Synthesizing Systems with Optimal Average-Case Behavior for Ratio Objectives

Christian von Essen
VERIMAG
Grenoble, France
EDMSTII
Universit Joseph Fourier, Grenoble, France
christian.vonessen@imag.fr

Barbara Jobstmann
CNRS/VERIMAG
Grenoble, France
barbara.jobstmann@imag.fr

We show how to automatically construct a system that satisfies a given logical specification and has an optimal average behavior with respect to a specification with ratio costs.

When synthesizing a system from a logical specification, it is often the case that several different systems satisfy the specification. In this case, it is usually not easy for the user to state formally which system she prefers. Prior work proposed to rank the correct systems by adding a quantitative aspect to the specification. A desired preference relation can be expressed with (i) a quantitative language, which is a function assigning a value to every possible behavior of a system, and (ii) an environment model defining the desired optimization criteria of the system, e.g., worst-case or average-case optimal.

In this paper, we show how to synthesize a system that is optimal for (i) a quantitative language given by an automaton with a ratio cost function, and (ii) an environment model given by a labeled Markov decision process. The objective of the system is to minimize the expected (ratio) costs. The solution is based on a reduction to Markov Decision Processes with ratio cost functions which do not require that the costs in the denominator are strictly positive. We find an optimal strategy for these using a fractional linear program.

1 Introduction

Quantitative analysis techniques are usually used to measure quantitative properties of systems, such as timing, performance, or reliability (cf. [7, 26, 8]). We use quantitative reasoning in the classically Boolean contexts of verification and synthesis because they allow us to distinguish systems with respect to "soft constraints" like robustness [11] or default behavior [10]. This is particularly helpful in synthesis, where a system is automatically derived from a specification, because quantitative specifications allow us to guide the synthesis tool towards a desired implementation.

In this paper we show how quantitative specifications based on ratio objectives can be used to guide the synthesis process. In particular, we present a technique to synthesize a system with an average-case behavior that satisfies a logical specification and optimizes a quantitative objective given by a ratio objective.

The synthesis problem can be seen as a game between two players: the system and the environment (the context in which the system operates). The system has a fixed set of interface variables with a finite domain to interact with its environment. The variables are partitioned into a set of input and output variables. The environment can modify the set of input variables. For instance, an input variable can indicate the arrival of some packet on a router on a given port or the request of a client to use a shared resource. Each assignment to the input variables is a possible move of the environment in the synthesis game. The system reacts to the behavior of the environment by changing the value of the

Johannes Reich and Bernd Finkbeiner (Eds): International Workshop on Interactions, Games and Protocols (iWIGP)
EPTCS 50, 2011, pp. 17–32, doi:10.4204/EPTCS.50.2

output variables. An assignment to the output variables is called an action of the system and describes a possible move of the system in the synthesis game. E.g., the system can grant a shared resource to Client C by setting a corresponding output variable. Environment and system change their variables in turns. In every step, first the system makes modification to the output variables, then the environment changes the input variables. The sequence of variable evaluations built up by this interplay is evaluated with respect to a specification. A logical (or qualitative) specification maps every sequence to 1 or 0, indicating whether the sequence satisfies the specification or not. For example, a sequence of evaluations in which the system grants a shared resource to two clients at the same time is mapped to 0 if the specification requires mutual exclusive access to this resource. The aim of the system in the synthesis game is to satisfy the specification independent of the choices of the environment. There might be several systems that can achieve this goal for a given specification. Therefore, Bloem et al. [10] proposed to add a quantitative specification in order to rank the correct systems. A quantitative specification maps every infinite sequence of variable evaluations to a value indicating how desirable this behavior is. In this paper, we study quantitative specifications resulting from ratio objectives. The idea is that a behavior of the system is mapped to two infinite sequences of values. The first sequence refers to events that were "good" for the system, while the second sequence refers to "bad" events within a behavior. For instance, consider a server processing requests from several clients. If the server receives a request it can be seen as a bad event, since it requires the server to process the request. On the other hand, every handled request is clearly a good event. Intuitively, the ratio objectives computes the long-run ratio between the sum of bad and the sum of good events. This ratio is the value of a behavior. A system can be seen as a set of behaviors. We can assign a value to a system by taking, e.g., the worst or the average value over all its behaviors. Given a way to evaluate a system, we can ask for a system that optimizes this value, i.e., a system that achieves a better value than any other system. Taking the worst value over the possible behaviors corresponds to assuming that the system is in an adversary environment. The average value is computed with respect to a probabilistic model of the environment [15]. In the average-case synthesis game, the environment player is replaced by a probabilistic player that is playing according to the probabilistic environment model.

In this paper, we present the first average-case synthesis algorithm for specifications that evaluate a behavior of the system with respect to the ratio of two cost functions [10]. This ratio objective allows us, e.g., to ask for a system that optimizes the ratio between requests and acknowledgments in a server-client system. For the average-case analysis, we present a new environment model, which is based on Markov decision processes and generalizes the one in [15]. We solve the average-case synthesis problem with ratio objective by reduction to Markov decision processes with ratio cost functions. For unichain Markov Decision Processes with ratio cost functions, we present a solution based on linear programming.

Related Work. Researchers have considered a number of formalisms for quantitative specifications [5, 12, 13, 14, 2, 3, 20, 22, 28] but most of them (except for [11]) do not consider long-run ratio objectives. In [11], the environment is assumed to be adversary, while we assume a probabilistic environment model. Regarding the environment model, there have been several notions of metrics for probabilistic systems and games proposed in the literature [4, 19]. The metrics measure the distance of two systems with respect to all temporal properties expressible in a logic, whereas we (like [15]) uses the quantitative specification to compare systems wrt the property of interest. In contrast to [15], we use ratio objectives and a more general environment model. Our environment model is the same as the one used for control and synthesis in the presence of uncertainty (cf. [6, 16, 9]). However, in this context usually only qualitative specifications are considered. MDPs with long-run average objectives are well studied.

The books [23, 30] present a detailed analysis of this topic. Cyrus Derman [18] studied MDPs with a fractional objective. This work differs in two aspects from ours: first, Derman requires that the payoff of the cost function of the denominator is always strictly positive and second, the objective function used in [18] is already given in terms of the expected cost of the first cost function to the expected cost of the second cost functions and not in terms of a single trace. De Alfaro [1] studies a model that is similar to ours but does not consider the synthesis problem. Finally, we would like to note that the two choices we have in a quantitative synthesis problem, namely the choice of the quantitative language and the choice of environment model are the same two choices that appear in weighted automata and max-plus algebras (cf. [21, 24, 17]).

2 Preliminaries

Words, Qualitative and Quantitative Languages. Given a finite alphabet Σ, a *word* $w = w_0 w_1 \dots$ is a finite or infinite sequence of elements of Σ. We use w_i to denote the $(i+1)$-th element in the sequence. If w is finite, then $|w|$ denotes the length of w, otherwise $|w|$ is infinity. We denote the empty word by ε, i.e., $|\varepsilon| = 0$. We use Σ^* and Σ^ω to denote the set of finite and infinite words, respectively. Given a finite word $w \in \Sigma^*$ and a finite or infinite word $v \in \Sigma^* \cup \Sigma^\omega$, we write wv for the concatenation of w and v. A *qualitative language* φ is a function $\varphi : \Sigma^\omega \to \mathbb{B}$ mapping every infinite word to 1 or 0. Intuitively, a qualitative language partitions the set of words into a set of good and a set of bad traces. A *quantitative language* [14] ψ is a function $\psi : \Sigma^\omega \to \mathbb{R}^+ \cup \{\infty\}$ associating to each infinite word a value from the extended non-negative reals.

Specifications and automata with cost functions. An *automaton* is a tuple $\mathscr{A} = (\Sigma, Q, q_0, \delta, F)$, where Σ is a finite *alphabet*, Q is a finite set of *states*, $q_0 \in Q$ is an *initial state*, $\delta : Q \times \Sigma \to Q$ is the *transition function*, and $F \subseteq Q$ is a set of *safe states*. We use $\delta^* : S \times L^* \to S$ to denote the closure of δ over finite words. Formally, given a word $w = w_0 \dots w_n \in \Sigma^*$, δ^* is defined inductively as $\delta^*(q, \varepsilon) = q$, and $\delta^*(q, w) = \delta(\delta^*(q, w_0 \dots w_{n-1}), w_n)$. We use $|\mathscr{A}|$ to denote the size of the automaton.

