Automated Verification and Strategy Synthesis for Probabilistic Systems

Marta Kwiatkowska[1] and David Parker[2]

[1] Department of Computer Science, University of Oxford, UK
[2] School of Computer Science, University of Birmingham, UK

Abstract. Probabilistic model checking is an automated technique to verify whether a probabilistic system, e.g., a distributed network protocol which can exhibit failures, satisfies a temporal logic property, for example, "the minimum probability of the network recovering from a fault in a given time period is above 0.98". Dually, we can also synthesise, from a model and a property specification, a strategy for controlling the system in order to satisfy or optimise the property, but this aspect has received less attention to date. In this paper, we give an overview of methods for automated verification and strategy synthesis for probabilistic systems. Primarily, we focus on the model of Markov decision processes and use property specifications based on probabilistic LTL and expected reward objectives. We also describe how to apply multi-objective model checking to investigate trade-offs between several properties, and extensions to stochastic multi-player games. The paper concludes with a summary of future challenges in this area.

1 Introduction

Probabilistic model checking is an automated technique for verifying quantitative properties of stochastic systems. Like conventional model checking, it uses a systematic exploration and analysis of a system model to verify that certain requirements, specified in temporal logic, are satisfied by the model. In probabilistic model checking, models incorporate information about the likelihood and/or timing of the system's evolution, to represent uncertainty arising from, for example, component failures, unreliable sensors, or randomisation. Commonly used models include Markov chains and Markov decision processes.

Properties to be verified against these models are specified in probabilistic temporal logics such as PCTL, CSL and probabilistic LTL. These capture a variety of quantitative correctness, reliability or performance properties, for example, "the maximum probability of the airbag failing to deploy within 0.02 seconds is at most 10^{-6}". Tool support, in the form of probabilistic model checkers such as PRISM [29] and MRMC [27], has been used to verify quantitative properties of a wide variety of real-life systems, from wireless communication protocols [19], to aerospace designs [9], to DNA circuits [32].

One of the key strengths of probabilistic model checking, in contrast to, for example, approximate analysis techniques based on Monte Carlo simulation, is

the ability to analyse quantitative properties in an *exhaustive* manner. A prime example of this is when analysing models that incorporate both probabilistic and nondeterministic behaviour, such as Markov decision processes (MDPs). In an MDP, certain unknown aspects of a system's behaviour, e.g., the scheduling between components executing in parallel, or the instructions issued by a controller at runtime, are modelled as nondeterministic choices. Each possible way of resolving these choices is referred to as a *strategy*. To verify a property ϕ on an MDP $\mathcal{M}$, we check that ϕ holds *for all* possible strategies of $\mathcal{M}$.

Alternatively, we can consider the dual problem of *strategy synthesis*, which finds *some* strategy of $\mathcal{M}$ that satisfies a property ϕ, or which optimises a specified objective. This is more in line with the way that MDPs are used in other fields, such as planning under uncertainty [34], reinforcement learning [42] or optimal control [8]. In the context of probabilistic model checking, the strategy synthesis problem has generally received less attention than the dual problem of verification, despite being solved in essentially the same way. Strategy synthesis, however, has many uses; examples of its application to date include:

(i) *Robotics*. In recent years, temporal logics such as LTL have grown increasingly popular as a means to specify tasks when synthesising controllers for robots or embedded systems [47]. In the presence of uncertainty, e.g. due to unreliable sensors or actuators, optimal controller synthesis can be performed using MDP model checking techniques [31].

(ii) *Security*. In the context of computer security, model checking has been used to synthesise strategies for malicious attackers, which represent flaws in security systems or protocols. Probability is also often a key ingredient of security; for example, in [41], probabilistic model checking of MDPs was used to generate PIN guessing attacks against hardware security modules.

(iii) *Dynamic power management*. The problem of synthesising optimal (randomised) control strategies to switch between power states in electronic devices can be solved using optimisation problems on MDPs [7] or, alternatively, with multi-objective strategy synthesis for MDPs [21].

In application domains such as these, probabilistic model checking offers various benefits. Firstly, as mentioned above, temporal logics provide an expressive means of formally specifying the goals of, for example, a controller or a malicious attacker. Secondly, thanks to formal specifications for models and properties, rigorous mathematical underpinnings, and the use of exact, exhaustive solution methods, strategy synthesis yields controllers that are guaranteed to be correct (at least with respect to the specified model and property). Such guarantees may be essential in the context of safety-critical systems.

Lastly, advantage can be drawn from the significant body of both past and ongoing work to improve the efficiency and scalability of probabilistic verification and strategy synthesis. This includes methods developed specifically for probabilistic model checking, such as symbolic techniques, abstraction or symmetry reduction, and also advances from other areas of computer science. For example, renewed interest in the area of synthesis for reactive systems has led to

significantly improved methods for generating the automata needed to synthesise strategies for temporal logics such as LTL. Parallels can also be drawn with verification techniques for timed systems: for example, UPPAAL [33], a model checker developed for verifying timed automata, has been used to great success for synthesising solutions to real-time task scheduling problems, and is in many cases superior to alternative state-of-the-art methods [1].

In this paper, we give an overview of methods for performing verification and strategy synthesis on probabilistic systems. Our focus is primarily on algorithmic issues: we introduce the basic ideas, illustrate them with examples and then summarise the techniques required to perform them. For space reasons, we restrict our attention to finite-state models with a discrete notion of time. We also only consider complete information scenarios, i.e. where the state of the model is fully observable to the strategy that is controlling it.

Primarily, we describe techniques for Markov decision processes. The first two sections provide some background material, introduce the strategy synthesis problem and summarise methods to solve it. In subsequent sections, we describe extensions to multi-objective verification and stochastic multi-player games. We conclude the paper with a discussion of some of the important topics of ongoing and future research in this area.

An extended version of this paper, which includes full details of the algorithms needed to perform strategy synthesis and additional worked examples, is available as [30]. The examples in both versions of the paper can be run using PRISM (and its extensions [15]). Accompanying PRISM files are available online [49].

2 Markov Decision Processes

In the majority of this paper, we focus on *Markov decision processes* (MDPs), which model systems that exhibit both *probabilistic* and *nondeterministic* behaviour. Probability can be used to model *uncertainty* from a variety of sources, e.g., the unreliable behaviour of an actuator, the failure of a system component or the use of randomisation to break symmetry.

Nondeterminism, on the other hand, models *unknown* behaviour. Again, this has many uses, depending on the context. When using an MDP to model and verify a randomised distributed algorithm or network protocol, nondeterminism might represent *concurrency* between multiple components operating in parallel, or *underspecification*, where some parameter or behaviour of the system is only partially defined. In this paper, where we focus mainly on the problem of strategy synthesis for MDPs, nondeterminism is more likely to represent the possible decisions that can be taken by a *controller* of the system.

Formally, we define an MDP as follows.

Definition 1 (Markov decision process). *A* Markov decision process *(MDP) is a tuple* $\mathcal{M}=(S,\overline{s},A,\delta_{\mathcal{M}},Lab)$ *where* S *is a* finite *set of states,* $\overline{s}\in S$ *is an initial state,* A *is a finite set of* actions, $\delta_{\mathcal{M}} : S\times A \to Dist(S)$ *is a (partial)*

probabilistic transition function, mapping state-action pairs to probability distributions over S, and $Lab : S \to 2^{AP}$ is a labelling function assigning to each state a set of atomic propositions taken from a set AP.

