Measuring and Synthesizing Systems in Probabilistic Environments*

Krishnendu Chatterjee[1], Thomas A. Henzinger[1,2],
Barbara Jobstmann[3], and Rohit Singh[4]

[1] IST Austria
[2] EPFL, Switzerland
[3] CNRS/Verimag, France
[4] IIT Bombay, India

Abstract. Often one has a preference order among the different systems that satisfy a given specification. Under a probabilistic assumption about the possible inputs, such a preference order is naturally expressed by a weighted automaton, which assigns to each word a value, such that a system is preferred if it generates a higher expected value. We solve the following optimal-synthesis problem: given an omega-regular specification, a Markov chain that describes the distribution of inputs, and a weighted automaton that measures how well a system satisfies the given specification under the given input assumption, synthesize a system that optimizes the measured value.

For safety specifications and measures that are defined by mean-payoff automata, the optimal-synthesis problem amounts to finding a strategy in a Markov decision process (MDP) that is optimal for a long-run average reward objective, which can be done in polynomial time. For general omega-regular specifications, the solution rests on a new, polynomial-time algorithm for computing optimal strategies in MDPs with mean-payoff parity objectives. We present some experimental results showing optimal systems that were automatically generated in this way.

1 Introduction

Quantitative reasoning is traditionally used to measure quantitative properties of systems, such as performance or reliability (cf. [1, 4, 30, 33]). More recently, quantitative reasoning has been shown useful also in the classically Boolean contexts of verification (where we ask if a given system satisfies a given specification) and synthesis (where a system is automatically derived from a specification) [6, 31]. In particular, by augmenting a Boolean specifications with a quantitative specifications, we can measure how "well" a system satisfies the specification. For example, among systems that respond to requests, we may prefer one system over another if it responds quicker, or it responds to more requests, or it issues fewer unrequested responses, etc. In synthesis, we can use such measures to guide the synthesis process towards deriving a system that is, in the desired sense, "optimal" among all systems that satisfy the specification [6].

* This research was supported by the European Union project COMBEST and the European Network of Excellence ArtistDesign.

T. Touili, B. Cook, and P. Jackson (Eds.): CAV 2010, LNCS 6174, pp. 380–395, 2010.

There are many ways to define a quantitative measure that captures the "goodness" of a system with respect to the Boolean specification, and particular measures can be quite different, but there are two questions every such measure has to answer: (1) how to assign a quantitative value to one particular behavior of a system (measure along a behavior) and (2) how to aggregate the quantitative values that are assigned to the possible behaviors of the system (measure across behaviors). Recall the response property. Suppose there is a sequence of requests along a behavior and we are interested primarily in response time, i.e., the quicker the system responds, the better. As measure (1) along a particular behavior, we may be interested in an average or the supremum (i.e., worst case) of all response times, or in any other function that aggregates all response times along a behavior into a single real value. The choice of measure (2) across behaviors is independent: we may be interested in an average of all values assigned to individual behaviors, or in the supremum, or again, in some other function. In this way, we can measure the average (across behaviors) of average (along a behavior) response times, or the average of worst-case response times, or the worst case of average response times, or the worst case of worst-case response times, etc. Note that these are the same two choices that appear in weighted automata and max-plus algebras (cf. [17, 25, 28]).

In previous work, we studied various measures (1) along a behavior. In particular, lexicographically ordered tuples of averages [6] and ratios [7] are of natural interest in certain contexts. Alur et al. [2] consider an automaton model with a quantitative measure (1) that is defined with respect to certain accumulation points along a behavior. However, in all of these cases, for measure (2) only the worst case (i.e., supremum) is considered. This comes natural as an extension of Boolean thinking, where a system fails to satisfy a property if even a single behavior violates the property. But in this way, we cannot distinguish between two systems that have the same worst cases across behaviors, but in one system almost all possible behaviors exhibit the worst case, while in the other only very few behaviors do so. In contrast, in performance evaluation one usually considers the average case across different behaviors.

For instance, consider a resource controller for two clients. Clients send requests, and the controller grants the resource to one of them at a time. Suppose we prefer, again, systems where requests are granted "as quickly as possible." Every controller that avoids simultaneous grants will have a behavior along which at least one grant is delayed by one step, namely, the behavior along which both clients continuously send requests. The best the controller can do is to alternate between the clients. Now, if systems are measured with respect to the worst-case behaviors, then a controller that always alternates between both clients, independent of the actual requests, is as good as a controller that tries to grant all requests immediately and only alternates when both clients request the resource at the same time. Clearly, if we wish to synthesize the preferred controller, we need to apply an average-case measure across behaviors.

In this paper, we present a measure (2) that averages across all possible behaviors of a system and solve the corresponding synthesis problem to derive an optimal system. In synthesis, the different possible behaviors of a system are caused by different input sequences. Therefore, in order to take a meaningful average across different behaviors, we need to assume a probability distribution over the possible input sequences. For example, if on input 0 a system has response time r_0, and on input 1 response time r_1, and input 0 is twice as likely as input 1, then the average response time is $(2r_0 + r_1)/3$.

The resulting synthesis problem is as follows: given a Boolean specification φ, a probabilistic input assumption μ, and a measure that assigns to each system M a value $\mathcal{V}_\mu^\varphi(M)$ of how "well" M satisfies φ under μ, construct a system M such that $\mathcal{V}_\mu^\varphi(M) \geq \mathcal{V}_\mu^\varphi(M')$ for all M'. We solve this problem for qualitative specifications that are given as ω-automata, input assumptions that are given as finite Markov chains, and a quantitative specification given as mean-payoff automata which defines a quantitative language by assigning values to behaviors. From the above three inputs we derive a measure that captures (1) an average along system behaviors as well as (2) an average across system behaviors; and thus we obtain a measure that induces a value for each system.

To our knowledge this is the first solution of a synthesis problem for an average-case measure across system behaviors. Technically the solution rests on a new, polynomial-time algorithm for computing optimal strategies in MDPs with mean-payoff parity objectives. In contrast to MDPs with mean-payoff objectives, where pure memoryless optimal strategies exist, optimal strategies for mean-payoff parity objectives in MDPs require infinite memory. It follows from our result that the infinite memory can be captured with a counter, and with this insight we develop the polynomial time algorithm for solving MDPs with mean-payoff parity objectives

Related works. Many formalisms for quantitative specifications have been considered in the literature [2, 8–11, 19, 20, 23, 24, 32]; most of these works (other than [2, 11, 19]) do not consider mean-payoff specifications and none of these works focus on how quantitative specifications can be used to obtain better implementations for the synthesis problem. Furthermore, several notions of metrics for probabilistic systems and games have been proposed in the literature [21, 22]; these metrics provide a measure that indicates how close are two systems with respect to all temporal properties expressible in a logic; whereas our work uses quantitative specification to compare systems wrt the property of interest. Similar in spirit but based on a completely different technique, is the work by Niebert et al. [34], who group behaviors into good and bad with respect to satisfying a given LTL specification and use a CTL*-like analysis specification to quantify over the good and bad behaviors. This measure of logical properties was used by Katz and Peled [31] to guide genetic algorithms to discover counterexamples and corrections for distributed protocols. Control and synthesis in the presence of uncertainty has been considered in several works such as [3, 5, 16]: in all these works, the framework consists of MDPs to model nondeterministic and probabilistic behavior, and the specification is a Boolean specification. In contrast to these works where the probabilistic choice represent uncertainty, in our work the probabilistic choice represent a model for the environment assumption on the input sequences that allows us to consider the system as a whole. Moreover, we consider quantitative objectives. MDPs with mean-payoff objectives are well studied. The books [26, 36] present a detailed analysis of this topic. We present a polynomial-time solution to a more general condition: the Boolean combination of mean-payoff and parity condition on MDPs.