The *run* ρ of $\mathscr{A}$ on an infinite word $w = w_0 w_1 w_2 \dots \in \Sigma^\omega$ is an infinite sequence of states $q_0 q_1 q_2 \dots$ such that q_0 is the initial state of $\mathscr{A}$ and $\forall i \geq 0 : \delta(q_i, w_i) = q_{i+1}$ holds. The run ρ is called *accepting* if for all $i \geq 0$, $q_i \in F$. A word w is accepting if the corresponding run is accepting. The *language of* $\mathscr{A}$, denoted by $\mathscr{L}_\mathscr{A}$, is the qualitative language $\mathscr{L}_\mathscr{A} : \Sigma^\omega \to \mathbb{B}$ mapping all accepting words to 1 and non-accepting words to 0, i.e., $\mathscr{L}_\mathscr{A}$ is the characteristic function of the set of all accepting words of $\mathscr{A}$. We assume without loss of generality that $Q \setminus F$ is closed under δ, i.e., $\forall s \in Q \setminus F, \forall a \in \Sigma : \delta(s, a) \in Q \setminus F$. Note that every automaton can be modified to meet this assumption by (i) adding a new state $q_\perp$ with a self-loop for every letter and (ii) redirecting every transition starting from $Q \setminus F$ to the new state $q_\perp$. The modified automaton accepts the same language as the original automaton.

Given an automaton $\mathscr{A} = (\Sigma, Q, q_0, \delta, F)$, a *cost function* $c : Q \times \Sigma \to \mathbb{N}$ is a function that maps every transition in $\mathscr{A}$ to a non-negative integer. We use automata with cost functions and *objective functions* to define quantitative languages (or properties). Intuitively, the objective function tells us how to summarize the costs along a run. Given an automation $\mathscr{A}$ and two cost functions c_1, c_2, the *ratio objective* [11] computes the ratio between the costs seen along a run of $\mathscr{A}$ on a word $w = w_0 w_1 w_2 \dots \in \Sigma^\omega$:

$$\mathscr{R}(w) := \lim_{m \to \infty} \liminf_{l \to \infty} \frac{\sum_{i=m}^{l} c_1(\delta^*(q_0, w_0 \dots w_i), w_{i+1})}{1 + \sum_{i=m}^{l} c_2(\delta^*(q_0, w_0 \dots w_i), w_{i+1})} \tag{1}$$

The ratio objective is a generalization of the long-run average objective (also known as mean-payoff

objective, cf. [33]). We use $\mathscr{R}^{\mathscr{A}}_{\frac{c_1}{c_2}}$ to denote the quantitative language defined by $\mathscr{A}$, c_1, c_2, and the ratio objective function. If $\mathscr{A}$, c_1, or c_2 are clear from the context, we drop them.

Intuitively, $\mathscr{R}$ computes the long-run ratio between the costs accumulated along a run. The first limit allows us to ignore a finite prefix of the run, which ensures that we only consider the long-run behavior. The 1 in the denominator avoids division by 0, if the accumulated costs are 0 and has no effect if the accumulated costs are infinite. We need the limit inferior here because the sequence of the limit might not converge. Consider the sequence $\rho = q^1r^2q^4r^8q^{16}\dots$, where q^k means that the State q is visited k-times. Assume State q and State r have the following costs: $c_1(q) = 0$, $c_2(q) = 1$, $c_1(r) = 1$ and $c_2(r) = 1$. Then, the value of $\rho_0 \dots \rho_i$ will alternate between 0 and 1 with increasing i and hence the sequence for $i \to \infty$ will not converge. The limit inferior of this sequence is 0.

Finite-state system and Correctness A *finite-state system* $\mathscr{S} = (S, L, s_0, A, \delta, \tau)$ consists of the automaton $\mathscr{A} = (L, S, s_0, \delta, S)$[1], an *output (or action) alphabet A*, and an *output function* $\tau : S \to A$ assigning to each state of the system a letter from the output alphabet. The alphabet of the automaton L is called the *input alphabet* of the system. Given an input word w, the *run of the system* $\mathscr{S}$ *on the word* w is simply the run of $\mathscr{A}$ on the word w. For every word w over the input alphabet, the system produces a word over the joint input/output alphabet. We use $\mathscr{O}_{\mathscr{S}}$ to denote the function mapping input words to the joint input/output word, i.e, given an input word $w = w_0w_1 \dots \in L^\omega$, $\mathscr{O}_{\mathscr{S}}(w)$ is the sequence of tuples $(l_0, a_0)(l_1, a_1) \dots \in (L \times A)^\omega$ such that (i) $l_i = w_i$ for all $i \geq 0$, (ii) $a_0 = \tau(s_0)$, and (iii) for all $i > 0$, $a_i = \tau(\delta^*(s_0, w_0 \dots w_{i-1})))$ holds.

Given a system $\mathscr{S}$ with input alphabet L and output alphabet A, and an automaton $\mathscr{A}$ with alphabet $\Sigma = L \times A$, we say that the system $\mathscr{S}$ *satisfies the specification* $\mathscr{A}$, denoted $\mathscr{S} \models \mathscr{A}$, if for all input words, the joint input/output word produced by the system $\mathscr{S}$ is accepted by the automaton $\mathscr{A}$, i.e., $\forall w \in L^\omega : (\mathscr{L}_{\mathscr{A}} \circ \mathscr{O}_{\mathscr{S}})(w) = 1$, where $\circ$ denotes the function composition operator.

Probability space. We use the standard definitions of probability spaces. A *probability space* is given by a tuple $\mathscr{P} := (\Omega, \mathscr{F}, \mu)$, where Ω is the set of *outcomes or samples*, $\mathscr{F} \subseteq 2^\Omega$ is the σ-algebra defining the set of *measurable events*, and $\mu \in \mathscr{F} \to [0,1]$ is a *probability measure* assigning a probability to each event such that $\mu(\Omega) = 1$ and for each countable set $E_1, E_2, \dots \in \mathscr{F}$ of disjoint events we have $\mu(\bigcup E_i) = \sum \mu(E_i)$. Recall that, since $\mathscr{F}$ is a σ-algebra, it satisfies the following three conditions: (i) $\emptyset \in \mathscr{F}$, (ii) $E \in \mathscr{F}$ implies $\Omega \setminus E \in \mathscr{F}$ for any event E, and (iii) the union of any countable set of events $E_1, E_2, \dots \in \mathscr{F}$ is also in $\mathscr{F}$, i.e., $\bigcup E_i \in \mathscr{F}$. Given a measurable function $f : \mathscr{F} \to \mathbb{R} \cup \{+\infty, -\infty\}$, we use $\mathbb{E}_{\mathscr{P}}[f]$ to denote the expected value of f under μ, i.e.,

$$\mathbb{E}_{\mathscr{P}}[f] = \int_\Omega f \, d\mu \tag{2}$$

If $\mathscr{P}$ is clear from the context we drop the subscript or replace it with the structure that defines $\mathscr{P}$. The integral used here is the Lebesgue Integral, which is commonly used to define the expected value of a random variable. Note that the expected value is always defined if the function f maps only to values in $\mathbb{R}^+ \cup \{\infty\}$.

Markov chains and Markov decision processes (MDP). Let $\mathscr{D}(S) := \{p : S \to [0,1] \mid \sum_{s \in S} p(s) = 1\}$ be the *set of probability distributions* over a set S.

[1] Note that the last element of this tuple is the set of safe states, i.e., every state is safe.

A *Markov decision process* is a tuple $\mathscr{M} = (S, s_0, A, \tilde{A}, p)$, where S is a finite set of *states*, $s_0 \in S$ is an *initial state*, A is the finite set of *actions*, $\tilde{A} : S \to 2^A$ is the *enabled action function* defining for each state s the set of enabled actions in s, and $p : S \times A \to \mathscr{D}(S)$ is the probabilistic *transition function*. For technical convenience we assume that every state has at least one enabled action, i.e., $\forall s \in S : |\tilde{A}(s)| \geq 1$. If $|\tilde{A}(s)| = 1$ for all states $s \in S$, then $\mathscr{M}$ is called a *Markov chain (MC)*. In this case, we omit A and $\tilde{A}$ from the definition of $\mathscr{M}$. Given a Markov chain $\mathscr{M}$, we say that $\mathscr{M}$ is *irreducible* if every state can be reached from any other. We say that it is *unichain* if it has at most one maximal set of states that can reach it other. We call an MDP unichain if every strategy induces a unichain MC.

