QUASY: Quantitative Synthesis Tool

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

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

Abstract. 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[1] 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 [6] 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.

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

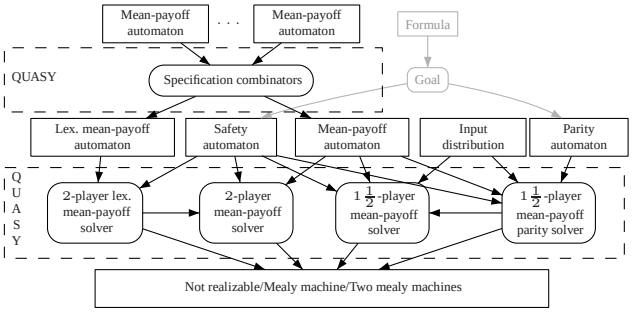

Fig. 1. Overview of input/output structure

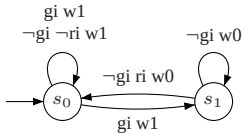

Fig. 2. A of quick reaction

k	Spec	Game	System	Time(s)
2	4	13	2	0.50
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[2] 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 in the accepted format. Labels of the form wk state that the edge has weight k. 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 later 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 [6]. QUASY significantly improves these runtimes, e.g., from 5 minutes to 16 seconds for game graphs with 800 states (6 clients). In the appendix we provide more examples and results for lexicographic mean-payoff and mean-payoff parity specifications.

[2] http://goal.im.ntu.edu.tw/

3 Implementation Details

QUASY is implemented in the programming language SCALA[3] 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 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 sum of weights player system can force to obtain within the next k steps starting from state s. The *k-step value* v_k, obtained by dividing this sum 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]. Therefore, 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.

[3] http://www.scala-lang.org/

MDPs with Mean-Payoff and Mean-Payoff-Parity Objective. For ergodic MDPs with mean-payoff-parity objective, we implemented the algorithm described in [6]. 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 [6]. For MDPs with mean-payoff, QUASY implements the strategy improvement algorithm (cf. [7], 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 [**?**]). 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 *Symposium on Formal Techniques in Real-Time and Fault Tolerant Systems*, 2002. http://www.uppaal.com/.
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*, abs/1006.1405, 2010.
6. K. Chatterjee, T. A. Henzinger, B. Jobstmann, and R. Singh. Measuring and synthesizing systems in probabilistic environments. In *CAV*, 2010.
7. Eugene A. Feinberg and Adam Shwartz. *Handbook of Markov Decision Processes: Methods and Applications*. Springer, 2001.
8. J. Filar and K. Vrieze. *Competitive Markov Decision Processes*. Springer, 1996.
9. A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In *TACAS*, 2006.
10. M. L. Puterman. *Markov Decision Processes*. John Wiley and Sons, 1994.
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. Accepted for publication.

12. U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Theor. Comput. Sci.*, 158(1-2):343–359, 1996.

Appendix : Details of the Tool

1 Dependencies and Architecture

Language, tools and installation. QUASY is written in Scala and uses a C++ based mean-payoff MDP Solver. All input and output automata can be visualized using the GOAL[1] graphical user interface. For compilation and installation: (a) installations of Scala, Java and C++ compilers are required; (b) A Scala runtime environment is required (c) installation of GOAL is required to visualize and manipulate inputs and outputs.

Source Code. The source code is structured in four main groups:

1. **Auxilary Datastructures** for game graphs and MDPs. *Class* **disjunct** provides an efficient way of representing and performing operations (checking implications and merging) on propositional disjunctions of both input and output variables (requests and grants) using **pos** and **neg** sets of indices corresponding to the positive and negative literals in the disjunction. E.g., $r0 \neg r2$ is represented as **disjunct(pos:**$\{\mathbf{0}\}$**,neg:**$\{\mathbf{2}\}$**)**. *Classes* **Edge**, **Action** and **State** implement generic edges, actions, and states used in the representations of game graph, automata, Mealy machines, and MDPs.
2. *Classes* **Game** and **MDP** are used as a generic representation for automaton, games, and MDPs. They provide:
   - methods for **building a game or an automaton** in memory from the specified GOAL format file and exporting an automaton or a game to GOAL format files.
   - functions for **converting an automaton to equivalent game graph** with as few states as possible using merging of Player-0 states with similar transition labels. This function first creates one Player-1 state for each original state in automaton and then for each state, and each minterm for input variables ($r0, r1, \dots$) finds edges whose input label is implied by this minterm. This map of minterms to sets of edge IDs is reduced to a map between sets of minterms and sets of edges. A new Player-0 state is created for each map entry in the final map.
   - methods to perform various **value iterations** and **finding an optimal positional strategy** for mean-payoff games, lexicographic mean-payoff games (with three different evaluation criteria), and discounted payoff games.
   - methods for performing **mean-payoff MDP policy iteration** and **stochastic shortest path algorithm** for ergodic parity MDP producing a pair of optimal positional strategies.
3. Object **Main** handles Input/Output for the tool in various modes of operation and provides product functions (prodAPPEND, prodMULT, prodADD, prodSafety etc.) for combining input specifications as a unified weighted or safety specification.
4. C++ based mean-payoff MDP solver: QUASY can also use an external mean-payoff MDP solver which utilizes sparse matrix techniques for the strategy improvement step in case of large game graphs. The implementation uses SparseLib[2] to perform sparse matrix operations efficiently. For this operation, the Scala application exports and imports the MDP in the format required by the C++ implementation.