An MDP models how the state of a system can evolve, starting from an initial state $\overline{s}$. In each state s, there is a choice between a set of enabled actions $A(s) \subseteq A$, where $A(s) \stackrel{\text{def}}{=} \{a \in A \mid \delta_{\mathcal{M}}(s,a) \text{ is defined}\}$. The choice of an action $a \in A$ is assumed to be nondeterministic. Once selected, a transition to a successor state s' occurs randomly, according to the probability distribution $\delta_{\mathcal{M}}(s,a)$, i.e., the probability that a transition to s' occurs is $\delta_{\mathcal{M}}(s,a)(s')$.

A *path* is a (finite or infinite) sequence of transitions $\pi = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots$ through MDP $\mathcal{M}$, i.e., where $s_i \in S$, $a_i \in A(s_i)$ and $\delta_{\mathcal{M}}(s_i,a_i)(s_{i+1})>0$ for all $i \in \mathbb{N}$. The $(i+1)$th state s_i of path π is denoted $\pi(i)$ and, if π is finite, $last(\pi)$ denotes its final state. We write $FPath_{\mathcal{M},s}$ and $IPath_{\mathcal{M},s}$, respectively, for the set of all finite and infinite paths of $\mathcal{M}$ starting in state s, and denote by $FPath_{\mathcal{M}}$ and $IPath_{\mathcal{M}}$ the sets of *all* such paths.

Example 1. Fig. 1 shows an MDP $\mathcal{M}$, which we will use as a running example. It represents a robot moving through terrain that is divided up into a 3×2 grid, with each grid section represented as one state. In each of the 6 states, one or more actions from the set $A = \{north, east, south, west, stuck\}$ are available, which move the robot between grid sections. Due to the presence of obstacles, certain actions are unavailable in some states or probabilistically move the robot to an alternative state. Action *stuck*, in states s_2 and s_3, indicates that the robot is unable to move. In Fig. 1, the probabilistic transition function is drawn as grouped, labelled arrows; where the probability is 1, it is omitted. We also show labels for the states, taken from the set $AP = \{hazard, goal_1, goal_2\}$.

Rewards and costs. We use *rewards* as a general way of modelling various additional quantitative measures of an MDP. Although the name "reward" suggests a quantity that it is desirable to maximise (e.g., profit), we will often use the same mechanism for *costs*, which would typically be minimised (e.g. energy consumption). In this paper, rewards are values attached to the actions available in each state, and we assume that these rewards are accumulated over time.

Definition 2 (Reward structure). *A* reward structure *for an MDP $\mathcal{M} = (S, \overline{s}, A, \delta_{\mathcal{M}}, Lab)$ is a function of the form $r : S \times A \to \mathbb{R}_{\geqslant 0}$.*

Strategies. We reason about the behaviour of MDPs using *strategies* (which, depending on the context, are also known as *policies*, *adversaries* or *schedulers*). A strategy resolves nondeterminism in an MDP, i.e., it chooses which action (or actions) to take in each state. In general, this choice can depend on the history of the MDP's execution so far and can be randomised.

Definition 3 (Strategy). *A* strategy *of an MDP $\mathcal{M} = (S, \overline{s}, A, \delta_{\mathcal{M}}, Lab)$ is a function $\sigma : FPath_{\mathcal{M}} \to Dist(A)$ such that $\sigma(\pi)(a)>0$ only if $a \in A(last(\pi))$.*

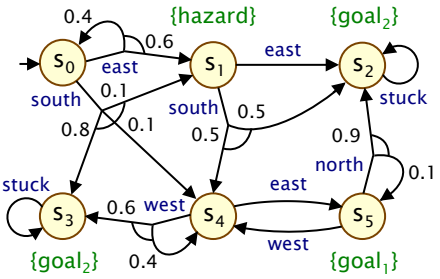

Fig. 1. Running example: an MDP $\mathcal{M}$ representing a robot moving about a 3×2 grid.

We denote by $\Sigma_{\mathcal{M}}$ the set of all strategies of $\mathcal{M}$, but in many cases we can restrict our attention to certain subclasses. In particular, we can classify strategies in terms of their use of *randomisation* and *memory*.

1. **randomisation**: we say that strategy σ is *deterministic* (or *pure*) if $\sigma(\pi)$ is a point distribution for all $\pi \in FPath_{\mathcal{M}}$, and *randomised* otherwise;
2. **memory**: a strategy σ is *memoryless* if $\sigma(\pi)$ depends only on $last(\pi)$ and *finite-memory* if there are finitely many *modes* such that $\sigma(\pi)$ depends only on $last(\pi)$ and the current mode, which is updated each time an action is performed; otherwise, it is *infinite-memory*.

Under a particular strategy σ of $\mathcal{M}$, all nondeterminism is resolved and the behaviour of $\mathcal{M}$ is fully probabilistic. Formally, we can represent this using an (infinite) *induced discrete-time Markov chain*, whose states are finite paths of $\mathcal{M}$. This leads us, using a standard construction [28], to the definition of a probability measure $Pr^{\sigma}_{\mathcal{M},s}$ over infinite paths $IPath_{\mathcal{M},s}$, capturing the behaviour of $\mathcal{M}$ from state s under strategy σ. We will also use, for a random variable $X : IPath_{\mathcal{M},s} \to \mathbb{R}_{\geqslant 0}$, the expected value of X from state s in $\mathcal{M}$ under strategy σ, denoted $\mathbb{E}^{\sigma}_{\mathcal{M},s}(X)$. If s is the initial state $\overline{s}$, we omit it and write $Pr^{\sigma}_{\mathcal{M}}$ or $\mathbb{E}^{\sigma}_{\mathcal{M}}$.

3 Strategy Synthesis for MDPs

We now explain the *strategy synthesis* problem for Markov decision processes and give a brief overview of the algorithms that can be used to solve it. From now on, unless stated otherwise, we assume a fixed MDP $\mathcal{M} = (S, \overline{s}, A, \delta_{\mathcal{M}}, Lab)$.

3.1 Property Specification

First, we need a way to formally specify a property of the MDP that we wish to hold under the strategy to be synthesised. We follow the approach usually adopted in probabilistic verification and specify properties using temporal logic. More precisely, we will use a fragment of the property specification language from the PRISM model checker [29], the full version of which subsumes logics such as PCTL, probabilistic LTL, PCTL* and others.

For any MDP $\mathcal{M}$, state $s \in S$ and strategy σ:		
$\mathcal{M}, s, \sigma \models \mathtt{P}_{\bowtie p}[\,\psi\,]$	$\iff$	$Pr^{\sigma}_{\mathcal{M},s}(\{\pi \in IPath_{\mathcal{M},s} \mid \pi \models \psi\}) \bowtie p$
$\mathcal{M}, s, \sigma \models \mathtt{R}^{r}_{\bowtie x}[\,\rho\,]$	$\iff$	$\mathbb{E}^{\sigma}_{\mathcal{M},s}(rew(r,\rho)) \bowtie x$
For any path $\pi = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \ldots \in IPath_{\mathcal{M}}$:		
$\pi \models \mathtt{true}$		always
$\pi \models b$	$\iff$	$b \in L(s_0)$
$\pi \models \psi_1 \wedge \psi_2$	$\iff$	$\pi \models \psi_1 \wedge \pi \models \psi_2$
$\pi \models \neg\psi$	$\iff$	$\pi \not\models \psi$
$\pi \models \mathtt{X}\,\psi$	$\iff$	$s_1 s_2 \ldots \models \psi$
$\pi \models \psi_1\ \mathtt{U}^{\leqslant k}\ \psi_2$	$\iff$	$\exists i \leqslant k\ .\ \big(\, s_i s_{i+1} \ldots \models \psi_2 \wedge (\,\forall j{<}i\ .\ s_j s_{j+1} \ldots \models \psi_1\,)\,\big)$
$\pi \models \psi_1\ \mathtt{U}\ \psi_2$	$\iff$	$\exists i \geqslant 0\ .\ \big(\, s_i s_{i+1} \ldots \models \psi_2 \wedge (\,\forall j{<}i\ .\ s_j s_{j+1} \ldots \models \psi_1\,)\,\big)$
For any reward structure r and path $\pi = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \ldots \in IPath_{\mathcal{M}}$:		
$rew(r, \mathtt{C}^{\leqslant k})(\pi)$	$\stackrel{\text{def}}{=}$	$\sum_{j=0}^{k-1} r(s_j, a_j)$
$rew(r, \mathtt{C})(\pi)$	$\stackrel{\text{def}}{=}$	$\sum_{j=0}^{\infty} r(s_j, a_j)$
$rew(r, \mathtt{F}\ b)(\pi)$	$\stackrel{\text{def}}{=}$	$\begin{cases} \infty & \text{if } \forall j \in \mathbb{N} : b \notin L(s_j), \\ \sum_{j=0}^{k-1} r(s_j, a_j) & \text{otherwise, where } k = \min\{j \mid b \in L(s_j)\} \end{cases}$

