A Constraint-Based Approach to Solving Games on Infinite Graphs

Tewodros A. Beyene
Technische Universtität München

Swarat Chaudhuri
Rice University

Corneliu Popeea
Technische Universtität München

Andrey Rybalchenko
Microsoft Research Cambridge and
Technische Universtität München

Abstract

We present a constraint-based approach to computing winning strategies in two-player graph games over the state space of infinite-state programs. Such games have numerous applications in program verification and synthesis, including the synthesis of infinite-state reactive programs and branching-time verification of infinite-state programs. Our method handles games with winning conditions given by safety, reachability, and general Linear Temporal Logic (LTL) properties. For each property class, we give a deductive proof rule that — provided a symbolic representation of the game players — describes a winning strategy for a particular player. Our rules are sound and relatively complete. We show that these rules can be automated by using an off-the-shelf Horn constraint solver that supports existential quantification in clause heads. The practical promise of the rules is demonstrated through several case studies, including a challenging "Cinderella-Stepmother game" that allows infinite alternation of discrete and continuous choices by two players, as well as examples derived from prior work on program repair and synthesis.

Categories and Subject Descriptors D.2.4 [*Software Engineering*]: Software/Program Verification; F.3.1 [*Logics and Meanings of Programs*]: Specifying and Verifying and Reasoning about Programs

Keywords program synthesis; games; reactive synthesis; proof rules; static analysis; Horn clauses

1. Introduction

Many fundamental questions in formal methods reduce to computing winning strategies in *turn-based graph games* [18], i.e., games where two players take turns in moving a token along the edges of a graph, and a player wins if the sequence of nodes visited by the token satisfies a certain ω-regular *winning condition*. For example:

- To synthesize a reactive system from a temporal specification [7, 38, 43], one constructs a graph game where the goal of one player is to satisfy the specification and the goal of the other is to violate it. The desired system is realizable if and only if the first player has a winning strategy in this game.
- The problem of verifying a branching-time property of a system is naturally framed as a graph game [14]. Here, one player models the existential path quantifiers in the property; the other player models the universal quantifiers. The system satisfies the property if and only if the existential player has a winning strategy.
- Graph games are a natural model for "open" systems [32] that explicitly model interactions between a controller (one player) and its environment (the other player). To prove such a system correct, we show that the controller has a strategy to enforce its requirements no matter how the environment behaves.

There is a rich literature on algorithmic approaches to graph games motivated by applications in formal methods [10, 12, 30, 47]. The majority of these approaches focus on decidable classes of games, such as games on finite graphs. This focus limits the applications of these techniques. For example, an algorithm that requires a finite game graph can only be applied to the verification and synthesis of finite-state systems. To use games in the analysis and synthesis of infinite-state programs, we need symbolic, abstraction-based algorithms for solving games on the state spaces of such programs. While a few such algorithms exist in the literature [12, 26], much more remains to be done on this topic.

This paper presents a new approach to this problem space. Our contribution is an algorithmic technique based on automated deduction for solving (turn-based) games over infinite-state symbolic transition systems.

Specifically, we target three classes of games over infinite graphs: *safety games*, *reachability games*, and *Linear Temporal Logic (LTL) games* [18]. These games differ in the winning condition for the player for whom we are computing a winning strategy (call this player Eve; the other player is called Adam). In a safety game, Eve wins a play (an infinite sequence of nodes visited by the game token) if and only if the play avoids a certain "unsafe" set of nodes. In a reachability game, a play is winning for Eve if and only if it reaches a certain target set of nodes. In LTL games, Eve

POPL '14, January 22–24, 2014, San Diego, CA, USA.

http://dx.doi.org/10.1145/2535838.2535860

wins a play if and only if the play satisfies an LTL property. We note that, LTL games subsume parity games, an important class of games where each node of the game graph is labeled with a "color" from the set $\{1, \ldots, N\}$, and a play is winning for Eve if and only if the minimum color seen infinitely often in the play is odd.

The importance of solving the above types of games to formal methods is well-established in the literature. For instance, the problem of solving a parity game over a program's state space is equivalent to that of verifying program properties written in the modal μ-calculus [14, 18] (note that the μ-calculus subsumes popular temporal logics like LTL, CTL, and CTL*). The solution of LTL games is also at the core of reactive synthesis from temporal specifications. Reachability and safety games are important special cases of LTL games that are sufficient for many applications, including program repair [21, 29], program synthesis [41], synthesis of interface specifications [1], and verification of the fragment of the μ-calculus without alternation of fixpoint quantifiers.