An *L-labeled Markov decision process* is a tuple $\mathscr{M} = (S, s_0, A, \tilde{A}, p, \lambda)$, where $(S, s_0, A, \tilde{A}, p)$ is a Markov decision process and $\lambda : S \to L$ is a labeling function such that $\mathscr{M}$ is deterministic with respect to λ, i.e, for all states s, s', s'' and every action a such that $s' \neq s''$, $p(s,a)(s') > 0$ and $p(s,a)(s'') > 0$ we have $\lambda(s') \neq \lambda(s'')$. Since we use L-labeled Markov decision process to represent the behavior of the environment, we require that in every state all actions are enabled, i.e., $\forall s \in S : \tilde{A}(s) = A$.

Sample runs and strategies A *(sample) run* ρ of $\mathscr{M}$ is an infinite sequence of tuples $(s_0, a_0)(s_1, a_1) \cdots \in (S \times A)^\omega$ of states and actions such that for all $i \geq 0$, (i) $a_i \in \tilde{A}(s_i)$ and (ii) $p(s_i, a_i)(s_{i+1}) > 0$. We use Ω to denote the set of all runs, and Ω_s for the set of runs starting at state s. A *finite run* of $\mathscr{M}$ is a prefix of some infinite run. To avoid confusion, we use v to refer to a finite run. Given a finite run v, the set $\gamma(v) := \{\rho \in \Omega \mid \exists \rho' \in \Omega : \rho = v\rho'\}$ of all possible infinite extensions of v is called the *cone set* of v. We use the usual extension of $\gamma(\cdot)$ to sets of finite words.

A *strategy* is a function $\pi : (S \times A)^* S \to \mathscr{D}(A)$ that assigns a probability distribution to all finite sequences in $(S \times A)^* S$. A strategy must refer only to enabled actions, i.e., for all sequences $w \in (S \times A)^*$, states $s \in S$, and actions $a \in A$, if $\pi(ws)(a) > 0$, then action a has to be enabled in s, i.e., $a \in \tilde{A}(s)$. A strategy π is *pure* if for all finite sequences $w \in (S \times A)^*$ and for all states $s \in S$, there is an action $a \in A$ such that $\pi(ws)(a) = 1$. A *memoryless* strategy is independent of the history of the run, i.e., for all $w, w' \in (S \times A)^*$ and for all $s \in S$, $\pi(ws) = \pi(w's)$ holds. A memoryless strategy can be represented as function $\pi : S \to \mathscr{D}(A)$. A pure and memoryless function can be represented by a function $\pi : S \to A$ mapping states to actions. An MDP $\mathscr{M} = (S, s_0, A, \tilde{A}, p)$ together with a pure and memoryless strategy $\pi : S \to A$ defines the Markov chain $\mathscr{M}^\pi = (S, s_0, A, \tilde{A}_\pi, p)$, in which only the actions prescribed in the strategy π are enabled, i.e., $\tilde{A}_\pi(s) = \{\pi(s)\}$. Note that every finite-state system $\mathscr{S}$ with input alphabet S and output alphabet A that refers only to enabled actions can be viewed as a strategy for $\mathscr{M}$. Vice-versa, an MDP with a pure and memoryless strategy π defines a finite state system $\mathscr{S}_\pi^{\mathscr{M}}$ with input alphabet S and output alphabet A.

Induced probability space, objective function, and optimal strategies. An MDP $\mathscr{M} = (S, s_0, A, \tilde{A}, p)$ together with a strategy π and a state $s \in S$ induces a probability space $\mathscr{P}^\pi_{\mathscr{M},s} = (\Omega^\pi_{\mathscr{M},s}, \mathscr{F}^\pi_{\mathscr{M},s}, \mu^\pi_{\mathscr{M},s})$ over the cone sets of the runs starting in s. Hence, $\Omega^\pi_{\mathscr{M},s} = S^\omega$. The probability measure of a cone set is the probability that the MDP starts from state s and follows the common prefix under the strategy π. By convention $\mathscr{P}^\pi_{\mathscr{M}} := \mathscr{P}^\pi_{\mathscr{M},s_0}$. If $\mathscr{M}$ is a Markov chain, then π is fixed (since there is only one available action in every state), and we simply write $\mathscr{P}_{\mathscr{M}}$.

An *objective function* of $\mathscr{M}$ is a measurable function $f : (S \times A)^\omega \to \mathbb{R}^+ \cup \{\infty\}$ that maps runs of $\mathscr{M}$ to values in $\mathbb{R}^+ \cup \{\infty\}$. We use $\mathbb{E}^\pi_{\mathscr{M},s}[f]$ to denote the expected value of f wrt the probability space induced by the MDP $\mathscr{M}$, a strategy π, and a state s.

We are interested in a strategy that has the least expected value for a given state. Given an MDP $\mathscr{M}$ and a state s, a strategy π is called *optimal* for objective f and state s if $\mathbb{E}^\pi_{\mathscr{M},s}[f] = \min_{\pi'} \mathbb{E}^{\pi'}_{\mathscr{M},s}[\mathscr{R}]$,

where π' ranges over all possible strategies.

Given an MDP $\mathscr{M} = (S, s_0, A, \tilde{A}, p)$ and two cost function $c_1 : S \times A \to \mathbb{N}$ and $c_2 : S \times A \to \mathbb{N}$, the *ratio payoff value* is the function $\mathscr{R} : (S \times A)^\omega \to \mathbb{R}^+ \cup \{\infty\}$ mapping every run ρ to a value in $\mathbb{R}^+ \cup \{\infty\}$ as follows:

$$\mathscr{R}_{\frac{c_1}{c_2}}(\rho) := \lim_{m\to\infty} \liminf_{l\to\infty} \frac{\sum_{i=m}^{l} c_1(\rho_i)}{1 + \sum_{i=m}^{l} c_2(\rho_i)} \tag{3}$$

We drop the subscript $\frac{c_1}{c_2}$ if c_1 and c_2 are clear from the context.

3 Synthesis with Ratio Objective in Probabilistic Environments

In this section, we first present a variant of the quantitative synthesis problem introduced in [10]. Then, we show how to solve the synthesis problem with safety and ratio specifications in a probabilistic environment described by an MDP.

The quantitative synthesis problem with probabilistic environments asks to construct a finite-state system $\mathscr{S}$ that satisfies a qualitative specification and optimizes a quantitative specification under the given environment. The specifications are qualitative and quantitative languages over letters in $(L \times A)$, where L and A are the input and output alphabet of $\mathscr{S}$, respectively.

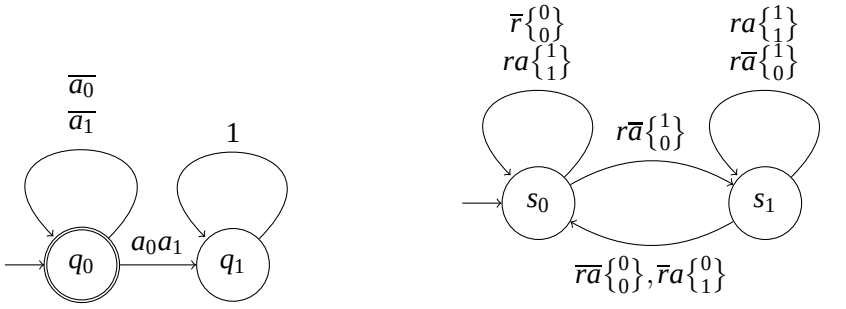
In order to compute the average behavior of a system, we assume a model of the environment. In [15], the environment model is a probability space $\mathscr{P} = (L^\omega, \mathscr{F}, \mu)$ over the input words L^ω of the system defined by a finite L-labeled Markov chain. This model assumes that the behavior of the environment is independent of the behavior of the system, which restricts the modeling possibilities. For instance, a client-server system, in which a client increases the probability of sending a request if it has not been served in the previous step, cannot be modeled using this approach. Therefore, *our environment model* is a function f_e that maps every system $f_s : L^* \to A$ to a probability space $\mathscr{P} = (L^\omega, \mathscr{F}, \mu)$ over the input words L^ω. Note that every finite-state system defines such a system function f_s but not vice versa. To describe a particular environment model f_e, we use a finite L-labeled Markov decision process. Once we have an environment model, we can define what it means for a system to satisfy a specification under a given environment.

