Better Quality in Synthesis through Quantitative Objectives[⋆]

Roderick Bloem[1], Krishnendu Chatterjee[2],
Thomas A. Henzinger[3], and Barbara Jobstmann[3]

[1]Graz University of Technology, [2]IST, Austria, [3]EPFL

Abstract. Most specification languages express only qualitative constraints. However, among two implementations that satisfy a given specification, one may be preferred to another. For example, if a specification asks that every request is followed by a response, one may prefer an implementation that generates responses quickly but does not generate unnecessary responses. We use quantitative properties to measure the "goodness" of an implementation. Using games with corresponding quantitative objectives, we can synthesize "optimal" implementations, which are preferred among the set of possible implementations that satisfy a given specification.

In particular, we show how automata with lexicographic mean-payoff conditions can be used to express many interesting quantitative properties for reactive systems. In this framework, the synthesis of optimal implementations requires the solution of lexicographic mean-payoff games (for safety requirements), and the solution of games with both lexicographic mean-payoff and parity objectives (for liveness requirements). We present algorithms for solving both kinds of novel graph games.

1 Introduction

Traditional specifications are Boolean: an implementation satisfies a specification, or it does not. This Manichean view is not entirely satisfactory: There are usually many different ways to satisfy a specification, and we may prefer one implementation over another. This is especially important when we automatically synthesize implementations from a specification, because we have no other way to enforce these preferences. In this paper, we add a quantitative aspect to system specification, imposing a preference order on the implementations that satisfy the qualitative part of the specification. Then, we present synthesis algorithms that construct, from a given specification with both qualitative and quantitative aspects, an implementation that (i) satisfies the qualitative aspect and (ii) is optimal or near-optimal with respect to the quantitative aspect. Along the way, we introduce and solve graph games with new kinds of objectives, namely, lexicographic mean-payoff objectives and the combination of parity and lexicographic mean-payoff objectives.

[⋆] This research was supported by the Swiss National Science Foundation (Indo-Swiss Research Program and NCCR MICS) and the European Union projects COMBEST and COCONUT.

Suppose we want to specify an arbiter for a shared resource. For each client i, the arbiter has an input r_i (request access) and an output g_i (access granted). A first attempt at a specification in LTL may be $\bigwedge_i \mathsf{G}(r_i \rightarrow \mathsf{F}\, g_i) \wedge \mathsf{G} \bigwedge_i \bigwedge_{j \neq i} (\neg g_i \vee \neg g_j)$. (All requests are granted eventually and two grants never occur simultaneously.) This specification is too weak: An implementation that raises all g_i signals in a round-robin fashion satisfies the specification but is probably undesired. The unwanted behaviors can be ruled out by adding the requirements $\bigwedge_i \mathsf{G}(g_i \rightarrow \mathsf{X}(\neg g_i \,\mathsf{W}\, r_i)) \wedge \bigwedge_i \neg g_i \,\mathsf{W}\, r_i$. (No second grant before a request.)

Such Boolean requirements to rule out trivial but undesirable implementations have several drawbacks: (i) they are easy to forget and difficult to get right (often leading to unrealizable specifications) and, perhaps more importantly, (ii) they constrain implementations unnecessarily, by giving up the abstract quality of a clean specification. In our example, we would rather say that the implementation should produce "as few unnecessary grants as possible" (where a grant g_i is unnecessary if there is no outstanding request r_i). We will add a quantitative aspect to specifications which allows us to say that. Specifically, we will assign a real-valued reward to each behavior, and the more unnecessary grants, the lower the reward.

A second reason that the arbiter specification may give rise to undesirable implementations is that it may wait arbitrarily long before producing a grant. Requiring that grants come within a fixed number of steps instead of "eventually" is not robust, because it depends on the step size of the implementation and the number of clients. Rather, we assign a lower reward to executions with larger distances between a request and corresponding grant. If we use rewards both for punishing unnecessary grants and for punishing late grants, then these two rewards need to be combined. This leads us to consider tuples of costs that are ordered lexicographically. We define the quantitative aspect of a specification using *lexicographic mean-payoff automata*, which assign a tuple of costs to each transition. The cost of an infinite run is obtained by taking, for each component of the tuple, the long-run average of all transition costs. Such automata can be used to specify both "produce as few unnecessary grants as possible" and "produce grants as quickly as possible," and combinations thereof.

If the qualitative aspect of the specification is a safety property, then synthesis requires the solution of *lexicographic mean-payoff games*, for which we can synthesize optimal solutions. (The objective is to minimize the cost of an infinite run lexicographically.) If the qualitative aspect is a liveness property, then we obtain *lexicographic mean-payoff parity games*, which must additionally satisfy a parity objective. We present the solution of these games in this paper. We show that lexicographic mean-payoff games are determined for memoryless strategies and can be decided in NP $\cap$ coNP, but that in general optimal strategies for lexicographic mean-payoff parity games require infinite memory. We prove, however, that for any given real vector $\boldsymbol{\varepsilon} > \mathbf{0}$, there exists a finite-state strategy that ensures a value within $\boldsymbol{\varepsilon}$ of the optimal value. This allows us to synthesize ε-optimal implementations, for any ε. The complexity class of the optimal synthesis problem is NP.

Related work. There are several formalisms for quantitative specifications in the literature [2, 5–8, 11, 12, 15, 16, 20]; most of these works (other than [2, 8, 11]) 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. Several notions of metrics have been proposed in the literature for probabilistic systems and games [13, 14]; 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 compares how good an implementation is with respect to a given specification. The work [10] considers non-zero-sum games with lexicographic ordering on the payoff profiles, but to the best of our knowledge, the lexicographic quantitative objective we consider for games has not been studied before.

2 Examples

After giving necessary definitions, we illustrate with several examples how quantitative constraints can be a useful measure for the quality of an implementation.

Alphabets, vectors, and lexicographic order. Let $\mathcal{I}$ and $\mathcal{O}$ be finite sets of *input* and *output signals*, respectively. We define the *input alphabet* $\Sigma_I = 2^{\mathcal{I}}$ and the *output alphabet* $\Sigma_O = 2^{\mathcal{O}}$. The joint alphabet Σ is defined as $\Sigma = 2^{\mathcal{I} \cup \mathcal{O}}$. Let $\mathbb{R}^d$ be the set of real vectors of dimension d with the usual lexicographic order.

Mealy machines. A *Mealy machine* is a tuple $M = \langle Q, q_0, \delta \rangle$, where Q is a finite set of states, $q_0 \in Q$ is the initial state, and $\delta \subseteq Q \times \Sigma_I \times \Sigma_O \times Q$ is a set of labeled edges. We require that the machine is *input enabled* and *deterministic*: $\forall q \in Q \,.\, \forall i \in \Sigma_I$, there exists a unique $o \in \Sigma_O$ and a unique $q' \in Q$ such that $\langle q, i, o, q' \rangle \in \delta$. Each input word $i = i_0 i_1 \cdots \in {\Sigma_I}^\omega$ has a unique *run* $q_0 i_0 o_0 q_1 i_1 o_1 \ldots$ such that $\forall k \geq 0 \,.\, \langle q_k, i_k, o_k, q_{k+1} \rangle \in \delta$. The corresponding *I/O word* is $i_0 \cup o_0, i_1 \cup o_1, \dots \in \Sigma^\omega$. The *language* of M, denoted by L_M, is the set of all I/O words of the machine. Given a language $L \subseteq \Sigma^\omega$, we say a Mealy machine M *implements* L if $L_M \subseteq L$.

