A Constraint-based Approach to Solving Games on Infinite Graphs

Tewodros Beyene\(^1\), Swarat Chaudhuri\(^3\), Corneliu Popeea\(^1\), and Andrey Rybalchenko\(^{1,2}\)

\(^1\)Technische Universität München
\(^2\)Microsoft Research Cambridge
\(^3\)Rice University

PUMA workshop
Alacati, Turkey

October 1, 2013
Motivation

Many fundamental questions reduce to solving turn-based graph games:

- modeling interactions between a controller and its environment
- verifying a branching-time property of a system
- synthesizing a reactive system from a temporal specification
- ...

In turn-based graph games:

- two players take turns
- a token is moved along the edges of a graph

Do the visited nodes satisfy a certain winning condition?
Many fundamental questions reduce to solving turn-based graph games:
- modeling interactions between a controller and its environment
- verifying a branching-time property of a system
- synthesizing a reactive system from a temporal specification
- ...

In turn-based graph games
- two players take turns
- a token is moved along the edges of a graph

Do the visited nodes satisfy a certain winning condition?
Many fundamental questions reduce to solving turn-based graph games:

- modeling interactions between a controller and its environment
- verifying a branching-time property of a system
- synthesizing a reactive system from a temporal specification
- ...

In turn-based graph games

- two players take turns
- a token is moved along the edges of a graph

Do the visited nodes satisfy a certain winning condition?
Motivation (cont)

- Majority of algorithmic approaches focus on decidable classes.
  - such as games on finite graphs
  - limits the scope of the applications
- To analyse and synthesize infinite-state systems:
  - symbolic, abstraction-based algorithms
  - solve games on infinite state spaces
- The talk is about an algorithmic approach based on automated deduction for solving games over infinite-state symbolic transition systems.
Motivation (cont)

- Majority of algorithmic approaches focus on decidable classes.
  - such as games on finite graphs
  - limits the scope of the applications
- To analyse and synthesize infinite-state systems:
  - symbolic, abstraction-based algorithms
  - solve games on infinite state spaces
- The talk is about an algorithmic approach based on automated deduction for solving games over infinite-state symbolic transition systems.
A 'Challenge' Example: Cinderella-Stepmother game

- Between Cinderella and her Stepmother.
- Involves 5 buckets arranged in a circle.
  - With a constant $c$ bucket capacity
  - all buckets empty initially
- Stepmother starts each round of play.
  - Splits 1 unit of additional water among the five buckets
  - If overflow in any one of the buckets - Stepmother wins
- If not, Cinderella empties two adjacent buckets.
  - If the game goes on forever without overflow - Cinderella wins
- More challenging for $1.5 \leq c < 3$. 
A 'Challenge' Example: Modeling the game

- Set of variables: $v = (b_1, b_2, b_3, b_4, b_5)$.
- Initial condition:
  \[\text{init}(v) = (b_1 = 0 \land \cdots \land b_5 = 0).\]
- Transition relation of Stepmother:
  \[\text{stepmother}(v, v') = (b'_1 + \cdots + b'_5 = b_1 + \cdots + b_5 + 1 \land b'_1 \geq b_1 \land \cdots \land b'_5 \geq b_5).\]
- Transition relation of Cinderella:
  \[\text{cinderella}(v, v') = \bigvee_{i \in \{1\ldots5\}} \left( b'_i = 0 \land b'_{(i+1)\%5} = 0 \right. \left. \land \left( \bigwedge_{j \in \{1\ldots5\}} \left( j \neq i \land j \neq (i + 1)\%5 \rightarrow b'_j = b_j \right) \right) \right).\]
- Overflow condition:
  \[\text{overflow}(v) = (b_1 > c \lor \cdots \lor b_5 > c).\]
A 'Challenge' Example: Type of games

Depending on the objective of the player we compute a strategy for.