2 Preliminaries

Alphabet, words, and languages. An *alphabet* Σ consists of a finite set of *letters* $\sigma \in \Sigma$. A *word* w over Σ is either a *finite* or *infinite* sequence of letters, i.e., $w \in \Sigma^* \cup \Sigma^\omega$. Given a word $w \in \Sigma^\omega$, we denote by w_i the letter at position i of w and by w^i the prefix of w of length i, i.e., $w^i = w_1 w_2 \dots w_i$. We denote by $|w|$ the length of the word w, i.e., $|w^i| = i$ and $|w| = \infty$, if w is infinite. A *qualitative language* L is a subset of Σ^ω. A *quantitative language* L [11] is a mapping from the set of words to the set of reals, i.e., $L : \Sigma^\omega \to \mathbb{R}$. Note that the characteristic function of a qualitative language L is a quantitative language mapping words to 0 and 1. Given a qualitative language L, we use L also to denote its characteristic function.

Automata with parity, safety, and mean-payoff objective. An *(finite-state) automaton* is a tuple $A = (\Sigma, Q, q_0, \Delta)$, where Σ is a *alphabet*, Q is a (finite) set of *states*, $q_0 \in Q$ is an *initial state*, and $\Delta : Q \times \Sigma \rightharpoonup Q$[1] is a *transition function* that maps a state and a letter to a successor state. The *run of* A *on a word* $w = w_0 w_1 \dots$ is a sequence of states $\rho = \rho_0 \rho_1 \dots$ such that (i) $\rho_0 = q_0$ and (ii) for all $0 \leq i \leq |w|$, $\Delta(\rho_i, w_i) = \rho_{i+1}$). A *parity automaton* is a tuple $A = ((\Sigma, Q, q_0, \Delta), p)$, where (Σ, Q, q_0, Δ) is a finite-state automaton and $p : Q \to \{0, 1, \dots, d\}$ is a *priority function* that maps every state to a natural number in $[0, d]$ called *priority*. A parity automaton A *accepts a word* w if the least priority of all states occurring infinitely often in the run ρ of A on w is even, i.e., $\min_{q \in \mathrm{Inf}(\rho)} p(q)$ is even, where $\mathrm{Inf}(\rho) = \{q \mid \forall i \exists j > i\, \rho_j = q\}$. The *language* of A denoted by L_A is the set of all words accepted by A. A *safety automaton* is a parity automaton with only priorities 0 and 1, and no transitions from priority-1 to priority-0 states. A *mean-payoff automaton* is a tuple $A = ((\Sigma, Q, q_0, \Delta), r)$, where (Σ, Q, q_0, Δ) is a finite-state automaton and $r : Q \times \Sigma \to \mathbb{N}$ is a *reward function* that associates to each transition of the automaton a *reward* $v \in \mathbb{N}$. A mean-payoff automaton assigns to each word w the long-run average of the rewards, i.e., for a word w let ρ be the run of A on w, then we have $L_A(w) = \frac{1}{n} \cdot \sum_{i=1}^{n} r(\rho_i, w_i)$, if w is finite, and $L_A(w) = \liminf_{n \to \infty} L_A(w^n)$ otherwise. Note that L_A is a function assigning values to words.

State machines and specifications. A *(finite-)state machine* (or *system*) with *input signals* I and *output signals* O is a tuple $M = (Q, q_0, \Delta, \lambda)$, where $(\Sigma_I, Q, q_0, \Delta)$ with $\Sigma_I = 2^I$ is a (finite-state) automaton and $\lambda : Q \times \Sigma_I \to \Sigma_0$ with $\Sigma_I = 2^I$ and $\Sigma_O = 2^O$ is a *labeling function* that maps every transition in Δ to an element in Σ_O. The sets Σ_I and Σ_O are called the *input and the output alphabet of* M, respectively. We denote the joint alphabet $2^{I \cup O}$ by Σ. Given an input word $w \in \Sigma_I^* \cup \Sigma_I^\omega$, let ρ by the run of M on w, the *outcome* of M on w, denoted by $\mathcal{O}_M(w)$, is the word $v \in \Sigma^* \cup \Sigma^\omega$ s.t. for all $0 \leq i \leq |w|$, $v_i = w_i \cup \lambda(\rho_i, w_i)$. So, $\mathcal{O}_M$ is the function mapping input words to outcomes. The *language* of M, denoted L_M, is the set of all outcomes of M.

We analyze state machines with respect to qualitative and quantitative specifications. *Qualitative specifications* are qualitative languages, i.e., subsets of Σ^ω or equivalently functions mapping words to 0 and 1. We consider ω-regular specifications given as safety or parity automata. Given a safety or parity automaton A and a state machine M, we say M *satisfies* L_A (written $M \models L_A$) if $L_M \subseteq L_A$ or equivalently $\forall w \in \Sigma_I^\omega$:

[1] Note that our automata are deterministic and complete to simplify the presentation.

$L_A(\mathcal{O}_M(w)) = 1$. *A quantitative specification* is given by a quantitative language L, i.e., a function that assigns values to words. Given a state machine M, we use function composition to relate L and M, i.e., $L \circ \mathcal{O}_M$ is mapping every input word w of M to the value assigned by L to the outcome of M on w. We consider quantitative specifications given by Mean-payoff automata.

Markov chains and Markov Decision Processes (MDP). A *probability distribution* over a finite set S is a function $d : S \to [0, 1]$ such that $\sum_{q \in Q} d(q) = 1$. We denote the set of all probabilistic distributions over S by $\mathcal{D}(S)$. A *Markov Decision Process (MDP)* $G = (S, s_0, E, S_1, S_P, \delta)$ consists of a finite set of *states* S, an *initial state* $s_0 \in S$, a set of *edges* $E \subseteq S \times S$, a partition (S_1, S_P) of the set S, and a probabilistic transition function δ: $S_P \to \mathcal{D}(S)$. The states in S_1 are the *player*-1 states, where player 1 decides the successor state; and the states in S_P are the *probabilistic* states, where the successor state is chosen according to the probabilistic transition function δ. So, we can view an MDP as a game between two players: player 1 and a *random player* that plays according to δ. We assume that for $s \in S_P$ and $t \in S$, we have $(s, t) \in E$ iff $\delta(s)(t) > 0$, and we often write $\delta(s, t)$ for $\delta(s)(t)$. For technical convenience we assume that every state has at least one outgoing edge. For a state $s \in S$, we write $E(s)$ to denote the set $\{t \in S \mid (s, t) \in E\}$ of possible successors. If the set $S_1 = \emptyset$, then G is called a *Markov Chain* and we omit the partition (S_1, S_P) from the definition. A *Σ-labeled MDP* (G, λ) is an MDP G with a labeling function $\lambda : S \to \Sigma$ assigning to each state of G a letter from Σ. We assume that labeled MDPs are deterministic and complete, i.e., (i) $\forall (s, s'), (s, s'') \in E$, $\lambda(s') = \lambda(s'') \to s' = s''$ holds, and (ii) $\forall s \in S, \sigma \in \Sigma$, $\exists s' \in S$ s.t. $(s, s') \in E$ and $\lambda(s') = \sigma$.

