Modelling and Analysis
(and towards Synthesis)

Marta Kwiatkowska
Department of Computer Science, University of Oxford

Dagstuhl meeting 15041, 19–23rd Jan 2015
Outline

• Introduction
  – The context: role of models in software engineering
  – Probabilistic models and quantitative verification
  – The PRISM model checker

• A taste of quantitative modelling, verification and synthesis
  – Markov decision process model
  – Goals and objectives in temporal logic
  – How to do verification and strategy synthesis

• Quantitative runtime verification
  – Verification meets autonomic computing

• Challenges and future directions
Users expect: **predictability & high integrity** in presence of
  - component failure
  - environmental uncertainty
  - conflicting requirements

**Probability useful to quantify**
  - unreliability
  - uncertainty

**Quantitative properties**
  - reliability, performance, quality of service, ...
  - “the probability of an airbag failing to deploy within 0.02s”

**How to ensure safety, dependability, performance?**
  - complex scenarios, recovery from faults, limited resources, ...
What is self-aware computing?

- **Best effort description: self-adaptive**
  - continuously **interacts** with its environment
    - including other components or human users
  - can **monitor** state of (internal and environmental) parameters
  - has **goals/objectives**
    - expressed formally in e.g. temporal logic
    - often quantitative
    - may conflict
  - takes **decisions** based on current state and external events
  - able to **adapt/reconfigure** so as to maintain the satisfaction of the objectives

- **NB reactive, parametric systems (no reflection)**
  - but multi-agent systems in AI have a notion of self
Rigorous software engineering

- **Verification and validation (V&V)**
  - Derive model, or extract from software artefacts
  - Verify correctness, validate if fit for purpose

Diagram:
- Informal requirements
- Formalise
- Formal specification
- Verification
- Simulation
- Validation
- Model
- Abstract
- Refine
- System
Model-centric approach

- **Employ (quantitative) formal models**
  - rigorous, unambiguous
  - can be derived or extracted from code
  - can also be used at runtime

- **Specify goals/objectives/properties in temporal logic:**
  - energy efficiency, performance, resource usage, …
  - *(energy)* “maximum expected energy consumption in 1 hr is at most 10mA”
  - *(performance)* “the packet will be delivered with high probability in 10ms”

- **Focus on automated, tool-supported methodologies**
  - design-time verification via **model checking**
  - **runtime** verification
  - **synthesis** from specifications
Model checking

- System
- Finite-state model
- Temporal logic specification
- Model checker e.g. SMV, Spin
- Result
- Counter-example

\[ \neg \mathit{EF} \text{ fail} \]
Design-time quantitative verification

System

Probabilistic model
e.g. Markov chain

Probabilistic temporal logic specification
e.g. PCTL, LTL

P<0.1 [ F fail ]

System requirements

Probabilistic model checker
e.g. PRISM

Result

Quantitative results

Counter-example
Historical perspective

- 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
    [Kwiatkowska, Norman, Parker, …]
  - ETMCC (now MRMC): model checking for continuous-time Markov chains
    [Baier, Hermanns, Haerkort, Katoen, …]

- Now mature area, of industrial relevance
  - successfully used by non-experts for many application domains, but full automation and good tool support essential
    - distributed algorithms, communication protocols, security protocols, biological systems, quantum cryptography, planning…
  - genuine flaws found and corrected in real-world systems
Tool support: PRISM

• **PRISM: Probabilistic symbolic model checker**
  – developed at Birmingham/Oxford University, since 1999
  – free, open source software (GPL), runs on all major OSs

• **Support for:**
  – models: DTMCs, CTMCs, MDPs, PTAs, SMGs, …
  – properties: PCTL/PCTL*, CSL, LTL, rPATL, costs/rewards, …

• **Features:**
  – simple but flexible high-level modelling language
  – user interface: editors, simulator, experiments, graph plotting
  – multiple efficient model checking engines (e.g. symbolic)

• **Many import/export options, tool connections**
  – MRMC, INFAMY, DSD, Petri nets, Matlab, …

• **See:** [http://www.prismmodelchecker.org/](http://www.prismmodelchecker.org/)
Probabilistic models: discrete time

- **Discrete–time Markov chains (DTMCs)**
  - discrete states + discrete probability
  - for: randomisation, unreliable communication media, ...