- **Safety games:**
  - requires only states with a certain property to be visited by all the plays
  - e.g. the property $G(\neg \text{overflow}(v))$ for Cinderella

- **Reachability games:**
  - requires a state with a certain property to be visited eventually by all the plays
  - e.g. the property $F(\text{overflow}(v))$ for Stepmother
Depending on the objective of the player we compute a strategy for:

- **Safety games:**
  - requires only states with a certain property to be visited by all the plays
  - e.g. the property $G(\neg \text{overflow}(v))$ for Cinderella

- **Reachability games:**
  - requires a state with a certain property to be visited eventually by all the plays
  - e.g. the property $F(\text{overflow}(v))$ for Stepmother
A 'Challenge' Example: Type of games

Depending on the objective of the player we compute a strategy for.

- **Safety games:**
  - requires only states with a certain property to be visited by all the plays
  - e.g. the property $G(\neg \text{overflow}(v))$ for Cinderella

- **Reachability games:**
  - requires a state with a certain property to be visited eventually by all the plays
  - e.g. the property $F (\text{overflow}(v))$ for Stepmother
LTL and Parity games:
- winning condition is an LTL property
- LTL games are an extremely challenging
  - solving them on finite graphs is 2EXPTIME-complete
- Parity games - an important special case
- each state is assigned a color (a number in \{1, \ldots, N\}).
- the winning condition - the minimum color seen infinitely often is odd
- e.g. no overflow or \(bucket_2\) is the only bucket where overflow occurs infinitely often.
Overview

- Game syntax and semantics.
- Proof rules for each type of game.
- Case study on the 'challenge' example.
- Implementation and Experimental results.
- Summary and future work.
A (two-player, turn-based, graph) game is a pair consisting of a symbolic transition system and a winning condition.

- The symbolic transition system
  - consists of two players; Adam and Eve
  - let \( v \) be a tuple of variables of the system
  - system states are valuations of \( v \)
  - assertion \( \text{init}(v) \) represents the initial states
  - the transition relations of Adam and Eve are given by assertions \( \text{adam}(v, v') \) and \( \text{eve}(v, v') \)

- The winning condition
  - given by a set of infinite sequences of system states
  - decides the type of game
A (two-player, turn-based, graph) game is a pair consisting of a symbolic transition system and a winning condition.

- **The symbolic transition system**
  - consists of two players; Adam and Eve
  - let \( \nu \) be a tuple of variables of the system
  - system states are valuations of \( \nu \)
  - assertion \( \text{init}(\nu) \) represents the initial states
  - the transition relations of Adam and Eve are given by assertions \( \text{adam}(\nu, \nu') \) and \( \text{eve}(\nu, \nu') \)

- **The winning condition**
  - given by a set of infinite sequences of system states
  - decides the type of game
A (two-player, turn-based, graph) game is a pair consisting of a symbolic transition system and a winning condition.

- The symbolic transition system
  - consists of two players; Adam and Eve
  - let \( v \) be a tuple of variables of the system
  - system states are valuations of \( v \)
  - assertion \( \text{init}(v) \) represents the initial states
  - the transition relations of Adam and Eve are given by assertions \( \text{adam}(v, v') \) and \( \text{eve}(v, v') \)

- The winning condition
  - given by a set of infinite sequences of system states
  - decides the type of game
A strategy $\sigma$ for Eve is a set of infinite trees such that:

- each root in $\sigma$ coincide with the set of initial states (roots are assumed to be on the first level of the tree)
- the set of successors of each tree node $s$ at an odd level consists of the following set of states.

\[ \{s' \mid (s, s') \models adam(v, v')\} \]

- the set of successors of each tree node $s$ at an even level consists of a non-empty subset of the following set of states.

\[ \{s' \mid (s, s') \models eve(v, v')\} \]

Such an infinite sequence is called a play $\pi$ determined by $\sigma$.