Plays and strategies. An infinite path, or a *play*, of the MDP G is an infinite sequence $\omega = s_0 s_1 s_2 \ldots$ of states such that $(s_k, s_{k+1}) \in E$ for all $k \in \mathbb{N}$. Note that we use ω for infinite sequences of states (plays) and v for finite sequences of states. We write Ω for the set of all plays, and for a state $s \in S$, we write $\Omega_s \subseteq \Omega$ for the set of plays starting at s. A *strategy* for player 1 is a function π: $S^* S_1 \to \mathcal{D}(S)$ that assigns a probability distribution to all finite sequences $v \in S^* S_1$ of states ending in a player-1 state. Player 1 follows π, if she make all her moves according to the distributions provided by π. A strategy must prescribe only available moves, i.e., for all $v \in S^*$, $s \in S_1$, and $t \in S$, if $\pi(vs)(t) > 0$, then $(s, t) \in E$. We denote by Π the set of all strategies for player 1. Once a starting state $s \in S$ and a strategy $\pi \in \Pi$ is fixed, the outcome of the game is a random walk ω_s^π for which the probabilities of every *event* $\mathcal{A} \subseteq \Omega$, which is a measurable set of plays, are uniquely defined. For a state $s \in S$ and an event $\mathcal{A} \subseteq \Omega$, we write $\mu_s^\pi(\mathcal{A})$ for the probability that a play belongs to $\mathcal{A}$ if the game starts from the state s and player 1 follow the strategy π, respectively. For a measurable function $f : \Omega \to \mathbb{R}$ we denote by $\mathbb{E}_s^\pi[f]$ the *expectation* of the function f under the probability measure $\mu_s^\pi(\cdot)$. Strategies that do not use randomization are called pure. A player-1 strategy π is *pure* if for all $v \in S^*$ and $s \in S_1$, there is a state $t \in S$ such that $\pi(vs)(t) = 1$. A *memoryless* player-1 strategy depends only on the current state, i.e., for all $v, v' \in S^*$ and for all $s \in S_1$ we have $\pi(vs) = \pi(v's)$. A memoryless strategy can be represented as a function π: $S_1 \to \mathcal{D}(S)$. A *pure memoryless strategy* is a strategy that is both pure and memoryless and can be represented as π: $S_1 \to S$.

Quantitative objectives. A quantitative objective is given by a measurable function $f : \Omega \to \mathbb{R}$. We consider several objectives based on priority and reward functions. Given a priority function $p : S \to \{0, 1, \ldots, d\}$, we defined the set of plays satisfying the parity objective as $\Omega_p = \{\omega \in \Omega \mid \min\big(p(\mathrm{Inf}(\omega))\big) \text{ is even}\}$. A *Parity objective* parity_p is the characteristic function of Ω_p. Given a reward function $r : S \to \mathbb{N} \cup \{\bot\}$, the *mean-payoff objective* mean_r for a play $\omega = s_1 s_2 s_3 \ldots$ is defined as $\mathsf{mean}_r(\omega) = \liminf_{n\to\infty} \frac{1}{n} \cdot \sum_{i=1}^{n} r(s_i)$, if for all $i > 0 : r(s_i) \neq \bot$, otherwise $\mathsf{mean}_r(\omega) = \bot$. Given a priority function p and a reward function r the *mean-payoff parity objective* $\mathsf{mp}_{p,r}$ assigns the long-run average of the rewards if the parity objective is satisfied; otherwise it assigns $\bot$. Formally, for a play ω we have

$$\mathsf{mp}_{p,r}(\omega) = \begin{cases} \mathsf{mean}_r(\omega) & \text{if } \mathsf{parity}_p(\omega) = 1, \\ \bot & \text{otherwise.} \end{cases}$$

For a reward function $r : S \to \mathbb{R}$ the *max objective* max_r assigns to a play the maximum reward that appears in the play. Note that since S is finite, the number of different rewards appearing in a play is finite and hence the maximum is defined. Formally, for a play $\omega = s_1 s_2 s_3 \ldots$ we have $\mathsf{max}_r(\omega) = \max\langle r(s_i)\rangle_{i\geq 0}$.

Values and optimal strategies. Given an MDP G, the *value* function V_G for an objective f is the function from the state space S to the set $\mathbb{R}$ of reals. For all states $s \in S$, let $\mathsf{V}_G(f)(s) = \sup_{\pi \in \Pi} \mathbb{E}_s^{\pi}[f]$. In other words, the value $\mathsf{V}_G(f)(s)$ is the maximal expectation with which player 1 can achieve her objective f from state s. A strategy π is *optimal* from state s for objective f if $\mathsf{V}_G(f)(s) = \mathbb{E}_s^{\pi}[f]$. For parity objectives, mean-payoff objectives, and max objectives pure memoryless optimal strategies exist in MDPs.

Almost-sure winning states. Given an MDP G and a priority function p, we denote by $W_G(\mathsf{parity}_p) = \{s \in S \mid \mathsf{V}_G(\mathsf{parity}_p)(s) = 1\}$, the set of states with value 1. These states are called the *almost-sure* winning states and an optimal strategy from the almost-sure winning states is called a almost-sure winning strategy. The set $W_G(\mathsf{parity}_p)$ for an MDP G with priority function p can be computed in $O(d \cdot n^{\frac{3}{2}})$ time, where n is the size of the MDP G and d is the number of priorities [14, 15]. For states in $S \setminus W_G(\mathsf{parity}_p)$ the parity objective is falsified with positive probability for all strategies, which implies that for all states in $S \setminus W_G(\mathsf{parity}_p)$ the value is less than 1 (i.e., $\mathsf{V}_G(\mathsf{parity}_p)(s) < 1$).

3 Measuring Systems

In this section, we start with an example to explain the problem and introduce our measure. Then, we define the measure formally and show finally, how to compute the value of a system with respect to the given measure.

Example 1. Recall the example from the introduction, where we consider a resource controller for two clients. Client i requests the resource by setting its request signal r_i. The resource is granted to Client i by raising the grant signal g_i. We require that the controller guarantees mutually exclusive access and that it is fair, i.e., a requesting client eventually gets access to the resource. Assume we prefer controllers that respond quickly. Fig. 1 shows a specification that rewards a quick response to request r_i. The

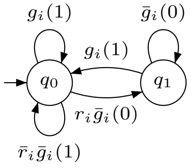

Fig. 1. Automaton A_i

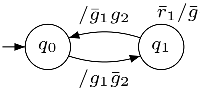

Fig. 2. System M_1

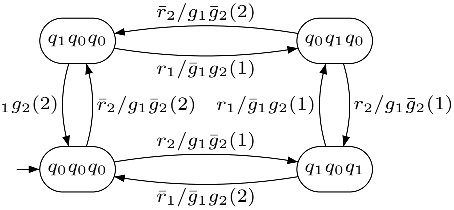

Fig. 3. Product of System M_1 with Specification A_1 and A_2

specification is given as a Mean-payoff automaton that measures the average delay between a request r_i and a corresponding grant g_i. Transitions are labeled with a conjunction of literals[2] and a reward in parentheses. In particular, whenever a request is granted the reward is 1, while a delay of the grant results in reward 0. The automaton assigns to each word in $(2^{\{r_i,g_i\}})^\omega$ the average reward. E.g., the value of the word $(r_i\bar{g}_i\ r_ig_i)^\omega$ is $(0+1)/2 = 1/2$. We can take two copies of this specification, one for each client, and assign to each word in $(2^{\{r_1,r_2,g_1,g_2\}})^\omega$ the sum of the average rewards. E.g., the word $(r_1\bar{r}_2\bar{g}_1g_2\ r_1\bar{r}_2g_1\bar{g}_2)^\omega$ gets an average reward of $1/2$ with respect to the first client and reward 1 with respect to the second client, which sums up to a total reward of $3/2$.