[1] http://goal.im.ntu.edu.tw/

[2] http://www.ing.unitn.it/ bertolaz/4-software/Cpp/docs/sparselib/sparselib.html

2 User Manual

In this section we describe the usage of the tool.

Format of files. The file format used by QUASY is based on the format used by GOAL for both inputs and outputs. The format for input specifications is presented below

```
<structure label-on="transition" type="fa">
<alphabet type="propositional">
  <prop>[r{NUMERIC}|g{NUMERIC}]</prop>
  ...
</alphabet>
<stateSet>
  <state sid="NUMERIC">
  [<label>NUMERIC(Parity)</label>]
</state>
...
</stateSet>
<transitionSet>
  <transition tid="NUMERIC">
    <from>NUMERIC(State ID)</from>
    <to>NUMERIC(State ID)</to>
    <read>[([¬]r{NUMERIC})* ([¬]g{NUMERIC})* w{NUMERIC}(v{NUMERIC})*]</read>
</transition>
...
</transitionSet>
<initialStateSet>
  <stateID>NUMERIC</stateID>
</initialStateSet>
<acc type="[buchi|parity]">
  <accSet> %For Buchi and parity acceptance conditions
    <stateID>NUMERIC(State ID)</stateID>
    ...
  </accSet>
</acc>
</structure>
```

By convention, we specify the weight on an edge using a label starting with the letter w followed by v's separating the components in case of multidimensional weights. In parity specifications, each state is labeled with its priority. In safety specifications all weights are assumed to be 1. An input specification represented by a safety or a weighted automaton can be constructed using GOAL GUI. The top of the screenshot in Figure 1 shows two weighted automata (w0.gff and w1.gff) rewarding quick grants, and a safety specification (safe2.gff) ensuring exclusive grants. The outputs of the tool (the game graphs and the Mealy machines) are also in GOAL format and can be visualized and simplified using GOAL.

Input Specifications. QUASY accepts qualitative specifications given as deterministic safety or parity automaton in GOAL format with atomic propositions from $\{r_0, r_1, \ldots, g_0, g_1, \ldots\}$. By convention r_i are input signals and g_i are output signals. We can use the tool GOAL to obtain the input automaton. E.g., the safety automaton in Figure 1 (on the top right) is obtained from the LTL specification $\texttt{always}(\neg g_0 \vee \neg g_1)$. Quantitative properties are specified by (lexicographic) mean-payoff automata. A mean-payoff automaton (MPA) 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 (LMPA) are a generalization of mean-payoff automata that map edges to tuples of weights.