Quantitative languages. A *quantitative language [8] over Σ* is a function $L : \Sigma^\omega \to V$ that associates to each word in Σ^ω a *value* from V, where $V \subset \mathbb{R}^d$ has a least element. Words with a higher value are more desirable than those with a lower value. In the remainder, we view an ordinary, *qualitative* language as a quantitative language that maps words in L to true $(= 1)$ and words not in L to false $(= 0)$. We often use a pair $\langle L, L' \rangle$ of a qualitative language L and a quantitative language $L' : \Sigma^\omega \to V$ as specification, where L has higher priority than L'. We can also view $\langle L, L' \rangle$ as quantitative language with $\langle L, L' \rangle(w) = \mathbf{0}$ if $L(w) = 0$, and $\langle L, L' \rangle(w) = L'(w) - v_\perp + \mathbf{1}$ otherwise, where $v_\perp$ is the minimal value in V. (Adding constant factors does not change the order between words).

We extend the definition of value to Mealy machines. As in verification and synthesis of qualitative languages, we take the worst-case behavior of the Mealy machine as a measure. Given a quantitative language L over Σ, the *value* of a Mealy machine M, denoted by $L(M)$, is $\inf_{w \in L_M} L(w)$.

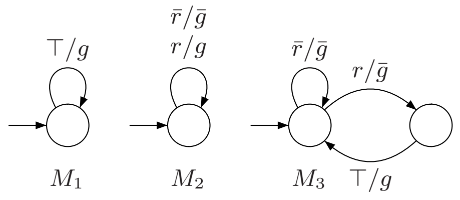

Fig. 1. Three Mealy machines that implement $\mathsf{G}(r \rightarrow g \vee \mathsf{X}\, g)$

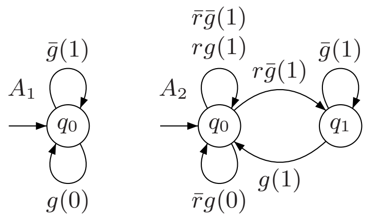

Fig. 2. Two specifications that provide different ways of charging for grants.

Lexicographic mean-payoff automata. We use lexicographic mean-payoff automata to describe quantitative languages. In lexicographic mean-payoff automata each edge is mapped to a reward. The automaton associates a run with a word and assigns to the word the average reward of the edges taken (as in mean-payoff games [17]). Unlike in mean-payoff games, rewards are vectors.

Formally, a *lexicographic mean-payoff automaton* of dimension d over Σ is a tuple $A = \langle\langle S, s_0, E\rangle, \boldsymbol{r}\rangle$, where S is a set of states, $E \subseteq S \times \Sigma \times S$ is a labeled set of edges, $s_0 \in S$ is the initial state, and $\boldsymbol{r} : E \rightarrow \mathbb{N}^d$ is a reward function that maps edges to d-vectors of natural numbers. Note that all rewards are non-negative. We assume that the automaton is complete and deterministic: for each s and σ there is exactly one s' such that $\langle s, \sigma, s'\rangle \in E$. A word $w = w_0 w_1 \cdots \in \Sigma^\omega$ has a unique run $\rho(w) = s_0 e_0 s_1 e_1 \ldots$ such that $s_i \in S$ and $e_i = \langle s_i, w_i, s_{i+1}\rangle \in E$ for all $i \geq 0$. The *lexicographic mean payoff* $LM(\rho)$ of a run ρ is defined as $LM(\rho) = \liminf_{n\rightarrow\infty} \frac{1}{n}\sum_{i=0}^{n} \boldsymbol{r}(e_i)$. The automaton defines a quantitative language with domain $\mathbb{R}^d$ by associating to every word w the value $L_A(w) = LM(\rho(w))$.

If the dimension of A is 1 and the range of L_A is $\{0, 1\}$ then, per definition, L_A defines a qualitative language. We say that A is a *safety automaton* if it defines a qualitative language and there is no path from an edge with reward 0 to an edge with reward > 0. Safety automata define safety languages [1]. Note that in general, ω-regular languages and languages expressible with mean-payoff automata are incomparable [8].

Example 1. Let us consider a specification of an arbiter with one client. In the following, we use r, $\bar{r}$, g, and $\bar{g}$ to represent that r or g are set to true and false, respectively and $\top$ to indicate that a signal can take either value. A slash separates input and output.

Take the specification $\varphi = \mathsf{G}(r \rightarrow g \vee \mathsf{X}\, g)$: every request is granted within two steps. The corresponding language L_φ maps a word $w = w_0 w_1, \ldots$ to true iff for every position i in w, if $r \in w_i$, then $g \in w_i \cup w_{i+1}$. Fig. 1 shows three implementations for L_φ. Machine M_1 asserts g continuously independent of r, M_2 responds to each request with a grant but keeps g low otherwise, and M_3 delays its response if possible.

We use a quantitative specification to state that we prefer an implementation that avoids unnecessary grants. Fig. 2 shows two mean-payoff automata, A_1 and

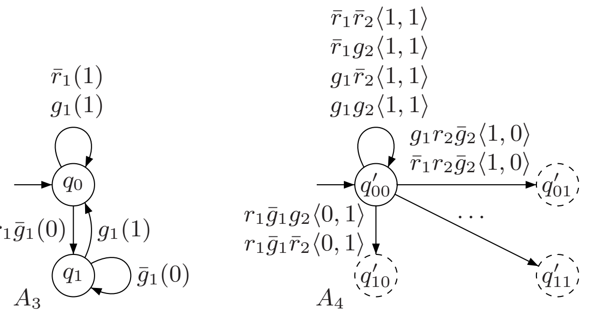
$\bar{r}_1(1)$
$g_1(1)$
q_0
$r_1\bar{g}_1(0)$
$g_1(1)$
q_1
$\bar{g}_1(0)$
A_3
$\bar{r}_1\bar{r}_2\langle 1,1\rangle$
$\bar{r}_1 g_2\langle 1,1\rangle$
$g_1\bar{r}_2\langle 1,1\rangle$
$g_1 g_2\langle 1,1\rangle$
$g_1 r_2\bar{g}_2\langle 1,0\rangle$
$\bar{r}_1 r_2\bar{g}_2\langle 1,0\rangle$
q'_{00}
q'_{01}
$r_1\bar{g}_1 g_2\langle 0,1\rangle$
$r_1\bar{g}_1\bar{r}_2\langle 0,1\rangle$
$\dots$
q'_{10}
q'_{11}
A_4

Fig. 3. A specification that rewards quick grants for a request from Client 1, and a specification that rewards quick grants for both clients, while giving priority to Client 1.