Definition 1 (Satisfaction)**.** *Given a finite-state system $\mathscr{S}$ with alphabets L and A, a qualitative specification φ over alphabet $L \times A$, and an environment model f_e, we say that $\mathscr{S}$* satisfies *φ under f_e (written $\mathscr{S} \models_{f_e} \varphi$[2]) iff $\mathscr{S}$ satisfies φ with probability* 1*, i.e.,*

$$\mathbb{E}_{f_e(\mathscr{S})}[\varphi \circ \mathscr{O}_{\mathscr{S}}] = 1.$$

Recall that $\mathscr{O}_{\mathscr{S}}$ denotes the function that maps input words to joint input/output words, and that φ is a qualitative specification, which maps (input/output) words to 0 or 1. Hence, $\varphi \circ \mathscr{O}_{\mathscr{S}}$ denotes the function that maps an input sequence to 1 if the behavior of the system $\mathscr{S}$ for this input word satisfies the specification φ. Otherwise, the input word is mapped to 0. The function $\mathbb{E}_{f_e(\mathscr{S})}[f]$ of some measurable function f denotes the expected value of f under the probability distribution induced by the system $\mathscr{S}$ under the environment model f_e. Hence, Definition 1 says that a system satisfies a specification under a probabilistic environment model if almost all behaviors of the system satisfy the specification, i.e., the probability that the system misbehaves is 0.

[2] Note that $\mathscr{S} \models_{f_e} \varphi$ and $\mathscr{S} \models \varphi$ coincide if (i) φ is prefix-closed (which is the case for the specifications, we consider here), and (ii) $f_e(\mathscr{S})$ assigns, for every finite word $w \in L^*$, a positive probability to the set of infinite words wL^ω.



(a) Automaton stating mutual exclusion (b) Automaton with cost fcts for client i

Figure 1: Specifications for the client-server example

Next, we define the value of a system with respect to a specification under an environment model and what it means for a system to optimize a specification. Then, we are ready to define the quantitative synthesis problem.

Definition 2 (Value of a system)**.** *Given a finite-state system $\mathcal{S}$ with alphabets L and A, a qualitative (φ) and a quantitative specification (ψ) over alphabet $L \times A$, and an environment model f_e, the* value of $\mathcal{S}$ with respect to φ and ψ under f_e *is defined as the expected value of the function $\psi \circ \mathcal{O}_{\mathcal{S}}$ in the probability space $f_e(\mathcal{S})$, if $\mathcal{S}$ satisfies φ, and ∞ otherwise. Formally,*

$$Value^{f_e}_{\varphi\psi}(\mathcal{S}) := \begin{cases} \mathbb{E}_{f_e(\mathcal{S})}[\psi \circ \mathcal{O}_{\mathcal{S}}] & \text{if } \mathcal{S} \models_{f_e} \varphi, \\ \infty & \text{otherwise.} \end{cases}$$

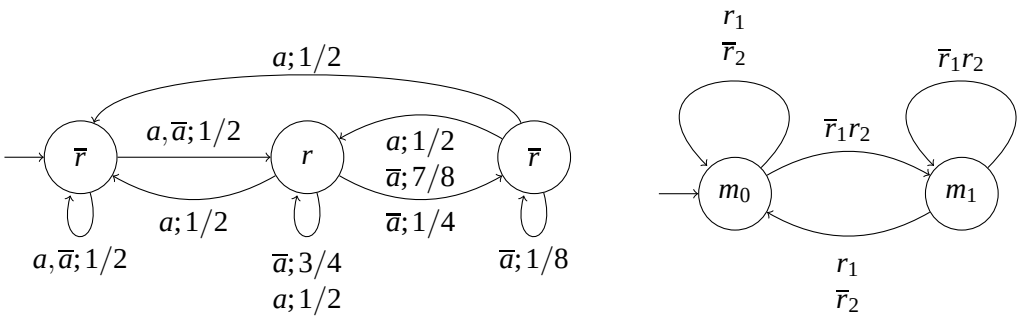
If φ is the set of all words, then we write $Value^{f_e}_{\psi}(\mathcal{S})$. Furthermore, we say $\mathcal{S}$ optimizes *ψ* wrt *f_e, if $Value^{f_e}_{\psi}(\mathcal{S}) \leq Value^{f_e}_{\psi}(\mathcal{S}')$ for all systems $\mathcal{S}'$.*

Definition 3 (Quantitative realizability and synthesis problem)**.** *Given a qualitative specification φ and a quantitative specification ψ over the alphabets $L \times A$ and an environment model f_e, the* realizability problem *asks to decide if there exists a finite-state system $\mathcal{S}$ with alphabets L and A such that $Value^{f_e}_{\varphi\psi}(\mathcal{S}) \neq \infty$. The* synthesis problem *asks to construct a finite-state system $\mathcal{S}$ (if it exists) s. t.*

1. *$Value^{f_e}_{\varphi\psi}(\mathcal{S}) \neq \infty$ and*
2. *$\mathcal{S}$ optimizes ψ wrt f_e.*

In the following, we give an example of a quantitative synthesis problem.

Server-client example. Consider a server-client system with two clients and one server. Each server-client interface consists of two variables r_i (request) and a_i (acknowledge). Client i sends a request by setting r_i to 1. The server acknowledges the request by setting a_i to 1. We require that the server does not acknowledge both clients at the same time. Hence, our qualitative specification demands mutual exclusion. Figure 1(a) shows an automaton stating the mutual exclusion property for a_1 and a_2. Edges are labeled with sets of evaluations of a_1 and a_2, e.g., $\overline{a}_1$ states that a_1 has to be 0 and a_2 can have either value, 1 and 0. States drawn with a double circle are safe states. Among all systems satisfying the mutual exclusion property, we ask for a system that minimizes the average ratio between requests and useful acknowledgments. An acknowledge is useful if it is sent as a response to a request. To express this property, we can give a quantitative language defined by an automaton with two cost functions (c_1, c_2)



(a) MDP of one client

(b) Implementation of a server for two clients. State labeling: $\tau(m_0) = a_1\overline{a}_2$ and $\tau(m_1) = \overline{a}_1 a_2$

Figure 2: Specifications and implementation for the client-server example

and the ratio objective (Eqn. 1). Figure 1(b) shows an automaton labeled with tuples representing the two cost functions c_1 and c_2 for one client. The first component of the tuples represents cost function c_1, the second component defines cost function c_2. The cost function c_1 is 1, whenever we see a request. The cost function c_2 is 1, when we see a "useful" acknowledge, which is an acknowledge that matches an unacknowledged request. E.g., every acknowledge in state s_1 is useful, since the last request has not been acknowledged yet. In state s_0 only acknowledgments that answer a direct request are useful and get cost 1 (in the second component). This corresponds to a server with a buffer that can hold exactly one request and that gets outdated after two steps and has to be dropped. State s_1 says that there is a request in the buffer. If there is no acknowledgment while the machine is in this state, then the request is lost. This means that a request has to be acknowledged in the step it is received or in the step after that.

Assume we know the expected behavior of the clients. E.g., in every step, Client 1 is expected to send a request with probability 0.5 independent of the acknowledgments. Client 2 changes its behavior based on the acknowledgments. We can describe the behavior of Client 2 by the labeled MDP shown in Figure 2(a). In the beginning the chance of getting a request from this client is 0.5. Once it has sent a request, i.e., it is in state r, the probability of sending a request again is very high until at least one acknowledgment is given. This is modeled by action $\overline{g}$ at state r having a probability of $3/4$ to get into state r again, and a probability of $1/4$ to not send a request in the next step. In this case, we move to the right $\overline{r}$ state. In this state, the probability of receiving a request from this client in the next step is even $7/8$. This means that if this client does not receive an acknowledgment after having sent a request, then the possibility of receiving another request from this client in the next two steps is $1-1/4*1/8 = 31/32$.

Consider the finite-state system $\mathscr{S}$ shown in Figure 2(b). It is an implementation of a server for two clients. The system has two states m_0 and m_1 labeled with $a_1\overline{a}_2$ and $\overline{a}_1 a_2$, respectively. We can compute the value of $\mathscr{S}$ using the following two lemmas (Lem. 1, Lem. 2).

