GAVS: Game Arena Visualization and Synthesis

Chih-Hong Cheng[1], Christian Buckl[2],
Michael Luttenberger[1], and Alois Knoll[1]

[1] Department of Informatics, Technischen Universität München
Boltzmann Str. 3, Garching D-85748, Germany
[2] fortiss GmbH, Guerickestr.25, D-80805 München, Germany
{chengch,knoll}@in.tum.de, buckl@fortiss.org, luttenbe@model.in.tum.de

Abstract. Reasoning on the properties of computer systems can often be reduced to deciding the winner of a game played on a finite graph. In this paper, we introduce GAVS, an open-source tool for the visualization of some of the most fundamental games on finite graphs used in theoretical computer science, including, e.g., reachability games and parity games. The main purpose of GAVS is educational, a fact which is emphasized by the graphical editor for both defining game graphs and also visualizing the computation of the winning sets. Nevertheless, the underlying solvers are implemented with scalability in mind using symbolic techniques where applicable.

1 Introduction

We present GAVS[1], a tool which allows to visualize and solve some of the most common two-player games encountered in theoretical computer science, amongst others reachability, Büchi and parity games.

The importance of these games results from the reduction of different questions regarding the analysis of computer systems to two-player games played on finite graphs. For example, liveness (safety) properties can easily be paraphrased as a game play on the control-flow graph of a finite program: will the system always (never) visit a given state no matter how the user might interact with the system? The resulting games are usually called (co-)reachability if only finite program runs are considered. Similarly, one obtains (co-)Büchi games when considering infinite runs. Another well-known example is the class of parity games which correspond to the model-checking problem of μ-calculus. Advantages of the game theoretic reformulation of these analysis problems are the easier accessibility and the broadened audience.

The main goal of GAVS is to further enhance these advantages by providing educational institutions with a graphical tool for both constructing game graphs and also visualizing standard algorithms for solving them step-by-step. Still, symbolic methods, where applicable, have been used in the implementation in order to ensure scalability.

[1] Short for "Game Arena Visualization and Synthesis".

A. Bouajjani and W.-N. Chin (Eds.): ATVA 2010, LNCS 6252, pp. 347–352, 2010.
© Springer-Verlag Berlin Heidelberg 2010

GAVS is released under the GNU General Public License (v3) and allows for an easy extension to novel algorithms. The software package is available at `http://www6.in.tum.de/~chengch/gavs`

2 Preliminaries and Supported Games

We briefly recapitulate the most important definitions regarding two-player games on finite graphs before explicitly enumerating the games supported by GAVS.

A *game graph* or *arena* is a directed graph $G = (V_0 \uplus V_1, E)$ whose nodes are partitioned into two classes V_0 and V_1. We only consider the case of two players in the following and call them player 0 and player 1 for simplicity. A *play* starting from node v_0 is simply a maximal path $\pi = v_0 v_1 \ldots$ in G where we assume that player i determines the *move* $(v_k, v_{k+1}) \in E$ if $v_k \in V_i$ $(i \in \{0, 1\})$. With $\mathrm{Occ}(\pi)$ / $\mathrm{Inf}(\pi)$ we denote the set of nodes visited / visited infinitely often by a play π. A *winning condition* defines when a given play π is *won* by player 0; if π is not won by player 0, it is won by player 1. A node v is won by player i if player i can always choose his moves in such a way that he wins any resulting play starting from v; the sets of nodes won by player i are denoted by W_i $(i \in \{0, 1\})$.