A_2 that define rewards for the behavior of an implementation. Note that we have summarized edges using Boolean algebra. For instance, an arc labeled g in the figure corresponds to the edges labeled rg and $\bar{r}g$. Automata A_1 and A_2 define quantitative languages that distinguish words by the frequency of grants and the condition under which they appear. Specification A_1 defines a reward of 1 except when a grant is given; A_2 only withholds a reward when a grant is given unnecessarily. Consider the words $w_1 = (rg, \bar{r}\bar{g})^\omega$ and $w_2 = (r\bar{g}, \bar{r}g, \bar{r}g)^\omega$. Specification A_1 defines the rewards $L_{A_1}(w_1) = 1/2$, and $L_{A_1}(w_2) = 1/3$. For A_2, we get $L_{A_2}(w_1) = 1$ and $L_{A_2}(w_2) = 2/3$. Both specifications are meaningful but they express different preferences, which leads to different results for verification and synthesis, as discussed in Section 4.

Recall the three implementations in Fig. 1. Each of them implements L_φ. For A_1, input r^ω gives the lowest reward. The values corresponding to the input/output word of M_1, M_2, and M_3 are 0, 0, and 1/2, respectively. Thus, A_1 prefers the last implementation. The values of the implementations for A_2 are minimal when the input is $\bar{r}^\omega$; they are 0, 1, and 1, respectively. Thus, A_2 prefers the last two implementations, but does not distinguish between them.

Example 2. Assume we want to specify an arbiter for two clients that answers requests within three steps. Simultaneous grants are forbidden. Formally, we have $\varphi = \bigwedge_{i\in\{1,2\}} \mathsf{G}\big(r_i \to \bigvee_{t\in\{0,1,2\}} \mathsf{X}^t\, g_i\big) \wedge \mathsf{G}(\neg g_1 \vee \neg g_2)$. We want grants to come as quickly as possible. Fig. 3 shows a mean-payoff automaton A_3 that rewards rapid replies to Client 1. Suppose we want to do the same for Client 2. One option is to construct a similar automaton A'_3 for Client 2 and to add the two resulting quantitative languages. This results in a quantitative language $L_{A_3} + L_{A'_3}$ that treats the clients equally. Suppose instead that we want to give Client 1 priority. In that case, we can construct a lexicographic mean-payoff automaton that maps a word w to a tuple $\langle s_1(w), s_2(w)\rangle$, where the first and second elements correspond to the payoff for Clients 1 and 2, resp. Part of this automaton, A_4, is shown in Fig. 3.

Automaton A_3 distinguishes words with respect to the maximal average distance between request and grant. For instance, $L_{A_3}((r_1 g_1, \bar{r}_1\bar{g}_1)^\omega) = 1$ and $L_{A_3}((r_1\bar{g}_1, \bar{r}_1 g_1)^\omega) = 1/2$. Automaton A_4 associates a vector to every word. For instance, $L_{A_4}((r_1 g_1 r_2 \bar{g}_2, \bar{r}_1 \bar{g}_1 \bar{r}_2 g_2)^\omega) = 1/2 \cdot (\langle 1, 0\rangle + \langle 1, 1\rangle) = \langle 1, 1/2\rangle$, which makes it preferable to the word $(r_1 \bar{g}_1 r_2 g_2, \bar{r}_1 g_1 \bar{r}_2 \bar{g}_2)^\omega$, which has value $\langle 1/2, 1\rangle$. This is what we expect: the first word gives priority to requests from Client 1, while the second gives priority to Client 2.

Example 3. Let us consider the liveness specification $\varphi = \mathsf{G}(r \rightarrow \mathsf{F}\, g)$ saying that every request must be granted eventually. This languages can usefully be combined with A_3, stating that grants must come quickly. It can also be combined with A_1 from Fig. 2 stating that grants should occur as infrequently as possible. A Mealy machine may emit a grant every k ticks, which gives a reward of $1 - 1/k$. Thus, there is an infinite chain of ever-better machines. There is no Mealy machine, however, that obtains the limit reward of 1. This can only be achieved by an implementation with infinite memory, for instance one that answers requests only in cycle 2^k for all k [9].

3 Lexicographic Mean-Payoff (Parity) Games

We show how to solve lexicographic mean-payoff games and lexicographic mean-payoff parity games, which we will need in Section 4 to solve the synthesis problem for quantitative specifications.

3.1 Notation and known results

Game graphs and plays. A *game graph* over the alphabet Σ is a tuple $G = \langle S, s_0, E\rangle$ consisting of a finite set of states S, partitioned into S_1 and S_2, representing the states of Player 1 and Player 2, an initial state $s_0 \in S$, and a finite set of labeled edges $E \subseteq S \times \Sigma \times S$. We require that the labeling is deterministic, i.e., if $\langle s, \sigma, t\rangle, \langle s, \sigma, t'\rangle \in E$, then $t = t'$. We write $\bar{E} = \{\langle s, t\rangle \mid \exists \sigma \in \Sigma : \langle s, \sigma, t\rangle \in E\}$. At S_1 states, Player 1 decides the successor state and at S_2 states, Player 2 decides the successor states. We assume that $\forall s \in S \,.\, \exists t \in S \,.\langle s, t\rangle \in \bar{E}$. A *play* $\rho = \rho_0 \rho_1 \dots \in S^\omega$ is an infinite sequence of states such that for all $i \geq 0$ we have $\langle \rho_i, \rho_{i+1}\rangle \in \bar{E}$. We denote the set of all plays by Ω.

The labels and the initial state are not relevant in this section. They are used later to establish the connection between specifications, games, and Mealy machines. They also allow us to view automata as games with a single player.

Strategies. Given a game graph $G = \langle S, s_0, E\rangle$, a *strategy for Player 1* is a function $\pi_1 : S^* S_1 \rightarrow S$ such that $\forall s_0 \dots s_i \in S^* S_1$ we have $\langle s_i, \pi_1(s_0 s_1 \dots s_i)\rangle \in \bar{E}$. A Player-2 strategy is defined similarly. We denote the set of all Player-p strategies by Π_p. The *outcome* $\rho(\pi_1, \pi_2, s)$ of π_1 and π_2 on G starting at s is the unique play $\rho = \rho_0 \rho_1 \dots$ such that for all $i \geq 0$, if $\rho_i \in S_p$, then $\rho_{i+1} = \pi_p(\rho_0 \dots \rho_i)$ and $\rho_0 = s$.