Consider the systems M_1 and M_2 in Fig. 2 and 4, respectively. Transitions are labeled with conjunctions of input and output literals separated by a slash. System M_1 alternates between granting the resource to Client 1 and 2. System M_2 grants the resource to Client 2, if only Client 2 is sending requests. By default it grants the resource to Client 1. If both clients request, then the controller alternates between them. Both systems are correct with respect to the functional requirements describe above: they are fair to both clients and guarantee that the resource is not accessed simultaneously.

Though, one can argue that System M_2 is better than M_1 because the delay between requests and grants is, for most input sequences, smaller than the delay in System M_1. For instance, consider the input trace $(\bar{r}_1r_2\ r_1\bar{r}_2)^\omega$. The response of System M_1 is $(g_1\bar{g}_2\ \bar{g}_1g_2)^\omega$. Looking at the product between the system M_1 and the specifications A_1 and A_2 shown in Fig. 3, we can see that this results in an average reward of 1. Similar, Fig. 5 shows the product of M_2, A_1, and A_2. System M_2 responds with $(\bar{g}_1g_2\ g_1\bar{g}_2)^\omega$ and obtains a reward of 2. Now, consider the sequence $(r_1r_2)^\omega$, which is the worst input sequence the environment can provide. In both systems, this sequences leads to a reward of 1, which is the lowest possible reward. So M_1 and M_2 cannot be distinguished with respect to their worst case behavior.

To measure a system with respect to its average behavior, we aim to average over the rewards obtained for all possible input sequences. Since we have infinite sequences, one way to average is the limit of the average over all finite prefixes. Note that this can only be done if we know the values of finite words with respect to the quantitative specification. For instance, for a finite-state machine M and a Mean-payoff automaton A, we can define the average as $\mathcal{V}_{\oslash}^{L_A}(M) := \lim_{n\to\infty} \frac{1}{|\Sigma_I|^n} \sum_{w\in\Sigma_I^n} L_A(\mathcal{O}_M(w^n))$. However, if we truly want to capture the average behavior, we need to know, how often the different

[2] Note that transitions depend only on the signals that appear in their labels.

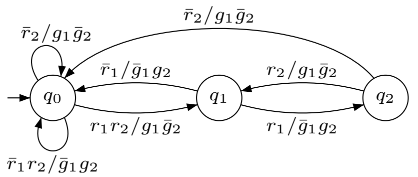

Fig. 4. System M_2 that prefers r_1

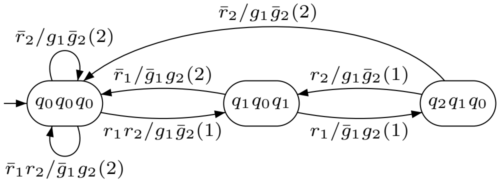
$\bar{r}_2/g_1\bar{g}_2(2)$
$\bar{r}_2/g_1\bar{g}_2(2)$
$\bar{r}_1/\bar{g}_1 g_2(2)$
$r_2/g_1\bar{g}_2(1)$
$q_0 q_0 q_0$
$q_1 q_0 q_1$
$q_2 q_1 q_0$
$r_1 r_2/g_1\bar{g}_2(1)$
$r_1/\bar{g}_1 g_2(1)$
$\bar{r}_1 r_2/\bar{g}_1 g_2(2)$

Fig. 5. Product of M_1, A_1, and A_2

parts of the system are used. This corresponds to knowing how likely the different input sequences are. The measure above assumes that all input sequences are "equally likely". In order to define measures that take the behavior of the environment into account, we use a probability measure on input words. In particular, we consider the probability space $(\Sigma_I^\omega, \mathcal{F}, \mu)$ over Σ_I^ω, where $\mathcal{F}$ is the σ-algebra generated by the cylinder sets of Σ^ω (which are the sets of infinite words sharing a common prefix) (in other words, we have the Cantor topology on Σ_I^ω) and μ is a probability measure defined on $(\Sigma^\omega, \mathcal{F})$. We use finite labeled Markov chains to define the probability measure μ.

Example 2. Recall the controller of Example 1. Assume we know that Client 1 is more likely to send requests than Client 2. We can represent such a behavior by assigning probabilities to the events in $\Sigma = 2^{\{r_1, r_2\}}$. Assume Client 1 sends requests with probability p_1 and Client 2 sends them with probability $p_2 < p_1$, independent of what has happened before. Then, we can build a labeled Markov chain with four states $S_p = \{s_0, s_1, s_2, s_3\}$ each labeled with a letter in Σ, i.e., $\lambda(s_0) = \bar{r}_1\bar{r}_2$, $\lambda(s_1) = \bar{r}_1 r_2$, $\lambda(s_2) = r_1\bar{r}_2$, and $\lambda(s_3) = r_1 r_2$., and the following transition probabilities: (i) $\delta(s_i)(s_0) = (1 - p_1) \cdot (1 - p_2)$, (ii) $\delta(s_i)(s_1) = (1 - p_1) \cdot p_2$, (iii) $\delta(s_i)(s_2) = p_1 \cdot (1 - p_2)$, and (iv) $\delta(s_i)(s_3) = p_1 \cdot p_2$, for all $i \in \{0, 1, 2, 3\}$.

Once we have a probability measure μ on the input sequences and the associated expectation measure $\mathbb{E}$, we can define a satisfaction relation between systems and specifications and a measure for a system with respect to a qualitative and a quantitative specification.

Definition 1 (Satisfaction). *Given a state machine M with input alphabet Σ_I, a qualitative specification φ, and a probability measure μ on $(\Sigma_I^\omega, \mathcal{F})$, we say that M* satisfies φ under μ *(written $M \models_\mu \varphi$) iff M satisfies φ with probability 1, i.e., $\mathbb{E}[\varphi \circ \mathcal{O}_M] = 1$, where $\mathbb{E}$ is the expectation measure for μ.*

Recall that we use a quantitative specification to describe how "good" a system is. Since we aim for a system that satisfies the given (qualitative) specification and is "good" in a given sense, we define the value of a machine with respect to a qualitative and a quantitative specification.

