# Gist: A Solver for Probabilistic Games

Krishnendu Chatterjee[1], Thomas A. Henzinger[1], Barbara Jobstmann[2], and Arjun Radhakrishna[1]

[1] IST Austria (Institute of Science and Technology Austria)
[2] CNRS/Verimag, France

**Abstract.** Gist is a tool that (a) solves the qualitative analysis problem of turn-based probabilistic games with $\omega$-regular objectives; and (b) synthesizes reasonable environment assumptions for synthesis of unrealizable specifications. Our tool provides the first and efficient implementations of several reduction-based techniques to solve turn-based probabilistic games, and uses the analysis of turn-based probabilistic games for synthesizing environment assumptions for unrealizable specifications.

## 1 Introduction

Gist (Game solver from IST) is a tool for (a) qualitative analysis of *turn-based probabilistic games ($2^1\!/\!_2$-player games)* with $\omega$-regular objectives, and (b) computing environment assumptions for synthesis of unrealizable specifications. The class of $2^1\!/\!_2$-player games arise in several important applications related to verification and synthesis of reactive systems. Some key applications are: (a) synthesis of stochastic reactive systems; (b) verification of probabilistic systems; and (c) synthesis of unrealizable specifications. We believe that our tool will be useful for the above applications.

**$2^1\!/\!_2$-player games.** $2^1\!/\!_2$-player games are played on a graph by two players along with probabilistic transitions. We consider $\omega$-regular objectives over infinite paths specified by parity, Rabin and Streett (strong fairness) conditions that can express all $\omega$-regular properties such as safety, reachability, liveness, fairness, and most properties commonly used in verification. Given a game and an objective, our tool determines whether the first player has a strategy to ensure that the objective is satisfied with probability 1, and if so, it constructs such a witness strategy. Our tool provides the first implementation of qualitative analysis (probability 1 winning) of $2^1\!/\!_2$-player games with $\omega$-regular objectives.

**Synthesis of environment assumptions.** The synthesis problem asks to construct a finite-state reactive system from an $\omega$-regular specification. In practice, initial specifications are often unrealizable, which means that there is no system that implements the specification. A common reason for unrealizability is that assumptions on the environment of the system are incomplete. The problem of

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

correcting an unrealizable specification $\Psi$ by computing an environment assumption $\Phi$ such that the new specification $\Phi \to \Psi$ is realizable was studied in [3]. The work [3] constructs an assumption $\Phi$ that constrains only the environment and is as weak as possible. Our tool implements the algorithms of [3]. We believe our implementation will be useful in analysis of realizability of specifications and computation of assumptions for unrealizable specifications.

## 2 Definitions

We first present the basic definitions of games and objectives.

**Game graphs.** A *turn-based probabilistic game graph* ($2^1\!/\!_2$*-player game graph*) $G = ((S, E), (S_0, S_1, S_P), \delta)$ consists of a directed graph $(S, E)$, a partition ($S_0$, $S_1$,$S_P$) of the finite set $S$ of states, and a probabilistic transition function $\delta$: $S_P \to \mathcal{D}(S)$, where $\mathcal{D}(S)$ denotes the set of probability distributions over the state space $S$. The states in $S_0$ are the *player*-0 states, where player 0 decides the successor state; the states in $S_1$ are the *player*-1 states, where player 1 decides the successor state; and the states in $S_P$ are the *probabilistic* states, where the successor state is chosen according to the probabilistic transition function $\delta$. We assume that for $s \in S_P$ and $t \in S$, we have $(s, t) \in E$ iff $\delta(s)(t) > 0$. The *turn-based deterministic game graphs* (*2-player game graphs*) are the special case of the $2^1\!/\!_2$-player game graphs with $S_P = \emptyset$.

**Objectives.** We consider the three canonical forms of $\omega$-regular objectives: Streett and its dual Rabin objectives; and parity objectives. The Streett objective consists of $d$ request-response pairs $\{ (Q_1, R_1), (Q_2, R_2), \ldots, (Q_d, R_d) \}$ where $Q_i$ denotes a request and $R_i$ denotes the corresponding response (each $Q_i$ and $R_i$ are subsets of the state space). The objective requires that if a request $Q_i$ happens infinitely often, then the corresponding response must happen infinitely often. The Rabin objective is its dual. The parity (or Rabin-chain objective) is the special case of Streett objectives when the set of request-responses $Q_1 \subset R_1 \subset Q_2 \subset R_2 \subset Q_3 \subset \cdots \subset Q_d \subset R_d$ form a chain.