Fig. 2. Inductive definition of the property satisfaction relation $\models$.

Definition 4 (Properties and objectives). *For the purposes of this paper, a* property *is a formula ϕ derived from the following grammar:*

$$
\begin{aligned}
\phi &::= \mathtt{P}_{\bowtie p}[\,\psi\,] \;\big|\; \mathtt{R}^{r}_{\bowtie x}[\,\rho\,] \\
\psi &::= \mathtt{true} \mid b \mid \psi \wedge \psi \mid \neg\psi \mid \mathtt{X}\,\psi \mid \psi\ \mathtt{U}^{\leqslant k}\ \psi \mid \psi\ \mathtt{U}\ \psi \\
\rho &::= \mathtt{C}^{\leqslant k} \;\big|\; \mathtt{C} \;\big|\; \mathtt{F}\ b
\end{aligned}
$$

where $b \in AP$ is an atomic proposition, $\bowtie \in \{\leqslant, <, \geqslant, >\}$, $p \in [0,1]$, r is a reward structure, $x \in \mathbb{R}_{\geqslant 0}$ and $k \in \mathbb{N}$. We refer to ψ and ρ as objectives*.*

A property is thus a single instance of either the $\mathtt{P}_{\bowtie p}[\,\psi\,]$ operator, which asserts that the probability of a path satisfying (LTL) formula ψ meets the bound $\bowtie p$, or the $\mathtt{R}^{r}_{\bowtie x}[\,\rho\,]$ operator, which asserts that the expected value of a reward objective ρ, using reward structure r, satisfies $\bowtie x$. For now, we forbid multiple occurrences of the $\mathtt{P}$ or $\mathtt{R}$ operators in the same property,[1] as would typically be permitted when using branching-time probabilistic logics such as PCTL or PCTL* for verification of MDPs. This is because our primary focus in this tutorial is not verification, but strategy synthesis, for which the treatment of branching-time logics is more challenging [4,10].

For an MDP $\mathcal{M}$, state s and strategy σ of $\mathcal{M}$, and property ϕ, we write $\mathcal{M}, s, \sigma \models \phi$ to denote that, when starting from s, and operating under σ, $\mathcal{M}$ satisfies ϕ. Generally, we are interested in the behaviour of $\mathcal{M}$ from its initial state $\overline{s}$, and we write $\mathcal{M}, \sigma \models \phi$ to denote that $\mathcal{M}, \overline{s}, \sigma \models \phi$. A formal definition of the satisfaction relation $\models$ is given in Fig. 2. Below, we identify several key classes of properties and explain them in more detail.

[1] We will relax this restriction, for multi-objective strategy synthesis, in Sec. 4.

Probabilistic reachability. For probabilistic properties $\mathtt{P}_{\bowtie p}[\,\psi\,]$, a simple but fundamental class of path formulae ψ are "until" formulae of the form $b_1\ \mathtt{U}\ b_2$, where b_1, b_2 are atomic propositions. Intuitively, $b_1\ \mathtt{U}\ b_2$ is true if a b_2-labelled state is eventually reached, whilst passing only through b_1 states. Particularly useful is the derived operator $\mathtt{F}\ b \equiv \mathtt{true}\ \mathtt{U}\ b$, representing *reachability*, i.e., a b state is eventually reached. Another common derived operator is $\mathtt{G}\ b \equiv \neg\mathtt{F}\ \neg b$, which captures *invariance*, i.e., that b always remains true. Also useful are step-bounded variants. For example, *step-bounded reachability*, expressed as $\mathtt{F}^{\leqslant k}\ b \equiv \mathtt{true}\ \mathtt{U}^{\leqslant k}\ b$, means that a b-state is reached within k steps.

Probabilistic LTL. More generally, for the probabilistic properties $\mathtt{P}_{\bowtie p}[\,\psi\,]$ defined in Defn. 4, ψ can be any formula in the temporal logic LTL. This allows a wide variety of useful properties to be expressed. These include, for example: (i) $\mathtt{G}\,\mathtt{F}\ b$ (infinitely often b); (ii) $\mathtt{F}\,\mathtt{G}\ b$ (eventually always b); (iii) $\mathtt{G}\ (b_1 \rightarrow \mathtt{X}\, b_2)$ (b_2 always immediately follows b_1); and (iv) $\mathtt{G}\ (b_1 \rightarrow \mathtt{F}\ b_2)$ (b_2 always eventually follows b_1). Notice that, in order to provide convenient syntax for expressing step-bounded reachability (discussed above), we explicitly add a step-bounded until operator $\mathtt{U}^{\leqslant k}$. This is not normally included in the syntax of LTL, but does not add to its expressivity (e.g., $b_1\ \mathtt{U}^{\leqslant 2}\ b_2 \equiv b_2 \vee (b_1 \wedge \mathtt{X}\, b_2) \vee (b_1 \wedge \mathtt{X}\, b_1 \wedge \mathtt{X}\,\mathtt{X}\, b_2)$).

Reward properties. As explained in Sec. 2, rewards (or dually, costs) are values assigned to state-action pairs that we assume to be accumulated over time. Properties of the form $\mathtt{R}^{r}_{\bowtie x}[\,\rho\,]$ refer to the *expected* accumulated value of a reward structure r. The period of time over which rewards are accumulated is specified by the operator ρ: for the first k steps ($\mathtt{C}^{\leqslant k}$), indefinitely ($\mathtt{C}$), or until a state labelled with b is reached ($\mathtt{F}\ b$). In the final case, if a b-state is *never* reached, we assume that the accumulated reward is infinite.

3.2 Verification and Strategy Synthesis

Classically, probabilistic model checking is phrased in terms of *verifying* that a model $\mathcal{M}$ satisfies a property ϕ. For an MDP, this means checking that ϕ holds for all possible strategies of $\mathcal{M}$.

Definition 5 (Verification). *The* verification *problem is: given an MDP $\mathcal{M}$ and property ϕ, does $\mathcal{M}, \sigma \models \phi$ hold for all possible strategies $\sigma \in \Sigma_{\mathcal{M}}$?*

In practice, this is closely related to the dual problem of *strategy synthesis*.