- **Markov decision processes (MDPs)**
  - discrete states + discrete probability + nondeterminism (e.g. for concurrency, control)
  - for: randomised distributed algorithms, security protocols, ...

- **Stochastic multi–player games (SMGs)**
  - discrete states + discrete probability + player nondeterminism (e.g. for collaboration and competition)
  - for: team working, coalitions, user–centric networks, smartgrid protocols, ...

NB all supported by PRISM
Probabilistic models: continuous time

- **Continuous-time Markov chains (CTMCs)**
  - discrete states + **exponentially distributed delays**
  - for: component failures, job arrivals, molecular reactions, ...

- **Interactive Markov chains (CTMCs)**
  - discrete states + exponentially distributed delays + **nondeterminism**
  - for: job arrivals in a distributed environment

- **Probabilistic timed automata (PTAs)**
  - discrete probability, nondeterminism + **real-time**
  - for wireless comm. protocols, embedded control systems, ...

NB  Shown in grey not supported by PRISM
Probabilistic models: continuous space

• Labelled Markov processes (LMPs)
  – continuous states + stochastically distributed jumps
  – for: aircraft controllers, search and rescue,…

• Probabilistic hybrid automata (PHAs)
  – discrete probability, nondeterminism + continuous flows (ODEs)
  – for embedded control systems, automotive controllers, …
  – Supported by tool chain PHAVER+PRISM

• Stochastic hybrid systems (PHAs)
  – continuous probability, nondeterminism + continuous flows
  – for embedded control systems, automotive controllers, …
Probability elsewhere

• In performance modelling
  – pioneered by Erlang, in telecommunications, ca 1910
  – models: typically continuous time Markov chains
  – emphasis on steady-state and transient probabilities

• In stochastic planning and decision support
  – cf Bellman equations, ca 1950s
  – models: Markov decision processes
  – emphasis on finding optimum policies

• In control engineering
  – models: stochastic hybrid systems
  – emphasis on finding optimal controllers

• Our focus distinctive, on automated verification
  – temporal logic specifications
  – automata-theoretic techniques
Model derivation techniques

• Models are typically state-transition systems (automata)
• Manual construction
  – derive a model from description
    • e.g. IEEE standards document
  – express in high-level language, then build
• Automated extraction
  – extract a model from software
    • using e.g. abstract interpretation, slicing, static analysis…
  – build a control flow graph
  – implemented in CProver + PRISM
• Challenges
  – state space explosion, infinite state systems
  – need to consider augmenting with additional information
    • action labels, state labels, time, probability, rate, etc
bool fail = false;
int c = 0;
int main ()
{
    // nondeterministic
    c = user-input (3);
    while (! fail && c > 0)
    {
        // probabilistic
        fail = Bernoulli (0.1);
        c --;
    }
}
Probabilistic program as MDP

Probabilistic program

MDP semantics

minimum/maximum probability of the program terminating with fail being true is 0 and 0.19, respectively
PRISM – Property specification

- **Temporal logic-based property specification language**
  - subsumes PCTL, CSL, probabilistic LTL, PCTL*, ...

