QUASY: Quantitative Synthesis Tool

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

[1] Institute of Science and Technology Austria, Austria
[2] École Polytechnique Fédéral de Lausanne, Switzerland
[3] CNRS/Verimag, France
[4] Indian Institute of Technology Bombay, India

Abstract. We present the tool QUASY, a quantitative synthesis tool. QUASY takes qualitative and quantitative specifications and automatically constructs a system that satisfies the qualitative specification and optimizes the quantitative specification, if such a system exists. The user can choose between a system that satisfies and optimizes the specifications (a) under all possible environment behaviors or (b) under the most-likely environment behaviors given as a probability distribution on the possible input sequences. QUASY solves these two quantitative synthesis problems by reduction to instances of 2-player games and Markov Decision Processes (MDPs) with quantitative winning objectives. QUASY can also be seen as a game solver for quantitative games. Most notable, it can solve lexicographic mean-payoff games with 2 players, MDPs with mean-payoff objectives, and ergodic MDPs with mean-payoff parity objectives.

1 Introduction

Quantitative techniques have been successfully used to measure quantitative properties of systems, such as timing, performance, or reliability (cf. [1, 9, 2]). We believe that quantitative reasoning is also useful in the classically Boolean contexts of verification and synthesis because they allow the user to distinguish systems with respect to "soft constraints" like robustness [4] or default behavior [3]. This is particularly helpful in synthesis, where a system is automatically derived from a specification, because the designer can use soft constraints to guide the synthesis tool towards a desired implementation.

QUASY[5] is the first synthesis tool taking soft constraints into account. Soft constraints are specified using quantitative specifications, which are functions that map infinite words over atomic propositions to a set of values. Given a (classical) qualitative specification φ and a quantitative specification ψ over signals $\mathcal{I} \cup \mathcal{O}$, the tool constructs a reactive system with input signals $\mathcal{I}$ and output signals $\mathcal{O}$ that satisfies φ (if such a system exists) and optimizes ψ either under the worse-case behavior [3] or under the average-case behavior [7] of the environment. The average-case behavior of the environment can be specified by a probability distribution μ of the input sequences.

In summary, QUASY is the first tool for quantitative synthesis, both under adversarial environment as well as probabilistic environment. The underlying techniques to achieve quantitative synthesis are algorithms to solve two-player games and MDPs with quantitative objectives. QUASY is the first tool that solves lexicographic mean-payoff games and ergodic MDPs with mean-payoff parity objectives.

[5] http://pub.ist.ac.at/quasy/

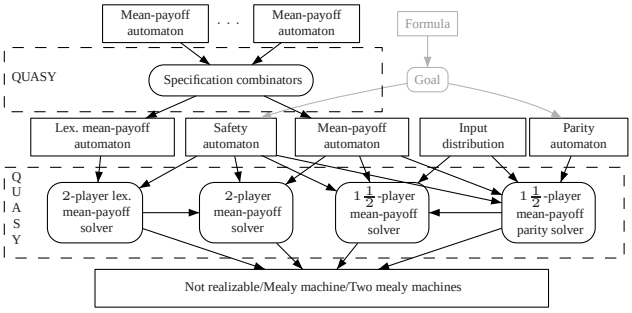

Fig. 1. Overview of input/output structure

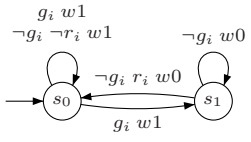

Fig. 2. An automaton rewarding a quick response to Client i

k	Spec	Game	System	Time(s)
3	8	35	4	0.81
4	16	97	8	1.64
5	32	275	16	3.43
6	64	793	32	15.90
7	128	2315	64	34.28

Fig. 3. Results for MDPs

2 Synthesis from Combined Specifications

QUASY handles several combinations of qualitative and quantitative specifications. We give a brief description of the input format the tool expects. Then, we summarize the implemented combinations (see Figure 1 for an overview) and give some results.

