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?

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?

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?

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 synthese 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 synthese 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:

$$\bar{init}(v) = (b_1 = 0 \wedge \cdots \wedge b_5 = 0).$$

- Transition relation of Stepmother:

$$\begin{aligned} stepmother(v, v') = (&b'_1 + \cdots + b'_5 = b_1 + \cdots + b_5 + 1 \\ &\wedge\ b'_1 \geq b_1 \wedge \cdots \wedge b'_5 \geq b_5). \end{aligned}$$

- Transition relation of Cinderella:

$$cinderella(v, v') =$$

$$\bigvee_{i \in \{1...5\}} \left(\begin{array}{l} b'_i = 0 \wedge b'_{(i+1)\%5} = 0 \\ \wedge \left(\bigwedge_{j \in \{1..5\}} \left(\begin{array}{c} j \neq i \wedge j \neq (i+1)\%5 \\ \rightarrow b'_j = b_j \end{array} \right) \right) \end{array} \right).$$

- Overflow condition:

$$overflow(v) = (b_1 > c \vee \cdots \vee b_5 > c).$$

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 $G(\neg overflow(v))$ for Cinderella
- Reachability games:
  - requires a state with a certain property to be visited eventually by all the plays
  - e.g. the property $F\ (overflow(v))$ 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 $G(\neg overflow(v))$ for Cinderella
- Reachability games:
  - requires a state with a certain property to be visited eventually by all the plays
  - e.g. the property $F\ (overflow(v))$ 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 $G(\neg \mathit{overflow}(v))$ for Cinderella
- Reachability games:
  - requires a state with a certain property to be visited eventually by all the plays
  - e.g. the property $F\ (\mathit{overflow}(v))$ for Stepmother

A 'Challenge' Example: Type of games (cont)

- 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.

Game syntax

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 $init(v)$ represents the initial states
  - the transition relations of Adam and Eve are given by assertions $adam(v, v')$ and $eve(v, v')$
- The *winning condition*
  - given by a set of infinite sequences of system states
  - decides the type of game

Game syntax

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 $init(v)$ represents the initial states
  - the transition relations of Adam and Eve are given by assertions $adam(v, v')$ and $eve(v, v')$
- The *winning condition*
  - given by a set of infinite sequences of system states
  - decides the type of game

Game syntax

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 $init(v)$ represents the initial states
  - the transition relations of Adam and Eve are given by assertions $adam(v, v')$ and $eve(v, v')$
- The *winning condition*
  - given by a set of infinite sequences of system states
  - decides the type of game

Game semantics

- A *strategy* σ for Eve is a set of infinite trees such that:
  - each root in σ 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* π determined by σ.
- Alternates between universal choices of Adam and existential choices of Eve.

Game semantics (cont)

- A strategy σ is *winning* if every play of σ is in the winning condition.
- For the given system and a winning condition formula φ, we write

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

when Eve has a winning strategy.

Proof rules

- 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 $safe(v)$ are visited by all plays.
- Requires an invariant assertion $inv(v)$.

$$
\begin{array}{ll}
S1: & init(v) \rightarrow inv(v) \\
S2: & inv(v) \wedge adam(v, v') \rightarrow safe(v') \wedge \exists v'' : eve(v', v'') \wedge inv(v'') \\
S3: & inv(v) \rightarrow safe(v)
\end{array}
$$

$$(init(v), eve(v, v'), adam(v, v')) \models G \; safe(v)$$

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 $round(v, v')$.

$$
\begin{aligned}
\text{R1}: &\quad init(v) \rightarrow inv(v) \\
\text{R2}: &\quad inv(v) \wedge \neg dst(v) \wedge adam(v, v') \wedge \neg dst(v') \rightarrow \\
&\qquad\qquad \exists v'' : eve(v', v'') \wedge inv(v'') \wedge round(v, v'') \\
\text{R3}: &\quad \textit{well-founded}(round(v, v'))
\end{aligned}
$$

$$(init(v), eve(v, v'), adam(v, v')) \models F\ dst(v)$$

Proof rules: Parity/LTL games

- To state the winning condition we assume:
  - the set of all states is partitioned into N subsets $p_1(v), \dots, p_N(v)$
  - N is an odd number
  - $p_1(v) \vee \dots \vee p_N(v)$ is valid
  - for each $1 \leq i < j \leq N$, $p_i(v) \wedge p_j(v)$ is unsatisfiable.
- The parity winning condition:
  - the subsets of states that are visited infinitely often are given as $p_{i_1}(v), \dots, p_{i_K}(v)$, and
  - the minimal identifier is odd, i.e., $\min\{i_1, \dots, i_K\}$ is odd.
- ... or formally as the LTL formula φ.