For each of the above types of games, we give a deductive proof rule that, given a symbolic representation of the game graph, symbolically represents a winning strategy in the game using quantified Horn constraints. The rule is then automated by applying the EHSF engine for automated deduction [3].

To understand how our rules work, consider a safety game where the objective of Eve is to satisfy the state property p at all points in all plays. To find a winning strategy for Eve, our rule for safety games computes an invariant inv that describes the set of states from which Eve can win the game. This invariant needs to satisfy the following criteria: (a) the initial condition of the game implies inv; (b) inv implies p; and (c) for all Adam transitions out of inv (let us say to a destination state σ), σ satisfies p and there is a Eve transition from σ back to inv.

Strategy computation in reachability games relies on well-founded transition invariants [39] to guarantee that a target state is reached after a finite number of rounds of the game. We solve LTL games with temporal objective φ by converting $\neg\varphi$ into a nondeterministic Büchi automaton, then performing a fair termination check on the product of this automaton and the game graph.

All of our rules are sound, meaning that if they derive a strategy for a player, then the player actually wins under the strategy, as well as relatively complete, meaning that they can always derive a winning strategy when one exists, assuming a suitably powerful assertion language.

From a practical point of view, the appeal of our rules is that they leverage the most recent developments in SMT-solving, invariant generation, and termination verification [3, 20]. Specifically, our implementation CONSYNTH feeds our proof rules to the EHSF engine in the form of Horn-like clauses (in some cases with existential quantifiers in clause heads). Solving the game now amounts to resolving these clauses to a bounded depth, proving the unsatisfiability of the resolvent, repeating the process and generalizing from proofs of unsatisfiability to a solution for the original clauses. EHSF does so using a combination of counterexample-guided abstraction-refinement (CEGAR), interpolation, and SMT solving, and with help from user-provided templates that capture high-level intuitions about the strategy.

We evaluate CONSYNTH using several challenging case studies, including the "Cinderella-Stepmother game" — an existing challenge problem for infinite-state graph games that allows infinite alternation of discrete and continuous choices by the two players — and games arising out of prior work on program repair [29] and synthesis [45].

Now we summarize the main contributions of the paper:

- We take on the problem of solving games over state spaces of infinite-state programs using the power of modern automated software analysis technology.
- We present three deductive proof rules for solving such games under the safety, reachability, and LTL winning conditions. Our rules are sound and relatively complete, and our automata-theoretic rule for LTL games avoids the need to determinize a Büchi automaton.
- We offer a prototype implementation of our rules on top of an existing automated deduction engine. We illustrate the promise of the system through several case studies using examples posed in prior work.

This paper is organized as follows. In Section 2, we describe the Cinderella-Stepmother game as a motivating example. Section 3 formally defines graph games and the strategy computation problem. Section 4 gives our proof rules for solving games and proves them correct, i.e., sound and relatively complete. Section 5 revisits the example from Section 2 and applies our rules to variants of it; Section 6 presents applications of our rules to repair and synthesis problems from prior work. Section 7 presents concrete experimental results. Related work is described in Section 8; we conclude with some discussion in Section 9.

2. The Cinderella-Stepmother game

In this section, we describe a synthesis problem that motivated this work, and that we use in a case study later in the paper. A version of the problem was previously posed by Rajeev Alur as a challenge problem for the software synthesis community (see Bodlaender et al. [4] and Hurkens et al. [28] for more on the problem).

The problem involves a turn-based game between the mythical Cinderella, and her nemesis, the Stepmother. The game setup involves five buckets arranged in a circle. Each bucket can hold up to c (a constant) units of water; initially, all buckets are empty. In each round of the game, Stepmother brings 1 unit of additional water and splits it among the five buckets. If any of the buckets overflow, Stepmother wins. If not, Cinderella empties two *adjacent* buckets. Cinderella wins if the game goes on forever.