- **Simple examples:**
  - $P_{\leq 0.01} [ F \text{ “crash” } ]$ – “the probability of a crash is at most 0.01”
  - $P_{\max > 0.999} [ F_{<10.5} \text{ “send” } ]$ – “the maximum probability of eventually sending in 10.5 time units is > 0.999” (QoS)

- **Usually focus on quantitative (numerical) properties:**
  - $P_{=?} [ F \text{ “crash” } ]$  
    “what is the probability of a crash occurring?”
  - then analyse trends in quantitative properties as system parameters vary
PRISM – Underlying techniques

• Construction and analysis of finite probabilistic models
  – specified in high-level modelling formalisms

• Basic ingredients
  – graph-based algorithms, e.g. reachability, qualitative verif.
  – numerical solution techniques, e.g. probability computation
  – usually rely on iterative methods: uniformisation-based for transient properties, Gauss–Seidel/etc. for linear equations
  – Monte Carlo simulation
  – simulation-based approximate model checking

• Symbolic implementation
  – data structures based on binary decision diagrams
  – fast construction + compact storage of huge models possible
  – exploit structure, regularity in high-level model
  – usually: up to $10^7$–$10^8$ states; sometimes: up to $10^{10}$ states
Approximate (statistical) model checking

- Approximate (statistical) probabilistic model checking
  - discrete event (Monte Carlo) simulation + sampling
- Two distinct approaches (both implemented in PRISM)
- Estimation [Hérault et al.]
  - approximate result for quantitative query such as $P_{=?} [\phi]$
  - plus a probabilistic guarantee regarding result precision
  - $\text{Prob}( |p_{\text{actual}} - p_{\text{estimated}}| \leq \epsilon ) \geq 1-\delta$
  - can also generate corresponding confidence intervals
- Hypothesis testing/acceptance sampling [Younes/Simmons]
  - applied to boolean-valued queries such as $P_{\sim p} [\phi]$
  - basic idea: stop sampling as soon as the result can be shown to be either true or false with high probability
  - sensitive to distance between bound $p$ and actual answer
  - also extended to Bayesian approaches [Jha et al.]
Approximate (statistical) model checking

• **Advantages**
  - much more **scalable** than conventional (numerical computation based) probabilistic model checking
  - (almost no scalability issues – no need to build model)
  - wider range of **model types** (anything that can be effectively simulated) and **property types**

• **Disadvantages**
  - **nondeterminism** difficult to handle
  - loss of **precision**: only approximate answers
  - lose ability to definitively establish **causal** relationships and identify **best/worst-case** scenarios
  - **speed**: possibly very high number of samples required to generate suitable accurate approximations
  - may be hard to estimate likelihood of **rare events**
Quantitative verification in action

- **DNA transducer gate** [Lakin et al, 2012]
  - DNA computing with a restricted class of DNA strand displacement structures
  - transducer design due to Cardelli
  - automatically found and fixed design error, using Microsoft’s DSD and PRISM

- **Microgrid demand management protocol** [TACAS12,FMSD13]
  - designed for households to actively manage demand while accessing a variety of energy sources
  - found and fixed a flaw in the protocol, due to lack of punishment for selfish behaviour
  - implemented in PRISM-games
Quantitative verification – Status

- **Tools/techniques widely applicable, since real software/systems are quantitative**
  - extensions/adaptations of model-based frameworks
  - new application domains

- **Analysis “quantitative” & “exhaustive”**
  - strength of mathematical proof
  - best/worst-case scenarios, not possible with simulation
  - identifying trends and anomalies

- **But**
  - the modelling phase time-consuming and error prone
  - potential ‘disconnect’ between model and the artefact
  - scalability continues to be hard to overcome
From verification to synthesis

• Shift towards quantitative model synthesis from specification
  – begin with simpler problems: strategy synthesis, template-based synthesis, etc
  – advantage: correct-by-construction

• Here consider the problem of strategy (controller) synthesis
  – i.e. “can we construct a strategy to guarantee that a given quantitative property is satisfied?”
  – instead of “does the model satisfy a given quantitative property?”
  – also parameter synthesis: “find optimal value for parameter to satisfy quantitative objective”

• Many application domains
  – robotics (controller synthesis from LTL/PCTL)
  – dynamic power management (optimal policy synthesis)
Markov decision processes (MDPs)

- Model **nondeterministic** as well as **probabilistic** behaviour
  - e.g. for concurrency, under-specification, abstraction...
  - extension of discrete-time Markov chains
  - nondeterministic choice between probability distributions