**Qualitative analysis.** The qualitative analysis for $2^1\!/\!_2$-player games is as follows: the input is a $2^1\!/\!_2$-player game graph, and an objective $\Phi$ (Streett, Rabin or parity objective), and the output is the set of states such that player 0 can ensure $\Phi$ with probability 1. For detailed description of game graphs, plays, strategies, objectives and notion of winning see [1]. We focus on qualitative analysis because: a) In applications like synthesis the relevant analysis is qualitative analysis: the goal is to synthesize a system that behaves correctly with probability 1; (b) Qualitative analysis for probabilistic games is independent of the precise probabilities, and thus robust with imprecision in the exact probabilities (hence resilient to probabilistic modeling errors). The qualitative analysis can be done with discrete graph theoretic algorithms. Thus qualitative analysis is more robust and efficient, and our tools implements these efficient algorithms.

## 3 Tool Implementation

Our tool presents a solution of the following two problems.

**Qualitative analysis of $2\frac{1}{2}$-player games.** Our tool presents the first implementation for the qualitative analysis of $2\frac{1}{2}$-player games with Streett, Rabin and parity objectives. We have implemented the linear-time reduction for qualitative analysis of $2\frac{1}{2}$-player Rabin and Streett games to 2-player Rabin and Streett games of [2], and the linear-time reduction for $2\frac{1}{2}$-player parity games to 2-player parity games of [4]. The 2-player Rabin and Streett games are solved by reducing them to the 2-player parity games using the LAR (latest appearance records) construction [5]. The 2-player parity games are solved using the tool PGSolver [6].

**Environment assumptions for synthesis.** Our tool implements a two-step algorithm for computing the environment assumptions as presented in [3]. The algorithm operates on the game graph that is used to answer the realizability question. First, a safety assumption that removes a minimal set of environment edges from the graph is computed. Second, a fairness assumption that puts fairness conditions on some of the remaining environment edges is computed. The problem of finding a minimal set of fair edges is computationally hard [3], and a reduction to $2\frac{1}{2}$-player games was presented in [3] to compute a locally minimal fairness assumption. The details of the implementation are as follows: given an LTL formula $\phi$, the conversion to an equivalent deterministic parity automaton is achieved through GOAL [8]. Our tool then converts the parity automaton into a 2-player parity game by splitting the states and transitions based on input and output symbols. Our tool then computes the safety assumption by solving a safety model-checking problem. The computation of the fairness assumption is achieved in the following steps:

– Convert the parity game with fairness assumption into a $2\frac{1}{2}$-player game.
– Solve the $2\frac{1}{2}$-player game (using our tool) to check whether the assumption is sufficient (if so, go to the previous step with a weaker fairness assumption).

The synthesized system is obtained from a witness strategy of the parity game. The flow is illustrated in Figure 1.

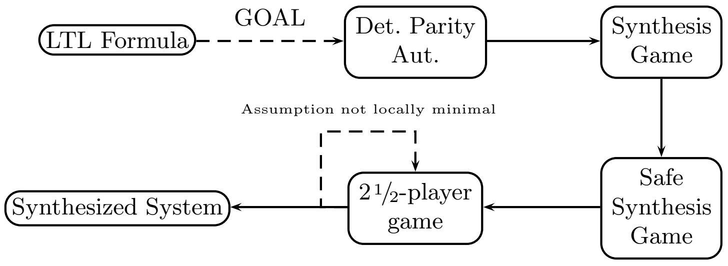
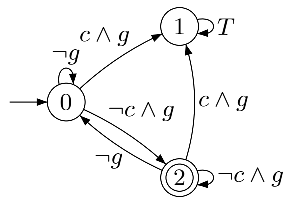
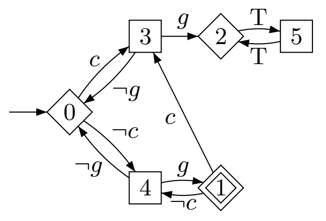