Definition 2 (Value). *Given a state machine M, a qualitative specification φ, quantitative specification ψ, and a probability measure μ on $(\Sigma_I^\omega, \mathcal{F})$, the* value of M with respect to φ and ψ under μ *is defined as the expectation of the function $\psi \circ \mathcal{O}_M$ under the probability measure μ if M* satisfies φ under μ*, and $\perp$ otherwise. Formally, let $\mathbb{E}$ be the expectation measure for μ, then*

$$\mathcal{V}_\mu^{\varphi\psi}(M) := \begin{cases} \mathbb{E}[\psi \circ \mathcal{O}_M] & \text{if } M \models_\mu \varphi, \\ \bot & \text{otherwise.} \end{cases}$$

If φ is the set of all words, then we write $\mathcal{V}_\mu^{\psi}(M)$. Furthermore, we say M optimizes *ψ* under *μ, if $\mathcal{V}_\mu^{\psi}(M) \geq \mathcal{V}_\mu^{\psi}(M')$ for all systems M'.*

We could consider here also the traditional satisfaction relation, i.e., $M \models \varphi$. We have algorithms for both notions but we focus on satisfaction under μ, since satisfaction with probability 1 is the natural correctness criterion, if we are given a probabilistic environment assumption. Note that for safety specifications the two notions coincide, because we assume that the labeled Markov chain defining the input distribution is complete.[3] For parity specifications, the results in this section would change only slightly if we replace $M \models_\mu \varphi$ by $M \models \varphi$. In particular, instead of analyzing a Markov chain with parity objective, we would have to analyze an automaton with parity objective. We discuss the the alternative synthesis algorithm in the conclusions.

Lemma 1. *Given a finite-state machine M, a safety or a parity automaton A, a mean-payoff automaton B, and a labeled Markov chain (G, λ_G) defining a probability measure μ on $(\Sigma_I^\omega, \mathcal{F})$, we can construct a Markov chain $G' = (S', s_0', E', \delta')$, a reward function r', and a priority function p' such that*

$$\mathcal{V}_\mu^{L_A, L_B}(M) = \begin{cases} 2 \cdot \mathsf{V}_{G'}(\mathsf{mean}_{r'})(s_0') & \text{if } A \text{ is a safety automaton,} \\ 2 \cdot \mathsf{V}_{G'}(\mathsf{mp}_{p',r'})(s_0') & \text{otherwise.} \end{cases}$$

We first build the product of M, A, B (cf. Fig. 3). Then, G' alternates between (1) moving according to G, which means choosing an input value according to the distribution given by G, and (2) moving in $M \times A \times B$ according to the chosen input. The reward given by B for this transition is assigned to the intermediate state. The priorities are copied from A. The value $\mathcal{V}_\mu^{L_B}(M)$ is twice the expectation $\mathsf{V}_{G'}(\mathsf{mean}_{r'})(s_0')$, since we have introduced 0-rewards in every second step. Using Lemma 1 and the fact that we can compute $\mathsf{V}_{G'}(\mathsf{mean}_{r'})(s_0')$ and $\mathsf{V}_{G'}(\mathsf{mp}_{p',r'})(s_0')$ in polynomial time for Markov chains [14, 26], we obtain the following results. Detailed proofs can be found in the technical report [12].

Theorem 1. *Given a finite-state machine M, a parity automaton A, a mean-payoff automaton B, and a labeled Markov chain (G, λ_G) defining a probability measure μ, we can compute the value $\mathcal{V}_\mu^{L_A, L_B}(M)$ in polynomial time. Furthermore, if (G, λ_G) defines a uniform input distribution, then $\mathcal{V}_\oslash^{L_B}(M) = \mathcal{V}_\mu^{L_B}(M)$*[4].

Example 3. Recall the two system M_1 and M_2 (Fig. 2 and 4, respectively) and the specification A (cf. Fig. 1) that rewards quick responses. The two systems are equivalent

[3] Recall that a Markov chain is complete, if in every state there is an edge for every input value. Since every edge has a positive probability, also every finite path has a positive probability and therefore a system violating a safety specification will have a value $\bot$. If the Markov chain is not complete (i.e., we are given an input distribution that assigns probability 0 to some finite input sequences), we require a simple pre-processing step that restricts our algorithms to the set of states satisfying the safety condition independent of the input assumption. This set can be computed in linear time by solving a safety game.

[4] We can show that this measure is invariant under transformations of the computation tree.

wrt the worst case behavior. Let us consider the average behavior: we build a Markov chain $G_\oslash$ that assigns $1/4$ to all events in $2^{\{r_1,r_2\}}$. To measure M_1, we take the product between $G_\oslash$ and $M_1 \times A$ (shown in Fig. 3). The product looks like the automaton in Fig. 3 with an intermediate state for each edge. This state is labeled with the reward of the edge. All transition leading to intermediate states have probability $1/2$, the other once have probability 1. So the expectation of being in a state is the same for all four main states (i.e., $1/8$) and half of it in the eight intermediate states (i.e., $1/16$). Four (intermediate) states have a reward of 1, four have a reward of 2. So we get a total reward of $4 \cdot 1/16 + 4 \cdot 2 \cdot 1/16 = 3/4$, and a system value of 1.5. This is expected when looking at Fig. 3 because each state has two inputs resulting in a reward of 2 and two inputs with reward 1. For System M_2, we obtain Markov chain similar to Fig. 5 but now the probability of the transitions corresponding to the self-loops on the initial state sum up to $3/4$. So it is more likely to state in the initial state, then to leave it. The expectation for being in the states (q_0, q_0, q_0),(q_1, q_0, q_1), and (q_2, q_1, q_0) are $2/3$, $2/9$, and $1/9$, respectively, and their expected rewards are $(2 + 2 + 2 + 1)/4 = 7/4$, $3/2$, and $3/2$, respectively. So, the total reward of System M_2 is $2/3{\cdot}7/4{+}2/9{\cdot}3/2{+}1/9{\cdot}3/2 = 1.67$, which is clearly better than the value of system M_1 for specification A.

4 Synthesizing Optimal Systems

In this section, we show how to construct a system that satisfies a qualitative specification and optimizing a quantitative specification under a given probabilistic environment. First, we reduce the problem to finding an optimal strategy in an MDP for a mean-payoff (parity) objective. Then, we show how to compute such a strategy using end components and a reduction to max objective. Finally, we provide a linear program that computes the value function of an MDP with max objective. This shows that MDPs with mean-payoff parity objective can be solved in polynomial time.

Lemma 2. *Given a safety (resp. parity) automaton A, a mean-payoff automaton B, and a labeled Markov chain (G, λ_G) defining a probability measure μ on $(\Sigma_I^\omega, \mathcal{F})$, we can construct a labeled MDP $(G', \lambda_{G'})$ with $G' = (S', s_0', E', S_1', S_P', \delta')$, a reward function r', and a priority function p' such that every pure strategy π that is optimal from state s_0' for the objective $\mathsf{mean}_{r'}$ (resp. $\mathsf{mp}_{p',r'}$) and for which $\mathbb{E}^\pi_{s_0'}(\mathsf{mean}_{r'}) \neq \bot$ (resp. $\mathbb{E}^\pi_{s_0'}(\mathsf{mp}_{p',r'}) \neq \bot$) corresponds to a state machine M that satisfies L_A under μ and optimizes L_B under μ.*

The construction of G' is very similar to the construction used in Lemma 1. Intuitively, G' alternates between mimicking a move of G and mimicking a move of $A \times B \times C$, where C is an automaton with $|\Sigma_O|$-states that pushes the output labels from transitions to states, i.e., the transition function δ_C of C is the largest transition function s.t. $\forall s, s', \sigma, \sigma' : \delta_C(s, \sigma) = \delta_C(s', \sigma') \rightarrow \sigma = \sigma'$. Priorities p' are again copied from A and rewards r' from B. The labels for $\lambda_{G'}$ are either taken from λ_G (in intermediate state) or they correspond to the transitions taken in C. Every pure strategy in G' fixes one output value for every possible input sequence. The construction of the state machine depends on the structure of the strategy. For pure memoryless strategies, the construction is straight forward. At the end of this section, we discuss how to deal with other strategies.