Lemma 1. *Given (i) a finite-state system $\mathscr{S}$ with alphabets L and A, (ii) an automaton $\mathscr{A}$ with alphabet $L \times A$, and (iii) an L-labeled MDP $\mathscr{M}$ defining an environment model for $\mathscr{S}$, there exists a Markov chain $\mathscr{M}_c$ and two cost functions c_1 and c_2 such that*

$$\mathscr{S} \models_{\mathscr{M}} \mathscr{L}_{\mathscr{A}} \quad \overset{\text{Def. 1}}{\iff} \quad \mathbb{E}^{\mathscr{S}}_{\mathscr{M}}[\mathscr{L}_{\mathscr{A}} \circ \mathscr{O}_{\mathscr{S}}] = 1 \quad \iff \quad \mathbb{E}_{\mathscr{M}_c}[\mathscr{R}_{\frac{c_1}{c_2}}] = 0$$

Proof idea: The Markov chain $\mathscr{M}_c$ is constructed by taking the synchronous product of $\mathscr{S}$, $\mathscr{A}$, and $\mathscr{M}$. In every state $(s,q,m) \in (S_{\mathscr{S}} \times Q_{\mathscr{A}} \times S_{\mathscr{M}})$, we take the action $a \in A$ given by the labeling function of the system $\tau(s)$ and move to a successor state for every input label $l \in L$ such that there exists a state m' in the MDP $\mathscr{M}$ with $\lambda(m') = l$ and $p(m,a)(m') > 0$. The corresponding successor states of the system- and the automaton-state components are $s' = \delta_{\mathscr{S}}(s,l)$ and $q' = \delta_{\mathscr{A}}(q,(l,a))$. The probability distribution of $\mathscr{M}_c$ is taken from the $\mathscr{M}$-component. The two cost functions are defined as follows: for state (s,q,m) and an action a we set $c_1((s,q,m),a) = 0$ and $c_2((s,q,m),a) = 1$, if q is a safe state in $\mathscr{A}$, otherwise $c_1((s,q,m),a) = 1$ and $c_2((s,q,m),a) = 0$. Intuitively, since the non-safe states of $\mathscr{A}$ are (by definition) closed under $\delta_{\mathscr{A}}$ and all actions in this set have the same cost, they all have the same value, namely ∞, so does every state from which there is a positive probability to reach this set.[3]

Lemma 2. *Given (i) a finite-state system $\mathscr{S}$ with alphabets L and A, (ii) an automaton $\mathscr{A}$ with alphabet $L \times A$ with two cost functions c_1 and c_2, and (iii) a L-labeled MDP $\mathscr{M}$ defining an environment model for $\mathscr{S}$, there exists a Markov chain $\mathscr{M}_c$ and two cost functions d_1 and d_2 such that*

$$\mathit{Value}^{\mathscr{M}}_{\mathscr{R}_{\frac{c_1}{c_2}}}(\mathscr{S}) \quad \overset{\text{Def. 2}}{=} \quad \mathbb{E}^{\mathscr{S}}_{\mathscr{M}}[\mathscr{R}_{\frac{c_1}{c_2}} \circ \mathscr{O}_{\mathscr{S}}] \quad = \quad \mathbb{E}_{\mathscr{M}_c}[\mathscr{R}_{\frac{d_1}{d_2}}]$$

Proof idea: The construction is the same as the one for Lem. 1 except for the cost functions. The cost functions are simply copied from the component referring to the automaton, e.g., given a state $(s,q,m) \in (S_{\mathscr{S}} \times Q_{\mathscr{A}} \times S_{\mathscr{M}})$ and an action $a \in A$, $d_1((s,q,m),a) = c_1(q)$ and $d_2((s,q,m),a) = c_2(q)$.

In Section 4, we show how to compute an optimal value for MDPs with ratio objectives in polynomial time. Since Markov chains with ratio objectives are a special case of MDPs with ratio objectives, we can first use Lem. 1 to check if $\mathscr{S} \models_{\mathscr{M}} \mathscr{L}_{\mathscr{A}}$. If the check succeeds, we then use Lem. 2 to compute the value $\mathrm{Value}^{\mathscr{M}}_{\mathscr{R}_{\frac{c_1}{c_2}}}(\mathscr{S})$. This algorithm leads to the following theorem.

Theorem 1 (System value)**.** *Given a finite-state system $\mathscr{S}$ with alphabets L and A, an automaton $\mathscr{A}$ with alphabet $L \times A$ defining a qualitative language, an automaton $\mathscr{B}$ with alphabet $L \times A$ and two cost functions c_1 and c_2 defining a quantitative language, and a L-labeled MDP $\mathscr{M}$ defining an environment model, we can compute value of $\mathscr{S}$ with respect to $\mathscr{L}_{\mathscr{A}}$ and $\mathscr{R}_{\frac{c_1}{c_2}}$ under $\mathscr{P}^{\mathscr{S}}_{\mathscr{M}}$ in time polynomial in the maximum of $|\mathscr{S}| \cdot |\mathscr{A}| \cdot |\mathscr{M}|$ and $|\mathscr{S}| \cdot |\mathscr{B}| \cdot |\mathscr{M}|$.*

In order to synthesize an optimal system, we construct an MDP from the environment model, the quantitative, and qualitative specifications similar to the constructions in Lem. 1 and 2. Any optimal strategy for this MDP with a value different from ∞ corresponds to a system that satisfies the qualitative specification and optimizes the quantitative specifications. In the next section, we will show that MDPs with ratio objectives have pure memoryless optimal strategies. Therefore, we need to consider only such strategies that are pure and memoryless. Given a pure and memoryless strategy, we build the corresponding system as follows: we reduce the set of enabled actions in each state to the single action

[3] Note that instead of an MDP with ratio objective, we could have also set up a two-player safety game here.

specified by the strategy. In each state, the enabled action defines the output function of the system. Instead of deciding the next state probabilistically, the system moves from one to the next state depending on the chosen input value.

In the next section we show how to compute an optimal strategy for a given MDP in time polynomial in the number of states. This result together with construction above leads to the following theorem.

Theorem 2 (Synthesis)**.** *Given an automaton $\mathscr{A}$ with alphabet $L \times A$ defining a qualitative language , an automaton $\mathscr{B}$ with alphabet $L \times A$ and two cost functions c_1 and c_2 defining a quantitative language, and a L-labeled MDP $\mathscr{M}$ defining an environment model, we can compute an optimal system $\mathscr{S}$ with respect to $\mathscr{L}_{\mathscr{A}}$ and $\mathscr{R}_{\frac{c_1}{c_2}}$ in time polynomial in $|\mathscr{A}| \cdot |\mathscr{B}| \cdot |\mathscr{M}|$.*

4 Calculating the best strategy

In this section we will first outline a proof showing that for every MDP there is a pure and memoryless optimal strategy for our payoff function. To this end, we argue how the proof given by [25] can be adapted to our case. After that we will show how we can calculate an optimal pure and memoryless strategy.

4.1 Pure and memoryless strategies suffice

In [25], Gimbert proved that in an MDP any payoff function mapping to $\mathbb{R}$ that is submixing and prefix independent admits optimal pure and memoryless strategies. Since our payoff function $\mathscr{R}$ may also take the value ∞, we cannot apply the result immediately. However, since $\mathscr{R}$ maps only to non-negative values and the set of measurable functions is closed under addition, multiplication, limit inferior and superior and division, provided that the divisor is not equal to 0, the expected value of $\mathscr{R}$ is always defined and the theory presented in [25] also applies in this case. Furthermore, to adapt the proof of [25] to minimizing the payoff function instead of maximizing it, one only needs to inverse the used inequalities and replace max by min. What remains to show is that $\mathscr{R}$ fulfills the following two properties.

Lemma 3 ($\mathscr{R}$ is submixing and prefix independent)**.** *Let $\mathscr{M} = (S, A, \tilde{A}, p)$ be a MDP and ρ be a run.*

1. *For every $i \geq 0$ the prefix of ρ up to i does not matter, i.e., $\mathscr{R}(\rho) = \mathscr{R}(\rho_i \rho_{i+1} \dots)$.*
2. *For every sequence of non-empty words $u_0, v_0, u_1, v_1 \dots \in (A \times S)^+$ such that $\rho = u_0 v_0 u_1 v_1 \dots$ we have that the payoff of the sequence is greater than or equal to the minimal payoff of sequences $u_0 u_1 \dots$ and $v_0 v_1 \dots$, i.e., $\mathscr{R}(\rho) \geq \min\{\mathscr{R}(u_0 u_1 \dots), \mathscr{R}(v_0 v_1 \dots)\}$.*

Proof. The first property follows immediately from the first limit in the definition of $\mathscr{R}$.