A strategy $\pi_p \in \Pi_p$ is memoryless if for any two sequences $\sigma = s_0 \dots s_i \in S^* S_p$ and $\sigma' = s'_0 \dots s'_{i'} \in S^* S_p$ such that $s_i = s'_{i'}$, we have $\pi_p(\sigma) = \pi_p(\sigma')$. We represent a memoryless strategy π_p simply as a function from S_p to S. A strategy is a *finite-memory strategy* if it needs only finite memory of the past, consisting of a finite-state machine that keeps track of the history of the play. The strategy chooses a move depending on the state of the machine and the location in the game. Strategies that are not finite-memory are called *infinite-memory strategies.*

Quantitative and qualitative objectives. We consider different objectives for the players. A *quantitative* objective f is a function $f : \Omega \to \mathbb{R}^d$ that assigns a vector of reals as reward to every play. We consider complementary objectives for the two players; i.e., if the objective for Player 1 is f, then the objective for Player 2 is $-f$. The goal of each player is to maximize her objective. Note that Player 2 tries to minimize f by maximizing the complementary $-f$. An objective $f : \Omega \to \{0, \pm 1\}$ that maps to the set $\{0, 1\}$ (or $\{0, -1\}$) is per definition a *qualitative objective*. Given a qualitative objective $f : \Omega \to V$ we say a play $\rho \in \Omega$ is *winning for Player 1* if $f(\rho) = \max(V)$ holds, otherwise the play is *winning for Player 2*.

Value. Given an objective f, the *Player-1 value of a state s for a strategy* π_1 is the minimal value Player 1 achieves using π_1 against all Player-2 strategies, i.e., $\mathcal{V}_1(f, s, \pi_1) = \inf_{\pi_2 \in \Pi_2} f(\rho(\pi_1, \pi_2, s))$. The *Player-1 value of a state s* is the maximal value Player-1 can ensure from state s, i.e., $\mathcal{V}_1(f, s) = \sup_{\pi_1 \in \Pi_1} \mathcal{V}_1(f, s, \pi_1)$. Player-2 values are defined analogously. If $\mathcal{V}_1(f, s) + \mathcal{V}_2(-f, s) = 0$ for all s, then the game is *determined* and we call $\mathcal{V}_1(f, s)$ the *value of s*.

Optimal, ε-optimal, and winning strategies. Given an objective f and a vector $\boldsymbol{\varepsilon} \geq \mathbf{0}$, a Player-1 strategy π_1 is *Player-1 $\boldsymbol{\varepsilon}$-optimal from a state s* if $\mathcal{V}_1(f, s, \pi_1) \geq \mathcal{V}_1(f, s) - \boldsymbol{\varepsilon}$. If π_1 is $\mathbf{0}$-optimal from s, then we call π_1 *optimal from s*. Optimality for Player-2 strategies is defined analogously. If $f : \Omega \to V$ is a qualitative objective, a strategy π_1 is *winning for Player 1* from s if $\mathcal{V}_1(f, s, \pi_1) = \max(V)$.

We now define various objectives.

Parity objectives. A *parity objective* consists of a *priority function* $p : S \to \{0, 1, \ldots, k\}$ that maps every state in S to a number (called *priority*) between 0 and k. We denote by $|p|$ the maximal priority (i.e., $|p| = k$). The objective function P of Player 1 maps a play ρ to 1 if the smallest priority visited infinitely often is even, otherwise ρ is mapped to 0.

Lexicographic mean-payoff objectives. A *lexicographic mean-payoff objective* consists of a *reward function* $\boldsymbol{r} : E \to \mathbb{N}^d$ that maps every edge in G to a d-vector (called *reward*) of natural numbers. We define $|\boldsymbol{r}| = \prod_{1 \leq i \leq d} \max_{e \in E} r_i(e)$, where $r_i(e)$ is the i-component of $\boldsymbol{r}(e)$. The objective function of Player 1 for a play ρ is the lexicographic mean payoff $LM_{\boldsymbol{r}}(\rho) = \liminf_{n \to \infty} \frac{1}{n} \sum_{i=0}^{n} \boldsymbol{r}(\langle \rho_i, \rho_{i+1} \rangle)$. If $d = 1$, then $LM_{\boldsymbol{r}}(\rho)$ is the *mean payoff* [17] and we refer to it as $M_{\boldsymbol{r}}(\rho)$.

Lexicographic mean-payoff parity objectives. A *lexicographic mean-payoff parity objective* has a priority function $p : S \to \{0, 1, \ldots, k\}$ and a reward function $\boldsymbol{r} : E \to \mathbb{N}^d$. The *lexicographic mean-payoff parity value* $LMP_{\boldsymbol{r}}(\rho)$ for Player 1 of a play ρ is the lexicographic mean-payoff $LM_{\boldsymbol{r}}(\rho)$ if ρ is winning for the parity objective (i.e., $P_p(\rho) = 1$), else the payoff is $-\mathbf{1}$. If $d = 1$, then $LMP_{\boldsymbol{r},p}(\rho)$ defines the *mean-payoff parity value* [9] and we write $MP_{\boldsymbol{r},p}(\rho)$. If p or $\boldsymbol{r}$ are clear from the context, we omit them.

Games and automata. A *game* is a tuple $\mathcal{G} = \langle G, f \rangle$ consisting of a game graph $G = \langle S, s_0, E \rangle$ and an objective f. An *automaton* is a game with only one player, i.e., $S = S_1$. We name games and automata after their objectives.

3.2 Lexicographic mean-payoff games

In this section, we prove that memoryless strategies are sufficient for lexicographic mean-payoff games and we present an algorithm to decide these games by a reduction to simple mean-payoff games. We first present the solution of lexicographic mean-payoff games with a reward function with two components, and then extend it to d-dimensional reward functions. Consider a lexicographic mean-payoff game $\mathcal{G}_{LM} = \langle \langle S, s_0, E \rangle, \boldsymbol{r} \rangle$, where $\boldsymbol{r} = \langle r_1, r_2 \rangle$ consists of two reward functions.

Memoryless strategies suffice. We show that memoryless strategies suffice by a reduction to a finite cycle forming game. Let us assume we have solved the mean-payoff game with respect to the reward function r_1. Consider a *value class* of r_1, i.e., a set of states having the same value with respect to r_1. It is not possible for Player 1 to move to a higher value class, and Player 1 will never choose an edge to a lower value class. Similarly, Player 2 does not have edges to a lower value class and will never choose edges to a higher value class. Thus, we can consider the sub-game for a value class.

Consider a value class of value ℓ and the sub-game induced by the value class. We now play the following finite-cycle forming game: Player 1 and Player 2 choose edges until a cycle C is formed. The payoff for the game is as follows: (1) If the mean-payoff value of the cycle C for r_1 is greater than ℓ, then Player 1 gets reward $|r_2| + 1$. (2) If the mean-payoff value of the cycle C for r_1 is smaller than ℓ, then Player 1 gets reward -1. (3) If the mean-payoff value of the cycle C for r_1 is exactly ℓ, then Player 1 gets the mean-payoff value for reward r_2 of the cycle C.

Lemma 1. The value of Player 1 for any state in the finite-cycle forming game is (*i*) strictly greater than -1 and (*ii*) strictly less than $|r_2| + 1$.

Lemma 2. Both players have memoryless optimal strategy in the finite-cycle forming game.