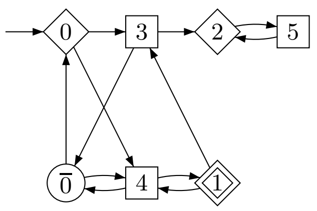
**Fig. 1.** An example illustrating the flow of the tool



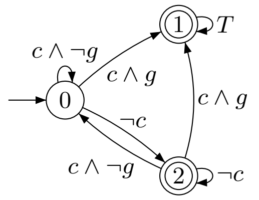
(a) Deterministic Parity Automaton



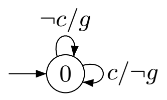
(b) 2-player Parity Game



(c) A $2\frac{1}{2}$-player game obtained for the fairness assumption which contains only $(0, 4)$



(d) Environment Assumption



(e) Transducer System

**Fig. 2.** An example that illustrates the tool flow

We illustrate the working of our tool on a simple example shown in Figure 2 Consider an LTL formula $\Phi = GF\texttt{grant} \wedge G(\texttt{cancel} \rightarrow \neg\texttt{grant})$, where $G$ and $F$ denote globally and eventually, respectively. The propositions `grant` and `cancel` are abbreviated as `g` and `c`, respectively. From $\Phi$ our tool obtains a deterministic parity automaton (Figure 2(a)) that accepts exactly the words that satisfies $\Phi$. The parity automaton is then converted into a parity game. In Figure 2(b), $\Box$ represents player 0 states and $\Diamond$ represents player 1 states. It can be shown that in this game no safety assumption required. We illustrate how to compute a locally minimal fairness assumption. Given an fairness assumption on edges, our tool reduces the game with the assumption to a $2^1/_2$-player parity game (see details in [3]). If the initial state in the $2^1/_2$-player game is in winning with probability 1 for player 0, then the assumption is sufficient. Figure 2(c) illustrates the $2^1/_2$-player game obtained with the fairness assumption on the edge $(0, 4)$. The $\bigcirc$ state is the probabilistic state with uniform distribution over its successors. The assumption on this edge is the minimal fairness assumption for the example. Our tool then converts this game back into an automaton to obtain the environment assumption as an automaton(Figure 2(d)). This assumption is equivalent to the formula $G(\neg(\texttt{cancel} \wedge \texttt{grant})) \implies GF(\neg\texttt{cancel})$. From a witness strategy in Figure 2(c) our tool obtains the system that implements the specification with the assumption (Figure 2(e)).

**Performance of** GIST**.** Our implementation of reduction of $2^1/_2$-player games to 2-player games is linear time and efficient, and the computationally expensive step is solving 2-player games. For qualitative analysis of $2^1/_2$-player games, GIST can handle game graphs of size that can be typically handled by tools solving 2-player games. Typical run-times for qualitative analysis of $2^1/_2$-player parity games of various sizes are summarized in Table 3. The games used were generated using the benchmark tools of PGSolver and then converting one-tenth of the states into probabilistic states.

| States | Edges | Runtime (sec.) | | |
|---|---|---|---|---|
| | | Avg. | Best | Worst |
| 1000 | 5000 | 1.17 | 0.63 | 1.59 |
| 5000 | 25000 | 15.94 | 11.10 | 19.46 |
| 10000 | 50000 | 51.43 | 39.38 | 62.61 |
| 20000 | 100000 | 282.24 | 267.40 | 310.11 |
| 50000 | 250000 | 2513.18 | 2063.40 | 2711.23 |

**Table 1.** Runtimes for solving $2^1/_2$-player parity games

In the case of synthesis of environment assumptions, the expensive step is the reduction of LTL formula to deterministic parity automata. Our tool can handle formulas that are handled by classical tools for translation of LTL formula to deterministic parity automata.

**Other features of** GIST**.** Our tool is compatible with several other game solving and synthesis tools: GIST is compatible with PGSolver and GOAL. Our tool provides a graphical interface to describe games and automata, and thus can also be used as a front-end to PGSolver to graphically describe games.