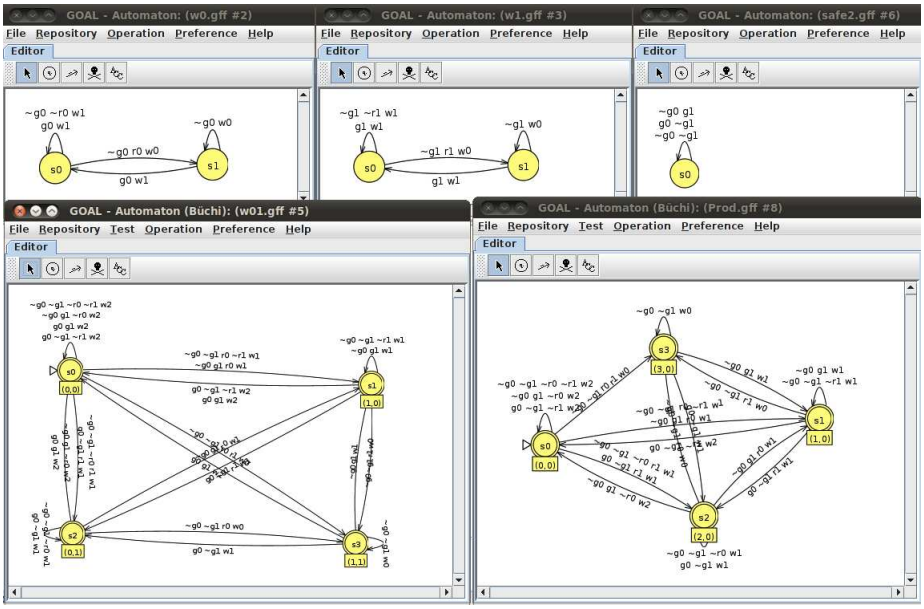

Fig. 1. Weighted(**w0** and **w1**) and Safety(**safe2**) Specifications, and their combinations obtained using **prodADD**(**w01** from **w0** and **w1**) and **prodMULT**(**Prod** from **w01** and **safe2**)

QUASY provides three operations to build larger specifications from smaller ones:

1. prodAPPEND: constructs the product of two LMPAs and appends their weights lexicographically.
2. prodADD: constructs the product of two MPAs and sums their weights.
3. prodMULT: constructs the product of an LMPA and an MPA and multiplies their weights. Since safety automata are assumed to have edges with weight 1, we use this operation to make the product of two safety automata.

Deterministic parity specifications can be constructed from LTL formulas or smaller specifications using GOAL.

Optimality. QUASY can use two definitions for optimality: (1) a system is optimal in the worst-case, or (2) a system is optimal in the average-case. In the later 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. Each line of the file starts with the id of a state followed by $2^{|\mathcal{I}|}$ entries assigning probabilities to all input minterms in lexicographic order. E.g., for $\mathcal{I} = \{r_0, r_1\}$, the line "`1 0.4 0.3 0.2 0.1`" states that in state `1`, the probability of reading input value $\bar{r}_0\bar{r}_1$ is 0.4, input $\bar{r}_0 r_1$ has a probability of 0.3, $P(r_0\bar{r}_1) = 0.2$, and $P(r_0 r_1) = 0.1$. States, for which no probability distribution is given, are assumed to have a uniform distribution over the input values.

Modes of Operation. Table 1 lists and describes the different modes of operation of the tool.

Table 1. Different Modes of Operation

SNo	Mode	Input Options	Description
1	prodADD	file1 file2 oFile	file1 and file2 are mean-payoff Automata(MPA). Product constructed using addition of weights
2	prodAPPEND	file1 file2 oFile	file1 and file2 are Lexicographic MPA (LMPA). Product constructed by appending the weights
3	prodMULT	file1 file2 oFile	file1 is a (L)MPA and file2 is a Safety Automaton. Product constructed by multiplication of weights
4	solveMPG	file1 file2 oFold	file1 is a MPA and file2 is a Safety Automaton. QUASY optimizes the worst case mean-payoff performance of the system and outputs an optimal Mealy machine
5	solveDPG	file1 file2 λ oFold	λ is the discount factor. QUASY optimizes the worst case discounted performance of the system and outputs an optimal Mealy machine
6	solveLMPGr	file1 file2 oFold	file1 is LMPA. Reduction to solveMPG case using a suitable base
7	solveLMPGc	file1 file2 oFold	file1 is LMPA. Similar to mode SolveLMPGr. Performs indirect comparison of value vectors using successive division by the base
8	solveLMPGl	file1 file2 oFold	file1 is LMPA. Similar to mode SolveLMPGc. Performs lexicographic comparison and carry shifting
9	solveMDP	file1 file2 fdist oFold	fdist is the distribution file for input minterms. QUASY optimizes the average case performance of the system and outputs an optimal Mealy machine
10	solveMDPP	file1 fileP fdist oFold	fileP is a Parity Automaton. QUASY optimizes the average case performance of the system and outputs two optimal Mealy machines to be combined with a counter to realize the actual system
11	solveMDPext	file1 file2 fdist oFold	Similar to mode SolveMDP. Uses the external C++ based MDP Solver.
12	solveMDPPext	file1 fileP fdist oFold	Similar to mode SolveMDPP. Uses the external C++ based MDP Solver.

