall ([\Box \neg off]_{\geq 1} \wedge [\rho_{slow}]_{\leq 0.5}) \rightarrow [\rho_{time}]_{\leq 3.2}$
  - yes, since $M_m \models_\exists ([\Box \neg off]_{\geq 1} \wedge [\rho_{slow}]_{\leq 0.5} \wedge [\rho_{time}]_{>3.2})$ is false

$$\frac{M_c \models_\forall \langle \Psi_A \rangle \quad \langle \Psi_A \rangle\, M_m\, \langle [\rho_{time}]_{\leq 3.2} \rangle}{M_c \,||\, M_m \models_{\forall fair} [\rho_{time}]_{\leq 3.2}}$$

with assumption

$$\Psi_A = [\Box \neg off]_{\geq 1} \wedge [\rho_{slow}]_{\leq 0.5}$$

PA M_m ("machine")



$\rho_{slow} = \{slow \mapsto 1\}$

$\rho_{time} = \{fast \mapsto 1, slow \mapsto 3, on \mapsto 5\}$

"Quantitative" assume-guarantee

- A more "quantitative" approach
  - use numerical queries to obtain best/worst-case bounds

$$\frac{M_1 \models_{\forall} \Psi_A \qquad \langle \Psi_A \rangle\ M_2\ \langle \Psi_G \rangle}{M_1 \,||\, M_2 \models_{\forall \text{fair}} \Psi_G}$$

- For example:
  - if Ψ_G is of the form $[\rho]_{\leq r}$ (upper bound on expected reward)
  - then, instead of checking premise 2: $\neg(M_2 \models_{\exists} (\Psi_A \wedge \neg \Psi_G))$
  - compute $\text{ExpTot}_{M_2}^{\max}(\rho \mid \Psi_A)$ (worst-case under assumption)
- In similar style, we can "optimise" our assumptions
  - e.g. if Ψ_A is of the form $[\phi]_{\geq p}$
  - we can compute $p^* = \Pr_{M_1}^{\min}(\phi)$
  - and use assumption $[\phi]_{\geq p^*}$ instead
- We can also study multi-objective LP, Pareto curves, …

Implementation + Case studies

- Extension of PRISM model checker
  - already supports LTL and reward properties for PAs/MDPs
  - added support for multi-objective model checking
    - using LP solvers (ECLiPSe/COIN-OR CBC/lpsolve)
  - fully-automated support for A/G reasoning in progress

- Two large case studies
  - randomised consensus algorithm [Aspnes & Herlihy]
    - maximum expected steps required in the first R rounds
  - Zeroconf network protocol
    - termination with (minimum) probability 1
    - minimum/maximum expected time for termination

- See: http://www.prismmodelchecker.org/files/tacas11/

Experimental results

Case study [parameters] & Property		Non-compositional		Compositional	
		States	Time (s)	LP size	Time (s)
Randomised consensus (3 proc.s) [R,K] "max. steps"	3, 2	114,559	20.5	43,712	12.1
	3, 12	507,919	1,361.6	92,672	284.9
	3, 20	822,607	time-out	131,840	901.8
	4, 2	3,669,649	728.1	260,254	118.9
	4, 12	29,797,249	mem-out	351,694	642.2
	4, 20	65,629,249	mem-out	424,846	1,697.0
ZeroConf [K] "termination"	4	57,960	8.7	155,458	23.8
	6	125,697	16.6	156,690	24.5
	8	163,229	19.4	157,922	25.5
ZeroConf [K] "max. time"	4	57,960	5.8	154,632	23.7
	6	125,697	13.3	155,600	24.2
	8	163,229	18.9	156,568	25.1

Experimental results

Case study [parameters] & Property		Non-compositional		Compositional	
		States	Time (s)	LP size	Time (s)
Randomised consensus (3 proc.s) [R,K] "max. steps"	3, 2	114,559	20.5	43,712	12.1
	3, 12	507,919	1,361.6	92,672	284.9
	3, 20	822,607	time-out	131,840	901.8
	4, 2	3,669,649	728.1	260,254	118.9
	4, 12	29,797,249	mem-out	351,694	642.2
	4, 20	65,629,249	mem-out	424,846	1,697.0
ZeroConf [K] "termination"	4	57,960	8.7	155,458	23.8
	6	125,697	16.6	156,690	24.5
	8	163,229	19.4	157,922	25.5
ZeroConf [K] "max. time"	4	57,960	5.8	154,632	23.7
	6	125,697	13.3	155,600	24.2
	8	163,229	18.9	156,568	25.1

- Compositional verification: faster, larger models, scales better

Experimental results

Case study [parameters] & Property		Non-compositional		Compositional	
		States	Result	LP size	Result
Randomised consensus (3 proc.s) [R,K] "max. steps"	3, 2	114,559	212.0	43,712	214.3
	3, 12	507,919	4,352	92,672	4,352
	3, 20	822,607	-	131,840	11,552
	4, 2	3,669,649	212.0	260,254	260.3
	4, 12	29,797,249	-	351,694	4,533
	4, 20	65,629,249	-	424,846	11,840
ZeroConf [K] "termination"	4	57,960	1.0	155,458	1.0
	6	125,697	1.0	156,690	1.0
	8	163,229	1.0	157,922	1.0
ZeroConf [K] "max. time"	4	57,960	14.28	154,632	17.33
	6	125,697	18.28	155,600	22.67
	8	163,229	22.28	156,568	28.00

- Compositional verification: yields good bounds on actual results

Conclusions

- Multi-objective model checking techniques for PAs/MDPs
  - simple, temporal-logic-based language with:
  - probabilistic ω-regular and expected total reward properties
  - verification, achievability and numerical queries
- Applications to controller synthesis
  - large case study: dynamic power management for disk-drive
- Compositional probabilistic verification
  - assume-guarantee framework for probabilistic automata
  - richer assumptions/guarantees; quantitative results
  - good experimental results: faster verification, larger models
- Current/future work
  - assumption generation via learning (done for safety prop.s)
  - continuous- or real-time models