## References

1. K. Chatterjee. *Stochastic ω-Regular Games.* PhD thesis, UC Berkeley, 2007.
2. K. Chatterjee, L. de Alfaro, and T. A. Henzinger. The complexity of stochastic Rabin and Streett games. In *ICALP*, 2005.
3. K. Chatterjee, T. A. Henzinger, and B. Jobstmann. Environment assumptions for synthesis. In *CONCUR*, 2008.
4. K. Chatterjee, M. Jurdziński, and T.A. Henzinger. Quantitative stochastic parity games. In *SODA*, 2004.
5. Y. Gurevich and L. Harrington. Trees, automata, and games. In *STOC*, 1982.
6. M. Lange and O. Friedmann. The pgsolver collection of parity game solvers. Technical report, Institut für Informatik, Ludwig-Maximilians-Universität, 2009.
7. The JUNG Framework Development Team. JUNG: Java universal network/graph framework. `http://jung.sourceforge.net`.
8. Y. Tsay, Y. Chen, M. Tsai, W. Chan, and C. Luo. Goal extended: Towards a research tool for omega automata and temporal logic. In *TACAS*, 2008.

## 4 Appendix: Details of the Tool

Gist is available for download at `http://pub.ist.ac.at/gist` for Unix-based architectures. All the libraries that Gist uses are packaged along with it.

### 4.1 Dependencies and Architecture

**Language, tools and installation.** Gist is written in Scala and it uses several other tools. For the graphical interface to draw game graphs and automata it uses the JUNG library [7] for graph layout algorithms. For translation of an LTL formula to a deterministic parity automata it uses GOAL [8]. The solution of 2-player parity games is achieved by using PGSolver [6]. For compilation and installation: (a) an installation of the Scala compiler and runtime environment is required; (b) the PGSolver build process requires an OCaml compiler to be installed; and (c) GOAL and JUNG require a Java runtime environment to be installed.

**Source code.** The source code of Gist is composed mainly of five modules:

1. Module **newgames** mainly consists of the classes for probabilistic $\omega$-regular games, i.e. games with Büchi, coBüchi, Rabin, Streett and parity objectives. Each of these classes contains routines for the reduction of the $2\frac{1}{2}$-player version to the 2-player version which preserves the probability 1 winning region of Player 1. Each of these classes also returns a witness strategy for the player as required.
2. Module **specification** consists of classes implementing the specifications for the synthesis problem, i.e. LTL formulae, Büchi automata and parity automata. The class for LTL formulae contains a routine to convert LTL formulae into an equivalent nondeterministic Büchi automata and the class for Büchi automata has a routine for converting it into a deterministic parity automaton. The parity automata class can generate the synthesis game (by splitting transitions) for the automaton as described in [3].
3. Module **synthesis** contains the classes relevant to the process of synthesis. The class for synthesis games contains routines (a) to compute transducers implementing the specification; (b) to compute minimal safety assumption and locally minimal fairness assumption in case of an unrealizable specification; (c) to check whether user-specified assumptions are sufficient to make the specification realizable; and (d) to get the assumptions as a Streett automaton.
4. Modules **gui** and **cui** contain classes for graphical and text based user interfaces. Most of the functionality in the **cui** module is contained in the **Console** class, which interprets a command line. The **gui** module contains forms and windows to display various automata and games; and provide an interface for the various operations on them.
5. Module **basic** contains the definitions which are needed by all other packages, namely, the classes for alphabet, symbols and generic automata.

In addition to these, there are other routines to parse and write automata and game graphs in files in a format that can be used with GOAL.

## 4.2 User Manual

In this section we describe the usage of the graphical and text-based interface of the tool.
**Format of files.** The file format used by the tool is based on the format used by GOAL. The format for games and automata structures is presented below:

```
<structure label-on="transition" type=["game"|"fa"]>
  <alphabet type="propositional">
    <prop type=["input"|"output"]>TEXT</prop>
    ...
  </alphabet>
  <stateSet>
    <state sid="NUMERIC">
      [<player>[0|1|-1]</player>]
      [<label>TEXT</label>]
    </state>
    ...
  </stateSet>
  <transitionSet>
    <transition tid="NUMERIC">
      <from>NUMERIC(State ID)</from>
      <to>NUMERIC(State ID)</to>
      <read>TEXT(Symbol)</read>
    </transition>
    ...
  </transitionSet>
  <initialStateSet>
    <stateID>NUMERIC</stateID>
  </initialStateSet>
  <acc type="[buchi|parity|rabin|streett]">
    <accSet> %Ony one set for Buchi acceptance condition.
      <stateID>NUMERIC(State ID)</stateID>
      ...
    </accSet>
    <accSet> %Multiple sets for Parity acceptance condition.
             %One for each priority
      <stateID>NUMERIC(State ID)</stateID>
      ...
    </accSet>
    <accSet> %Multiple sets for Rabin and Streett acceptance conditions.
             %Different format from the other conditions.
      <E>
```

```
        <stateID>NUMERIC(State ID)</stateID>
        ...
      </E>
      <F>
        <stateID>NUMERIC(State ID)</stateID>
        ...
      </F>
    ...
    </accSet>
  </acc>
</structure>
```