Definition 6 (Strategy synthesis). *The* strategy synthesis *problem is: given MDP $\mathcal{M}$ and property ϕ, find, if it exists, a strategy $\sigma \in \Sigma_{\mathcal{M}}$ such that $\mathcal{M}, \sigma \models \phi$.*

Verification and strategy synthesis for a property ϕ on MDP $\mathcal{M}$ can be done in essentially the same way, by computing *optimal values* for either probability or expected reward objectives, defined as follows:

$$Pr^{\min}_{\mathcal{M},s}(\psi) = \inf_{\sigma \in \Sigma_{\mathcal{M}}} \{Pr^{\sigma}_{\mathcal{M},s}(\psi)\} \qquad \mathbb{E}^{\min}_{\mathcal{M},s}(rew(r,\rho)) = \inf_{\sigma \in \Sigma_{\mathcal{M}}} \{\mathbb{E}^{\sigma}_{\mathcal{M},s}(rew(r,\rho))\}$$
$$Pr^{\max}_{\mathcal{M},s}(\psi) = \sup_{\sigma \in \Sigma_{\mathcal{M}}} \{Pr^{\sigma}_{\mathcal{M},s}(\psi)\} \qquad \mathbb{E}^{\max}_{\mathcal{M},s}(rew(r,\rho)) = \sup_{\sigma \in \Sigma_{\mathcal{M}}} \{\mathbb{E}^{\sigma}_{\mathcal{M},s}(rew(r,\rho))\}$$

When s is the initial state $\overline{s}$, we omit the subscript s.

Verifying, for example, property $\phi = \mathtt{P}_{\geqslant p}[\,\psi\,]$ against $\mathcal{M}$ or, dually, synthesising a strategy for $\phi' = \mathtt{P}_{\leqslant p}[\,\psi\,]$ can both be done by computing $Pr^{\min}_{\mathcal{M}}(\psi)$. For the former, $\mathcal{M}$ satisfies ϕ if and only if $Pr^{\min}_{\mathcal{M}}(\psi) \geqslant p$. For the latter, there exists a strategy σ satisfying ϕ' if and only if $Pr^{\min}_{\mathcal{M}}(\psi) \leqslant p$, in which case we can take σ to be a corresponding *optimal strategy*, i.e., one that achieves the optimal value. In general, therefore, rather than fix a specific bound p, we often simply aim to compute an optimal value and accompanying optimal strategy. In this case, we adapt the syntax of properties to include *numerical queries*.

Definition 7 (Numerical query). *Let ψ, r and ρ be as specified in Defn. 4. A* numerical query *takes the form* $\mathtt{P}_{\min=?}[\,\psi\,]$, $\mathtt{P}_{\max=?}[\,\psi\,]$, $\mathtt{R}^{r}_{\min=?}[\,\rho\,]$ *or* $\mathtt{R}^{r}_{\max=?}[\,\rho\,]$ *and yields the optimal value for the probability/reward objective.*

In the rest of this section, we describe how to compute optimal values and strategies for the classes of properties described above. We also explain which class of strategies suffices for optimality in each case (i.e., the smallest class of strategies which is guaranteed to contain an optimal one). This is important both in terms of the tractability of the solution methods, and the size and complexity of the controller that we might wish to construct from the synthesised strategy. As mentioned earlier, an extended version of this paper, available from [49], presents full details of these methods. Coverage of this material can also be found in, for example, [21,5,2] and standard texts on MDPs [6,26,38].

3.3 Strategy Synthesis for Probabilistic Reachability

To synthesise optimal strategies for probabilistic reachability, it suffices to consider *memoryless deterministic* strategies. For this class of properties, and for those covered in the following subsections, the bulk of the work for strategy synthesis actually amounts to computing optimal values. An optimal strategy is extracted either after or during this computation.

Calculating optimal values proceeds in two phases: the first *precomputation* phase performs an analysis of the underlying graph structure of the MDP to identify states for which the probability is 0 or 1; the second performs numerical computation to determine values for the remaining states. The latter can be done using various methods: (i) by solution of a *linear programming* problem; (ii) *policy iteration*, which builds a sequence of strategies (i.e., policies) with increasingly high probabilities until an optimal one is reached; (iii) *value iteration*, which computes increasingly precise approximations to the exact probabilities.

The method used to construct an optimal strategy σ^* depends on how the probabilities were computed. Policy iteration is the simplest case, since a strategy is constructed as part of the algorithm. For the others, minimum probabilities are straightforward – we choose the locally optimal action in each state:

$$\sigma^*(s) = \arg\min_{a \in A(s)} \textstyle\sum_{s' \in S} \delta(s,a)(s') \cdot Pr^{\min}_{\mathcal{M},s'}(\mathtt{F}\ b)$$

Maximum probabilities require more care, but simple adapations to precomputation and value iteration algorithms yield an optimal strategy.

For step-bounded reachability, memoryless strategies do not suffice: we need to consider the class of *finite-memory deterministic* strategies. Computation of optimal probabilities (and an optimal strategy) for step-bounded reachability amounts to working backwards through the MDP and determining, at each step, and for each state, which action yields optimal probabilities. In fact, this amounts simply to performing a fixed number of steps of value iteration.

Example 2. We return to the MDP $\mathcal{M}$ from Fig. 1 and synthesise a strategy satisfying the property $\mathtt{P}_{\geqslant 0.4}[\,\mathtt{F}\ \mathit{goal}_1\,]$. To do so, we compute $Pr^{\max}_{\mathcal{M}}(\mathtt{F}\ \mathit{goal}_1)$, which equals 0.5. This is achieved by the memoryless deterministic strategy that picks *east* in s_0, *south* in s_1 and *east* in s_4 (there is no choice to make in states s_2, s_3 and any action can be taken in s_5, since goal_1 has already been reached).

Next, we consider label goal_2 and a numerical query $\mathtt{P}_{\max=?}[\,\mathtt{F}^{\leqslant k}\ \mathit{goal}_2\,]$ with a step-bounded reachability objective. We find that $Pr^{\max}_{\mathcal{M}}(\mathtt{F}^{\leqslant k}\ \mathit{goal}_2)$ is 0.8, 0.96 and 0.99 for $k = 1, 2$ and 3, respectively. Taking $k = 3$ as an example, the optimal strategy is deterministic, but finite-memory. For example, if we arrive at state s_4 after 1 step, action *east* is optimal, since it reaches goal_2 with probability 0.9. If, on the other hand, we arrive in s_4 after 2 steps, it is better to take *west*, since it would be impossible to reach goal_2 within $k - 2 = 1$ steps.

3.4 Strategy Synthesis for Probabilistic LTL

To synthesise an optimal strategy of MDP $\mathcal{M}$ for an LTL formula ψ, we reduce the problem to the simpler case of a reachability property on the product of $\mathcal{M}$ and an ω-automaton representing ψ. Here, we describe the approach of [2], which uses *deterministic Rabin automata* (DRAs) and computes the probability of reaching *accepting end components*. Since the minimum probability of an LTL formula can be expressed as the maximum probability of a negated formula:

$$Pr^{\min}_{\mathcal{M}}(\psi) \;=\; 1 - Pr^{\max}_{\mathcal{M}}(\neg\psi)$$

we only need to consider the computation of maximally optimal probabilities.