$$
\begin{aligned}
\varphi \;=\; & GFp_1(v) \\
& \vee\; GFp_3(v) \wedge FG\neg(p_1(v) \vee p_2(v)) \\
& \dots \\
& \vee\; GFp_N(v) \wedge FG\neg(p_1(v) \vee \dots \vee p_{N-1}(v))
\end{aligned}
$$

Proof rules: Parity/LTL games (cont)

- Negate φ and translate $\neg\varphi$ to the Büchi automaton $\mathcal{B}$.
  - represented using assertions over the program counter of the automaton $pc_{\mathcal{B}}$ and the system variables v
  - initial condition given by $init_{\mathcal{B}}(pc_{\mathcal{B}})$
  - transition relation given by $next_{\mathcal{B}}(pc_{\mathcal{B}}, v, pc'_{\mathcal{B}})$.
  - $acc_{\mathcal{B}}(pc_{\mathcal{B}})$ represents the accepting states.
- Given a play $\pi = s_1, s_2, \ldots,$ run of $\mathcal{B}$ on π is defined as $q_0, q_1, q_2, \ldots$ such that:
  - $q_0 \models init_{\mathcal{B}}(pc_{\mathcal{B}})$,
  - $(q_{i-1}, s_i, q_i) \models next_{\mathcal{B}}(pc_{\mathcal{B}}, v, pc'_{\mathcal{B}})$ for each $i \geq 1$.
- Apply Büchi acceptance condition
- $\mathcal{B}$ accepts a play π if there exists an accepting run on π.
  - here, if $\mathcal{B}$ accepts π then $\pi \not\models \varphi$.

Proof rules: Parity/LTL games (cont)

Find assertions $inv(w)$, $aux(w, w', v'')$, $round(w, w', w'')$, and $fair(w, w')$ where $w = (v, pc_{\mathcal{B}})$ such that:

$$
\begin{aligned}
\mathrm{B1}: &\quad init(v) \wedge init_{\mathcal{B}}(pc_{\mathcal{B}}) \wedge next_{\mathcal{B}}(pc_{\mathcal{B}}, v, pc'_{\mathcal{B}}) \rightarrow inv(v, pc'_{\mathcal{B}}) \\
\mathrm{B2}: &\quad inv(w) \wedge adam(v, v') \wedge next_{\mathcal{B}}(pc_{\mathcal{B}}, v', pc'_{\mathcal{B}}) \rightarrow \\
&\qquad\qquad \exists v'' : eve(v', v'') \wedge aux(w, w', v'') \\
\mathrm{B3}: &\quad aux(w, w', v'') \wedge next_{\mathcal{B}}(pc'_{\mathcal{B}}, v'', pc''_{\mathcal{B}}) \rightarrow inv(w'') \wedge round(w, w', w'') \\
\mathrm{B4}: &\quad round(w, w', w'') \wedge (acc_{\mathcal{B}}(pc_{\mathcal{B}}) \vee acc_{\mathcal{B}}(pc'_{\mathcal{B}})) \rightarrow fair(w, w'') \\
\mathrm{B5}: &\quad fair(w, w') \wedge round(w', w'', w''') \rightarrow fair(w, w''') \\
\mathrm{B6}: &\quad \textit{well-founded}(fair(w, w'))
\end{aligned}
$$

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

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.

$$
\begin{aligned}
init(v, r) &= (\bar{init}(v) \wedge r = 1) \\
eve(v, r, v', r') &= cinderella(v, v') \wedge \textsc{RelT}(rel)(v, r, v', r') \\
adam(v, r, v', r') &= (stepmother(v, v') \wedge r' = r)
\end{aligned}
$$

Case Study: Cinderella-Stepmother game
Safety objective: Round strategy (cont)

$$\begin{aligned}\textsc{RelT}(\mathit{rel})(v, r, v', r') = (&r = 1 \land r' = ?_1 \land c_1(v, v') \lor \\ &r = 2 \land r' = ?_2 \land c_2(v, v') \lor \\ &r = 3 \land r' = ?_3 \land c_3(v, v') \lor \\ &r = 4 \land r' = ?_4 \land c_4(v, v') \lor \\ &r = 5 \land r' = ?_5 \land c_5(v, v'))\end{aligned}$$