**Graphical Interface.** The graphical interface for Gist consists of a window for each kind of game graph, automata, and formula the tool handles. When Gist is invoked, a window is shown with buttons for each kind. A window for a specific kind contains buttons that represent relevant actions that can be performed. There are also generic options such as saving and loading.

For automata and game graphs, the window contains an area in which the graph is laid out visually. The layout can be changed by dragging the vertices and the edges of the graph. Gist uses the layout algorithms of JUNG to automatically layout the graph. The layout algorithms can be chosen by right-clicking on the window and selecting **Layout** from a pop-up menu that appears. Also, sets of vertices or edges can be highlighted for other operations (such as finding sufficiency of assumptions containing these edges) by choosing **Highlight Mode** on the pop-up menu.

The tool also includes interfaces for building automata and games graphically. In these windows, one can insert states or edges into a structure by selecting the appropriate mode from the pop-up menu. When an edge is created, the user can label the edge appropriately. The alphabet for the symbols (for labeling edges) must be set before the edges are created. States and edges can also be deleted using the **Delete mode**.

**Text-based Interface.** The text-based interface for Gist is an interactive prompt. The user can define and use variable for any object. Variables need not be declared before use. All variable names need to begin with a $. The syntax for the statements is defined below.

```
Variable := $[a-zA-Z0-9]*
Statement :=   Variable               --Prints the value of the variable
             | Variable = Variable    --Assignment
             | Variable = Expression  --Assignment
Expression := Object Action
Object :=   "LTL" | "BuchiAutomaton" | "ParityAutomaton"
              | "SynthesisGame" | "StreettAutomaton" | "ParityGame"
              | "RabinGame" | "StreettGame"
Action := readFile ... | writeFile ... | help | ...
```

The "action" as seen in the above syntax definition varies depending upon the object. The **help** action for any object displays all the other actions available for this object along with an explanation.

All objects which represent games have the following actions: **winningRegion**, **cooperativeWinningRegion**, and **toDeterministicGame**. The action **winningRegion** takes an argument, either 0 or 1 (for a player), and computes the set of states from which the player wins with probability 1. The action **cooperativeWinningRegion** is invoked only for 2-player games, and it computes the set of states such that there is a path to satisfy the objective of player 0. The action **toDeterministicGame** is invoked on $2\frac{1}{2}$-player games and it returns a 2-player game in which probability 1 winning of player 0 is preserved. In addition, the action **winningStrategy** computes the winning strategy of each player in 2-player games, and the probability 1 winning strategy in $2\frac{1}{2}$-player games.

The objects for Büchi automata have an action **toParityAutomaton** to convert it into equivalent deterministic parity automata. Similarly, the objects for LTL formulae and parity automata have actions to convert them into nondeterministic Büchi automata and Synthesis games respectively. The objects for Synthesis games have actions related to synthesis and computation of environment assumptions.

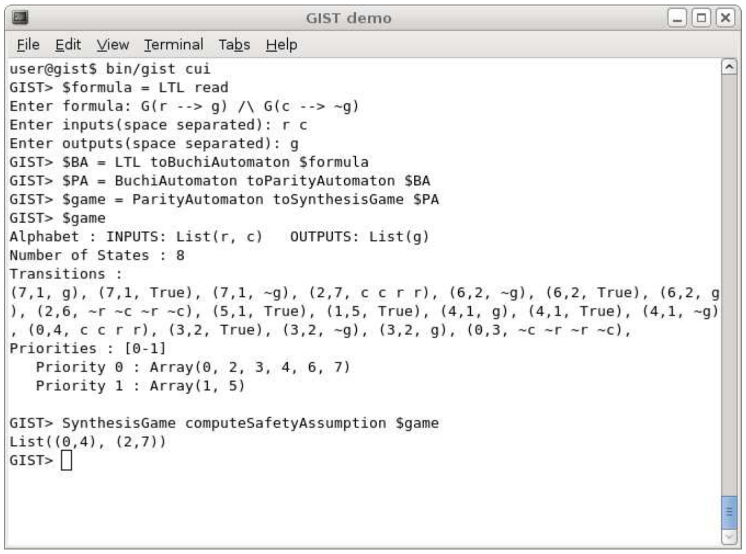
The text-based interface for GIST is also available online at `http://pub.ist.ac.at/gist`. Figure 3 shows the screenshot for the text-interface with input and output for Example 1 (described in the following subsection). Figure 4 shows the screenshot of the web interface for a similar example.