A DRA $\mathcal{A}$ with alphabet α represents a set of infinite words $\mathcal{L}(\mathcal{A}) \subseteq \alpha^\omega$. For any LTL formula ψ using atomic propositions from AP, we can construct [45,18,5] a DRA $\mathcal{A}_\psi$ with alphabet 2^{AP} that represents it, i.e., such that an infinite path $\pi = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \ldots$ of $\mathcal{M}$ satisfies ψ if and only if $\mathit{Lab}(s_0)\mathit{Lab}(s_1)\mathit{Lab}(s_2)\ldots$ is in $\mathcal{L}(\mathcal{A}_\psi)$. We then proceed by building the (synchronous) product $\mathcal{M}\otimes\mathcal{A}_\psi$ of $\mathcal{M}$ and $\mathcal{A}_\psi$. The product is an MDP with state space $S \times Q$, where Q is the set of states of the DRA. We then have:

$$Pr^{\max}_{\mathcal{M}}(\psi) = Pr^{\max}_{\mathcal{M}\otimes\mathcal{A}_\psi}(\mathtt{F}\ \mathit{acc})$$

where *acc* is an atomic proposition labelling *accepting end components* of $\mathcal{M}\otimes\mathcal{A}_\psi$. An end component [2] is a strongly connected sub-MDP of $\mathcal{M}$, and whether it is accepting is dictated by the acceptance condition of $\mathcal{A}_\psi$. Computing $Pr^{\max}_{\mathcal{M}}(\psi)$ thus reduces to identifying the set of all end components (see, e.g., [2,5]) and calculating the maximum probability of reaching the accepting ones.

To build an optimal strategy maximising the probability of an LTL formula, we need to consider *finite-memory deterministic* strategies. An optimal strategy of this class is constructed in two steps. First, we find a *memoryless* deterministic strategy for the product $\mathcal{M}\otimes\mathcal{A}_\psi$, which maximises the probability of reaching accepting end components (and then stays in those end components, visiting each state infinitely often). Then, we convert this to a finite-memory strategy, with one mode for each state $q \in Q$ of the DRA $\mathcal{A}_\psi$.

Example 3. Again, using the running example (Fig. 1), we synthesise a strategy for the LTL property $\mathtt{P}_{\geqslant 0.05}[\,(\mathtt{G}\ \neg \mathit{hazard}) \wedge (\mathtt{G}\ \mathtt{F}\ \mathit{goal}_1)\,]$, which aims to both avoid the *hazard*-labelled state and visit the goal_1 state infinitely often. The maximum probability, from the initial state, is 0.1. In fact, for this example, a memoryless strategy suffices for optimality: we choose *south* in state s_0, which leads to state s_4 with probability 0.1. We then remain in states s_4 and s_5 indefinitely by choosing actions *east* and *west*, respectively.

3.5 Strategy Synthesis for Reward Properties

The techniques required to perform strategy synthesis for expected reward properties $\mathtt{R}^r_{\bowtie x}[\,\rho\,]$ are, in fact, quite similar to those required for the probabilistic reachability properties, described in Sec. 3.3. For the case where $\rho = \mathtt{F}\ b$, techniques similar to those for $\mathtt{P}_{\bowtie p}[\,\mathtt{F}\ b\,]$ are used: first, a graph based analysis of the model (in this case, to identify states of the MDP from which the expected reward is infinite), and then methods such as value iteration or linear programming. The resulting optimal strategy is again memoryless and deterministic.

For the case $\rho = \mathtt{C}$, where rewards are accumulated indefinitely, we need to identify end components containing non-zero rewards, since these can result in the expected reward being infinite. Subsequently, computation is similar to the case of $\rho = \mathtt{F}\ b$ above. For step-bounded properties $\rho = \mathtt{C}^{\leqslant k}$, the situation is similar to probabilities for step-bounded reachability; optimal strategies are deterministic, but may need finite memory, and optimal expected reward values can be computed recursively in k steps.

Example 4. We synthesise an optimal strategy for minimising the number of *moves* that the robot makes (i.e., the number of actions taken in the MDP) before reaching a goal_2 state. We use a reward structure *moves* that maps all state-action pairs to 1, and a numerical query $\mathtt{R}^{\mathit{moves}}_{\min=?}[\,\mathtt{F}\ \mathit{goal}_2\,]$. This yields the optimal value $\frac{19}{15}$, achieved by the memoryless deterministic strategy that chooses *south*, *east*, *west* and *north* in states s_0, s_1, s_4 and s_5 respectively.

4 Multi-objective Strategy Synthesis

In this section, we describe *multi-objective* strategy synthesis for MDPs, which generates a strategy σ that simultaneously satisfies multiple properties of the kind discussed in the previous section. We first describe the case for LTL properties and then summarise some extensions.

Definition 8 (Multi-objective LTL). *A* multi-objective LTL property *is a conjunction* $\phi = \mathtt{P}_{\bowtie_1 p_1}[\,\psi_1\,] \wedge \ldots \wedge \mathtt{P}_{\bowtie_n p_n}[\,\psi_n\,]$ *of probabilistic LTL properties. For MDP* $\mathcal{M}$ *and strategy* σ, $\mathcal{M}, \sigma \models \phi$ *if* $\mathcal{M}, \sigma \models \mathtt{P}_{\bowtie_1 p_1}[\,\psi_1\,]$ *for all* $1 \leqslant i \leqslant n$.

An algorithm for multi-objective LTL strategy synthesis was given in [20], although here we describe an adapted version, based on [22], using deterministic Rabin automata. The overall approach is similar to standard (single-objective) LTL strategy synthesis in that it constructs a product automaton and reduces the problem to (multi-objective) reachability.

First, we ensure that all n properties $\mathtt{P}_{\bowtie_i p_i}[\,\psi_i\,]$ contain only lower probability bounds $\bowtie \in \{\geqslant, >\}$, by negating LTL formulae as required (e.g., replacing $\mathtt{P}_{<p}[\,\psi\,]$ with $\mathtt{P}_{>1-p}[\,\neg\psi\,]$). Next, we build a DRA $\mathcal{A}_{\psi_i}$ for each LTL formula ψ_i, and construct the product MDP $\mathcal{M}' = \mathcal{M} \otimes \mathcal{A}_{\psi_1} \otimes \cdots \otimes \mathcal{A}_{\psi_n}$. We then consider each combination $X \subseteq \{1, \ldots, n\}$ of objectives and find the end components of $\mathcal{M}'$ that are accepting for all DRAs $\{\mathcal{A}_i \mid i \in X\}$. We create a special sink state for X in $\mathcal{M}'$ and add transitions from states in the end components to the sink.

The problem then reduces to a multi-objective problem on $\mathcal{M}'$ for n reachability properties $\mathtt{P}_{\bowtie_1 p_1}[\,\mathtt{F}\ acc_1\,], \ldots, \mathtt{P}_{\bowtie_n p_n}[\,\mathtt{F}\ acc_n\,]$, where acc_i represents the union of, for each set X containing i, the sink states for X. This can be done by solving a linear programming (LP) problem [20].

Optimal strategies for multi-objective LTL may be *finite-memory* and *randomised*. A strategy can be constructed directly from the solution of the LP problem. Like for LTL objectives (in Sec. 3.4), we obtain a memoryless strategy for the product MDP and then convert it to a finite-memory one on $\mathcal{M}$.

We now summarise several useful extensions and improvements.

(i) *Boolean combinations* of LTL objectives (rather than conjunctions, as in Defn. 8) can be handled via a translation to disjunctive normal form [20,22].

(ii) *expected reward objectives* can also be supported, in addition to LTL properties. The LP-based approach sketched above has beeen extended [22] to include reward objectives of the form $\mathtt{R}^r_{\bowtie x}[\,\mathtt{C}\,]$. An alternative approach, based on value iteration, rather than LP [23], allows the addition of step-bounded reward objectives $\mathtt{R}^r_{\bowtie x}[\,\mathtt{C}^{\leqslant k}\,]$ (and also provides significant gains in efficiency for both classes of properties).