Alternates between universal choices of Adam and existential choices of Eve.
A strategy $\sigma$ is *winning* if every play of $\sigma$ is in the winning condition. For the given system and a winning condition formula $\varphi$, we write

$$(\text{init}(v), \text{eve}(v, v'), \text{adam}(v, v')) \models \varphi$$

when Eve has a winning strategy.
3 proof rules - one for each type of game.

Conclude that Eve has a winning strategy.

Imposes implication and well-foundedness conditions on auxiliary assertions.

Sound and relatively complete.
Proof rules: Safety games

- Only states from \( \text{safe}(v) \) are visited by all plays.
- Requires an invariant assertion \( \text{inv}(v) \).

\[
\begin{align*}
\text{S1} & : \quad \text{init}(v) \rightarrow \text{inv}(v) \\
\text{S2} & : \quad \text{inv}(v) \land \text{adam}(v, v') \rightarrow \text{safe}(v') \land \exists v'' : \text{eve}(v', v'') \land \text{inv}(v'') \\
\text{S3} & : \quad \text{inv}(v) \rightarrow \text{safe}(v)
\end{align*}
\]

\((\text{init}(v), \text{eve}(v, v'), \text{adam}(v, v')) \models G \text{ safe}(v)\)
Proof rules: Reachability games

- A certain set of states called $dst(v)$ is eventually reached by each play.
- Requires an invariant assertion $inv(v)$ together with a binary relation $round(v, v')$.

\[
\begin{align*}
R1 & : \ init(v) \rightarrow inv(v) \\
R2 & : \ inv(v) \land \neg dst(v) \land adam(v, v') \land \neg dst(v') \rightarrow \\
& \quad \exists v'': eve(v', v'') \land inv(v'') \land round(v, v'') \\
R3 & : \ well-founded(round(v, v')) \\
\end{align*}
\]

\[(init(v), eve(v, v'), adam(v, v')) \models F \ dst(v)\]
To state the winning condition we assume:
- the set of all states is partitioned into \( N \) subsets \( p_1(v), \ldots, p_N(v) \)
- \( N \) is an odd number
- \( p_1(v) \lor \cdots \lor p_N(v) \) is valid
- for each \( 1 \leq i < j \leq N \), \( p_i(v) \land p_j(v) \) is unsatisfiable.

The parity winning condition:
- the subsets of states that are visited infinitely often are given as \( p_{i_1}(v), \ldots, p_{i_K}(v) \), and
- the minimal identifier is odd, i.e., \( \min\{i_1, \ldots, i_K\} \) is odd.

... or formally as the LTL formula \( \varphi \).

\[
\varphi = GFp_1(v) \\
\lor GFp_3(v) \land FG\neg(p_1(v) \lor p_2(v)) \\
\ldots \\
\lor GFp_N(v) \land FG\neg(p_1(v) \lor \cdots \lor p_{N-1}(v))
\]
Negate $\varphi$ and translate $\neg \varphi$ to the Büchi automaton $B$.

- represented using assertions over the program counter of the automaton $pc_B$ and the system variables $v$
- initial condition given by $\text{init}_B(pc_B)$
- transition relation given by $\text{next}_B(pc_B, v, pc'_B)$.
- $\text{acc}_B(pc_B)$ represents the accepting states.

Given a play $\pi = s_1, s_2, \ldots$, run of $B$ on $\pi$ is defined as $q_0, q_1, q_2, \ldots$ such that:

- $q_0 \models \text{init}_B(pc_B)$,
- $(q_{i-1}, s_i, q_i) \models \text{next}_B(pc_B, v, pc'_B)$ for each $i \geq 1$.

Apply Büchi acceptance condition

$B$ accepts a play $\pi$ if there exists an accepting run on $\pi$.