## 4.3 Examples to Illustrate the Usage of GIST

In this section, we present two examples to illustrate the usage of GIST. We have chosen small examples for the simplicity of the presentation to illustrate the usage of GIST. These examples demonstrate the usage of GIST for computation of environment assumptions for synthesis and uses solution of $2\frac{1}{2}$-player games. In these examples, we compute the assumptions for two unrealizable specifications given in LTL. Both the specifications are about request-response systems and are chosen to illustrate safety and fairness assumptions respectively.

*Example 1.* Consider a request-response system in which there are two inputs, `request` and `cancel`, and one output `grant`. Now, consider the specification $G(\texttt{request} \rightarrow \texttt{grant}) \wedge G(\texttt{cancel} \rightarrow \neg\texttt{grant})$. This specification is unrealizable: any input in which both `request` and `cancel` are set at the same time does not have an output which satisfies the specification. We can compute an environment assumption for this specification using GIST. Intuitively, we would want an assumption that says `request` and `cancel` must not be set at the same time provided the specification was not already violated earlier. We show that the assumption can be computed automatically by GIST.

To compute the assumption using GIST, we select LTL formula from the main window of options and then enter the formula above, specifying the inputs and



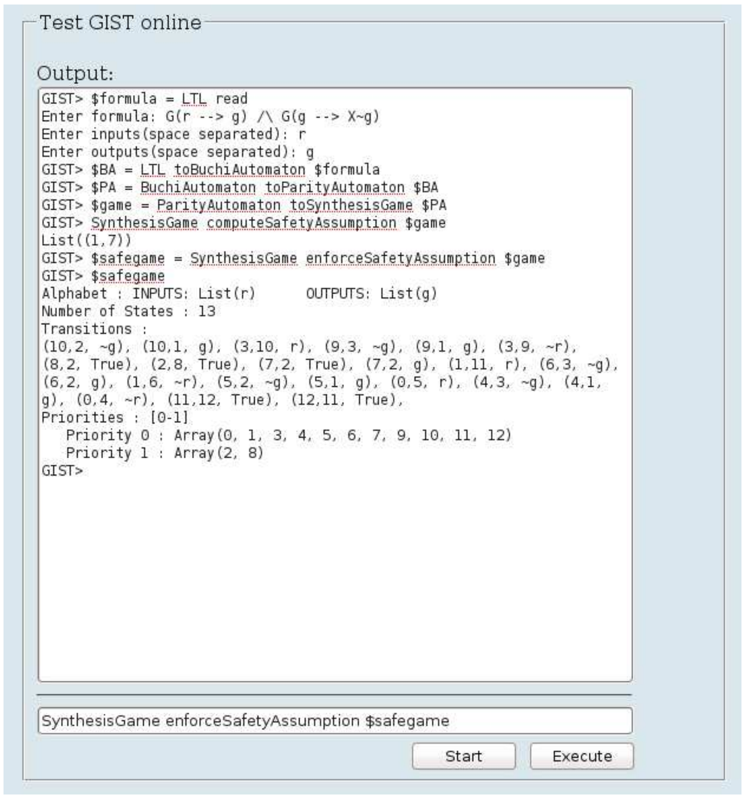
```
user@gist$ bin/gist cui
GIST> $formula = LTL read
Enter formula: G(r --> g) /\ G(c --> ~g)
Enter inputs(space separated): r c
Enter outputs(space separated): g
GIST> $BA = LTL toBuchiAutomaton $formula
GIST> $PA = BuchiAutomaton toParityAutomaton $BA
GIST> $game = ParityAutomaton toSynthesisGame $PA
GIST> $game
Alphabet : INPUTS: List(r, c)   OUTPUTS: List(g)
Number of States : 8
Transitions :
(7,1, g), (7,1, True), (7,1, ~g), (2,7, c c r r), (6,2, ~g), (6,2, True), (6,2, g
), (2,6, ~r ~c ~r ~c), (5,1, True), (1,5, True), (4,1, g), (4,1, True), (4,1, ~g)
, (0,4, c c r r), (3,2, True), (3,2, ~g), (3,2, g), (0,3, ~c ~r ~r ~c),
Priorities : [0-1]
   Priority 0 : Array(0, 2, 3, 4, 6, 7)
   Priority 1 : Array(1, 5)

GIST> SynthesisGame computeSafetyAssumption $game
List((0,4), (2,7))
GIST> 
```