Proof. The result can be obtained from the result of Björklund et al. [3]. From Theorem 5.1 and the comment in Section 7.2 it follows that in any finite-cycle forming game in which the outcome depends only on the vertices that appear in the cycle (modulo cyclic permutations) we have that memoryless optimal strategies exist for both players. Our finite-cycle forming game satisfies the conditions. □

Lemma 3. The following assertions hold.

1. If the value of the finite-cycle forming game is β at a state s, then the value of the lexicographic mean-payoff game is $\langle \ell, \beta \rangle$ at s.
2. A memoryless optimal strategy of the finite-cycle forming game is optimal for the lexicographic mean-payoff game.

The proof can be found in [4].

Reduction to mean-payoff games. We now sketch a reduction of lexicographic mean-payoff games to mean-payoff games for optimal strategies. We reduce the reward function $\boldsymbol{r} = \langle r_1, r_2 \rangle$ to a single reward function r^*. We ensure that if the mean-payoff difference of two cycles C_1 and C_2 for reward r_1 is positive, then the difference in reward assigned by r^* exceeds the largest possible difference in the mean-payoff for reward r_2. Consider two cycles C_1 of length n_1 and C_2 of length n_2, such that the sum of the r_1 rewards of C_i is α_i. Since all rewards are integral, $|\frac{\alpha_1}{n_1} - \frac{\alpha_2}{n_2}| > 0$ implies $|\frac{\alpha_1}{n_1} - \frac{\alpha_2}{n_2}| \geq \frac{1}{n_1 \cdot n_2}$. Hence we multiply the r_1 rewards by $m = |S|^2 \cdot |r_2| + 1$ to obtain $r^* = m \cdot r_1 + r_2$. This ensures that if the mean-payoff difference of two cycles C_1 and C_2 for reward r_1 is positive, then the difference exceeds the difference in the mean-payoff for reward r_2. Observe that we restrict our attention to cycles only since we have already proven that optimal memoryless strategies exist.

We can easily extend this reduction to reduce lexicographic mean-payoff games with arbitrarily many reward functions to mean-payoff games. The following theorem follows from this reduction in combination with known results for mean payoff parity games [17, 21].

Theorem 1 (Lexicographic mean-payoff games). *For all lexicographic mean-payoff games $\mathcal{G}_{LM} = \langle \langle S, s_0, E \rangle, \boldsymbol{r} \rangle$, the following assertions hold.*

1. (Determinacy.) *For all states $s \in S$, $\mathcal{V}_1(LMP, s) + \mathcal{V}_2(-LMP, s) = \mathbf{0}$.*
2. (Memoryless optimal strategies.) *Both players have memoryless optimal strategies from every state $s \in S$.*
3. (Complexity). *Whether the lexicographic mean-payoff value vector at a state $s \in S$ is at least a rational value vector $\boldsymbol{v}$ can be decided in NP $\cap$ coNP.*
4. (Algorithms). *The lexicographic mean-payoff value vector for all states can be computed in time $O(|S|^{2d+3} \cdot |E| \cdot |\boldsymbol{r}|)$.*

3.3 Lexicographic Mean-Payoff Parity Games

Lexicographic mean-payoff parity games are a natural lexicographic extension of mean-payoff parity games [9]. The algorithmic solution for mean-payoff parity games is a recursive algorithm, where each recursive step requires the solution of a parity objective and a mean-payoff objective. The key correctness argument of the algorithm relies on the existence of memoryless optimal strategies for parity and mean-payoff objectives. Since memoryless optimal strategies exist for lexicographic mean-payoff games, the solution of mean-payoff parity games extends to lexicographic mean-payoff parity games: in each recursive step, we replace the mean-payoff objective by a lexicographic mean-payoff objective. Thus, we have the following result.

Theorem 2 (Lexicographic mean-payoff parity games). *For all lexicographic mean-payoff parity games $\mathcal{G}_{LMP} = \langle \langle S, s_0, E \rangle, \boldsymbol{r}, p \rangle$, the following assertions hold.*

1. (Determinacy). *$\mathcal{V}_1(LMP, s) + \mathcal{V}_2(-LMP, s) = \mathbf{0}$ for all state $s \in S$.*
2. (Optimal strategies). *Optimal strategies for Player 1 exist but may require infinite memory; finite-memory optimal strategies exist for Player 2.*

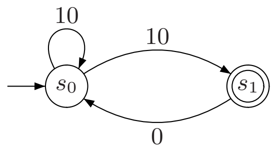

Fig. 4. Game in which the optimal strategy requires infinite memory.

3. (Complexity). *Whether the value at a state $s \in S$ is at least the vector $\boldsymbol{v}$ can be decided in coNP.*
4. (Algorithms). *The value for all states can be computed in time $O\big(|S|^{|p|} \cdot (\min\{|S|^{\frac{|p|}{3}} \cdot |E|, |S|^{O(\sqrt{S})}\} + |S|^{2d+3} \cdot |E| \cdot |\boldsymbol{r}|)\big)$.*

In the following, we prove two properties of mean-payoff parity games that are interesting for synthesis. For simplicity, we present the results for mean-payoff parity games. The results extend to lexicographic mean-payoff parity games as in Theorem 2. First, we show that the algorithm of [9] can be adapted to compute finite-memory strategies that are ε-optimal. Then, we show that Player 1 has a finite-memory optimal strategy if and only if she has a memoryless optimal strategy.

Finite-memory ε-optimal strategy. In mean-payoff parity games, though optimal strategies require infinite memory for Player 1, there is a finite-memory ε-optimal strategy for every $\varepsilon > 0$. The proof of this claim is obtained by a more detailed analysis of the optimal strategy construction of [9]. The optimal strategy constructed in [9] for Player 1 can be intuitively described as follows. The strategy is played in rounds, and each round has three phases: (a) playing a memoryless optimal mean-payoff strategy; (b) playing a strategy in a subgame; (c) playing a memoryless attractor strategy to reach a desired priority. Then the strategy proceeds to the next round. The length of the first phase is monotonically increasing in the number of rounds, and it requires infinite memory to count the rounds. Given an $\varepsilon > 0$, we can fix a bound on the number of steps in the first phase that ensures a payoff within ε of the optimal value. Hence, a finite-memory strategy can be obtained.

We illustrate the idea with an example. Consider the example shown in Fig. 4 where we have a game graph where all states belong to Player 1. The goal of Player 1 is to maximize the mean-payoff while ensuring that state s_1 is visited infinitely often. An optimal strategy is as follows: the game starts in round 1. In each round i, the edge $s_0 \rightarrow s_0$ is chosen i times, then the edge $s_0 \rightarrow s_1$ is chosen once, and then the game proceeds to round $i + 1$. Any optimal strategy in the game shown requires infinite memory. However, given $\varepsilon > 0$, in every round the edge $s_0 \rightarrow s_0$ can be chosen a fixed number K times such that $K > \frac{10}{\varepsilon} - 2$. Then the payoff is $\frac{10 \cdot K + 10}{K+2} = 10 - \frac{10}{K+2} \geq 10 - \varepsilon$ (since $K > \frac{10}{\varepsilon} - 2$); which is within ε of the value. It may also be noted that given $\varepsilon > 0$, the finite-memory optimal strategy can be obtained as follows. We apply the recursive algorithm to solve the game to obtain two memoryless strategies: one for the mean-payoff strategy and other for the attractor strategy. We then specify the bound (depending on

ε) on the number of steps for the mean-payoff strategy for each phase (this requires an additional $O(\frac{1}{\varepsilon})$ time for the strategy description after the recursive algorithm).