For the second property we partition $\mathbb{N}$ into U and V such that U contains the indexes of the parts of ρ that belong to a u_k for some $k \in \mathbb{N}$ and such that V contains the other indexes. Formally, we define $U := \bigcup_{i \in \mathbb{N}} U_i$ where $U_0 := \{k \in \mathbb{N} \mid 0 \leq k < |u_0|\}$ and $U_i := \{max(U_{i-1}) + |v_{i-1}| + k \mid 1 \leq k \leq |u_i|\}$. Let $V := U \setminus \mathbb{N}$ be the other indexes.

Now we look at the payoff from m to l for some $m \leq l \in \mathbb{N}$, i.e. $\mathscr{R}^l_m := (\sum_{i=m\dots l} c_1(\rho_i))/(1 + \sum_{i=m\dots l} c_2(\rho_i))$. We can divide the sums into two parts, the one belonging to U and the one belonging to V and we get

$$\mathscr{R}^l_m = \frac{\left(\sum_{i \in \{m\dots l\} \cap U} c_1(\rho_i)\right) + \left(\sum_{i \in \{m\dots l\} \cap V} c_1(\rho_i)\right)}{1 + \left(\sum_{i \in \{m\dots l\} \cap U} c_2(\rho_i)\right) + \left(\sum_{i \in \{m\dots l\} \cap V} c_2(\rho_i)\right)}$$

We now define the sub-sums between the parentheses as $u_1 := \sum_{i\in\{m\ldots l\}\cap U} c_1(\rho_i)$, $u_2 := \sum_{i\in\{m\ldots l\}\cap U} c_2(\rho_i)$, $v_1 := \sum_{i\in\{m\ldots l\}\cap V} c_1(\rho_i)$ and $v_2 := \sum_{i\in\{m\ldots l\}\cap V} c_2(\rho_i)$. Then we receive

$$\mathscr{R}_m^l = \frac{u_1+v_1}{1+u_2+v_2}$$

We will now show

$$\mathscr{R}_m^l \geq \min\left\{\frac{u_1}{u_2+1}, \frac{v_1}{v_2+1}\right\}$$

Without loss of generality we can assume $u_1/(u_2+1) \geq v_1/(v_2+1)$, then we have to show that

$$\frac{u_1+v_1}{1+u_2+v_2} \geq \frac{v_1}{v_2+1}.$$

This holds if and only if $(u_1+v_1)(1+v_2) = u_1+v_1+u_1v_2+v_1v_2 \geq v_1(1+u_2+v_2) = v_1+v_1u_2+v_1v_2$ holds. By subtracting v_1 and v_1v_2 from both sides we receive $u_1+u_1v_2 = u_1(1+v_2) \geq u_2v_1$. If u_2 is equal to 0 then this holds because u_1 and v_2 are greater than or equal to 0. Otherwise, this holds if and only if $u_1/u_2 \geq v_1/(1+v_2)$ holds. In general, we have $u_1/u_2 \geq u_1/(u_2+1)$. From the assumption we have $u_1/(u_2+1) \geq v_1/(v_2+1)$ and hence $u_1/u_2 \geq v_1/(v_2+1)$. The original claim follows because we have shown this for any pair of m and l. □

Theorem 3 (There is always a pure and memoryless optimal strategy)**.** *For each MDP with the ratio payoff function, there is a pure and memoryless optimal strategy.*

Proof. See [25] □

4.2 Reduction of MDP to a Linear Fractional Program

In this section, we show how to calculate a pure and memoryless optimal strategy for an MDP with ratio objective by reducing the problem to a fractional linear programming problem. A fractional linear programming problem is similar to a linear programming problem, but the function that one wants to optimize is the fraction of two linear functions. A fractional linear programming problem can be reduced to a series of conventional linear programming problems to calculate the optimal value.

We present the reduction only for unichain MDPs. The extension to general MDPs is based on end-components [1] and the fact that end-components have an optimal unichain strategy.

Our reduction uses the fact that an MDP with a pure and memoryless strategy induces a Markov chain and that the runs of a Markov chain have a special property akin to the law of large numbers, which we can use to calculate the expected value.

Definition 4 (Random variables of MCs)**.** *Let $p^n(s)$ be the probability of being in state s at step n and let $p^*(s) := lim_{n\to\infty}\frac{1}{n}\sum_{i=0}^{n-1} p^i(s)$. This is called the* Cesaro limit *of p^n. Let further v_s^n denote the* number of visits *to state s up to time n.*

We have the following lemma describing the long-run behavior of unichain Markov chains [31, 29].

Lemma 4 (Expected number of visits of a state and well-behaved runs)**.** *For every infinite run of a unichain Markov chain, the fraction of visits to a specific state s equals $p^*(s)$ almost surely, i.e., $P(\lim_{l\to\infty}\frac{v_s^l}{l} = p^*(s)) = 1$. We call the set of runs that have this property* well-behaved*.*

When we calculate the expected payoff, we only need to consider well-behaved words as shown in the following lemma.

Lemma 5. *Let N denote the set of runs that are not well-behaved. Then*

$$\mathbb{E}_{\mathscr{M}}[\mathscr{R}] = \int_{\Omega_{\mathscr{M}} \setminus N} \mathscr{R} \, d\mu_{\mathscr{M}}$$

Proof. The probability measure of the set of well-behaved words is 1. Hence the probability measure of the complement of this set, i.e., N, has to be 0. Sets like these are called *null sets*. A classical result says that *null sets* do not need to be considered for the Lebesgue integral. □

For a well-behaved run, i.e., for every run that we need to consider when calculating the expected value, we can calculate the payoff in the following way.

Lemma 6 (Calculating the payoff of a well-behaved run)**.** *Let ρ be a well-behaved run of a unichain Markov chain. Denote by $\pi : S \to A$ the only action available at a state. Then*

$$\mathscr{R}(\rho) = \frac{\sum_{s \in S} p^*(s) c_1(s, \pi(s))}{\lim_{l \to \infty} \frac{1}{l} + \sum_{s \in S} p^*(s) c_2(s, \pi(s))}$$

Proof. By definition of $\mathscr{R}$ we have

$$\mathscr{R}(\rho) = \lim_{m \to \infty} \liminf_{l \to \infty} \frac{\sum_{i=l}^{m} c_1(\rho_i)}{1 + \sum_{i=l}^{m} c_2(\rho_i)}$$

We now assume that the Markov chain consists of one maximal recurrence class. We can do this because every non-recurrent state will not influence $\mathscr{R}(\rho)$, because ρ is well-behaved and because $\mathscr{R}$ is prefix independent. Hence

$$\mathscr{R}(\rho) = \liminf_{l \to \infty} \frac{\sum_{i=0}^{l} c_1(\rho_i)}{1 + \sum_{i=0}^{l} c_2(\rho_i)}$$

We can calculate the sums in a different way: we take the sum over the states and count how often we visit one state, i.e.,

$$\frac{\sum_{i=0}^{l} c_1(\rho_i)}{1 + \sum_{i=0}^{l} c_2(\rho_i)} = \frac{\sum_{s \in S} c_1(s, \pi(s)) \nu_s^l}{1 + \sum_{s \in S} c_2(s, \pi(s)) \nu_s^l} = \frac{\sum_{s \in S} c_1(s, \pi(s)) (\nu_s^l / l)}{1/l + \sum_{s \in S} c_2(s, \pi(s)) (\nu_s^l / l)}$$