We can model the Cinderella-Stepmother game using the following symbolic transition system. Let v be a set of system variables that represent the amount of water in the five buckets, $v = (b_1, b_2, b_3, b_4, b_5)$. All the buckets are initially empty — this fact is specified as the initial condition

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

The transition relation of Stepmother represents a nondeterministic choice of buckets in which 1 unit of additional water is added:

$$stepmother(v, v') = (b'_1 + \cdots + b'_5 = b_1 + \cdots + b_5 + 1 \wedge b'_1 \geq b_1 \wedge \cdots \wedge b'_5 \geq b_5).$$

The transition relation of the Cinderella player represents a nondeterministic choice of two consecutive buckets that are emptied.

$$cinderella(v, v') = \bigvee_{i \in \{1\ldots5\}} \left(\begin{array}{l} b'_i = 0 \wedge b'_{(i+1)\%5} = 0 \\ \wedge \left(\bigwedge_{j \in \{1..5\}} \left(\begin{array}{c} j \neq i \wedge j \neq (i+1)\%5 \\ \rightarrow b'_j = b_j \end{array} \right) \right) \end{array} \right).$$

The condition that one of the buckets overflows is described by the assertion

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

Safety game We observe that in the above game, Cinderella wants to enforce a *safety property* — specifically, the property $G(\neg overflow(v))$ — in every play of the game. This property is Cinderella's *winning condition*. Games are classified according to the winning condition of the player for whom we want to compute

a strategy. Specifically, suppose we want to compute a strategy for Cinderella. In that case, we are trying to solve a *safety game*.

Reachability game Now suppose we want to compute a strategy for Stepmother instead. We note that the winning condition for Stepmother is the reachability property $F\ overflow(v)$. The game is a *reachability game*.

LTL and parity games It is easy to define generalizations of the game where the winning condition for a player is a general Linear Temporal Logic (LTL) property. Such a game is called an *LTL game*. LTL games are an extremely challenging class of games — the problem of solving such games on finite game graphs is 2EXPTIME-complete [38]. The intuitive reason for this hardness is that it requires a conversion from an LTL formula to a nondeterministic Büchi automaton (an exponential blowup) and then the determinization of this automaton (another exponential blowup).

An important special case of LTL games is *parity games* [18]. Here, each state of the transition system is assigned a color (a number in $\{1, \ldots, N\}$), and the winning condition for a play is that the minimum color seen infinitely often in the play is odd. (The condition can be stated in LTL in an obvious way.) In Section 7, we use ConSynth on a parity game generalizing our original game.

Discussion From the determinacy of the classes of graph games that we study [35], it follows that for every value of c, either Cinderella or Stepmother has a winning strategy in each of the above games. Now we give some intuitions about what such a winning strategy would look like in the game as originally stated. The discussion of how to automatically solve the problem using ConSynth is postponed until Section 5.

First note that if $c < 1.5$ units, then Stepmother wins. Her strategy is as follows: in the first round, she divides 1 unit into two non-adjacent buckets. Then no matter what Cinderella does, there will be a bucket with 0.5 units at the end of the round, and Stepmother can cause a spill in second round by adding 1 unit in that bucket. If $c \geq 3$ units, Cinderella wins: she can just select the buckets in a round-robin order, emptying two buckets in each round, and this strategy is winning no matter what Stepmother does.

The problem becomes more challenging for $1.5 \leq c < 3$. We leave this case as a challenge for the reader — it will soon be apparent that it is highly nontrivial. In such cases, fully automated strategy synthesis seems unrealistic, and computer-*assisted* proofs driven by user-provided hints or templates are more plausible. This is the strategy that our approach takes.

3. Preliminaries

In this section we formally define games and strategies, and also sketch the deduction framework used to automate them.

3.1 Games

Syntax A *(two-player, turn-based, graph) game* is a pair consisting of a symbolic transition system and a *winning condition*:

- We consider symbolic transition systems that are composed from two players, Adam and Eve. Let v be a tuple of variables. We assume that valuations of v describe states of the system under consideration. (For simplicity, we do not distinguish between variables controlled by Eve and Adam.)

  We represent the initial states of the transition system by an assertion $init(v)$. The transition relations of Adam and Eve are given by assertions $adam(v, v')$ and $eve(v, v')$, respectively.
- A *winning condition* obj for a game is given by a set of infinite sequences of system states. A game is said to be a safety game, a reachability game, and an LTL game respectively when its winning condition is a safety property, a reachability property, and a general LTL property.