- here, if $B$ accepts $\pi$ then $\pi \not\models \varphi$. 
Proof rules: Parity/LTL games (cont)

Find assertions \( \text{inv}(w) \), \( \text{aux}(w, w', v'') \), \( \text{round}(w, w', w'') \), and \( \text{fair}(w, w') \) where \( w = (v, pc_B) \) such that:

\[
\begin{align*}
B1 : & \quad \text{init}(v) \land \text{init}_B(pc_B) \land \text{next}_B(pc_B, v, pc'_B) \rightarrow \text{inv}(v, pc'_B) \\
B2 : & \quad \text{inv}(w) \land \text{adam}(v, v') \land \text{next}_B(pc_B, v', pc'_B) \rightarrow \exists v'' : \text{eve}(v', v'') \land \text{aux}(w, w', v'') \\
B3 : & \quad \text{aux}(w, w', v'') \land \text{next}_B(pc'_B, v'', pc''_B) \rightarrow \text{inv}(w'') \land \text{round}(w, w', w'') \\
B4 : & \quad \text{round}(w, w', w'') \land (\text{acc}_B(pc_B) \lor \text{acc}_B(pc'_B)) \rightarrow \text{fair}(w, w'') \\
B5 : & \quad \text{fair}(w, w') \land \text{round}(w', w'', w'''_B) \rightarrow \text{fair}(w, w'''_B) \\
B6 : & \quad \text{well-founded}(\text{fair}(w, w')) \\
\end{align*}
\]

\[
(init(v), eve(v, v'), adam(v, v')) \models \varphi
\]
Case Study: Cinderella-Stepmother game
Safety objective: Round strategy

- $c = 3$ for the bucket capacity.
- An auxiliary variable $r$ for a pair of buckets to be emptied.
- A user-provided template for Cinderella adds guard for each disjunct and updates the round variable.

$$\text{init}(v, r) = (\overline{\text{init}}(v) \land r = 1)$$
$$\text{eve}(v, r, v', r') = \text{cinderella}(v, v') \land \text{RELT}(\text{rel})(v, r, v', r')$$
$$\text{adam}(v, r, v', r') = (\text{stepmother}(v, v') \land r' = r)$$
**Case Study: Cinderella-Stepmother game**

**Safety objective: Round strategy (cont)**

\[ \text{REL}_T(\text{rel})(v, r, v', r') = (r = 1 \land r' = ?_1 \land c_1(v, v') \lor r = 2 \land r' = ?_2 \land c_2(v, v') \lor r = 3 \land r' = ?_3 \land c_3(v, v') \lor r = 4 \land r' = ?_4 \land c_4(v, v') \lor r = 5 \land r' = ?_5 \land c_5(v, v')) \]

- Template parameters are denoted by “?”-marks.
- Our tool returns a solution \(?_1 = 4, ?_2 = 1, ?_3 = 1, ?_4 = 3, ?_5 = 1\).
- The corresponding strategy is 1&2 - 4&5 - 3&4 - 1&2, ...
Case Study: Cinderella-Stepmother game
Safety objective: Second strategy

- $c = 2$ for the bucket capacity.
- Template based on the previous move of Cinderella and Stepmother.

$$inv(v) \land stepmother(v, v') \rightarrow safe(v') \land \exists v'': cinderella(v', v'') \land inv(v'')$$

- The template looks like

$$\text{REL}_T(rel)(v, v', v'') = (b_1 = 0 \land b_2 = 0 \land T_{12}(v', v'') \lor b_2 = 0 \land b_3 = 0 \land T_{23}(v', v'') \lor b_3 = 0 \land b_4 = 0 \land T_{34}(v', v'') \lor b_4 = 0 \land b_5 = 0 \land T_{45}(v', v'') \lor b_5 = 0 \land b_1 = 0 \land T_{51}(v', v'')).$$
Let us see one part of the template, e.g., $T_{12}$

- In the previous round emptied buckets 1 and 2. ($b_1 = 0 \land b_2 = 0$)
- During the next round empty another pair of buckets.
  - either the pair of buckets 3 and 4 ($b_3'' = 0 \land b_4'' = 0$)
  - or the pair of buckets 4 and 5 ($b_4'' = 0 \land b_5'' = 0$)
- Deciding between the two is not straightforward.
  - The game solving approach handles it using the specified template.
- Formulated the formula $T_{12}$ is provided as follows.