- Formally, an MDP is a tuple
  - \((S, s_{\text{init}}, \text{Act}, \delta, L)\)

- where:
  - \(S\) is a set of states
  - \(s_{\text{init}} \in S\) is the initial state
  - \(\delta : S \times \text{Act} \to \text{Dist}(S)\) is a (partial) transition probability function
  - \(L : S \to 2^{\text{AP}}\) is a labelling function
  - \(\text{Act}\) is a set of actions, \(\text{AP}\) is a set of atomic propositions
  - \(\text{Dist}(S)\) is the set of discrete probability distributions over \(S\)
Paths and strategies

- **A (finite or infinite) path** through an MDP
  - is a sequence \((s_0...s_n)\) of (connected) states
  - represents an execution of the system
  - resolves both the probabilistic and nondeterministic choices

- **A strategy \(\sigma\) (aka. “adversary” or “policy”)** of an MDP
  - is a resolution of nondeterminism only
  - is (formally) a mapping from finite paths to distributions
  - induces a fully probabilistic model
  - i.e. an (infinite–state) Markov chain over finite paths
  - on which we can define a probability space over infinite paths
Classification of strategies

- **Strategies are classified according to**
- randomisation:
  - $\sigma$ is **deterministic** (pure) if $\sigma(s_0...s_n)$ is a point distribution, and randomised otherwise
- memory:
  - $\sigma$ is **memoryless** (simple) if $\sigma(s_0...s_n) = \sigma(s_n)$ for all $s_0...s_n$
  - $\sigma$ is **finite memory** if there are finitely many modes such as $\sigma(s_0...s_n)$ depends only on $s_n$ and the current mode, which is updated each time an action is performed
  - otherwise, $\sigma$ is **infinite memory**

- **A strategy $\sigma$ induces, for each state $s$ in the MDP:**
  - a set of infinite paths $\text{Path}^\sigma(s)$
  - a probability space $\text{Pr}^{\sigma}_s$ over $\text{Path}^\sigma(s)$
Example strategy

- Fragment of induced Markov chain for strategy which picks b then c in $s_1$

Finite-memory, deterministic
Running example

- **Example MDP**
  - robot moving through terrain divided into 3 x 2 grid

States:
- \( s_0, s_1, s_2, s_3, s_4, s_5 \)

Actions:
- north, east, south, west, stuck

Labels
- (atomic propositions): hazard, goal\(_1\), goal\(_2\)
Properties and objectives

• The syntax:

- $\phi ::= P_{\sim p}[\psi] \mid R_{\sim r}[\rho]$
- $\psi ::= \text{true} \mid a \mid \psi \land \psi \mid \neg \psi \mid X\psi \mid \psi U^{\leq k} \psi \mid \psi U \psi$
- $\rho ::= F b \mid C \mid C^{\leq k}$

- $\psi$ is true with probability $\sim p$
- Expected reward is $\sim r$
- Where $b$ is an atomic proposition, used to identify states of interest, $p \in [0,1]$ is a probability, $\sim \in \{<,>,\leq,\geq\}$, $k \in \mathbb{N}$, and $r \in \mathbb{R}_{\geq 0}$
- $F b \equiv \text{true} U b$

• We refer to $\phi$ as property, $\psi$ and $\rho$ as objectives
  - (branching time more challenging for synthesis)
Properties and objectives

• Semantics of the probabilistic operator $P$
  – can only define probabilities for a specific strategy $\sigma$
  – $s \models P_{\sim p}[\psi]$ means “the probability, from state $s$, that $\psi$ is true for an outgoing path satisfies $\sim p$ for all strategies $\sigma$”
  – formally $s \models P_{\sim p}[\psi] \iff Pr_{s}(\psi) \sim p$ for all strategies $\sigma$
  – where we use $Pr_{s}(\psi)$ to denote $Pr_{s}(\{ \omega \in Path_{s} | \omega \models \psi \})$