Specifications. QUASY accepts qualitative specifications given as deterministic safety or parity automaton in GOAL[6] format. LTL properties can be translated into the required format using GOAL. Quantitative properties are specified by (lexicographic) mean-payoff automata. A mean-payoff automaton A is a deterministic automaton with weights on edges that maps a word v to the average over the weights encountered along the run of A on v. Lexicographic mean-payoff automata [3] are a generalization of mean-payoff automata. They map edges to tuples of weights. Figure 2 shows a mean-payoff automaton with alphabet $2^{\{r_i,g_i\}}$ in the accepted format (GOAL). Labels of the form wk, for $k \in \mathbb{N}$, state that the edge has weight k. E.g., the edge from state s_0 to s_1 labeled $\neg g_i\ \neg r_i\ w1$ states that if r_i and g_i are false, then we can follow this edge and encoder a weight of 1. QUASY can combine a set of mean-payoff automata to (i) a lexicographic mean-payoff automaton or (ii) a mean-payoff automaton representing the sum of the weights.

Optimality. The user can choose between two modes: (1) the construction of a worst-case optimal system, or (2) the construction of an average-case optimal system. In the latter case, the user needs to provide a text file that assigns to each state of the specification a probability distribution over the possible input values. For states that do not appear in the file, a uniform distribution over the input values is assumed.

Combinations and Results. The tool can construct worst-case optimal systems for mean-payoff and lexicographic mean-payoff specifications combined with safety specifications. For the average-case, QUASY accepts mean-payoff specifications combined with safety and parity specifications. Figure 3 shows (in Column 5) the time needed to construct a resource controller for k clients that (i) guarantees mutually exclusive access to the resource and (ii) optimizes the reaction time in a probabilistic environment, in which clients with lower id are more likely to send requests than clients with higher id. The quantitative specifications were built from k copies of the automaton shown in Figure 2. Column 2-4 in Figure 3 show the size of the specifications, the size of the corresponding game graphs, and the size of the final Mealy machines, respectively. These specifications were also used for preliminary experiments reported in [7]. QUASY significantly improves these

[6] GOAL is a graphical tool for manipulating Büchi automata and temporal formulae. It is available at http://goal.im.ntu.edu.tw/. We use it as graphical front-end for QUASY.

runtimes, e.g., from 5 minutes to 16 seconds for game graphs with 800 states (6 clients). We provide more examples and results at http://pub.ist.ac.at/quasy/.

3 Implementation Details

QUASY is implemented in the programming language SCALA[10] with an alternative mean-payoff MDP solver in C++. It transforms the input specifications into a game graph. States are represented explicitly. Labels are stored as two pairs of sets pos and neg corresponding to the positive and negative literals, respectively, for both input and output alphabets to facilitate efficient merging and splitting of edges during the game transformations. The games are solved using one of the game solvers described below. If a game is winning for the system, QUASY constructs a winning strategy, transforms it into a reactive system, and outputs it in GOAL format.

Two-Player Mean-Payoff Games. For two-player games with mean-payoff objectives, we have implemented the value iteration algorithm of [12]. The algorithm runs in steps. In every step k, the algorithm computes for each state s the minimal reward r_k player system can obtain within the next k steps starting from state s independent of the choice of the other player. The reward is the sum of the weights seen along the next k steps. The *k-step value* v_k, obtained by dividing the reward r_k by k, converges to the actual value of the game. After $n^3 \cdot W$ steps, where n is the size of the state space and W is the maximal weight, the k-step value uniquely identifies the value of the game [12]. Since the theoretical bound can be large and the value often converges before the theoretical bound is reached, we implemented the following *early stopping criteria*. The actual value of a state is a rational $\frac{e}{d}$ with $d \in \{1, \ldots, n\}$ and $e \in \{1, \ldots, d \cdot W\}$, and it is always in a $\frac{n \cdot W}{k}$-neighbourhood of the approximated value v_k [12]. So, we can fix the value of a state, if, for all d, there is only one integer in the interval $[d \cdot (v_k - \frac{n \cdot W}{k}), d \cdot (v_k + \frac{n \cdot W}{k})]$ and the integers obtained by varying d correspond to the same rational number. Fixing the value of these states, leads to a faster convergence. We implemented this criterion alternatively also by storing all possible values in an array and performing a binary search for a unique value. This method requires more memory but increases the speed of checking convergence.