$$T_{12}(v', v'') = (b_3'' = 0 \land b_4'' = 0 \land ?_5 \ast b_5' + ?_2 \ast b_2' \leq ?_6 \ast 1 \lor$$
$$b_4'' = 0 \land b_5'' = 0 \land ?_1 \ast b_1' + ?_3 \ast b_3' \leq ?_6 \ast 1)$$

- Our tool returns a solution $?_1 = 1, ?_2 = 1, ?_3 = 1, ?_5 = 1, ?_6 = 1$. 
Case Study: Cinderella-Stepmother game
Reachability objective

- $c = 1.4$ for the bucket capacity.
- Instantiate the proof rule as follows:
  \[
  \text{eve}(v, v') = \text{stepmother}(v, v') \\
  \text{adam}(v, v') = \text{cinderella}(v, v')
  \]

- A template corresponding to the existentially quantified clause.
  \[
  \text{REL}(\text{rel})(v, v', v'') = (?_1 + \cdots + ?_5 = 1 \land \\
  \bigwedge_{i \in \{1..5\}} (b''_i = b'_i + ?_i) \land \bigwedge_{i \in \{1..5\}} ?_i \geq 0)
  \]

- Our tool returns a solution
  \[
  ?_1 = 0.8, ?_2 = 0, ?_3 = 0.1, ?_4 = 0, ?_5 = 0.1.
  \]
Case Study: Cinderella-Stepmother game
Parity objective

- A state without overflow: \((\text{color} = 0) \iff \neg \text{overflow}(v)\).
- A state with overflow such that \(i\) is the smallest index from those that correspond to buckets that have overflown: \((\text{color} = i)\).
- The resulting state-partitioning groups states with different priority levels indicated by \(p(i)\):
  \[
p(i) = (\text{color} = i), \quad \text{for } i \in \{0, \ldots, 2\}
\]
  \[
p(3) = (\text{color} = 3 \lor \text{color} = 4 \lor \text{color} = 5).
\]
- The winning condition \(\text{win}(i)\) is defined as follows.
  \[
  \text{win}(i) = (GF \ p(i) \land \bigwedge_{j \in \{0, \ldots, i-1\}} FG \neg p(j))
  \]
we define the objective for the Cinderella player $\text{win}(0) \lor \text{win}(2)$.

The formula corresponding to the Cinderella’s objective:

$$\varphi = (GF \, p(0) \lor (GF \, p(2) \land FG \, \neg p(1) \land FG \, \neg p(0))).$$

Our tool finds the same strategy as the second winning strategy for the Cinderella player.
Other applications

- Synthesis of reactive programs from temporal specifications.
- Program repair game with safety objective.
- Concurrent program repair game with safety and response objectives.
- Synthesis of synchronization game with safety objective.
The EHSF engine

- Proof rules are automated using the EHSF engine
- Resolves forall-exists Horn-like clauses extended with well-foundedness criteria
- Example:

\[
x \geq 0 \rightarrow \exists y : x \geq y \land \text{rank}(x, y), \quad \text{rank}(x, y) \rightarrow \text{ti}(x, y),
\]
\[
\text{ti}(x, y) \land \text{rank}(y, z) \rightarrow \text{ti}(x, z), \quad \text{dwf}(\text{ti}).
\]

- Maps each predicate symbol into a constraint over \( v \).
- Maps both \( \text{rank}(x, y) \) and \( \text{ti}(x, y) \) to the constraint \( (x \geq 0 \land y \geq x - 1) \) for the example.
The EHSF engine (cont)