• $R_{\sim r}[\cdot]$ means “the expected value of $\cdot$ satisfies $\sim r$”

• Some examples:
  – $P_{\geq 0.4}[F\text{ “goal”}]$ “probability of reaching goal is at least 0.4”
  – $R_{< 5}[C\leq 60]$ “expected power consumption over one hour is below 5”
  – $R_{\leq 10}[F\text{ “end”}]$ “expected time to termination is at most 10”
Verification and strategy synthesis

• The verification problem is:
  – Given an MDP $M$ and a property $\phi$, does $M$ satisfy $\phi$ for all possible strategies $\sigma$?

• The synthesis problem is dual:
  – Given an MDP $M$ and a property $\phi$, find, if it exists, a strategy $\sigma$ such that $M$ satisfies $\phi$ under $\sigma$

• Verification and strategy synthesis is achieved using the same techniques, namely computing optimal values for probability objectives, i.e. for $\phi = P_{\sim p}[\psi]$:
  – $Pr_s^{\min}(\psi) = \inf_{\sigma} Pr_s^{\sigma}(\psi)$
  – $Pr_s^{\max}(\psi) = \sup_{\sigma} Pr_s^{\sigma}(\psi)$

• Expectations (reward objectives $R_{\sim r}[\psi]$) are similar, omitted
Verification and strategy synthesis

• The verification problem is:
  − Given an MDP $M$ and a property $\phi$, does $M$ satisfy $\phi$ for all possible strategies $\sigma$?

• The synthesis problem is dual:
  − Given an MDP $M$ and a property $\phi$, find, if it exists, a strategy $\sigma$ such that $M$ satisfies $\phi$ under $\sigma$.

• In particular, we have
  − $M$ satisfies $\phi = P_{\geq q}[\psi]$ iff $Pr_{s}^{\text{min}}(\psi) \geq q$
  − There exists a strategy satisfying $\phi = P_{\geq q}[\psi]$ iff $Pr_{s}^{\text{max}}(\psi) \geq q$
  − then take optimal strategy

\[
\begin{array}{c|c|c}
0 & Pr_{s}^{\text{min}}(\psi) & Pr_{s}^{\text{max}}(\psi) \\
q & & 1
\end{array}
\]
Computing reachability for MDPs

• Computation of probabilities $\Pr_s^{\text{max}}(F b)$ for all $s \in S$

• Step 1: pre–compute all states where probability is 1 or 0
  – graph-based algorithms, yielding sets $S^\text{yes}$, $S^\text{no}$

• Step 2: compute probabilities for remaining states ($S^?$)
  – (i) solve linear programming problem
  – (i) approximate with value iteration
  – (iii) solve with policy (strategy) iteration

• 1. Precomputation (for $\Pr_s^{\text{max}}$):
  – algorithm Prob1E computes $S^\text{yes}$
    • there exists a strategy for which the probability of "F b" is 1
  – algorithm Prob0A computes $S^\text{no}$
    • for all strategies, the probability of satisfying "F b" is 0
Example goal:
\[ P \geq 0.4 \ [ F \ \text{goal}_1 ] \]

So compute:
\[ \Pr S_{s_{\max}}(F \ \text{goal}_1) \]
Example goal:
\[ P_{\geq 0.4} \left[ F \text{ goal}_1 \right] \]

So compute:
\[ \text{Pr}_{\text{s max}}(F \text{ goal}_1) \]
2. Numerical computation

- compute probabilities $\Pr_s^{\max}(F b)$
- for remaining states $S' = S \setminus (S^{\text{yes}} \cup S^{\text{no}})$
- obtained as the unique solution of the linear programming (LP) problem:

$$\text{minimize } \sum_{s \in S'} x_s \text{ subject to the constraints:}$$

$$x_s \geq \sum_{s' \in S'} \delta(s, a)(s') \cdot x_{s'} + \sum_{s' \in S^{\text{yes}}} \delta(s, a)(s')$$

for all $s \in S'$ and for all $a \in A(s)$

- This can be solved with standard techniques
  - e.g. Simplex, ellipsoid method, branch-and-cut
Example – Reachability (LP)