(iii) *numerical multi-objective queries* generalise the numerical queries explained in Defn. 7. For example, rather than synthesising a strategy satisfying property $\mathtt{P}_{\bowtie_1 p_1}[\,\psi_1\,] \wedge \mathtt{P}_{\bowtie_2 p_2}[\,\psi_2\,]$, we can instead synthesise a strategy that maximises the probability of ψ_1, whilst simultaneously satisfying $\mathtt{P}_{\bowtie_2 p_2}[\,\psi_2\,]$. The LP-based methods mentioned above are easily extended to handle numerical queries by adding an objective function to the LP problem.

(iv) *Pareto queries* [23] produce a *Pareto curve* (or an approximation of it) illustrating the trade-off between multiple objectives. For example, if we want to maximise the probabilities of two LTL formulae ψ_1 and ψ_2, the Pareto curve comprises points (p_1, p_2) such that there is a strategy σ with $Pr^\sigma_{\mathcal{M}}(\psi_1) \geqslant p_1$

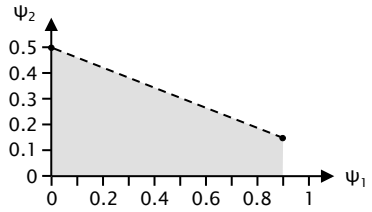

Fig. 3. Pareto curve (dashed line) for maximisation of the probabilities of LTL formulae $\psi_1 = \texttt{G}\ \neg \mathit{hazard}$ and $\psi_2 = \texttt{G F}\ \mathit{goal}_1$ (see Ex. 5).

and $Pr^{\sigma}_{\mathcal{M}}(\psi_2) \geqslant p_2$, but, if either bound p_1 or p_2 is increased, no strategy exists without decreasing the other bound.

We refer the reader to the references given above for precise details of the algorithms and any restrictions or assumptions that may apply.

Example 5. Previously, in Ex. 3, we synthesised a strategy for the LTL property $\texttt{P}_{\geqslant 0.05}[\,(\texttt{G}\ \neg \mathit{hazard}) \wedge (\texttt{G F}\ \mathit{goal}_1)\,]$ and found that the maximum achievable probability was 0.1. Let us now consider each conjunct of the LTL formula as a separate objective and synthesise a strategy satisfying the multi-objective LTL property $\texttt{P}_{\geqslant 0.7}[\,\texttt{G}\ \neg \mathit{hazard}\,] \wedge \texttt{P}_{\geqslant 0.2}[\,\texttt{G F}\ \mathit{goal}_1\,]$. For convenience, we will abbreviate the objectives to $\psi_1 = \texttt{G}\ \neg \mathit{hazard}$ and $\psi_2 = \texttt{G F}\ \mathit{goal}_1$.

Following the procedure outlined at the start of this section, we find that there *is* a strategy satisfying $\texttt{P}_{\geqslant 0.7}[\,\psi_1\,] \wedge \texttt{P}_{\geqslant 0.2}[\,\psi_2\,]$. To give an example of one such strategy, we consider a numerical multi-objective query that maximises the probability of satisfying ψ_2 whilst satisfying $\texttt{P}_{\geqslant 0.7}[\,\psi_1\,]$. The optimal value (maximum probability for ψ_2) is $\frac{41}{180} \approx 0.2278$, which is obtained by a randomised strategy that, in state s_0, picks *east* with probability approximately 0.3226 and *south* with probability approximately 0.6774.

Finally, we also show, in Fig. 3, the Pareto curve obtained when maximising the probabilities of both ψ_1 and ψ_2. The grey shared area shows all points (x, y) for which there is a strategy satisfying $\texttt{P}_{\geqslant x}[\,\psi_1\,] \wedge \texttt{P}_{\geqslant y}[\,\psi_2\,]$. Points along the top edge of this region, shown as a dashed line in the figure, form the Pareto curve. We also mark, as black circles, points $(Pr^{\sigma}_{\mathcal{M}}(\psi_1), Pr^{\sigma}_{\mathcal{M}}(\psi_2))$ for specific *deterministic* strategies of $\mathcal{M}$. The leftmost circle is the strategy described in the first part of Ex. 2, and the rightmost one is the strategy from Ex. 3.

5 Controller Synthesis with Stochastic Games

So far, we have assumed that the nondeterministic choices in the model represent the choices available to a single entity, such as a controller. In many situations, it is important to consider decisions being made by multiple entities, possibly with conflicting objectives. An example is the classic formulation of the *controller synthesis* problem, in which a *controller* makes decisions about how to control, for example, a manufacturing plant, and must respond to nondeterministic behaviour occurring in the *environment* of the plant.

It is natural to model and analyse such systems using game-theoretic methods, which are designed precisely to reason about the strategic decisions of competing agents. From a modelling point of view, we generalise MDPs to *stochastic games*, in which nondeterministic choices are resolved by multiple *players* and the resulting behaviour is probabilistic (MDPs can thus be seen as 1-player stochastic games). We restrict our attention here to *turn-based* (as opposed to *concurrent*) stochastic games, in which a single player is responsible for the nondeterministic choices available in each state. In line with the rest of the paper, we assume finite-state models and total information.

Definition 9 (SMG). *A (turn-based)* stochastic multi-player game *(SMG) is a tuple* $\mathcal{G} = (\Pi, S, (S_i)_{i\in\Pi}, \overline{s}, A, \delta_{\mathcal{G}}, Lab)$*, where* $S, \overline{s}, A, \delta_G$ *and* Lab *are as for an MDP, in Defn. 1,* Π *is a finite set of* players *and* $(S_i)_{i\in\Pi}$ *is a partition of* S*.*

An SMG $\mathcal{G}$ evolves in a similar way to an MDP, except that the nondeterministic choice in each state s is resolved by the player that *controls* that state (the player i for which $s \in S_i$). Like MDPs, we reason about SMGs using strategies, but these are defined separately for each player: a strategy σ_i for player i is a function mapping finite paths ending in a state from S_i to a distribution over actions A. Given strategies $\sigma_1, \ldots, \sigma_k$ for multiple players from Π, we can combine them into a single strategy $\sigma = \sigma_1, \ldots, \sigma_k$. If a strategy σ comprises strategies for all players of the game (sometimes called a *strategy profile*), we can construct, like for an MDP, a probability space $Pr^{\sigma}_{\mathcal{G}}$ over the infinite paths of $\mathcal{G}$.

For strategy synthesis on SMGs, we generate strategies either for an individual player, or for a *coalition* $C \subseteq \Pi$ of players. We extend the definition of properties given in Defn. 4 in the style of the logic rPATL [14].

Definition 10 (Multi-player strategy synthesis). *For a property* $\mathtt{P}_{\bowtie p}[\,\psi\,]$ *or* $\mathtt{R}^{r}_{\bowtie x}[\,\rho\,]$ *and a coalition* $C \subseteq \Pi$ *of players, (zero-sum)* multi-player strategy synthesis *is expressed by a query* $\langle\!\langle C\rangle\!\rangle \mathtt{P}_{\bowtie p}[\,\psi\,]$ *or* $\langle\!\langle C\rangle\!\rangle \mathtt{R}^{r}_{\bowtie x}[\,\rho\,]$*. For example,* $\langle\!\langle C\rangle\!\rangle \mathtt{P}_{\bowtie p}[\,\psi\,]$ *asks "does there exist a strategy* σ_1 *for the players in* C *such that, for all strategies* σ_2 *for the players* $\Pi\backslash C$*, we have* $Pr^{\sigma_1,\sigma_2}_{\mathcal{G}}(\psi) \bowtie p$*?".*

Intuitively, if $\langle\!\langle C\rangle\!\rangle \mathtt{P}_{\bowtie p}[\,\psi\,]$ is true, then the players in C can collectively *guarantee* that $\mathtt{P}_{\bowtie p}[\,\psi\,]$ holds, regardless of what the other players do. Like for the other classes of strategy synthesis described in this paper, we can also use numerical queries for stochastic multi-player games. We write, for example, $\langle\!\langle C\rangle\!\rangle \mathtt{P}_{\max=?}[\,\psi\,]$ to denote the maximum probability of ψ that players in C can guarantee, regardless of the actions of the other players.