**Fig. 3.** Example to illustrate the text-based interface

outputs. This formula is then converted into a nondeterministic Büchi automaton and then to a deterministic parity automaton, and finally to a synthesis game. In this game, we attempt to compute the safety assumption. The safety assumption is highlighted (green arrows in a box; (0,4) and (2,7)) as shown in Figure 5. As shown in Figure 5, the safety assumption includes all the edges where `request` and `cancel` are set at the same time. But, if there has been an instance of a `request` not being granted already, then there is no restriction on the inputs. This is the same assumption as was expected intuitively. Now, we can obtain a synthesis game where the safety assumption is enforced. In this new game, if the fairness assumption is computed the output shows no fairness assumption is necessary. A transducer that implements the modified specification can be obtained from the solution of this game.

*Example 2.* Consider the request-response system as in Example 1. But, with the specification $(GF\texttt{grant}) \wedge G(\texttt{cancel} \rightarrow \neg\texttt{grant})$. This specification says that we should have infinitely many grants and that at every step, if `cancel` is set, then there should be no grant at that step. This specification is also unrealizable as any input where the `cancel` is always set has no acceptable output. We can see that if `cancel` is not set always after a point, then the specification becomes realizable. This condition can be computed using Gist following the same steps as in the above example: first the tool finds that no safety assumption is necessary, and then it computes the fairness assumption in the synthesis game. The fairness assumption is computed internally by reduction to $2\frac{1}{2}$-player games. The fairness assumption is highlighted (by green arrow



```
GIST> $formula = LTL read
Enter formula: G(r --> g) /\ G(g --> X~g)
Enter inputs(space separated): r
Enter outputs(space separated): g
GIST> $BA = LTL toBuchiAutomaton $formula
GIST> $PA = BuchiAutomaton toParityAutomaton $BA
GIST> $game = ParityAutomaton toSynthesisGame $PA
GIST> SynthesisGame computeSafetyAssumption $game
List((1,7))
GIST> $safegame = SynthesisGame enforceSafetyAssumption $game
GIST> $safegame
Alphabet : INPUTS: List(r)      OUTPUTS: List(g)
Number of States : 13
Transitions :
(10,2, ~g), (10,1, g), (3,10, r), (9,3, ~g), (9,1, g), (3,9, ~r),
(8,2, True), (2,8, True), (7,2, True), (7,2, g), (1,11, r), (6,3, ~g),
(6,2, g), (1,6, ~r), (5,2, ~g), (5,1, g), (0,5, r), (4,3, ~g), (4,1,
g), (0,4, ~r), (11,12, True), (12,11, True),
Priorities : [0-1]
   Priority 0 : Array(0, 1, 3, 4, 5, 6, 7, 9, 10, 11, 12)
   Priority 1 : Array(2, 8)
GIST>
```

```
SynthesisGame enforceSafetyAssumption $safegame
```