The following theorem follows from Lemma 2 and the fact that MDPs with mean-payoff objective have pure memoryless optimal strategies and they can be computed in polynomial time (cf. [26]).

Theorem 2. *Given a safety automaton A, a mean-payoff automaton B, and a labeled Markov chain (G, λ_G) defining a probability measure μ, we can construct a finite-state machine M (if one exists) in polynomial time that satisfies L_A under μ and optimizes L_B under μ.*

MDPs with mean-payoff parity objectives. It follows from Lemma 2 that if the qualitative specification is a parity automaton, along with the Markov chain for probabilistic input assumption, and mean-payoff automata for quantitative specification, then the solution reduces to solving MDPs with mean-payoff parity objective. In the following we provide an algorithmic solution of MDPs with mean-payoff parity objective. We first present few basic results on MDPs.

End components of MDPs. Given an MDP $G = (S, s_0, E, S_1, S_P, \delta)$, a set $U \subseteq S$ of states is an *end component* [16, 18] if U is δ-closed (i.e., for all $s \in U \cap S_P$ we have $E(s) \subseteq U$) and the sub-game of G restricted to U (denoted $G \upharpoonright U$) is strongly connected. We denote by $\mathcal{E}(G)$ the set of end components of an MDP G. Given any strategy (memoryless or not), with probability 1 the set of states visited infinitely often along a play is an end component. More precisely, given an MDP G, for all states $s \in S$ and all strategies $\pi \in \Pi$, we have $\mu_s^\pi(\{\omega \mid \mathrm{Inf}(\omega) \in \mathcal{E}(G)\}) = 1$ [16, 18]. Furthermore, for an end component $U \in \mathcal{E}(G)$, consider the memoryless strategy π_U that plays in any state s in $U \cap S_1$ all edges in $E(s) \cap U$ uniformly at random. In the Markov chain obtained by fixing π_U, the end component U is a closed connected recurrent set.

Lemma 3. *Given an MDP G and an end component $U \in \mathcal{E}(G)$, the strategy π_U ensures that for all states $s \in U$, we have $\mu_s^{\pi_U}(\{\omega \mid \mathrm{Inf}(\omega) = U\}) = 1$.*

It follows that the strategy π_U ensures that from any starting state s, any other state t is reached in finite time with probability 1. From Lemma 3 we can conclude that in an MDP the value for mean-payoff parity objectives can be obtained by computing values for end-components and then applying the maximal expectation to reach the values of the end components.

Lemma 4. *Consider an MDP G with state space S, a priority function p, and reward function r such that (a) G is an end-component (i.e., S is an end component) and (b) the minimum priority in S is even. Then the value for mean-payoff parity objective for all states coincide with the value for mean-payoff objective, i.e., for all states s we have $\mathsf{V}_G(\mathsf{mp}_{p,r})(s) = \mathsf{V}_G(\mathsf{mean}_r)(s)$.*

The proof idea is to take two strategies: one for the mean-payoff and one for the parity objective, and combine them to produce the optimal value for the mean-payoff parity objective, which is equal to the optimal mean-payoff value. We take an optimal pure memoryless strategy π_m for the mean-payoff objective and a pure memoryless strategy π_S for the stochastic shortest path to reach the states with the minimum priority (which is even). Observe that (i) under the strategy π_S, from any state s we can reach the minimum (even) priority in finite time with probability 1; (ii) the mean-payoff value for all states is the same, because strategy π_U (from Lemma 3) ensures that every state can reach every other state in finite time with probability 1; and (iii) the strategy π_m ensures that for any $\varepsilon > 0$, there exists $j(\varepsilon) \in \mathbb{N}$ such that if π_m is played for any $\ell \geq j(\varepsilon)$ steps then the expected average of the rewards for ℓ steps is within ε of optimal mean-payoff value. So, we can achieve the optimal mean-payoff value and satisfies the parity objective by alternating between the two strategies, if we ensure to play π_m "long enough". Let β be the maximum absolute value of the rewards. The optimal strategy for mean-payoff objective is played in rounds, each having two stages. The strategy for round i is as follows: *(Stage 1)* First play the strategy π_S till the minimum priority state is reached. *(Stage 2)* Let $\varepsilon_i = 1/i$. If the game was in the first stage in this (i-th round) for k_i steps, then play the strategy π_m for ℓ_i steps such that $\ell_i \geq \max\{j(\varepsilon_i), i \cdot k_i \cdot \beta\}$. Then the strategy proceeds to round $i + 1$. This strategy guarantees the satisfaction of the parity objective and the optimal mean-payoff value. A full proof can be found in [12].

The above lemma shows that in an end component if the minimum priority is even, then the value for mean-payoff parity and mean-payoff objective coincide if we consider the sub-game restricted to the end component.

Computing best end-component values. We first compute a set S^* such that every end component U with $\min(p(U))$ is even is a subset of S^*. We also compute a function $f^* : S^* \to \mathbb{R}^+$ that assigns to every state $s \in S^*$ the mean-payoff parity value that can be obtained by visiting only states of an end component that contains s. The computation of S^* and f^* is as follows: *(1)* S_0^* is the set of maximal end-components with priority 0 and for a state $s \in S_0^*$ the function f^* assigns the mean-payoff value when the sub-game is restricted to S_0^* (by Lemma 4 we know that if we restrict the game to the end-components, then the mean-payoff values and mean-payoff parity values coincide); *(2)* for $i \geq 0$, let S_{2i}^* be the set of maximal end components with states with priority $2i$ or more and that contains at least one state with priority $2i$, and f^* assigns the mean-payoff value of the MDP restricted to the set of end components S_{2i}^*. The set $S^* = \bigcup_{i=0}^{\lfloor d/2 \rfloor} S_{2i}^*$. This gives the values under the end-component consideration, and to compute the maximal reachability expectation we present the following reduction.

Transformation to MDPs with $\max$ **objective.** Given an MDP $G = (S, s_0, E, S_1, S_P, \delta)$ with a positive reward function $r : S \to \mathbb{R}^+$ and a priority function $p : S \to \{0, \dots, d\}$, and let S^* and f^* be the output of the above procedure. We construct an MDP $\overline{G} = (\overline{S}, s_0, \overline{E}, \overline{S}_1, S_P, \delta)$ with a reward function $\overline{r}$ as follows: $\overline{S} = S \cup \widehat{S}^*$ (i.e., the set of states consists of the state space S and a copy $\widehat{S}^*$ of S^*), $\overline{E} = E \cup \{(s, \widehat{s}) \mid s \in S^* \cap S_1$and $\widehat{s}$ is the copy of s in $\widehat{S^*}\} \cup \{(\widehat{s}, \widehat{s}) \mid \widehat{s} \in \widehat{S}^*\}$ (i.e., along with edges E, for all player 1 states s in S^* there is an edge to its copy $\widehat{s}$ in $\widehat{S}^*$, and all states in $\widehat{S}^*$ are absorbing states), $\overline{S}_1 = S_1 \cup \widehat{S}^*$, $\overline{r}(s) = 0$ for all $s \in S$ and $\overline{r}(\widehat{s}) = f^*(s)$, where $\widehat{s}$ is the copy of s. This construction ensures that $\mathsf{V}_G(\mathsf{mp}_{p,r})(s) = \mathsf{V}_{\overline{G}}(\mathsf{max}_{\overline{r}})(s)$. We refer the reader to [12] for a detailed proof.

