Quantitative Verification: Formal Guarantees for Timeliness, Reliability and Performance

Dave Parker
University of Birmingham

FMICS 2014, Florence, September 2014
Quantitative Verification

Formal Guarantees for Timeliness, Reliability and Performance

A Knowledge Transfer Report from the London Mathematical Society and Smith Institute for Industrial Mathematics and System Engineering

By Gethin Norman and David Parker
Outline

- Quantitative verification
- Probabilistic model checking
- Case studies
- Challenges & current directions
- Verification vs. controller synthesis
Quantitative verification

• Adds quantitative aspects (to models and properties)
  – probability, time, costs, rewards, ...

• Probability
  – physical components can fail
  – communication media are unreliable
  – algorithms/protocols use randomisation

• Time
  – delays, time-outs, failure rates, ...

• Costs & rewards
  – energy consumption, resource usage, ...
  – profit, incentive schemes, ...
Quantitative verification

• Correctness properties are quantitative
  – “the probability of an airbag failing to deploy within 0.02 seconds of being triggered is at most 0.001”
  – “with probability 0.99, the packet arrives within 10 ms”

• Beyond correctness:
  – reliability, timeliness, performance, efficiency, ...
  – “the expected energy consumption of the sensor is…”
  – “the probability of the robot visiting all sites in < 10 min…”
Probabilistic model checking
Model checking

• Automated verification: model checking
  – exhaustive construction/analysis of finite-state model
  – correctness properties expressed in temporal logic
  – very successful in practice

• Why it works
  – temporal logic: expressive, tractable
  – fully automated, tools available
  – not just verification, but falsification (bug hunting) via counterexamples

\[ A[G (\text{trigger} \rightarrow X\text{ deploy})] \]
Probabilistic model checking

- Construction and analysis of probabilistic models
  - for example: discrete-time Markov chains (DTMCs)
  - transitions labelled with probabilities
  - from a description in a high-level modelling language

- Correctness properties expressed in probabilistic temporal logic, e.g. PCTL
  - \( \text{trigger} \rightarrow P_{\geq 0.999} [ F^{\leq 2} \text{deploy} ] \)
  - “the probability of the airbag deploying within 2 time units of being triggered is at least 0.999”
Probalistic model checking

• A (brief) early history
  – late 80s, early 90s: first underlying theory developed
  – late 90s: first PROBMIV workshops
  – 2000: first versions of PRISM and MRMC released

• What advances have been made?
• What are the strengths and weaknesses?
• Where and why does it work well?
Probabilistic model checking

• Flexible and widely applicable
  – techniques developed for many probabilistic models
  – and many types of properties, temporal logics, etc.

- discrete-time Markov chains (DTMCs)
- continuous-time Markov chains (CTMCs)
- Markov decision processes (MDPs)
- probabilistic automata (PAs)
- probabilistic timed automata (PTAs)
- continuous-time MDPs (CTMDPs)
- interactive Markov chains (IMCs)
- Markov automata (MAs)
- probabilistic hybrid automata (PHAs)
- stochastic multiplayer games (SMGs)
- PCTL, LTL, PCTL*, CSL, aCSL, PTCTL, MiTL, PATL, rPATL, …
Probabilistic model checking

• Flexible and widely applicable
  – techniques developed for many probabilistic models
  – and many types of properties, temporal logics, etc.

• Draws upon many different methods
  – and overlaps with many different disciplines

| graph algorithms, linear equations, linear programming, numerical fixed points, integral equations, differential equations, numerical approximations, … | model checking, performance analysis, optimisation, artificial intelligence & planning, control theory, machine learning, … |
Probabilistic model checking

• Flexible and widely applicable
  – techniques developed for many probabilistic models
  – and many types of properties, temporal logics, etc.

• Draws upon many different methods
  – and overlaps with many different disciplines

• Usable and efficient tool support available
  – PRISM, MRMC, Modest Toolset, ...
  – applied in many different application domains
Key strengths

• As for conventional model checking:
  – fully automated techniques and tools
  – precise, unambiguous models/properties

• Yields numerical results
  – (probabilities, response times, etc.)
  – results show trends, flaws, anomalies
  – numerical results are "exact"

• Combines numerical & exhaustive analysis
  – e.g. exhaustive search over reachable states or resolutions of nondeterminism
  – also: probabilistic counterexamples

\[ \text{trigger} \to P_{\geq 0.999} [ F^{\leq 2} \text{ deploy} ] \]
Case studies
Case study: Bluetooth

• Device discovery between a pair of Bluetooth devices
  – performance essential for this phase

• Complex discovery process
  – two asynchronous 28-bit clocks
  – pseudo-random hopping between 32 frequencies
  – random waiting scheme to avoid collisions
  – 17,179,869,184 initial configurations

• Probabilistic model checking (PRISM)
  – “probability discovery time exceeds 6s is always < 0.001”
  – “worst-case expected discovery time is at most 5.17s”
Case study: An airbag system