**Fig. 4.** GIST web interface

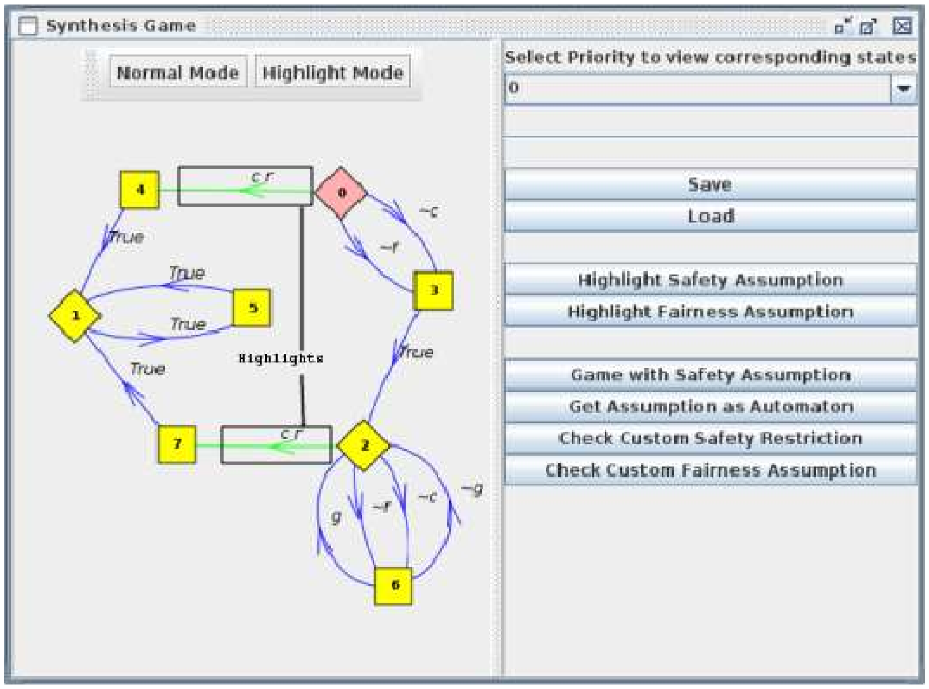

Synthesis Game
Normal Mode
Highlight Mode
Select Priority to view corresponding states
0
Save
Load
Highlight Safety Assumption
Highlight Fairness Assumption
Game with Safety Assumption
Get Assumption as Automaton
Check Custom Safety Restriction
Check Custom Fairness Assumption
Highlights
c r
True
¬c
¬r
g
¬g

**Fig. 5.** Example 1. The safety assumption is highlighted

in a box; (0,4)) in the screenshot Figure 6. The computed assumption can be interpreted as follows: the highlighted edge must be taken infinitely often if the source vertex of the edge is visited infinitely often. Translating this into propositions, it means that at any step, `cancel` cannot be set forever in the future.

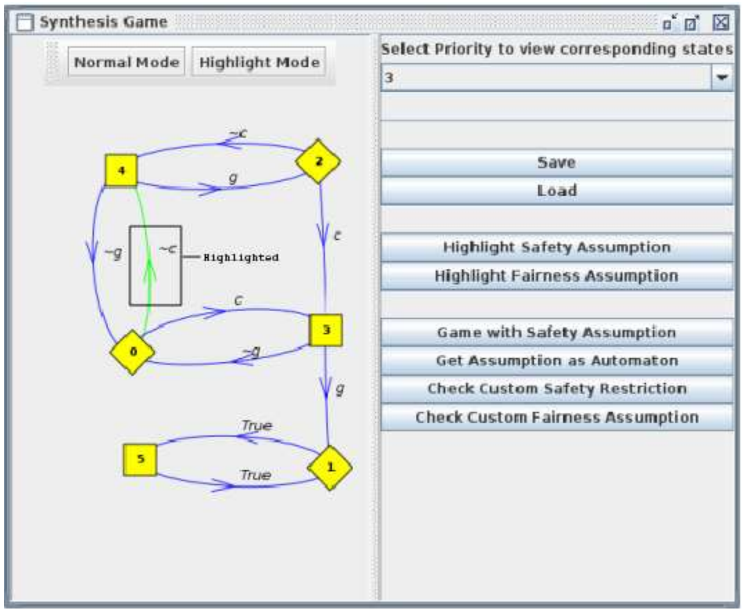

Synthesis Game
Normal Mode
Highlight Mode
~c
g
~g
~c
Highlighted
c
c
~g
g
True
True
4
2
3
0
1
5
Select Priority to view corresponding states
3
Save
Load
Highlight Safety Assumption
Highlight Fairness Assumption
Game with Safety Assumption
Get Assumption as Automaton
Check Custom Safety Restriction
Check Custom Fairness Assumption

**Fig. 6.** Example 2. The fairness assumption is highlighted