Now we take lim instead of liminf. We will see later that the sequence converges for $l \to \infty$ and hence lim and liminf have the same value. Because both sides of the fraction are finite values we can safely draw the limit into the fraction, i.e.,

$$\begin{aligned}(\dagger) \lim_{l \to \infty} \left(\frac{\sum_{s \in S} c_1(s, \pi(s)) (\nu_s^l / l)}{1/l + \sum_{s \in S} c_2(s, \pi(s)) (\nu_s^l / l)} \right) &= \frac{\lim_{l \to \infty} \left(\sum_{s \in S} c_1(s, \pi(s)) (\nu_s^l / l) \right)}{\lim_{l \to \infty} (1/l + \sum_{s \in S} c_2(s, \pi(s)) (\nu_s^l / l))} \\ &= \frac{\sum_{s \in S} c_1(s, \pi(s)) \lim_{l \to \infty} (\nu_s^l / l)}{\lim_{l \to \infty} (1/l) + \sum_{s \in S} c_2(s, \pi(s)) \lim_{l \to \infty} (\nu_s^l / l)}\end{aligned}$$

Finally, by the definition of well-behaved runs we have $\lim_{l \to \infty} \frac{\nu_s^l}{l} = p^*(s)$. Hence

$$\frac{\sum_{s \in S} c_1(s, \pi(s)) \lim_{l \to \infty} (\nu_s^l / l)}{\lim_{l \to \infty} (1/l) + \sum_{s \in S} c_2(s, \pi(s)) \lim_{l \to \infty} (\nu_s^l / l)} = \frac{\sum_{s \in S} c_1(s, \pi(s)) p^*(s)}{\lim_{l \to \infty} (1/l) + \sum_{s \in S} c_2(s, \pi(s)) p^*(s)}$$

The limit diverges to ∞ if and only if the second costs are all equal to zero and at least one first cost is not. In this case the original definition of $\mathscr{R}$ diverges and hence $\mathscr{R}$ and the last expression are the same. Otherwise the last expression converges, hence † converges, ergo liminf and lim of this sequence are the same. □

Note that the previous lemma implies that the value of a well-behaved run is independent of the actual run. In other words, on the set of well-behaved runs of a unichain Markov chain the payoff function is constant[4]. Ergo the expected value of such a Markov chain is equal to the payoff of any of its well-behaved runs.

Theorem 4 (Expected payoff of a MDP and a strategy)**.** *Let $\mathscr{M}$ be a MDP such that every pure and memoryless strategy induces an unichain MC. Let further p^* denote the Cesaro limit of p^n of the induced Markov chain. Then for every pure and memoryless strategy π*

$$\mathbb{E}^{\pi}_{\mathscr{M}}[\mathscr{R}] = \frac{\sum_{s\in S} c_1(s,\pi(s))p^*(s)}{\lim_{l\to\infty}(1/l) + \sum_{s\in S} c_2(s,\pi(s))p^*(s)}$$

Proof. This follows from the previous lemma and the fact that $\mathscr{R}$ is constant on any well-behaved run. □

Note that this means that an expected value is ∞ if and only if the second cost of every action in the recurrence class of the Markov chain is 0 and there is at least one first cost that is not.

Using this lemma, we are now able to transform the MDP into a fractional linear program. This is done in the same way as is done for the expected average payoff case (cf. [30]). We define variables $x(s,a)$ for every state $s \in S$ and every available action $a \in \tilde{A}(s)$. This variable intuitively corresponds to the probability of being in state s and choosing action a at any time. Then we have, for example $p^*(s) = \sum_{a\in\tilde{A}(s)} x(s,a)$. We need to restrict this set of variables. First of all, we always have to be in some state and choose some action, i.e., the sum over all $x(s,a)$ has to be one. The second set of restrictions ensures that we have a stationary distribution, i.e., the sum of the probabilities of going out of (i.e., being in) a state is equal to the sum of the probabilities of moving into this state.

Definition 5 (Fractional Linear program for MDP)**.** *Let $\mathscr{M}$ be an unichain MDP such that every Markov chain induced by any strategy contains at least one non-zero second cost. Then we define the following fractional linear program for it.*

$$\textit{Minimize } \frac{\sum_{s\in S}\sum_{a\in\tilde{A}(s)} x(s,a)c_1(s,a)}{\sum_{s\in S}\sum_{a\in\tilde{A}(s)} x(s,a)c_2(s,a)} \tag{4}$$

subject to

$$\sum_{s\in S}\sum_{a\in\tilde{A}(S)} x(s,a) = 1 \tag{5}$$

$$\sum_{a\in\tilde{A}(s)} x(s,a) = \sum_{s'\in S}\sum_{a\in\tilde{A}(s')} x(s',a)p(s',a)(s) \quad \forall s \in S \tag{6}$$

There is a correspondence between pure and memoryless strategies and basic feasible solutions to the linear program[5]. That is, the linear program always has a solution because every positional strategy corresponds to a solution. See [30] for a detailed analysis of this in the expected average reward case.

Once we have calculated a solution of the linear program, we can calculate the strategy as follows.

Definition 6 (Strategy from solution of linear program)**.** *Let $x(s,a)$ be the solutions to the linear program. Then we define the strategy as follows.*

[4]Note that the fact that any payoff function that is prefix-independent is constant almost surely on each irreducible Markov chain has already been proved by [25]

[5]A feasible solution is one that fulfills the linear equations that every solution is subject to.

$$\pi(s) = \begin{cases} \textit{arbitrary} & \textit{if } x(s,a) = 0 \textit{ for every enabled action } a \\ a & \textit{if } x(s,a) > 0 \end{cases}$$

Note that this is well defined because for each state s there is at most one action a such that $x(s,a) > 0$ because of the bijection (modulo the action of transient states) between basic feasible solutions and strategies and because the optimal strategy is always pure and memoryless.

4.3 From LFP to LP

Since solvers to linear fractional programs are not common and there are good free solvers to linear programs, we presented a method of converting a linear fractional program to a sequence of linear programs that calculate the solution. This algorithm is due to [27]. Let $f(x)$ denote the value of Eqn. 4 under variable assignment x.

Input: feasible solution x_0, MDP $\mathscr{M}$
Output: Variable assignment, optimal solution
$n \leftarrow 0$
repeat
  $g \leftarrow f(x_n)$
  $n \leftarrow n+1$
  Solve

$$\text{Minimize } \sum_{s \in S} \sum_{a \in \tilde{A}(s)} x_n(s,a) c_s^1 - g \sum_{s \in S} \sum_{a \in \tilde{A}(s)} x_n(s,a) c_s^2$$

  subject to Eqn. 5 and Eqn. 6.
until $f(x_{n-1}) = f(x_n)$;
return x_n, $f(x_n)$

4.4 Preliminary Implementation

We have developed a tool that can handle (finite) unichain MDPs with ratio objectives based on the approach presented in this paper. Our tool is implemented in Haskell and uses the *GNU Linear Programming Kit* to solve the resulting linear programs.

We made some initial experiments using the server-client example from Section 3. In the case of two clients we have a MDP with 24 states and 288 edges. Building and solving this system takes less than 100 milliseconds on a Laptop with an Intel Core 2 Duo P8600 clocked at 2.40 GHz. The resulting machine behaves as follows: If it receives only one request at the start, then it acknowledges this request immediately. Whenever Client 2, i.e., the complicated client, sends a request, then it also receives the acknowledgment, with one exception: When Client 1 has an outstanding request, i.e., if its qualitative specification is in state s_1, and if Client 2 has no outstanding request, then Client 1 receives the acknowledgment, The expected value is roughly $1.2 = 12/10$. This means that, out of 12 requests, 10 can be served, which means 83.3%.

5 Conclusions and Future Work

We have presented a technique to automatically synthesize system that satisfy a qualitative specification and optimize a quantitative specification under a given environment model. Our technique can handle qualitative specifications given by an automaton with a set of safe states, and quantitative specifications defined by an automaton with ratio objective.

Currently, we are working on a better representation of the input specifications. In particular, we are aiming for a symbolic representation that would allow us to use a combined symbolic and explicit approach, which has shown to be very effective for MDP with long-run average objective [32]. Furthermore, we are extending the presented approach to qualitative specification describe by arbitrary ω-regular specifications.

References

[1] L. de Alfaro (1997): *Formal Verification of Probabilistic Systems*. Ph.D. thesis, Stanford University.

[2] Luca de Alfaro (1998): *Stochastic Transition Systems*. In: Davide Sangiorgi & Robert de Simone, editors: *CONCUR, Lecture Notes in Computer Science* 1466, Springer, pp. 423–438. Available at `http://link.springer.de/link/service/series/0558/bibs/1466/14660423.htm`.