GAVS supports the computation of W_0, W_1 for the following games:

1. Games defined w.r.t. a set of target states F:
   - *Reachability game*: player 0 wins a play π if $\mathrm{Occ}(\pi) \cap F \neq \emptyset$.
   - *Co-reachability (safety) game*: player 0 wins a play π if $\mathrm{Occ}(\pi) \cap F = \emptyset$.
   - *Büchi game*: player 0 wins a play π on G if $\mathrm{Inf}(\pi) \cap F \neq \emptyset$.
2. Games defined w.r.t. a coloring $c : V \to \mathbf{N}$ of G:
   - *Weak-parity game*: player 0 wins a play π if $\max(c(\mathrm{Occ}(\pi)))$ is even.
   - *Parity game*: player 0 wins a play π if $\max(c(\mathrm{Inf}(\pi)))$ is even.
3. Games defined w.r.t. finite families $E = \{E_1, \ldots, E_k\}$ and $F = \{F_1, \ldots, F_k\}$ of subsets of V:
   - *Staiger-Wagner game*: player 0 wins a play π if $\mathrm{Occ}(\pi) \in F$.
   - *Muller game*: player 0 wins a play π if $\mathrm{Inf}(\pi) \in F$.
   - *Streett game*: player 0 wins a play π if $\bigwedge_{j=1}^{k} \mathrm{Inf}(\pi) \cap F_i \neq \emptyset \Rightarrow \mathrm{Inf}(\pi) \cap E_i \neq \emptyset$.

It is well-known that all these games are determined, i.e., $W_0 \cup W_1 = V$. We refer the reader to [4] for a thorough treatment of these games and their relationship to each other.

We close this section by recalling the definition of attractor which is at the heart of many algorithms for solving the games mentioned above: for $i \in \{0, 1\}$ and $X \subseteq V$, the map $\mathrm{attr}_i(X)$ is defined by

$$\mathrm{attr}_i(X) := X \cup \{v \in V_i \mid vE \cap X \neq \emptyset\} \cup \{v \in V_{1-i} \mid \emptyset \neq vE \subseteq X\},$$

i.e., $\mathrm{attr}_i(X)$ extends X by all those nodes from which either player i can move to X within one step or player $1 - i$ cannot prevent to move within the next step. (vE denotes the set of successors of v.) Then $\mathrm{Attr}_i(X) := \bigcup_{k \in \mathbf{N}} \mathrm{attr}_i^k(X)$ contains all nodes from which player i can force any play to visit the set X.

3 Software Architecture

GAVS consists of three major parts: (a) a graphical user interface (GUI), (b) solvers for different winning conditions, and (c) a two-way translation function between graphical representations and internal formats acceptable by the engine. The GUI is implemented using the JGraphX library [3] and acts as a front-end for the different game solvers. Every game solver is implemented as a separate back-end. We give a brief description of their implementation where we group back-ends sharing similar implementation approaches:

- Symbolic techniques: Algorithms of this type are implemented using JDD [2], a Java-based package for binary decision diagrams (BDDs). The supported winning conditions include reachability, safety, Büchi, weak-parity, and Staiger-Wagner.
- Explicit state operating techniques: Algorithms of this type are implemented based on direct operations over the graph structure.
  - **Parity game.** In addition to an inefficient version which enumerates all possibilities using BDDs, we have implemented the discrete strategy improvement algorithm adapted from [5]; the algorithm allows the number of nodes/vertices to exceed the number of colors in a game.
- Reduction techniques: Algorithms of these games are graph transformations to other games with different types of winning conditions.
  - **Muller game.** The algorithm performs reductions to parity games using the latest appearance record (LAR) [4].
  - **Streett game.** The algorithm performs reductions to parity games using the index appearance record (IAR) [4].

GAVS can easily be extended by additional game solvers. We briefly sketch how this can be done.

Every game engine interacts with the GUI via the method `EditorAction.java`. When invoked, an intermediate object of the data type `mxGraph` (the predefined data structure for a graph in JGraphX) is retrieved. We then translate the graph from `mxGraph` to a simpler structure (defined in `BuechiAutomaton.java`) which offers a simple entry for users to extend GAVS with new algorithms.