In order to solve $\overline{G}$ with the objective $\mathsf{max}_{\overline{r}}$, we set up the following linear program, which can be solved with a standard LP solver (e.g., [29]).

Linear programming for the $\max$ **objective in** $\overline{G}$**.** The following linear program characterizes the value function $\mathsf{V}_{\overline{G}}(\mathsf{max}_{\overline{r}})$. Observe that we have already restricted ourselves to the almost-sure winning states $W_G(\mathsf{parity}_p)$, and below we assume $W_G(\mathsf{parity}_p) = S$. For all $s \in \overline{S}$ we have a variable x_s and the objective function is $\min \sum_{s \in \overline{S}} x_s$. The set of linear constraints are as follows: (1) $\forall s \in \overline{S} : x_s \geq 0$, (2) $\forall s \in \widehat{S}^* : x_s = \overline{r}(s)$, (3) $\forall s \in \overline{S}_1, (s,t) \in \overline{E} : x_s \geq x_t$, and (4) $\forall s \in \overline{S}_P : x_s = \sum_{t \in \overline{S}} \overline{\delta}(s)(t) \cdot x_t$. The correctness proof of this linear program can be found in [12].

Lemma 5. *Given a MDP with a mean-payoff parity objective, the value function for the mean-payoff parity objective can be computed in polynomial time.*

Note that the optimal strategies constructed for mean-payoff parity requires memory, but the memory requirement is captured by a counter (which can be represented by a state machine with state space $\mathbb{N}$). The optimal strategy as described in Lemma 4 plays two memoryless strategies, and each strategy is played a number of steps which can be stored in a counter. Furthermore, we can show that the decision problem, whether there exists an optimal pure memoryless strategy is NP-complete; the upper bound follows from Theorem 1; the lower bound follows from a reduction of the directed subgraph homeomorphism problem [27]. Lemma 2 and Lemma 5 yield the following theorem.

Theorem 3. *Given a Parity specification A, a Mean-payoff specification B, and a labeled Markov chain (G, λ) defining a probability measure μ on $(\Sigma_I^\omega, \mathcal{F})$, we can construct a state machine M (if one exists) in polynomial time that satisfies L_A under μ and optimizes L_B under μ.*

5 Experimental Results

The aim of this section is to show which types of systems, we can construct using qualitative and quantitative specifications under probabilistic environment assumptions. We have implemented the approach for specifications consisting of a safety automaton A and a mean-payoff automaton B, and where the assumption μ is given as a set of probability distributions d_s over input letters for each state s of B. Our implementation is in Scala [35]. It takes automata in GOAL-format [37] as input and first builds the product of A and B. Then, it construct the corresponding MDP G and computes an optimal pure memoryless strategy using policy iteration for multi-chain MDPs [26]. Finally, if the value of the strategy is different from $\bot$, then it converts the strategy to a finite-state machine M which satisfies L_A (under μ) and is optimal for B under μ.

Priority-driven controller. In our first experiment, we took as the quantitative specification B the product of the specifications A_1 and A_2 from Example 1 (Fig. 1), where we sum the weights on the edges. The qualitative specification is a safety automaton A ensuring mutually exclusive grants. We assumed the constant probabilities $P(\{r_1 = 1\}) = 0.4$ and $P(\{r_2 = 1\}) = 0.3$ for the events $r_1 = 1$ and $r_2 = 1$, respectively. The optimal machine[5] constructed by the tool is shown in Fig. 6. This system behaves like a priority-driven scheduler, which always grants the resource to the

[5] State q_0 and q_1 are simulation equivalent but our tool does not minimize state machines yet.

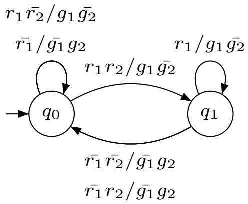
$r_1\bar{r_2}/g_1\bar{g_2}$
$\bar{r_1}/\bar{g_1}g_2$
$r_1/g_1\bar{g_2}$
$r_1r_2/g_1\bar{g_2}$
q_0
q_1
$\bar{r_1}\bar{r_2}/\bar{g_1}g_2$
$\bar{r_1}r_2/\bar{g_1}g_2$

Fig. 6. Optimal machine

Table 1. Results for n-client without response constraints

n	$A \times B$	G	M	Value
2	4	13	2	1.854
3	8	35	4	2.368
4	16	97	8	2.520
5	32	275	16	2.534
6	64	793	32	2.534

Table 2. Results for n-client with response constraints

n	$A \times B$	G	M	Value
2	3	11	3	1.850
3	34	156	16	2.329
4	125	557	125	2.366

client that is more likely to send requests, if she is requesting it, otherwise the resource is granted to the other client. This is optimal because client 1 is more likely to send requests and so missing a request from client 2 is better than missing a request from 1.

Fair controller. In the second experiment, we added response constraints to the safety specification. The constraints are given as safety automata that require that every request is granted within two steps. We added one automaton C_i for each client i and the final qualitative specification was $A \times C_1 \times C_2$. The optimal machine the tool constructs is System M_2 of Example 1 (Fig. 4). System M_2 follows the request sent, if only a single request is send. If both clients request simultaneously, it alternates between g_1 and g_2. If none of the clients is requesting it grants g_1. Recall that system M_1 and M_2 from Example 1 exhibit the same worst-case behavior, so a synthesis approach based on optimizing the worst-case behavior would not be able to construct M_2.

General controllers. We reran both experiments for several clients. Again, the quantitative specification was the product of A_i's. We used a skewed probability distribution with $P(\{r_n = 1\}) = 0.3$ and $P(\{r_i = 1\}) = P(\{r_{i+1} = 1\}) + 0.1$ for $1 \le i \le 6$ and the qualitative specification required mutual exclusion. Table 1 shows the number of clients (n), the size of the specification ($A \times B$), the size of the corresponding MDP (G), and the size of the resulting machine (M) and the optimal value (Value). The runs took between least than a second to a couple of minutes. The systems generated as a result of this experiment have an intrinsic priority to granting requests in order of probabilities from largest to smallest. Table 2 shows the results when adding response constraints that require that every request has to be granted within the next n steps. This experiment leads to quite intelligent systems which prioritize with the most probable input request but slowly the priority shifts to the next request variable cyclically resulting into servicing any request in n steps when there are n clients. Note that these systems are (as expected) quite a bit larger than the corresponding priority-driven controllers.

6 Conclusions and Future Work

In this paper we showed how to measure and synthesize systems under probabilistic environment assumptions wrt qualitative and quantitative specifications. We considered the satisfaction of the qualitative specification with probability 1 ($M \models_\mu \varphi$). Alternatively, we could have considered the satisfaction of the qualitative specification with certainty ($M \models \varphi$). For safety specification the two notions coincide, however, they are different for parity specification. The notion of satisfaction of the parity specification with certainty and optimizing the mean-payoff specification can be obtained similar to the solution of mean-payoff parity games [13] by replacing the solution of mean-payoff games by solution of MDPs with mean-payoff objectives. However, since solving MDPs with parity specification for certainty is equivalent to solving two-player parity games, and no polynomial time algorithm is known for parity games, the algorithmic solution for the satisfaction of the qualitative specification with certainty is computationally expensive as compared to the polynomial time algorithm for MDPs with mean-payoff parity objectives. Moreover, under probabilistic assumption satisfaction with probability 1 is the natural notion.