Multi-player strategy synthesis can be solved using rPATL model checking. Details for the the full logic rPATL (which allows nested operators and includes additional reward operators, but omits $\mathtt{C}^{\leqslant k}$ and $\mathtt{C}$) are given in in [14]. Basically, model checking reduces to the analysis of *zero-sum* properties on a stochastic 2-player game in which player 1 corresponds to C and player 2 to $\Pi\backslash C$. For reachability properties (i.e., $\langle\!\langle C\rangle\!\rangle \mathtt{P}_{\bowtie p}[\,\mathtt{F}\ b\,]$), *memoryless deterministic* strategies suffice and optimal values and strategies can be computed either with value iteration or strategy iteration [16,17]. For an LTL property ψ, we again reduce

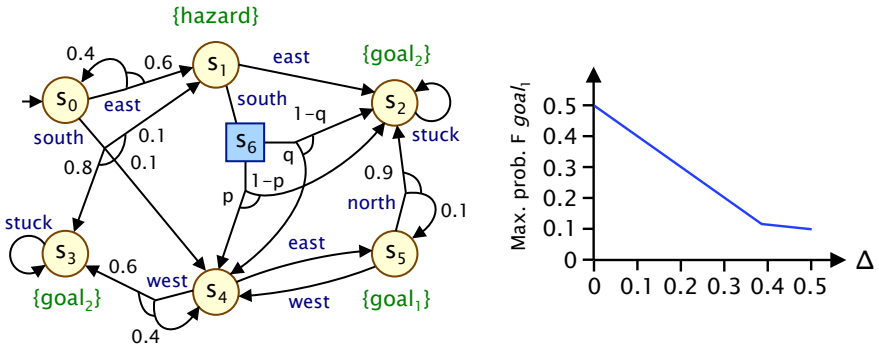

Fig. 4. Left: A stochastic 2-player game modelling an unknown probability interval. Right: The maximum value with which *ctrl* can guarantee reaching $goal_1$ for probability interval $[p, q] = [0.5-\Delta, 0.5+\Delta]$. See Ex. 6 for details.

the problem to a simpler one on the product of the game $\mathcal{G}$ and a deterministic automaton representing ψ. Unlike MDPs, there is no notion of end components. Instead, we can either use the strategy improvement algorithm of [13] to directly compute probabilities for Rabin objectives, or convert the DRA to a deterministic parity automaton and compute probabilities for parity objectives [12].

Example 6. To give an example of strategy synthesis using stochastic games, we consider a simple extension of the MDP $\mathcal{M}$ from the running example (Fig. 1). We assume that the existing choices in the MDP are made by a player *ctrl*, and we add a second player *env*, which represents nondeterministic aspects of the environment. Recall that transition probabilities in $\mathcal{M}$ model uncertain outcomes of robot actions due to obstacles. For example, when action *south* is taken in state s_1 of $\mathcal{M}$, the MDP only moves in the intended direction (to s_4) with probability 0.5; otherwise, it moves to s_2. Let us now instead assume that this probability (of going to s_4) can vary in some interval $[p, q]$.

Fig. 4 (left) shows how we can model this as a stochastic two-player game.[2] States controlled by player *ctrl* are, as before, shown as circles; those controlled by player *env*, are shown as squares. When action *south* is taken in state s_1, we move to a new state s_6, in which player *env* can choose between two actions, one that leads to s_4 with probability p and one that does so with probability q. Since players are allowed to select actions at random, this means player *env* can effectively cause the transition to occur with any probability in the range $[p, q]$.

In Ex. 2, we computed the maximum probability of reaching $goal_1$ as 0.5. Now, we will consider the maximum probability that *ctrl* can guarantee, regardless of the choices made by *env*, i.e., the maximum probability of reaching $goal_1$, if the probability of moving to state s_4 after action *south* can be *any* value in $[p, q]$. This is done with the query $\langle\langle ctrl \rangle\rangle \mathtt{P}_{\max=?}[\,\mathtt{F}\ goal_1\,]$. We compute this value for various intervals $[p, q]$ centred around the original value of 0.5, i.e., we

[2] This notion can be captured more cleanly by annotating transitions directly with probability intervals [40], or with more general specifications of uncertainty [39]. Here, we just aim to give a simple illustration of using a stochastic 2-player game.

let $p = 0.5-\Delta, q = 0.5+\Delta$ and vary Δ. Fig. 4 (right) plots the result. From inspection of the game, we can deduce that the plot corresponds to the function $\min(0.5-\Delta, 0.15-0.1\Delta)$. This means that, if $\Delta \geqslant \frac{7}{18}$ (i.e., if $p \leqslant \frac{1}{9}$), then it is better to switch to the strategy that picks *south* in state s_0, rather than *east*.

6 Challenges and Directions

In this paper, we have given a brief overview of strategy synthesis for probabilistic systems, pointing to some promising application areas, highlighting the benefits that can be derived from existing work on probabilistic verification, and summarising the algorithmic techniques required for a variety of useful strategy synthesis methods. We invite the reader to consult the extended version of this paper [30] for further details.

As noted at the start, the current presentation makes a number of simplifying assumptions. We conclude by reviewing some of the key challenges in the area of strategy synthesis for probabilistic systems.

- *Partial observability.* In this paper, we assumed a *complete information* setting, where the state of the model (and the states of its history) are fully visible when a strategy chooses an action to take. In many situations, this is unrealistic, which could lead to strategies being synthesised that are not feasible in practice. Although fundamental decision problems are undecidable in the context of partial observability [3], practical implementations have been developed for a few cases [11,24] and some tool support exists [36]. Developing efficient methods for useful problem classes is an important challenge.
- *Robustness and uncertainty.* In many potential applications of strategy synthesis, such as the generation of controllers in embedded systems, it may be difficult to formulate a precise model of the stochastic behaviour of the system's environment. Thus, developing appropriate models of uncertainty, and corresponding methods to synthesise strategies that are robust in these environments, is important. We gave a very simple illustration of uncertain probabilistic behaviour in Sec. 5. Developing more sophisticated approaches is an active area of research [46,37].
- *Continuous time and space.* In this paper, we focused on discrete-time probabilistic models. Verification techniques have also been developed for models that incorporate both nondeterminism and continuous notions of time, including probabilistic timed automata [35], interactive Markov chains [25] and Markov automata [43]. Similarly, progress is being made on verification techniques for models with continuous state spaces, and hybrid models that mix both discrete and continuous elements [48,44]. Developing efficient strategy synthesis techniques for such models will bring the benefits of the methods discussed in this paper to a much wider range of application domains.

Acknowledgments. The authors are supported in part by ERC Advanced Grant VERIWARE and EU FP7 project HIERATIC.