Theorem 3. *For all lexicographic mean-payoff parity games and for all $\varepsilon > 0$, there exists a finite-memory ε-optimal strategy for Player 1. Given $\varepsilon > 0$, a finite-memory ε-optimal strategy can be constructed in time $O(|S|^{|p|} \cdot |E|^{2d+6} \cdot |\boldsymbol{r}| + \frac{1}{\varepsilon})$.*

Optimal finite-memory and memoryless strategies. Consider a mean-payoff parity game $\mathcal{G} = \langle\langle S, s_0, E\rangle, r, p\rangle$. Our goal is to show that if there is a finite-memory optimal strategy for Player 1, then there is a memoryless optimal strategy for Player 1. Suppose there is a finite-memory optimal strategy $\widehat{\pi}_1$ for Player 1. Consider the finite graph $\widehat{\mathcal{G}}$ obtained by fixing the strategy $\widehat{\pi}_1$. ($\widehat{\mathcal{G}}$ is obtained as the synchronous product of the given game graph and finite-state strategy automaton for $\widehat{\pi}_1$.) For a state $s \in S$, consider any cycle $\widehat{C}$ in $\widehat{\mathcal{G}}$ that is reachable from $\langle s, q_0\rangle$ (where q_0 is the initial memory location) and $\widehat{C}$ is executed to ensure that Player 1 does not achieve a payoff greater than the value of the game from s. We denote by $\widehat{C}|_{\mathcal{G}}$ the sequence of states in $\mathcal{G}$ that appear in $\widehat{C}$. We call a cycle C of $\mathcal{G}$ that appears in $\widehat{C}|_{\mathcal{G}}$ a *component cycle of* $\widehat{C}$. We have the following properties about the cycle $\widehat{C}$ and its component cycles.

1. $\min(p(\widehat{C}|_{\mathcal{G}}))$ is even.
2. Suppose there is a component cycle C of $\widehat{C}$ such that the average of the rewards of C is greater than the average of the rewards of $\widehat{C}$. If Player 2 fixes a finite-memory strategy that corresponds to the execution of cycle $\widehat{C}$, then an infinite-memory strategy can be played by Player 1 that pumps the cycle C longer and longer to ensure a payoff that is equal to the average of the weights of C. The infinite memory strategy ensures that all states in $\widehat{C}|_{\mathcal{G}}$ are visited infinitely often, but the long-run average of the rewards is the average of the rewards of C. This would imply that for the cycle $\widehat{C}$, Player 1 can switch to an infinite-memory strategy and ensure a better payoff.
3. If there is component cycle C of $\widehat{C}$ such that $\min(p(C)) > \min(p(\widehat{C}|_{\mathcal{G}}))$, then the cycle segment of C can be ignored from $\widehat{C}$ without affecting the payoff.
4. Suppose we have two component cycles C_1 and C_2 in $\widehat{C}$ such that $\min(p(C_1)) = \min(p(C_2)) = \min(p(\widehat{C}|_{\mathcal{G}}))$, then one of the cycles can be ignored without affecting the payoff.

It follows from above that if the finite-memory strategy $\widehat{\pi}_1$ is an optimal one, then it can be reduced to a strategy π_1' such that if Player 2 fixes a finite-memory counter-optimal strategy π_2, then every cycle C in the finite graph obtained from fixing π_1' and π_2 is also a cycle in the original game graph. Since finite-memory optimal strategies exist for Player 2, a correspondence of the value of the game and the value of the following finite-cycle forming game can be established. The finite-cycle forming game is played on $\mathcal{G}$ and the game stops when a cycle C is formed, and the payoff is as follows: if $\min(p(C))$ is even, then the payoff for Player 1 is the average of the weights of the C, otherwise the payoff for Player 1 is $\mathbf{-1}$. The existence of pure memoryless optimal strategy in the finite-cycle forming game can be obtained from the results of Björklund et al. [3]. This concludes the proof of the following theorem.

Theorem 4. *For all lexicographic mean-payoff parity games, if Player 1 has a finite-memory optimal strategy, then she has a memoryless optimal strategy.*

It follows from Theorem 4 that the decision whether there is a finite-memory optimal strategy for Player 1 is in NP. The NP procedure goes as follows: we guess the value v_0 of state s_0 and verify that the value at s_0 is no more than v_0. We can decide in coNP whether the value at a state is at least v, for $v \in \mathbb{Q}$. Thus, we can decide in NP whether the value at state s_0 is no more than v_0 (as it is the complement). Then, we guess a memoryless optimal strategy for Player 1 and verify (in polynomial time) that the value is at least v_0 given the strategy.

4 Quantitative Verification and Synthesis

We are interested in the verification and the synthesis problem for quantitative specifications given by a lexicographic mean-payoff (parity) automaton. In the following simple lemma we establish that these automata also suffice to express qualitative properties.

Lemma 4. *Let $A = \langle G, p\rangle$ be a deterministic parity automaton and let $A' = \langle G', \boldsymbol{r}\rangle$ be a lexicographic mean-payoff automaton. We can construct a lexicographic mean-payoff parity automaton $A \times A' = \langle G \times G', \boldsymbol{r}, p\rangle$, where $G \times G'$ is the product graph of G and G' such that for any word w and associated run ρ, $LMP_{A\times A'}(\rho) = -\mathbf{1}$ if the run of w is lost in A, and $LM_{A'}(\rho')$ otherwise, where ρ' is the projection of ρ on G'.*

Note that $\langle L_A, L_{A'}\rangle = L_{A\times A'} + \mathbf{1}$, assuming that $\inf_{w\in\Sigma^\omega} L_{A'}(w) = 0$. If A is a safety automaton, the language $\langle L_A, L_{A'}\rangle$ can be presented by a lexicographic mean-payoff automaton (see Example 4). Thus, lexicographic mean-payoff automata suffice to express both a quantitative aspect and a safety aspect of a specification. Lexicographic mean-payoff parity automata can be used to introduce a quantitative aspect to liveness specifications and thus to the usual linear-time temporal logics.

Example 4. Let us resume Example 1. Fig. 5 shows a safety automaton B for the specification $\mathsf{G}(r \rightarrow g \vee \mathsf{X}\, g)$. It also shows the mean-payoff automaton C for $\langle L_B, L_{A_2}\rangle$. (See Fig. 2 for the definition of A_2.)