• Failure analysis for a car airbag system
  – TRW Automotive + Uni Konstanz/Swinburne [Aljazzar et al.'09]
  – compared design variants with one/two crash evaluators

• Methods used
  – probabilistic FMEA (Failure Mode and Effects Analysis)
  – probabilistic model checking (CTMCs + PRISM + counter-examples) used for a more formal and efficient approach
  – ASIL D (Automated Safety Integration Level D) for unintended airbag deployment, formulated in CSL

• Results & conclusions
  – detected violations, identified critical aspect (with cex.s)
  – language/tools suffice, difficulties with temporal logic
Further case studies

• Software reliability evaluation
  – e.g. industrial process control system [ABB, Koziolek et al.'12]

• Performance analysis & optimisation
  – e.g. cloud resource management [Fujitsu, Kikuchi et al.'11]
  – e.g. dynamic power management

• Network & communication protocols

• Security: e.g. anonymity networks, pin cracking

• Robotics: e.g. motion navigation planning

• Systems biology & DNA computing

• See:  www.prismmodelchecker.org/casestudies/
Challenges & directions
Challenges & directions

1. **Scalability and efficiency**
   - efficient data structures (e.g. symbolic)
   - parallelisation, multi-core, GPUs, ... 
   - statistical model checking (simulation-based)
   - abstraction and compositional frameworks

2. **Robustness and accuracy**
   - parametric probabilistic verification
   - probabilistic models with uncertainty
   - counterexamples/witnesses/certificates
Challenges & directions

3. Mainstream languages
   - many tools rely on custom modelling languages
   - increased (e.g. industrial) take-up need better support for mainstream programming/modelling languages
   - some work on UML, SysML, AADL (often via translation)

4. Cyber-physical systems
   - embedded sensing/control + close interaction with physical environment: automotive, avionics, medical, ...
   - combination of discrete/continuous aspects brings many challenges for modelling, analysis and verification
Controller synthesis
Controller synthesis

• Verification vs. synthesis
  – verification = check that a (model of) system satisfies a specification of correctness
  – synthesis = build a "correct-by-construction" system directly from a correctness specification

• Controller synthesis
  – generate a controller/scheduler that chooses actions such that a correctness specification is \textit{guaranteed} to hold
  – build a \textit{probabilistic model} incorporating both the controller and the system being controlled
  – formally specify \textit{correctness} properties in temporal logic
Markov decision processes

- Markov decision processes (MDPs)
  - generalise DTMCs by adding **nondeterminism**

- Nondeterminism: unknown behaviour
  - concurrency, abstraction, user input, control

- Strategies (or "policies", "adversaries", "schedulers")
  - resolve nondeterminism based on current history
Verification vs. controller synthesis

• Two (dual) problems:

• 1. Verification
  – quantify over all possible strategies (i.e. worst-case)
  – \( P_{<0.01} [ F_{err} ] \): “the probability of error is always < 0.01”
  – applications: randomised communication protocols, randomised distributed algorithms, security, ...

• 2. Controller (strategy) synthesis
  – \( P_{<0.01} [ F_{err} ] \): "does there exist a strategy for which the probability of an error occurring is < 0.01?"
  – applications: robotics, power management, security, ...
Multiple objectives

• Multi-objective controller synthesis
  – and/or multi-objective probabilistic model checking
  – investigate trade-offs between conflicting objectives

• Examples
  – “is there a strategy such that the probability of message transmission is > 0.95 and the expected battery life > 10 hrs?”
  – e.g. “maximum probability of message transmission, assuming expected battery life-time is > 10 hrs?”
  – e.g. "Pareto curve for maximising probability of transmission and expected battery life-time”
Applications

• Examples of PRISM-based controller synthesis

Synthesis of dynamic power management controllers [TACAS'11]

Motion planning for a service robot using LTL [IROS'14]

Synthesis of team formation strategies [CLIMA'11, ATVA'12]

Minimise disk drive energy consumption, subject to constraints on:
(i) expected job queue size;
(ii) expected number of lost jobs

Pareto curve:
x="probability of completing task 1";
y="probability of completing task 2";
z="expected size of successful team"
Other extensions

• Controller synthesis with stochastic games
  – player 1 = controller (as for MDPs)
  – player 2 = environment ("uncontrollable" actions)
  – more generally: models competitive and/or collaborative behaviour between multiple players

• Controller synthesis with multi-strategies
  – strategies which can choose between multiple actions at each time step
  – flexible/adaptable strategy, whilst still guaranteeing some property
  – uses penalty schemes to measure permissivity
Conclusions

• Quantitative verification
  – probabilistic model checking
  – formal methods for correctness, reliability, performance, ...
  – flexible approach, wide range of applications
  – exact numerical results + exhaustive analysis

• Challenges and directions
  – scalability + efficiency: state space explosion
  – accuracy + robustness
  – user friendly languages for model/property specification
  – controller synthesis: correctness by construction