For symbolic algorithms, we offer methods for translating the graph to BDDs. After the algorithm is executed, GAVS contains mechanisms to annotate the original game graph via the winning region encoded by a BDD, and visualize the result. To redirect the result of synthesis back to the GUI, a map structure with the type `HashMap<String, HashSet<String>>` is required, where the key is the source vertex and the value set contains destination vertices, describing the edges that should be labeled by GAVS. For explicit state operating algorithms, mechanisms follow analogously.

4 Example: Working with GAVS

Due to page limits, we give two small yet representative examples on using GAVS. We refer the reader to the GAVS homepage [1] for a full-blown tutorial, and serveral examples.

4.1 Example: Safety Games

We give a brief description of how to use GAVS for constructing a safety game and solving it step-by-step with the assist of Figure 1.

In the first step, the user constructs the game graph by simply drawing it using the graphical interface, similar to Figure 1-a: the states V_1 are called plant states and are of rectangular shape, while the states V_0 are called control states and are of circular shape.

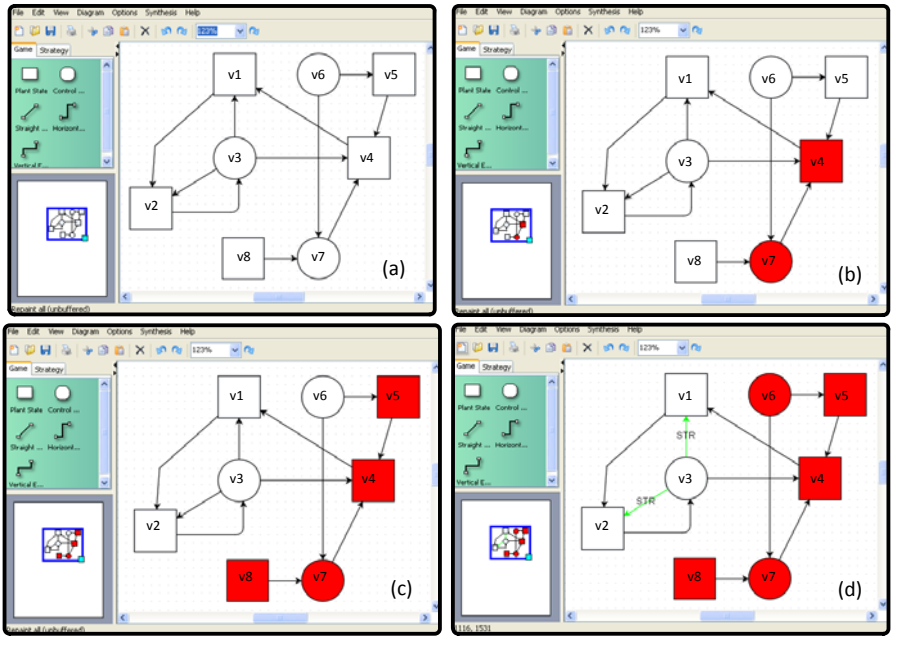

Fig. 1. An example for constructing and executing a safety game

Next, the user specifies the target nodes F, i.e., the nodes which player 0 tries to avoid. GAVS supports both graphical and textual methods[2]. In Figure 1-b, states v_4 and v_7 are painted by the user with red color, offering an graphical description of risk states.

[2] Graphical specification is only available with reachability, safety, and Büchi winning conditions; for weak-parity and parity games, colors of vertices can also be labeled directly on the game graph.