Output of the tool. QUASY synthesizes an optimal Mealy machine (or two Mealy machines in the case of parity specifications). The output file (listed below) are stored in the folder that is given as the last argument to the tool:

- *Prod.gff* : product of the quantitative and qualitative specifications given to the tool.
- *Game.gff* : 2-player game graph (played between system and environment) obtained from the product.
- *OptimalStrategy.gff* : game graph comprising of the edges corresponding to the optimal strategy fixed by the system.
- *MealyMachine.gff* : an optimal Mealy machine corresponding to mean-payoff (or discounted payoff) measure. This machine might be optimal with respect to average case or worst case performance evaluation (decided by the mode of operation).

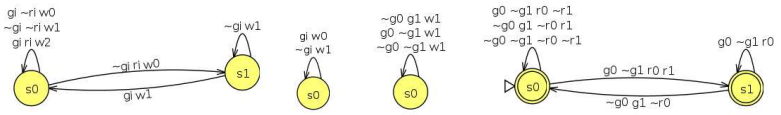

Fig. 2. (a) A_i of quick reaction; (b) A_i of by default low; (c) A_φ for mutual exclusion; (d) worst-case optimal controller

- *MealyMachineParity.gff* : the Mealy machine corresponding to the shortest stochastic path from each state to a state with the minimum even priority in the mean-payoff parity MDP.
- All inputs to the tool (the specifications and the distribution used) are also copied into this folder for further evaluation.

3 Examples

We use a resource controller to show the different combinations and the input format of the specifications the tool accepts. The controller uses two input signals r_0, r_1 and two output signals g_0, g_1 to controls the access of two clients to a shared resource. Client i requests the resource by setting signal r_i to high. The controller grants the resource to Client i by raising the output signal g_i.

Basic specifications. Assume we require that the clients have mutually exclusive access to the resource, i.e., $\varphi = \texttt{always}(\neg g_0 \vee \neg g_1)$. Assume we would also like to optimize the reaction time of the controller. We build a quantitative specification by combining instances of the property ψ_i that captures the reaction time of the controller to Client i. E.g., for the property ψ_i, we give the mean-payoff automaton shown in Figure 2(a) (in the format accepted by our tool). The automaton assigns to each trace a value corresponding to the average distance between request r_i and g_i. The shorter the distance in a trace, the larger the value of the trace.

Worst-case optimal controller. Assume we aim for a controller that gives priority to the client with the lowest number. Then, we take an instance of the specification ψ_i for each client i and build a lexicographic mean-payoff automaton B using QUASY. Now, we ask the tool to build a system that satisfy the mutual exclusion property (given by A_φ) and is optimal for specification B under an adversary environment. At every step, the resulting system grants the resource to the highest priority client that either has a pending grant or is sending a request in this time step. Assume **w0.gff** and **w1.gff** are the files for the automaton A_0 and A_1 shown in Figure 2(a), and **safe2.gff** is the file holding to the mutual exclusion specification for 2 clients (shown in Figure 2(c)). Then, QUASY produces the Mealy machine shown in Figure 2(d) (and stores it in the output file results/MealyMachine.gff) if used with following arguments:

```
scala quasy.Main prodAPPEND ./w0.gff ./w1.gff ./w01.gff
scala quasy.Main solveLMPGr ./w01.gff ./safe2.gff results/
```

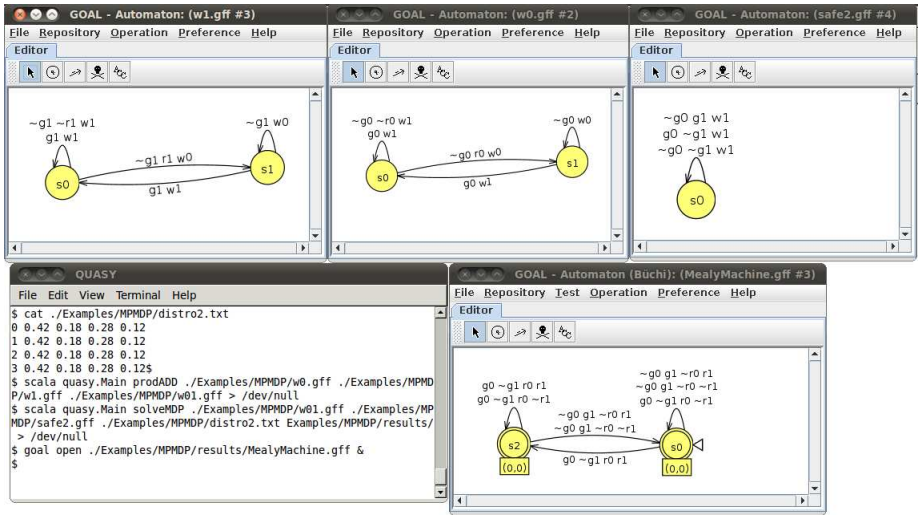

Fig. 3. QUASY I/O for 2-client Average-case optimal controller

Building the product specification, solving the game, and constructing the optimal system takes less than a minute.