[3] Luca de Alfaro, Thomas A. Henzinger & Rupak Majumdar (2003): *Discounting the Future in Systems Theory*. In: Jos C. M. Baeten, Jan Karel Lenstra, Joachim Parrow & Gerhard J. Woeginger, editors: *ICALP, Lecture Notes in Computer Science* 2719, Springer, pp. 1022–1037. Available at `http://link.springer.de/link/service/series/0558/bibs/2719/27191022.htm`.

[4] Luca de Alfaro, Rupak Majumdar, Vishwanath Raman & Mariëlle Stoelinga (2007): *Game Relations and Metrics*. In: *LICS*, IEEE Computer Society, pp. 99–108. Available at `http://doi.ieeecomputersociety.org/10.1109/LICS.2007.22`.

[5] Rajeev Alur, Aldric Degorre, Oded Maler & Gera Weiss (2009): *On Omega-Languages Defined by Mean-Payoff Conditions*. In: Luca de Alfaro, editor: *FOSSACS, Lecture Notes in Computer Science* 5504, Springer, pp. 333–347. Available at `http://dx.doi.org/10.1007/978-3-642-00596-1_24`.

[6] C. Baier, M. Größer, M. Leucker, B. Bollig & F. Ciesinski (2004): *Controller Synthesis for Probabilistic Systems*. In: *IFIP TCS*, pp. 493–506.

[7] Christel Baier, Boudewijn R. Haverkort, Holger Hermanns & Joost-Pieter Katoen (2010): *Performance evaluation and model checking join forces*. *Commun. ACM* 53(9), pp. 76–85. Available at `http://doi.acm.org/10.1145/1810891.1810912`.

[8] G. Behrmann, J. Bengtsson, A. David, K. G. Larsen, P. Pettersson & W. Yi. (2002): UPPAAL *Implementation Secrets*. In: *Formal Techniques in Real-Time and Fault Tolerant Systems*.

[9] Andrea Bianco & Luca de Alfaro (1995): *Model Checking of Probabalistic and Nondeterministic Systems*. In: P. S. Thiagarajan, editor: *FSTTCS, Lecture Notes in Computer Science* 1026, Springer, pp. 499–513. Available at `http://dx.doi.org/10.1007/3-540-60692-0_70`.

[10] Roderick Bloem, Krishnendu Chatterjee, Thomas A. Henzinger & Barbara Jobstmann (2009): *Better Quality in Synthesis through Quantitative Objectives*. In: Ahmed Bouajjani & Oded Maler, editors: *CAV, Lecture Notes in Computer Science* 5643, Springer, pp. 140–156. Available at `http://dx.doi.org/10.1007/978-3-642-02658-4_14`.

[11] Roderick Bloem, Karin Greimel, Thomas A. Henzinger & Barbara Jobstmann (2009): *Synthesizing robust systems*. In: *FMCAD*, IEEE, pp. 85–92. Available at `http://dx.doi.org/10.1109/FMCAD.2009.5351139`.

[12] Arindam Chakrabarti, Krishnendu Chatterjee, Thomas A. Henzinger, Orna Kupferman & Rupak Majumdar (2005): *Verifying Quantitative Properties Using Bound Functions*. In: Dominique Borrione & Wolfgang J. Paul, editors: *CHARME, Lecture Notes in Computer Science* 3725, Springer, pp. 50–64. Available at `http://dx.doi.org/10.1007/11560548_7`.

[13] Krishnendu Chatterjee, Luca de Alfaro, Marco Faella, Thomas A. Henzinger, Rupak Majumdar & Mariëlle Stoelinga (2006): *Compositional Quantitative Reasoning*. In: *QEST*, IEEE Computer Society, pp. 179–188. Available at `http://doi.ieeecomputersociety.org/10.1109/QEST.2006.11`.

[14] Krishnendu Chatterjee, Laurent Doyen & Thomas A. Henzinger (2008): *Quantitative Languages*. In: Michael Kaminski & Simone Martini, editors: *CSL*, *Lecture Notes in Computer Science* 5213, Springer, pp. 385–400. Available at http://dx.doi.org/10.1007/978-3-540-87531-4_28.

[15] Krishnendu Chatterjee, Thomas A. Henzinger, Barbara Jobstmann & Rohit Singh (2010): *Measuring and Synthesizing Systems in Probabilistic Environments*. In: Tayssir Touili, Byron Cook & Paul Jackson, editors: *CAV*, *Lecture Notes in Computer Science* 6174, Springer, pp. 380–395. Available at http://dx.doi.org/10.1007/978-3-642-14295-6_34.

[16] Costas Courcoubetis & Mihalis Yannakakis (1990): *Markov Decision Processes and Regular Events (Extended Abstract)*. In: Mike Paterson, editor: *ICALP*, *Lecture Notes in Computer Science* 443, Springer, pp. 336–349. Available at http://dx.doi.org/10.1007/BFb0032043.

[17] R. A. Cuninghame-Green (1979): *Minimax algebra*. In: *Lecture Notes in Economics and Mathematical Systems*, 166, Springer-Verlag.

[18] C. Derman (1962): *On Sequential Decisions and Markov Chains*. *Management Science* 9(1), pp. 16–24.

[19] Josee Desharnais, Vineet Gupta, Radha Jagadeesan & Prakash Panangaden (2004): *Metrics for labelled Markov processes*. *Theor. Comput. Sci.* 318(3), pp. 323–354. Available at http://dx.doi.org/10.1016/j.tcs.2003.09.013.

[20] M. Droste & P. Gastin (2007): *Weighted automata and weighted logics*. *Theoretical Computer Science* 380, pp. 69–86. Available at http://dx.doi.org/10.1016/j.tcs.2007.02.055.

[21] M. Droste, W. Kuich & H. Vogler (2009): *Handbook of Weighted Automata*. Springer Publishing Company, Incorporated.

[22] Manfred Droste, Werner Kuich & George Rahonis (2008): *Multi-Valued MSO Logics OverWords and Trees*. *Fundam. Inform.* 84(3-4), pp. 305–327. Available at http://iospress.metapress.com/content/j9652453g663425m/.

[23] J. Filar & K. Vrieze (1996): *Competitive Markov Decision Processes*. Springer-Verlag.

[24] Stephane Gaubert & Max Plus (1997): *Methods and Applications of (MAX, +) Linear Algebra*. In: Rüdiger Reischuk & Michel Morvan, editors: *STACS*, *Lecture Notes in Computer Science* 1200, Springer, pp. 261–282. Available at http://dx.doi.org/10.1007/BFb0023465.

[25] Hugo Gimbert (2007): *Pure Stationary Optimal Strategies in Markov Decision Processes*. In: Wolfgang Thomas & Pascal Weil, editors: *STACS*, *Lecture Notes in Computer Science* 4393, Springer, pp. 200–211. Available at http://dx.doi.org/10.1007/978-3-540-70918-3_18.

[26] A. Hinton, M. Kwiatkowska, G. Norman & D. Parker (2006): *PRISM: A Tool for Automatic Verification of Probabilistic Systems*. In: *TACAS*.

[27] J. R. Isbell & W. H. Marlow (1956): *Attrition games*. *Naval Research Logistics Quarterly* 3, pp. 71–94.

[28] Orna Kupferman & Yoad Lustig (2007): *Lattice Automata*. In: Byron Cook & Andreas Podelski, editors: *VMCAI*, *Lecture Notes in Computer Science* 4349, Springer, pp. 199–213. Available at http://dx.doi.org/10.1007/978-3-540-69738-1_14.

[29] J.R. Norris (2003): *Markov Chains*. Cambridge University Press.

[30] M. L. Puterman (1994): *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience.

[31] H. C. Tijms (2003): *A First Course in Stochastic Models*. Chichester: Wiley.

[32] Ralf Wimmer, Bettina Braitling, Bernd Becker, Ernst Moritz Hahn, Pepijn Crouzen, Holger Hermanns, Catuscia Dhama & Oliver E. Theel (2010): *Symblicit Calculation of Long-Run Averages for Concurrent Probabilistic Systems*. In: *QEST*, IEEE Computer Society, pp. 27–36. Available at http://dx.doi.org/10.1109/QEST.2010.12.

[33] Uri Zwick & Mike Paterson (1996): *The Complexity of Mean Payoff Games on Graphs*. *Theor. Comput. Sci.* 158(1&2), pp. 343–359. Available at http://dx.doi.org/10.1016/0304-3975(95)00188-3.