Let $x_i = \Pr_{s_i}^{\text{max}}(F \text{ goal}_1)$

$S^{\text{yes}}$: $x_4 = x_5 = 1$

$S^{\text{no}}$: $x_2 = x_3 = 0$

For $S^? = \{x_0, x_1\}$:

Minimise $x_0 + x_1$ subject to:

- $x_0 \geq 0.4 \cdot x_0 + 0.6 \cdot x_1$ (east)
- $x_0 \geq 0.1 \cdot x_1 + 0.1$ (south)
- $x_1 \geq 0.5$ (south)
- $x_1 \geq 0$ (east)

Example:

$P_{\geq 0.4} [ F \text{ goal}_1 ]$

So compute:

$\Pr_{s}^{\text{max}}(F \text{ goal}_1)$
Let $x_i = \Pr_{s_i}^{\max}(F \text{ goal}_1)$

$S^{\text{yes}}$: $x_4=x_5=1$

$S^{\text{no}}$: $x_2=x_3=0$

For $S^? = \{x_0, x_1\}$:

Minimise $x_0 + x_1$ subject to:

- $x_0 \geq x_1$ (east)
- $x_0 \geq 0.1 \cdot x_1 + 0.1$ (south)
- $x_1 \geq 0.5$ (south)
Example – Reachability (LP)

Let \( x_i = Pr_{s_i}^{\text{max}}(F_{\text{goal}_1}) \)

\( S^{\text{yes}}: x_4 = x_5 = 1 \)

\( S^{\text{no}}: x_2 = x_3 = 0 \)

For \( S^? = \{x_0, x_1\} \):

Minimise \( x_0 + x_1 \) subject to:

- \( x_0 \geq x_1 \)
- \( x_0 \geq 0.1 \cdot x_1 + 0.1 \)
- \( x_1 \geq 0.5 \)

Solution:

(\( x_0, x_1 \) = (0.5, 0.5)

i.e.

\( Pr_{s_0}^{\text{max}}(F_{\text{goal}_1}) = 0.5 \)
Strategy synthesis

- Compute optimal probabilities $\Pr_s^{\max}(F b)$ for all $s \in S$

- To compute the optimal strategy $\sigma^*$, choose the \textit{locally optimal} action in each state
  - in general depends on the method used to compute the optimal probabilities
  - i.e. policy iteration constructs the optimal strategy
  - for max probabilities, adaptation of precomputation needed

- For reachability
  - memoryless strategies suffice

- For step-bounded reachability
  - need finite-memory strategies
  - typically requires \textit{backward} computation for a fixed number of steps
Example – Strategy

Optimal strategy:

- $s_0 : \text{east}$
- $s_1 : \text{south}$
- $s_2 : -$ (stuck)
- $s_3 : -$ (stuck)
- $s_4 : \text{east}$
- $s_5 : -$ (stuck)

Graph:

- $s_0$ to $s_1$ with arrow labeled east and probability 0.6
- $s_1$ to $s_2$ with arrow labeled east and probability 0.5
- $s_2$ to $s_3$ with arrow labeled north and probability 0.9
- $s_3$ to $s_4$ with arrow labeled west and probability 0.1
- $s_4$ to $s_5$ with arrow labeled west and probability 0.4
- $s_0$ to $s_0$ with arrow labeled south and probability 0.8
- $s_1$ to $s_1$ with arrow labeled south and probability 0.1
- $s_2$ to $s_2$ with arrow labeled stuck
- $s_3$ to $s_3$ with arrow labeled stuck
- $s_4$ to $s_4$ with arrow labeled stuck
- $s_5$ to $s_5$ with arrow labeled stuck

Equations:

- $x_0 \geq x_1$ (east)
- $x_1 \geq 0.5$ (south)
Example – Bounded reachability

Example:
\[ P_{\text{max}=?} [ F_{\leq 3} \text{ goal}_2 ] \]

So compute:
\[ \Pr_{s_4}^{\text{max}}(F_{\leq 3} \text{ goal}_2) = 0.99 \]

Optimal strategy
is finite-memory:
s_4 (after 1 step): east
s_4 (after 2 steps): west

Computation more involved
May need to choose a different action on successive visits
Strategy synthesis for LTL objectives

- Reduce to the problem of reachability on the product of MDP $M$ and an omega-automaton representing $\psi$
  - for example, deterministic Rabin automaton (DRA)

- Need only consider computation of maximum probabilities $\Pr_s^{\text{max}}(\psi)$
  - since $\Pr_s^{\text{min}}(\psi) = 1 - \Pr_s^{\text{max}}(\neg \psi)$

- To compute the optimal strategy $\sigma^*$
  - find memoryless deterministic strategy on the product
  - convert to finite-memory strategy with one mode for each state of the DRA for $\psi$
Example – LTL

- $P_{\geq 0.05} \left[ (G \neg \text{hazard}) \land (GF \text{ goal}_1) \right]$
  - avoid hazard and visit goal$_1$ infinitely often

- $Pr_{s_0}^{\text{max}}((G \neg \text{hazard}) \land (GF \text{ goal}_1)) = 0.1$

Optimal strategy:
(in this instance, memoryless)

$s_0 : \text{south}$
$s_1 : -$  
$s_2 : -$  
$s_3 : -$  
$s_4 : \text{east}$  
$s_5 : \text{west}$
Multi-objective strategy synthesis

- Consider conjunctions of probabilistic LTL formulas $P^{\sim p}[\psi]$
  - require all conjuncts to be satisfied
- Reduce to a multi-objective reachability problem on the product of MDP M and the omega-automata representing the conjuncts
  - convert (by negation) to formulas with lower probability bounds ($\geq$, $>$), then to DRA
  - need to consider all combinations of objectives
- The problem can be solved using LP methods [TACAS07] or via approximations to Pareto curve [ATVA12]
  - strategies may be finite memory and randomised
  - continue as for single-objectives to compute the strategy $\sigma^*$
  - find memoryless deterministic strategy on the product
  - convert to finite-memory strategy
**Example – Multi-objective**

- **Multi-objective formula**
  - \( P_{\geq 0.7} [ G \neg \text{hazard} ] \land P_{\geq 0.2} [ GF \text{ goal}_1 ] \) ? True (achievable)

- **Numerical query**
  - \( P_{\text{max}=?} [ GF \text{ goal}_1 ] \) such that \( P_{\geq 0.7} [ G \neg \text{hazard} ] \) ? \(~0.2278\)

- **Pareto query**
  - for \( P_{\text{max}=?} [ G \neg \text{hazard} ] \land P_{\text{max}=?} [ GF \text{ goal}_1 ] \) ?
Example – Multi-objective strategies

Strategy 1
(deterministic)

\[
\begin{align*}
\psi_1 &= G \neg \text{hazard} \\
\psi_2 &= GF \text{goal}_1
\end{align*}
\]
Example – Multi-objective strategies

Strategy 2
(deterministic)

\begin{itemize}
  \item $s_0 :$ south
  \item $s_1 :$ south
  \item $s_2 :$ stuck
  \item $s_3 :$ stuck
  \item $s_4 :$ east
  \item $s_5 :$ west
\end{itemize}

$\psi_1 = G \neg$ hazard

$\psi_2 = GF \text{ goal}_1$
Example – Multi-objective strategies

Optimal strategy: (randomised)
- $s_0 : 0.3226 :$ east  $0.6774 :$ south
- $s_1 : 1.0 :$ south
- $s_2 :$ stuck
- $s_3 :$ stuck
- $s_4 : 1.0 :$ east
- $s_5 : 1.0 :$ west

Graphical representation:

$\psi_1 = G \neg$ hazard
$\psi_2 = GF \text{ goal}_1$
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

Is design-time verification sufficient?