Semantics We present the semantics of games in two steps. First, we define strategies of the individual players. A *strategy* σ for Eve is a set of infinite trees over the states of the system that satisfies the following conditions:

- The roots of trees in σ coincide with the set of initial states, and are considered to be on the first level of the tree. (Here, the level of a node is the length of the path to the root plus one.)
- 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.
$$\{s' \mid (s, s') \models eve(v, v')\}$$

Thus, a strategy for Eve alternates between universal choices of Adam and existential choices of Eve. We call each infinite sequence of system states that starts at a root of a strategy σ and follows some branch a *play* π determined by σ.

A strategy σ for Eve is *winning* if every play determined by σ is included in the winning condition.

For the given system and a formula φ that describes a winning condition in some temporal logic, we write
$$(init(v), eve(v, v'), adam(v, v')) \models \varphi$$
when Eve has a winning strategy.

We also consider Adam's perspective. A strategy σ for Adam is defined in a similar way. The roots of σ represent a non-empty subset of $init(v)$. σ alternates between existential choices of Adam and universal choices of Eve. If a tree node s is on an odd level, then its successors form a non-empty subset of $\{s' \mid (s, s') \models adam(v, v')\}$. Otherwise, the set of successors is $\{s' \mid (s, s') \models eve(v, v')\}$.

3.2 The EHSF engine

Our proof rules are automated using the EHSF [3] engine for resolving forall-exists Horn-like clauses extended with well-foundedness criteria.

We skip the syntax and semantics of the clauses targeted by this system — see [3] for more details. Instead, we illustrate these clauses with the following example:

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

Intuitively, these clauses represent an assertion over the interpretation of "query symbols" $rank$ and ti (the predicate dwf represents disjunctive well-foundedness, and is not a query symbol). The semantics of these clauses maps each predicate symbol occurring in them into a constraint over v.

Specifically, the above set of clauses has a solution that maps both $rank(x, y)$ and $ti(x, y)$ to the constraint $(x \geq 0 \wedge y \geq x - 1)$.

EHSF resolves clauses like the above using a CEGAR scheme to discover witnesses for existentially quantified variables. The refinement loop collects a global constraint that declaratively determines which witnesses can be chosen. The chosen witnesses are used to replace existential quantification, and then the resulting universally quantified clauses are passed to a solver for such clauses. At this step, we can benefit from emergent tools in the area of solving Horn clauses over decidable theories, e.g., HSF [19] or μZ [27]. Such a solver either finds a solution, i.e., a model for uninterpreted relations constrained by the clauses, or returns a counterexample,

which is a resolution tree (or DAG) representing a contradiction. EHSF turns the counterexample into an additional constraint on the set of witness candidates, and continues with the next iteration of the refinement loop. Notably, this refinement loop conjoins constraints that are obtained for all discovered counterexamples. This way EHSF guarantees that previously handled counterexamples are not rediscovered and that a wrong choice of witnesses can be mended.

For the existential clause above, EHSF introduces a witness/Skolem relation sk over variables x and y, i.e., $x \geq 0 \wedge sk(x,y) \rightarrow x \geq y \wedge rank(x,y)$. In addition, since for each x such that $x \geq 0$ holds we need a value y, we require that such x is in the domain of the Skolem relation using an additional clause $x \geq 0 \rightarrow \exists y : sk(x,y)$. In the EHSF approach, the search space of a skolem relation $sk(x,y)$ is restricted by a template function $\text{TEMPL}(sk)(x,y)$. In general, a template is required for both the Skolem relation as well as for its guard. For this paper, templates for the guards can be derived automatically as the projection from the template of the corresponding Skolem relation, therefore we omit them from our presentation.

4. Proof rules for games

In this section, we present proof rules for three kind of games: safety, reachability and parity/LTL games. These proof rules conclude that Eve has a winning strategy by imposing implication and well-foundedness conditions on auxiliary assertions over system variables. For each proof rule we prove its soundness, i.e., a winning strategy exists if the premises are satisfied by some auxiliary assertions, and relative completeness, i.e., if a winning strategy exists then auxiliary assertions satisfying the premises exist under an assumption that the assertion language of our choice is sufficiently expressive. Such correctness criteria are standard for temporal proof rules [33].