In our future work, we will implement our algorithm for MDPs with mean-payoff parity conditions and develop a tool for synthesizing systems in probabilistic environments with ω-regular specifications. In the course of developing this tool, it will be interesting to study subclasses of specifications for which we can construct finite-state systems (i.e., systems without counters). We will also explore the use of a logical framework to express quantitative properties to simplify stating quantitative specifications.

References

1. de Alfaro, L.: Temporal logics for the specification of performance and reliability. In: Reischuk, R., Morvan, M. (eds.) STACS 1997. LNCS, vol. 1200, pp. 165–176. Springer, Heidelberg (1997)
2. Alur, R., Degorre, A., Maler, O., Weiss, G.: On omega-languages defined by mean-payoff conditions. In: de Alfaro, L. (ed.) FOSSACS 2009. LNCS, vol. 5504, pp. 333–347. Springer, Heidelberg (2009)
3. Baier, C., Größer, M., Leucker, M., Bollig, B., Ciesinski, F.: Controller synthesis for probabilistic systems. In: IFIP TCS, pp. 493–506 (2004)
4. Baier, C., Katoen, J.-P.: Principles of Model Checking. Representation and Mind Series. The MIT Press, Cambridge (2008)
5. Bianco, A., de Alfaro, L.: Model checking of probabilistic and nondeterministic systems. In: Thiagarajan, P.S. (ed.) FSTTCS 1995. LNCS, vol. 1026, pp. 499–513. Springer, Heidelberg (1995)
6. Bloem, R., Chatterjee, K., Henzinger, T.A., Jobstmann, B.: Better quality in synthesis through quantitative objectives. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 140–156. Springer, Heidelberg (2009)
7. Bloem, R., Greimel, K., Henzinger, T.A., Jobstmann, B.: Synthesizing robust systems. In: FMCAD'09 (2009)
8. Chakrabarti, A., Chatterjee, K., Henzinger, T.A., Kupferman, O., Majumdar, R.: Verifying quantitative properties using bound functions. In: Borrione, D., Paul, W. (eds.) CHARME 2005. LNCS, vol. 3725, pp. 50–64. Springer, Heidelberg (2005)
9. Chakrabarti, A., de Alfaro, L., Henzinger, T.A., Stoelinga, M.: Resource interfaces. In: Alur, R., Lee, I. (eds.) EMSOFT 2003. LNCS, vol. 2855, pp. 117–133. Springer, Heidelberg (2003)
10. Chatterjee, K., de Alfaro, L., Faella, M., Henzinger, T.A., Majumdar, R., Stoelinga, M.: Compositional quantitative reasoning. In: QEST, pp. 179–188 (2006)
11. Chatterjee, K., Doyen, L., Henzinger, T.A.: Quantitative languages. In: Kaminski, M., Martini, S. (eds.) CSL 2008. LNCS, vol. 5213, pp. 385–400. Springer, Heidelberg (2008)
12. Chatterjee, K., Henzinger, T., Jobstmann, B., Singh, R.: Measuring and synthesizing systems in probabilistic environments. In: CoRR, arXiv:1004.0739 (2010)

13. Chatterjee, K., Henzinger, T.A., Jurdzinski, M.: Mean-payoff parity games. In: LICS, pp. 178–187 (2005)
14. Chatterjee, K., Jurdziński, M., Henzinger, T.A.: Simple stochastic parity games. In: Baaz, M., Makowsky, J.A. (eds.) CSL 2003. LNCS, vol. 2803, pp. 100–113. Springer, Heidelberg (2003)
15. Chatterjee, K., Jurdziński, M., Henzinger, T.A.: Quantitative stochastic parity games. In: SODA'04, pp. 121–130. SIAM, Philadelphia (2004)
16. Courcoubetis, C., Yannakakis, M.: Markov decision processes and regular events. In: Paterson, M. (ed.) ICALP 1990. LNCS, vol. 443, pp. 336–349. Springer, Heidelberg (1990)
17. Cuninghame-Green, R.A.: Minimax algebra. Lecture Notes in Economics and Mathematical Systems, vol. 166. Springer, Heidelberg (1979)
18. de Alfaro, L.: Formal Verification of Probabilistic Systems. PhD thesis, Stanford University (1997)
19. de Alfaro, L.: Stochastic transition systems. In: Sangiorgi, D., de Simone, R. (eds.) CONCUR 1998. LNCS, vol. 1466, pp. 423–438. Springer, Heidelberg (1998)
20. de Alfaro, L., Henzinger, T.A., Majumdar, R.: Discounting the future in systems theory. In: ICALP'03 (2003)
21. de Alfaro, L., Majumdar, R., Raman, V., Stoelinga, M.: Game relations and metrics. In: LICS, pp. 99–108. IEEE Computer Society Press, Los Alamitos (2007)
22. Desharnais, J., Gupta, V., Jagadeesan, R., Panangaden, P.: Metrics for labelled markov systems. In: Baeten, J.C.M., Mauw, S. (eds.) CONCUR 1999. LNCS, vol. 1664, pp. 258–273. Springer, Heidelberg (1999)
23. Droste, M., Gastin, P.: Weighted automata and weighted logics. Theoretical Computer Science 380, 69–86 (2007)
24. Droste, M., Kuich, W., Rahonis, G.: Multi-valued MSO logics over words and trees. Fundamenta Informaticae 84, 305–327 (2008)
25. Droste, M., Kuich, W., Vogler, H.: Handbook of Weighted Automata. Springer Publishing Company, Incorporated, Heidelberg (2009)
26. Filar, J., Vrieze, K.: Competitive Markov Decision Processes. Springer, Heidelberg (1996)
27. Fortune, S., Hopcroft, J.E., Wyllie, J.: The directed subgraph homeomorphism problem. Theor. Comput. Sci., 111–121 (1980)
28. Gaubert, S.: Methods and applications of (max, +) linear algebra. In: Reischuk, R., Morvan, M. (eds.) STACS 1997. LNCS, vol. 1200, pp. 261–282. Springer, Heidelberg (1997)
29. Glpk (gnu linear programming kit), http://www.gnu.org/software/glpk/
30. Haverkort, B.R.: Performance of Computer Communication Systems: A Model-Based Approach. John Wiley & Sons, Inc., New York (1998)
31. Katz, G., Peled, D.: Code mutation in verification and automatic code correction. In: TACAS 2010 (to appear, 2010)
32. Kupferman, O., Lustig, Y.: Lattice automata. In: Cook, B., Podelski, A. (eds.) VMCAI 2007. LNCS, vol. 4349, pp. 199–213. Springer, Heidelberg (2007)
33. Kwiatkowska, M., Norman, G., Parker, D.: PRISM: Probabilistic model checking for performance and reliability analysis. ACM SIGMETRICS Perform. Evaluation Review (2009)
34. Niebert, P., Peled, D., Pnueli, A.: Discriminative model checking. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 504–516. Springer, Heidelberg (2008)
35. Odersky, M., Spoon, L., Venners, B.: Programming in Scala. Artima (2008), http://www.scala-lang.org/
36. Puterman, M.L.: Markov Decision Processes. John Wiley and Sons, Chichester (1994)
37. Tsay, Y.-K., Chen, Y.-F., Tsai, M.-H., Wu, K.-N., Chan, W.-C.: GOAL: A graphical tool for Büchi automata and temporal formulae. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 466–471. Springer, Heidelberg (2007), http://goal.im.ntu.edu.tw