- 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.

$$\begin{aligned}inv(v) \wedge stepmother(v, v') \rightarrow safe(v') \;\wedge \exists v'' : &\; cinderella(v', v'') \\ &\wedge inv(v'')\end{aligned}$$

- The template looks like

$$\begin{aligned}\mathrm{RELT}(rel)(v, v', v'') = (&b_1 = 0 \wedge b_2 = 0 \wedge T_{12}(v', v'') \vee \\ &b_2 = 0 \wedge b_3 = 0 \wedge T_{23}(v', v'') \vee \\ &b_3 = 0 \wedge b_4 = 0 \wedge T_{34}(v', v'') \vee \\ &b_4 = 0 \wedge b_5 = 0 \wedge T_{45}(v', v'') \vee \\ &b_5 = 0 \wedge b_1 = 0 \wedge T_{51}(v', v'')).\end{aligned}$$

Case Study: Cinderella-Stepmother game
Safety objective: Second strategy (cont)

Let us see one part of the template, e.g., T_{12}

- In the previous round emptied buckets 1 and 2. $(b_1 = 0 \wedge b_2 = 0)$
- During the next round empty another pair of buckets.
  - either the pair of buckets 3 and 4 $(b''_3 = 0 \wedge b''_4 = 0)$
  - or the pair of buckets 4 and 5 $(b''_4 = 0 \wedge b''_5 = 0)$
- Deciding between the two is not straightforward.
  - The game solving approach handles it using the specified template.
- Formalized the formula T_{12} is provided as follows.

$$\begin{aligned} T_{12}(v', v'') = (&b''_3 = 0 \wedge b''_4 = 0 \wedge ?_5 * b'_5 + ?_2 * b'_2 \leq ?_6 * 1 \vee \\ &b''_4 = 0 \wedge b''_5 = 0 \wedge ?_1 * b'_1 + ?_3 * b'_3 \leq ?_6 * 1) \end{aligned}$$

- 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:

$$
\begin{aligned}
\mathit{eve}(v, v') &= \mathit{stepmother}(v, v') \\
\mathit{adam}(v, v') &= \mathit{cinderella}(v, v')
\end{aligned}
$$

- A template corresponding to the existentially quantified clause.

$$
\begin{aligned}
\mathrm{RelT}(\mathit{rel})(v, v', v'') = (&?_1 + \cdots + ?_5 = 1 \wedge \\
&\bigwedge_{i \in \{1..5\}} (b''_i = b'_i + ?_i) \wedge \bigwedge_{i \in \{1..5\}} ?_i \geq 0)
\end{aligned}
$$

- 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: $(color = 0) \leftrightarrow \neg overflow(v)$.
- A state with overflow such that i is the smallest index from those that correspond to buckets that have overflown: $(color = i)$.
- The resulting state-partitioning groups states with different priority levels indicated by $p(i)$:

$$\begin{aligned} p(i) &= (color = i), \qquad \text{for } i \in \{0, \ldots, 2\} \\ p(3) &= (color = 3 \vee color = 4 \vee color = 5). \end{aligned}$$

- The winning condition $win(i)$ is defined as follows.

$$win(i) = (GF\ p(i) \wedge \bigwedge_{j \in \{0,..,i-1\}} FG \neg p(j))$$

Case Study: Cinderella-Stepmother game
Parity objective (cont)

- we define the objective for the Cinderella player $win(0) \vee win(2)$.
- The formula corresponding to the Cinderella's objective:

$$\varphi = (GF\ p(0) \vee (GF\ p(2) \wedge FG\ \neg p(1) \wedge FG\ \neg p(0))).$$

- 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:

$$x \geq 0 \rightarrow \exists y : x \geq y \wedge rank(x, y), \qquad rank(x, y) \rightarrow ti(x, y),$$
$$ti(x, y) \wedge rank(y, z) \rightarrow ti(x, z), \qquad dwf(ti).$$

- Maps each predicate symbol into a constraint over v.
- Maps both $rank(x, y)$ and $ti(x, y))$ to the constraint $(x \geq 0 \wedge y \geq x - 1)$ for the example.

The EHSF engine (cont)

- Resolves clauses using a CEGAR scheme to discover witnesses for existentially quantified variables.
  - 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

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

- 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.