Average-case optimal controller. If we want to construct a controller that optimizes the average-case behavior, we start again with two instances of the quantitative specification ψ_i. We build a mean-payoff automaton B (using QUASY) that computes to the sum of the weights. The qualitative specification is again the automaton A_φ ensuring mutually exclusive grants. We assume that client 0 sends a request with constant probability 0.4 and client 1 with 0.3. This gives the following distribution over the minterm in $2^{\{r_0,r_1\}}$: 0.42, 0.18, 0.28, and 0.12. The optimal machine constructed by QUASY behaves like a priority-driven controller that always grants the resource to the client that is more likely to send a request, if this client is requesting it, otherwise the resource is granted to the other client. The Mealy machine and the used input specifications are shown in Figure 3.

Average-case optimal fair controller. We require the controller to give mutually exclusive grants and to be fair, i.e., every client that sends a request can eventually access to the resource. We write this property formally as $\forall i : \texttt{always}(r_i \rightarrow \texttt{eventually}(g_i))$ and use GOAL to translate it into a parity automaton. We aim for a controller that sets the grant signals to low by default. We build the quantitative specification from instances of the mean-payoff automaton A_i shown in Figure 2(b). For every client i, we take one copy of A_i and sum the weights with the input distribution being same as in the previous example. The Mealy machine and the used input specifications are shown in Figure 4. The optimal mealy machine for mean payoff does not grant any request at all since this strategy gives maximum reward. The optimal mealy machine for parity specification delays a request and grants the pending request as soon as possible (state $s2$ corresponds to *pending request **r1*** and $s3$ corresponds to *pending request **r0***). It toggles between which one to delay after every execution when no request is pending (moving from states $s0$ to $s1$

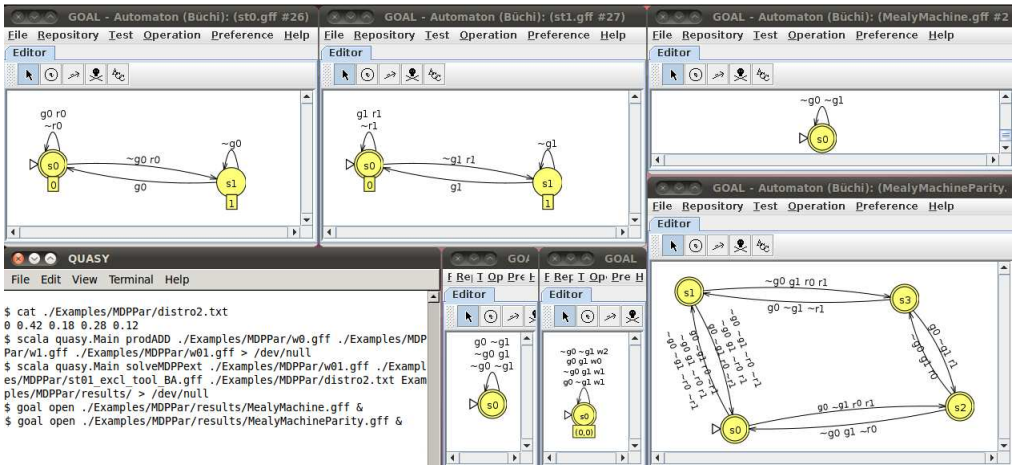

Fig. 4. QUASY I/O for 2-client Average-case optimal controller

and back to $s0$). The actual system is a combination of these two machines with a counter as mentioned earlier. The synthesis process for 2 clients took around 3 seconds, given the input specification comprising of 8 states, the MDP with 26 states, producing two mealy machines comprising of 8 and 4 states respectively. For 3 clients with input specification of size 22 producing an MDP of size 198, QUASY synthesized the two mealy machines of sizes 20 and 10 respectively in 15 seconds. Note that however, creating parity specifications using GOAL (intersection of buchi automata followed by determinization) is a very expensive process and major portion of the time taken by the synthesis process goes into building such specifications for multiple clients. Once we have such specifications, QUASY can quickly optimizate the corresponding measure and find the best reactive system.