References

1. Abdeddaim, Y., Kerbaa, A., Maler, O.: Task graph scheduling using timed automata. In: Proc. IPDPS'03 (2003)
2. de Alfaro, L.: Formal Verification of Probabilistic Systems. Ph.D. thesis, Stanford University (1997)
3. Baier, C., Bertrand, N., Größer, M.: On decision problems for probabilistic büchi automata. In: Proc. FOSSACS'08. LNCS, vol. 4962, pp. 287–301. Springer (2008)
4. Baier, C., Größer, M., Leucker, M., Bollig, B., Ciesinski, F.: Controller synthesis for probabilistic systems. In: Proc. TCS'06. pp. 493–5062. Kluwer (2004)
5. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press (2008)
6. Bellman, R.: Dynamic Programming. Princeton University Press (1957)
7. Benini, L., Bogliolo, A., Paleologo, G., De Micheli, G.: Policy optimization for dynamic power management. IEEE Trans. CADICS 8(3), 299–316 (2000)
8. Bertsekas, D.: Dynamic Programming and Optimal Control, Volumes 1 and 2. Athena Scientific (1995)
9. Bozzano et al., M.: The COMPASS approach: Correctness, modelling and performability of aerospace systems. In: Proc. SAFECOMP'09. pp. 173–186. Springer (2009)
10. Brázdil, T., Brožek, V., Forejt, V., Kučera, A.: Stochastic games with branching-time winning objectives. In: Proc. LICS'06. pp. 349–358. IEEE CS Press (2006)
11. Cerný, P., Chatterjee, K., Henzinger, T., Radhakrishna, A., Singh, R.: Quantitative synthesis for concurrent programs. In: Proc. CAV'11. pp. 243–259 (2011)
12. Chatterjee, K., Henzinger, T.: Strategy improvement and randomized subexponential algorithms for stochastic parity games. In: Proc. STACS'06 (2006)
13. Chatterjee, K., Henzinger, T.: Strategy improvement for stochastic Rabin and Streett games. In: Proc. CONCUR'06. pp. 375–389 (2006)
14. Chen, T., Forejt, V., Kwiatkowska, M., Parker, D., Simaitis, A.: Automatic verification of competitive stochastic systems. Formal Methods in System Design 43(1), 61–92 (2013)
15. Chen, T., Forejt, V., Kwiatkowska, M., Parker, D., Simaitis, A.: PRISM-games: A model checker for stochastic multi-player games. In: Proc. TACAS'13 (2013)
16. Condon, A.: The complexity of stochastic games. Information and Computation 96(2), 203–224 (1992)
17. Condon, A.: On algorithms for simple stochastic games. DIMACS Series in Discrete Mathematics and Theoretical Computer Science 13, 51–73 (1993)
18. Daniele, M., Giunchiglia, F., Vardi, M.: Improved automata generation for linear temporal logic. In: Proc. CAV'99. LNCS, vol. 1633, pp. 249–260. Springer (1999)
19. Duflot, M., Kwiatkowska, M., Norman, G., Parker, D.: A formal analysis of Bluetooth device discovery. STTT 8(6), 621–632 (2006)
20. Etessami, K., Kwiatkowska, M., Vardi, M., Yannakakis, M.: Multi-objective model checking of Markov decision processes. LMCS 4(4), 1–21 (2008)
21. Forejt, V., Kwiatkowska, M., Norman, G., Parker, D.: Automated verification techniques for probabilistic systems. In: SFM'11. LNCS, vol. 6659, pp. 53–113 (2011)
22. Forejt, V., Kwiatkowska, M., Norman, G., Parker, D., Qu, H.: Quantitative multi-objective verification for probabilistic systems. In: Proc. TACAS'11 (2011)
23. Forejt, V., Kwiatkowska, M., Parker, D.: Pareto curves for probabilistic model checking. In: Proc. ATVA'12. LNCS, vol. 7561, pp. 317–332. Springer (2012)
24. Giro, S., Rabe, M.: Verification of partial-information probabilistic systems using counterexample-guided refinements. In: Proc. ATVA'12. LNCS, Springer (2012)

25. Hermanns, H.: Interactive Markov Chains and the Quest for Quantified Quality, LNCS, vol. 2428. Springer Verlag (2002)
26. Howard, R.: Dynamic Programming and Markov Processes. The MIT Press (1960)
27. Katoen, J.P., Hahn, E., Hermanns, H., Jansen, D., Zapreev, I.: The ins and outs of the probabilistic model checker MRMC. In: Proc. QEST'09. IEEE CS Press (2009)
28. Kemeny, J., Snell, J., Knapp, A.: Denumerable Markov Chains. Springer-Verlag, 2nd edn. (1976)
29. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: Proc. CAV'11. LNCS, vol. 6806, pp. 585–591. Springer (2011)
30. Kwiatkowska, M., Parker, D.: Automated verification and strategy synthesis for probabilistic systems (extended version) (2013), available from [49]
31. Lahijanian, M., Wasniewski, J., Andersson, S., Belta, C.: Motion planning and control from temporal logic specifications with probabilistic satisfaction guarantees. In: Proc. ICRA'10. pp. 3227–3232 (2010)
32. Lakin, M., Parker, D., Cardelli, L., Kwiatkowska, M., Phillips, A.: Design and analysis of DNA strand displacement devices using probabilistic model checking. Journal of the Royal Society Interface 9(72), 1470–1485 (2012)
33. Larsen, K., Pettersson, P., Yi, W.: UPPAAL in a nutshell. International Journal on Software Tools for Technology Transfer 1(1-2), 134152 (1997)
34. Masuam, Kolobov, A.: Planning with Markov Decision Processes: An AI Perspective. Morgan & Claypool (2012)
35. Norman, G., Parker, D., Sproston, J.: Model checking for probabilistic timed automata. Formal Methods in System Design (2012), to appear
36. Poupart, P.: Exploiting Structure to Efficiently Solve Large Scale Partially Observable Markov Decision Processes. Ph.D. thesis, University of Toronto (2005)
37. Puggelli, A., Li, W., Sangiovanni-Vincentelli, A., Seshia, S.: Polynomial-time verification of PCTL properties of MDPs with convex uncertainties. In: CAV'13 (2013)
38. Puterman, M.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley and Sons (1994)
39. Satia, J., Lave Jr., R.: Markovian decision processes with uncertain transition probabilities. Oper. Res. 21, 728740 (1970)
40. Sen, K., Viswanathan, M., Agha, G.: Model-checking Markov chains in the presence of uncertainties. In: Proc. TACAS'06. pp. 394–410 (2006)
41. Steel, G.: Formal analysis of PIN block attacks. Theoretical Computer Science 367(1-2), 257–270 (2006)
42. Sutton, R., Barto, A.: Reinforcement Learning: An Introduction. MIT Press (1998)
43. Timmer, M., Katoen, J.P., van de Pol, J., Stoelinga, M.: Efficient modelling and generation of markov automata. In: Proc. CONCUR'12. pp. 364–379 (2012)
44. Tkachev, I., Abate, A.: Formula-free finite abstractions for linear temporal verification of stochastic hybrid systems. In: proc. HSCC'13. pp. 283–292 (2013)
45. Vardi, M., Wolper, P.: Reasoning about infinite computations. Information and Computation 115(1), 1–37 (1994)
46. Wolff, E., Topcu, U., Murray, R.: Robust control of uncertain Markov decision processes with temporal logic specifications. In: Proc. CDC'12. pp. 3372–3379 (2012)
47. Wongpiromsarn, T., Topcu, U., Murray, R.: Receding horizon temporal logic planning. IEEE Trans. Automat. Contr. 57(11), 2817–2830 (2012)
48. Zhang, L., She, Z., Ratschan, S., Hermanns, H., Hahn, E.M.: Safety verification for probabilistic hybrid systems. Eur. J. Control 18(6), 572–587 (2012)
49. `http://www.prismmodelchecker.org/files/stratsynth/`