Two-Player Lexicographic Mean-Payoff Games. For two-player lexicographic mean-payoff games, we have implemented three variants of value iterations. First, a straight forward adaption of the reduction described in [3]: given a lexicographic weight function $\overrightarrow{w}$ with two components $\overrightarrow{w}|_1$ and $\overrightarrow{w}|_2$, we construct a single weight function w defined by $w = c \cdot \overrightarrow{w}|_1 + \overrightarrow{w}|_2$, where the constant $c = n^2 \cdot W + 1$ depends on the maximal weight W in $\overrightarrow{w}|_2$ and the number n of states of the automaton. The other two variants keep the components separately and iterate over tuples. In the second version, we add the tuples component-wise and compare two tuples by evaluating them as a sum of successive division by powers of the base c. This avoids handling large integers but requires high precision. In the third version, we use the following addition modulo c on tuples to obtain a correct value iteration:

$$\begin{pmatrix} a_1 \\ a_2 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} a_1 + b_1 + ((a_2 + b_2) \text{ div } c) \\ (a_2 + b_2) \bmod c \end{pmatrix}.$$

We use lexicographic comparison because in many cases we do not need to compare all components to decide the ordering between two tuples. Furthermore, it allows us to handle large state spaces and large weights, which would lead to an overflow otherwise.

MDPs with Mean-Payoff and Mean-Payoff-Parity Objective. For ergodic MDPs with mean-payoff-parity objective, we implemented the algorithm described in [7]. QUASY produces two mealy machines A and B as output: (i) A is optimal wrt the mean-payoff objective and (ii) B almost-surely satisfies the parity objective. The actual system corresponds to a combination of the two mealy machines based on inputs from the environment switching over from one mealy machine to another based on a counter as explained in [7]. For MDPs with mean-payoff, QUASY implements the strategy improvement algorithm (cf. [8], Section 2.4) using two different methods to compute an improvement step of the algorithm: (i) Gaussian elimination that requires the complete probability matrix to be stored in memory (works well for dense and small game graphs) and (ii) GMRES iterative sparse matrix equation solver (works very well for sparse and large game graphs with optimizations as explained in [6]). The strategy for parity objective is computed using a reverse breadth first search from the set of least even-parity states ensuring that in every state we choose an action which shortens the distance to a least even-parity state.

4 Future Work

We will explore different directions to improve the performance of QUASY. In particular, a recent paper by Brim and Chaloupka [5] proposes a set of heuristics to solve mean-payoff games efficiently. It will be interesting to see if these heuristics can be extended to lexicographic mean-payoff games. Furthermore, Wimmer et al. [11] recently developed an efficient technique for solving MDP with mean-payoff objectives based on combining symbolic and explicit computation. We will investigate if symbolic and explicit computations can be combined for to MDPs with mean-payoff parity objectives as well.

References

1. C. Baier, B. R. Haverkort, H. Hermanns, and J.-P. Katoen. Performance evaluation and model checking join forces. *Commun. ACM*, 53(9), 2010.
2. G. Behrmann, J. Bengtsson, A. David, K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL implementation secrets. In *Formal Techniques in Real-Time and Fault Tolerant Systems*, 2002.
3. R. Bloem, K. Chatterjee, T. A. Henzinger, and B. Jobstmann. Better quality in synthesis through quantitative objectives. In *CAV*, pages 140–156, 2009.
4. R. Bloem, K. Greimel, T. A. Henzinger, and B. Jobstmann. Synthesizing robust systems. In *FMCAD*, 2009.
5. L. Brim and J. Chaloupka. Using strategy improvement to stay alive. *CoRR*, 1006.1405, 2010.
6. P. Černý, K. Chatterjee, T. Henzinger, A. Radhakrishna, and R. Singh. Quantitative synthesis for concurrent programs. Technical Report IST-2010-0004, IST Austria, 2010.
7. K. Chatterjee, T. A. Henzinger, B. Jobstmann, and R. Singh. Measuring and synthesizing systems in probabilistic environments. In *CAV*, 2010.
8. Eugene A. Feinberg and Adam Shwartz. *Handbook of Markov Decision Processes: Methods and Applications*. Springer, 2001.
9. A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In *TACAS*, 2006.
10. The Scala programming language. http://www.scala-lang.org/.
11. R. Wimmer, B. Braitling, B. Becker, E. M. Hahn, P. Crouzen, H. Hermanns, A. Dhama, and O. Theel. Symblicit calculation of long-run averages for concurrent probabilistic systems. In *QEST*, 2010.
12. U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Theor. Comput. Sci.*, 158(1-2):343–359, 1996.