4.1 Quantitative Verification

We now consider the verification problem for quantitative specifications. For qualitative specifications, the verification problem is whether an implementation satisfies the specification for all inputs. For quantitative specifications, the problem generalizes to the question if an implementation can achieve a given value independent of the inputs.

Let $A = \langle\langle S, s_0, E\rangle, \boldsymbol{r}, p\rangle$ be a lexicographic mean-payoff parity automaton and let $M = \langle Q, q_0, \delta\rangle$ be a Mealy machine. The *quantitative verification problem* is to determine $L_A(M)$. The corresponding decision problem is whether

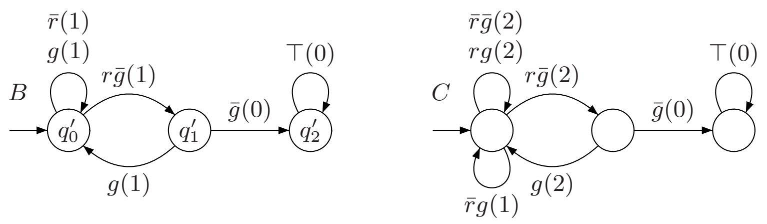

Fig. 5. Safety automaton B for $\mathsf{G}(r \to g \vee \mathsf{X}\, g)$ and automaton C for $\langle L_B, L_{A_2} \rangle$.

$L_A(M) \geq c$ for a given cutoff value c. Clearly, verification of qualitative languages is a special case in which the cutoff value is 1.

Theorem 5. *The value $L_A(M)$ can be computed in time $O(|S| \cdot |Q| \cdot |E| \cdot |\delta| \cdot d \cdot \lg(|Q| \cdot |\delta| \cdot |\boldsymbol{r}|))$.*

Proof. We reduce the lexicographic mean-payoff parity automata to a mean-payoff parity automaton A' using the reduction stated in Section 3.2 and build the product automaton of A' and M. Then, we check if it contains a cycle that is not accepted by the parity algorithm [19]. If so, we return $\mathbf{-1}$. If not, in the second step we find the minimal mean-weight cycle [18].

Example 5. In Example 1, we computed the values of Implementations M_1, M_2, and M_3 (Fig. 1) for the specifications A_1 and A_2 given in Fig. 2. Specification A_1 requires the number of grants to be minimal. Under this specification, M_3 is preferable to both other implementations because it only produces half as much grants in the worst case. Unfortunately, A_1 treats a grant the same way regardless of whether a request occurred. Thus, this specification does not distinguish between M_1 and M_2. Specification A_2 only punishes "unnecessary" grants, which means that A_2 prefers M_2 and M_3 to M_1.

A preference between the eagerness of M_2 and the laziness of M_3 can be resolved in either direction. For instance, if we combine the two quantitative languages using addition, lazy implementations are preferred.

4.2 Quantitative Synthesis

In this section, we show how to automatically construct an implementation from a quantitative specification given by a lexicographic mean-payoff (parity) automaton. First, we show the connection between automata and games, and between strategies and Mealy machines, so that we can use the theory from Sections 3 to perform synthesis. Then, we define different notions of synthesis and give their complexity bounds.

We will show the polynomial conversions of an automata to a game and of a strategy to a Mealy machines using an example.

Example 6. Fig. 6(left) shows the game $\mathcal{G}$ corresponding to the automaton C shown in Fig. 5. Note: The alphabet 2^{AP} has been split into an input alphabet 2^{I} controlled by Player 2 (squares) and an output alphabet 2^{O} controlled by Player 1

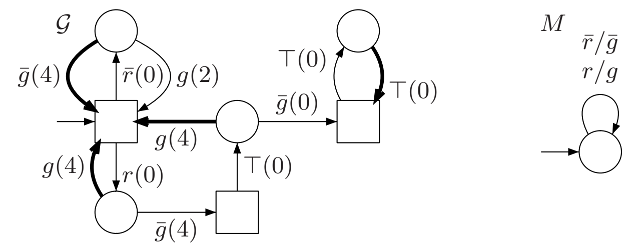

Fig. 6. A game (optimal strategy shown in bold) and corresponding Mealy machine

(circles). Accordingly, each edge e of C is split into two edges e_2 and e_1; the reward of e_2 is zero and the reward of e_2 is double the reward of e. It should be clear that with the appropriate mapping between runs, the payoff remains the same. Because we want a Mealy machine, the input player makes the first move.

The figure also shows an optimal strategy (bold edges) for $\mathcal{G}$ with payoff 2. The right side of the figure shows the Mealy machine M corresponding to the strategy. It is constructed by a straightforward collection of inputs and chosen outputs. It is easily verified that $L_C(M) = 2$.

Definition 1. *Let L be a quantitative language and let $\boldsymbol{c} \in \mathbb{R}^d$ be a cutoff value. We say that L is* $\boldsymbol{c}$-realizable *if there is a Mealy machine M such that $L(M) \geq \boldsymbol{c}$. We say that L is* limit-$\boldsymbol{c}$-realizable *if for all $\boldsymbol{\varepsilon} > 0$ there is a Mealy machine M such that $L(M) + \boldsymbol{\varepsilon} \geq \boldsymbol{c}$.*

Suppose the supremum of $L(M)$ over all Mealy machines M exists, and denote it by $\boldsymbol{c}^*$. We call L realizable (limit-realizable) *if L is $\boldsymbol{c}^*$-realizable (limit-$\boldsymbol{c}^*$-realizable). A Mealy machine M with value $L(M) \geq \boldsymbol{c}^*$ ($L(M) + \boldsymbol{\varepsilon} \geq \boldsymbol{c}^*$) is called* optimal *(*$\boldsymbol{\varepsilon}$-optimal*, resp.).*

Clearly, realizability implies limit-realizability. Note that by the definition of supremum, L is limit-$\boldsymbol{c}^*$-realizable iff $\boldsymbol{c}^*$ is defined. Note also that realizability for qualitative languages corresponds to realizability with cutoff 1. Synthesis is the process of constructing an optimal ($\boldsymbol{\varepsilon}$-optimal) Mealy machine. Note that for a cutoff value $\boldsymbol{c}$, if L is $\boldsymbol{c}$-realizable, then we have that $L(M) \geq \boldsymbol{c}$ for any optimal Mealy machine M. If L is limit-$\boldsymbol{c}$-realizable, then $L(M_\varepsilon) + \boldsymbol{\varepsilon} \geq \boldsymbol{c}$ holds for any $\boldsymbol{\varepsilon}$-optimal Mealy machine M_ε.

Example 7. We have already seen an example of a realizable specification expressed as a mean-payoff automaton (See Figs. 2 and 5 and Example 4.) Example 3 shows a language that is only limit-realizable.

For the combination of safety and quantitative specifications, we have Theorem 6.