- Resolves clauses using a CEGAR scheme to discover witnesses for existentially quantified variables.
  - The space of witnesses is provided by some 'template'.
- Refinement loop collects a global constraint that declaratively determines which witnesses to choose.
  - A chosen witnesses replace existential quantification.
  - The resulting universally quantified clauses are passed to a solver for such clauses. e.g., HSF.
- Such a solver either finds a solution or returns a counterexample.
  - Counterexample are turned into an additional constraint on the set of witness candidates, and
  - Continues with the next iteration of the refinement loop.
- Refinement loop conjoins constraints that are obtained for all discovered counterexamples.
  - Wrong choice of witnesses can be mended.
  - Previously handled counterexamples are not rediscovered.
**Experiment**

- **GSolve**: a proof-of-concept implementation of the approach.
- Implemented in SICStus Prolog.
- Relies on an implementation of the E-HSF algorithm to solve Horn clauses over linear inequalities.
- Uses SMT solvers for handling non-linear constraints, i.e., the Z3 and the Barcelogic solvers.
- Experiments run on an Intel Core 2 Duo machine, clocked at 2.53 GHz, with 4 GB of RAM.
### Results

<table>
<thead>
<tr>
<th>Id</th>
<th>Game</th>
<th>Player $p$</th>
<th>Objective for player $p$</th>
<th>Time (z3)</th>
<th>Time (Barcelogic)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Cinderella ($c = 3$)</td>
<td>Cinderella</td>
<td>$G \neg\text{overflow}$</td>
<td>3.2s</td>
<td>1.2s</td>
</tr>
<tr>
<td>P2</td>
<td>Cinderella ($c = 2$)</td>
<td>Cinderella</td>
<td>$G \neg\text{overflow}$</td>
<td>1m52s</td>
<td>1m52s</td>
</tr>
<tr>
<td>P3</td>
<td>Cinderella ($c = 1.4$)</td>
<td>Stepmother</td>
<td>$F \text{ overflow}$</td>
<td>18s</td>
<td>1m14s</td>
</tr>
<tr>
<td>P4</td>
<td>Cinderella ($c = 1.4$)</td>
<td>Cinderella</td>
<td>$\text{win}(0) \lor \text{win}(2)$</td>
<td>7m16s</td>
<td>SysError</td>
</tr>
<tr>
<td>P5</td>
<td>Cinderella ($c = 1.4$)</td>
<td>Cinderella</td>
<td>$F \ at \rightarrow \text{dest}$</td>
<td>4.7s</td>
<td>4.7s</td>
</tr>
<tr>
<td>P6</td>
<td>Robot-1d (yr0,yh0,ydst,e=10)</td>
<td>Robot</td>
<td>$G \neg\text{error}$</td>
<td>T/O</td>
<td>1s</td>
</tr>
<tr>
<td>P7</td>
<td>Repair-Lock</td>
<td>Program</td>
<td>$G \neg\text{error}$</td>
<td>0.3s</td>
<td>0.3s</td>
</tr>
<tr>
<td>P8</td>
<td>Repair-Critical</td>
<td>Program</td>
<td>$G \neg\text{error}$</td>
<td>17.7s</td>
<td>16.9s</td>
</tr>
<tr>
<td>P9</td>
<td>Repair-Critical</td>
<td>Program</td>
<td>$G \ (at \rightarrow F \neg at \rightarrow)$</td>
<td>53.3s</td>
<td>3m6s</td>
</tr>
<tr>
<td>P10</td>
<td>Synth-Synchronization</td>
<td>Program</td>
<td>$G \neg\text{error}$</td>
<td>T/O</td>
<td>1s</td>
</tr>
</tbody>
</table>

- **GSolve** has always succeeded in finding a strategy using one of the two solvers.
Summary and Future work

- A new algorithmic approach which comprises:
  - a set of sound and relatively complete proof rules; and
  - automation on top of an existing automated deduction engine

- Demonstrate the practical promise through a few case studies.

- Prototypic and many avenues for future work remain open.
  - engineering it for greater scalability
  - applying to reactive synthesis questions in embedded systems and robotics.
  - synergy between our approach and abstraction-based and automata-theoretic approaches.