F.D.A. Steps Up Oversight of Infusion Pumps

Over the last five years, [...] 710 patient deaths linked to problems with the devices. Some of those deaths involved patients who suffered drug overdoses accidentally, either because of incorrect dosage entered or because the device’s software malfunctioned.

Manufacturers [...] issued 79 recalls, among the highest for any medical device.

Pump producers now typically conduct ‘simulated’ testing of its devices by users.

Published: April 23, 2010

The New York Times
Quantitative runtime verification

- **Autonomic systems (legacy systems, etc)**
  - require increasingly demanding **non-functional** requirements
    - performance, dependability, utility, ...
  - need to **adapt** to changing scenarios
    - workload, environment, objectives, ...

- **Key observations**: offline qualitative verification methods do **not** suffice

- **Instead integrate** [ICSE09] [FASE09]:
  - **multi-objective quantitative runtime** analysis
    - rigorous exhaustive checks
  - **self-adaptive** system capabilities
    - adaptation decisions based on (non-linear) behaviour
Background: autonomic systems

System objectives (policies)

monitor–analyse–plan–execute
autonomic control loop
System objectives (policies)

PRISM-driven
monitor-analyse-plan-execute autonomic control loop

PRISM-driven
monitor-analyse-plan-execute autonomic control loop

Legacy IT system
Integration

- Monitor
- Select subset of models and configurations
- Carry out PRISM experiment
- Choose best configuration
- Enforce chosen configuration

System objectives (policies)

Autonomic manager

Parameterised family of PRISM models

Legacy IT system

Manageability adaptors

Sensors

Effectors

1. Monitor
2. Select subset of models and configurations
3. Carry out PRISM experiment
4. Choose best configuration
5. Enforce chosen configuration
• **Development method: Qosmos, telemedicine appl [TSE 2011]**
  - assume **exposed** control parameters
  - **model-driven** generation of adaptor code
  - system **objectives** specified by administrator/designer
    - e.g. minimise power consumption and maximise duration
  - at runtime, **monitor** via quantitative verification and **influence** the system by modifying control parameters
  - **predict**, select **optimal** configuration
  - incorporate **learning** (KAMI) to improve predictive power

• **Incremental techniques for better efficiency**
  - incremental model construction and model checking
  - uses constraint solving and MILP
  - implemented in PRISM (MDPs, SMGs)
Quantitative verification – Trends

• Being ‘younger’, generally lags behind conventional verification
  – much smaller model capacity
  – compositional reasoning in infancy
  – automation of model extraction/adaptation very limited
  – runtime monitoring frameworks still limited

• Tool usage on the increase, in academic/industrial contexts
  – real-time verification/synthesis in embedded systems
  – probabilistic verification in security, reliability, performance

• Shift towards greater automation
  – specification mining, model extraction, synthesis, verification, ...

• But many challenges remain!
Future directions

• Many challenges remain
  – computational runtime steering, away from danger states, in addition to online model repair
  – effective model abstraction/reduction techniques
  – scalability of monolithic/runtime verification
  – approximate methods
  – sampling–based methods, combination with automata–based

• More challenges not covered in this lecture
  – correct–by–construction model synthesis from specifications
  – parameter synthesis
  – more expressive models and logics
  – code generation from models
  – new application domains, …

• and more…
Further material

- **Projects**
  - PRISM [www.prismmodelchecker.org](http://www.prismmodelchecker.org)
  - ERC AdG VERIWARE [www.veriware.org](http://www.veriware.org)
  - EPSRC programme grant Mobile Robotics (with Paul Newman)

- **Reading**
  - [DTMCs/MDPs/LTL] Principles of Model Checking by Baier and Katoen, MIT Press 2008
• Recent work on controller synthesis