Theorem 6. *Let $A = \langle\langle S, s_0, E\rangle, \boldsymbol{r}\rangle$ be a lexicographic mean-payoff automaton of dimension d, and let $\boldsymbol{c}$ be a cutoff value. The following assertions hold.*

1. *L_A is realizable (hence limit-realizable); L_A is $\boldsymbol{c}$-realizable iff L_A is limit-$\boldsymbol{c}$-realizable.*
2. *$\boldsymbol{c}$-realizability (and by (1) limit-$\boldsymbol{c}$-realizability) of L_A are decidable in $NP \cap coNP$.*

3. *An optimal Mealy machine can be constructed in time $O(|E|^{4d+6} \cdot |\boldsymbol{r}|)$.*

The first results follow from the existence of memoryless optimal strategies for lexicographic mean-payoff games. The second and third results follows from the complexity and algorithms of solving these games. (See Theorem 1.) For liveness, we have the following result.

Theorem 7. *Let $A = \langle\langle S, s_0, E\rangle, \boldsymbol{r}, p\rangle$ be a lexicographic mean-payoff parity automaton of dimension d and let $\boldsymbol{c}$ be a cutoff value. The following assertions hold.*

1. *L_A is limit-realizable, but it may not be realizable; limit-$\boldsymbol{c}$-realizability of L_A does not imply $\boldsymbol{c}$-realizability.*
2. *Realizability and $\boldsymbol{c}$-realizability of L_A are decidable in NP, and limit-$\boldsymbol{c}$-realizability of L_A is decidable in coNP.*
3. *For $\boldsymbol{\varepsilon} > 0$, an $\boldsymbol{\varepsilon}$-optimal Mealy machine can be constructed in time $O(|S|^{|p|} \cdot |E|^{4d+6} \cdot |\boldsymbol{r}| + \frac{1}{\varepsilon})$. If L_A is realizable, then an optimal Mealy machine can be constructed in time $O(|S|^{|p|} \cdot |E|^{4d+6} \cdot |\boldsymbol{r}|)$.*

Explanation: Following Theorem 4, realizability and $\boldsymbol{c}$-realizability can be computed in NP. We have that L_A is limit-$\boldsymbol{c}$-realizable iff $\boldsymbol{c}$ is not higher than the value of the initial state, which can be decided in coNP. (Theorem 2.) Limit-realizability follows from Theorem 3.

Example 8. In Example 3 we discussed the specification $\varphi = \mathsf{G}(r \to \mathsf{F}\, g)$. In combination with the quantitative language given by A_3 in Fig. 3, this specification is optimally realizable by a finite implementation: implementations M_1 and M_2 from Fig. 1 are two examples. The combination of φ and the quantitative language given by A_1 in Fig. 2 only yields a specification that is optimally limit-realizable. Automaton A_1 prefers as few as possible requests. An implementation that is optimal within $1/k$ could simply give a request every k cycles. It may not be useful to require that something happens as infrequently as possible in the context of liveness specifications. Instead, more subtle approaches are necessary; in this case we could require that *unnecessary* grants occur as little as possible. (Cf. A_2 in Fig. 2.)

5 Conclusions and Future Work

We introduced a measure for the "goodness" of an implementation by adding quantitative objectives to a qualitative specification. Our quantitative objectives are mean-payoff objectives, which are combined lexicographically. Mean-payoff objectives are relatively standard and, as we demonstrated, sufficiently expressive for our purposes. Other choices, such as discounted objectives [12], are possible as well. These give rise to different expressive powers for specification languages [8].

Finally, we have taken the worst-case view that the quantitative value of an implementation is the worst reward of all runs that the implementation may produce. There are several alternatives. For instance, one could take the average-case view of assigning to an implementation some expected value of the cost taken over all possible runs, perhaps relative to a given input distribution. Another option may be to compute admissible strategies. It can be shown that such strategies do not exist for all mean-payoff games, but they may exist for an interesting subset of these games.

References

1. B. Alpern and F. B. Schneider. Defining liveness. *Information Processing Letters*, 1985.
2. Rajeev Alur, Aldric Degorre, Oded Maler, and Gera Weiss. On omega-languages defined by mean-payoff conditions. In *FOSSACS*, pages 333–347, 2009.
3. H. Björklund, S. Sandberg, and S. Vorobyov. Memoryless determinacy of parity and mean payoff games: a simple proof. *Theor. Comput. Sci.*, 2004.
4. R. Bloem, K. Chatterjee, T. A. Henzinger, and B. Jobstmannn. Better quality in synthesis through quantitative objectives. In *CoRR, abs/0904.2638*, 2009.
5. A. Chakrabarti, K. Chatterjee, T. A. Henzinger, O. Kupferman, and R. Majumdar. Verifying quantitative properties using bound functions. In *CHARME*, LNCS 3725, pages 50–64. Springer, 2005.
6. A. Chakrabarti, L. de Alfaro, T. A. Henzinger, and M. Stoelinga. Resource interfaces. In *EMSOFT*, LNCS 2855, pages 117–133. Springer, 2003.
7. K. Chatterjee, L. de Alfaro, M. Faella, T.A. Henzinger, R. Majumdar, and M. Stoelinga. Compositional quantitative reasoning. In *QEST*, pages 179–188. IEEE Computer Society Press, 2006.
8. K. Chatterjee, L. Doyen, and T. A. Henzinger. Quantitative languages. In *Proc. 22nd international workshop on Computer Science Logic (CSL'08)*, 2008.
9. K. Chatterjee, T. A. Henzinger, and M. Jurdzinski. Mean-payoff parity games. In *Annual Symposium on Logic in Computer Science (LICS)*, 2005.
10. K. Chatterjee, T.A. Henzinger, and M. Jurdziński. Games with secure equilibria. In *LICS'04*, pages 160–169. IEEE, 2004.
11. L. de Alfaro. How to specify and verify the long-run average behavior of probabilistic systems. In *LICS 98*, pages 454–465. IEEE Computer Society Press, 1998.
12. L. de Alfaro, T. A. Henzinger, and R. Majumdar. Discounting the future in systems theory. In *ICALP'03*, 2003.
13. L. de Alfaro, R. Majumdar, V. Raman, and M. Stoelinga. Game relations and metrics. In *LICS*, pages 99–108. IEEE Computer Society Press, 2007.
14. J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. Metrics for labelled Markov systems. In *CONCUR*, vol. 1664 of LNCS, pages 258–273. Springer, 1999.
15. M. Droste and P. Gastin. Weighted automata and weighted logics. *Theoretical Computer Science*, 380:69–86, 2007.
16. M. Droste, W. Kuich, and G. Rahonis. Multi-valued MSO logics over words and trees. *Fundamenta Informaticae*, 84:305–327, 2008.
17. A. Ehrenfeucht and J. Mycielski. Positional strategies for mean payoff games. *International Journal of Game Theory*, 1979.
18. R. M. Karp. A characterization of the minimum cycle mean of a digraph. *Discrete Mathematics*, 1978.
19. V. King, O. Kupferman, and M. Y. Vardi. On the complexity of parity word automata. In *Foundations of Software Science and Computation Structures*, 2001.
20. O. Kupferman and Y. Lustig. Lattice automata. In *VMCAI*, LNCS 4349, pages 199–213. Springer, 2007.
21. U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 1996.