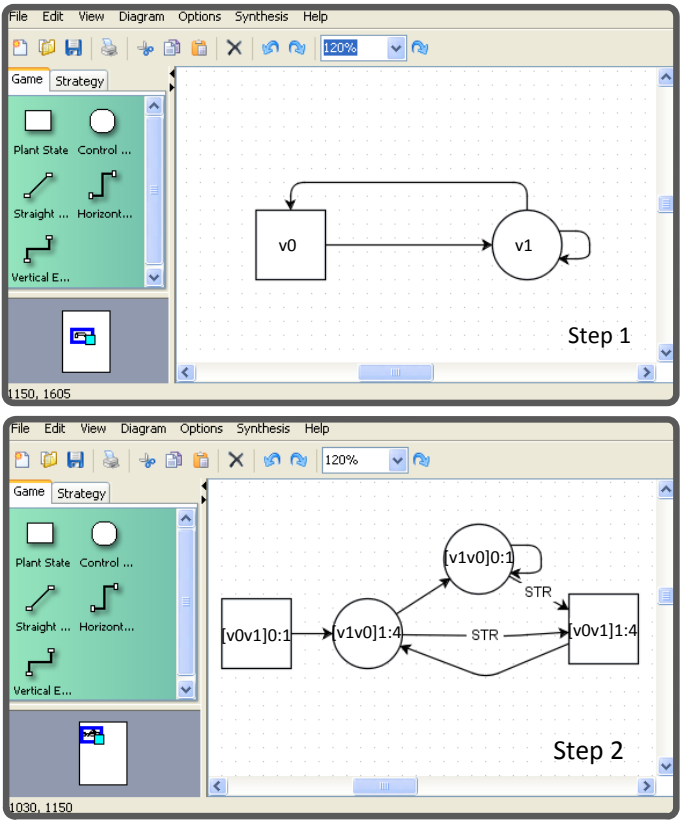

Fig. 2. A Muller game and its synthesized result in the form of the parity game

Finally, GAVS can be used to either compute the winning set W_1 immediately or to guide the user through the computation of W_1 step-by-step: the computation of $\text{Attr}_1(\{v_4, v_7\}) = \text{attr}_1^2(\{v_4, v_7\})$ is shown in Figures 1-b to 1-d with the corresponding nodes highlighted in red. For games with positional strategies, a winning strategy is shown automatically on the graph (edges labeled with "STR"), for instance, Figure 1-d shows the winning strategy for the safety game: edges (v_3, v_2) and (v_3, v_1) are highlighted as safe transitions.

4.2 Example: Muller Games

For Muller and Streett games, instead of generating strategies directly, game reductions are implemented for clearer understanding regarding the meaning of strategies. This is due to the fact that the generated FSM strategies for Muller and Streett games require memory which is factorial to the number of states, making them difficult for users to comprehend.

We indicate how Muller game reduction is applied in GAVS. Consider Figure 2, where a Muller game is shown in step 1 with the winning condition $\{\{v_0, v_1\}\}$. The reduced parity game generated by GAVS is shown in step 2, where each vertex is of the format `"[Vertex Permutation] LAR index : Color"`. By interpreting the strategy using LAR, it is clear that for player 0, the generated strategy is a positional move (v_1, v_0) on vertex v_1.

5 Concluding Remarks

As the name of the tool suggests, Game Arena Visualization and Synthesis (GAVS), which is served for both research and educational purposes, is designed to provide a unified platform to connect visualization and synthesis for games. We are also interested in the visualization and the synthesis of pushdown games, which will be our next step.

Acknowledgments. We thank Javier Esparza, Michael Geisinger, Jia Huang, and Shun-Ching Yang for their preliminary evaluations on GAVS. The first author is supported by the DFG Graduiertenkolleg 1480 (PUMA).

References

1. GAVS: Game Arena Visualization and Synthesis, `http://www6.in.tum.de/~chengch/gavs/`
2. JDD: Java BDD and Z-BDD library, `http://javaddlib.sourceforge.net/jdd/`
3. JGraphX: Java Graph Drawing Component, `http://www.jgraph.com/jgraph.html`
4. Grädel, E., Thomas, W., Wilke, T. (eds.): Automata, Logics, and Infinite Games: A Guide to Current Research. LNCS, vol. 2500. Springer, Heidelberg (2002)
5. Vöge, J., Jurdziński, M.: A discrete strategy improvement algorithm for solving parity games. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 202–215. Springer, Heidelberg (2000)