4.1 Safety games

We consider a safety game for which Eve has a winning strategy if only states from $safe(v)$ are visited by all plays.

We present the corresponding proof rule in Figure 1. The proof rule relies on an invariant assertion $inv(v)$ that represents a set of states reached by Eve in a winning strategy. We connect the invariant assertion with the reachable states by resorting to reasoning by induction on the number of steps to reach a state. The condition S1 requires that the initial state of the game are considered in $inv(v)$. S2 represent the induction step. Here, we require that for every step from $inv(v)$ executed by Adam there exist a step by Eve that leads back to $inv(v)$. Of course, since the winning condition requires that all states of a play need to satisfy $safe(v)$, we require that all states reached after Adam made a step as well as $inv(v)$ satisfy the assertion $safe(v)$. The former condition is enforced by a conjunct $safe(v')$ in the head of S2. The later condition is guaranteed by S3.

Theorem 1 (Correctness of rule RULESAFE)**.** *The proof rule* RULESAFE *is sound and relatively complete.*

Proof. We split the proof into two parts: soundness and completeness.

Soundness We prove the soundness by contradiction. Assume that there exists an assertion $inv(v)$ that satisfies the premises of RULESAFE, yet the conclusion of RULESAFE does not hold. That is, there is no winning strategy for Eve. Hence, there exists a strategy σ for Adam in which each play reaches a state that violates $safe(v)$. This strategy σ alternates between existential choices of Adam and universal choices of Eve. Let $aux(v)$ be a set of states for which σ provides existentially chosen successors wrt. Adam.

We derive a contradiction by relying on a certain play π that is determined by σ. The play π is constructed iteratively. We start from some root state s_1 of σ, which also satisfies the initial condition $init(v)$. Note that $s_1 \models inv(v)$, due to S1, and $s_1 \models aux(v)$ due to σ. Each iteration round extends the play obtained so far by two states, say s' and s''. We maintain a condition that each such s'' satisfies $inv(v)$ and $aux(v)$. Let s be the last state of the play π constructed so far. Due to our condition, we have $s \models inv(v) \wedge aux(v)$. Then, σ determines a successor state s' such that $(s,s') \models adam(v,v')$, and S2 guarantees that there exists a state s'' such that $(s',s'') \models eve(v,v')$ and $s'' \models inv(v)$. Furthermore, s'' satisfies $aux(v)$ due to σ. Finally, from S2 and S3 follows that $s' \models safe(v)$ and $s'' \models safe(v)$, respectively.

By iteratively constructing π using the above step we obtain a play that satisfies the strategy σ. Thus, we obtain a contradiction, since according to our construction all states in π satisfies $safe(v)$, however σ guarantees that each play eventually reaches a state that violates $safe(v)$.

Completeness Assume that Eve has a winning strategy, say σ, i.e., the conclusion of RULESAFE holds. We prove the completeness claim by showing how to construct $inv(v)$ that satisfies the premises of RULESAFE.

This strategy σ alternates between universal choices of Adam and existential choices of Eve. Let $inv(v)$ be a set of states for which σ provides universally chosen successors wrt. Adam. Since σ is a winning strategy, all states satisfying $inv(v)$ also satisfy $safe(v)$, i.e., $inv(v)$ satisfies S3. $inv(v)$ satisfies S1, since σ guarantees that Eve wins from every initial state. Now we consider an arbitrary state s that satisfies $inv(v)$. σ guarantees that for every successor s' of s wrt. Adam there exists a successor s'' wrt. Eve such that $s'' \models inv(v)$. Furthermore, since σ is winning, we have $s' \models safe(v)$. Thus we conclude that $inv(v)$ satisfies the condition S2 as well. □

4.2 Reachability games

In contrast to safety games, the winning condition of reachability games ensures that a certain set of states called $dst(v)$ is eventually reached by each play. Reasoning about such eventuality properties demands the use of well-founded orders.

We present a rule RULEREACH for proving that Eve has a winning strategy for a reachability property given by an LTL formula $F\,dst(v)$ in Figure 2. RULEREACH requires an invariant assertion $inv(v)$ together with a binary relation $round(v,v')$. Similarly to RULESAFE, we use $inv(v)$ to keep track of states that are reached by Eve. This is captured by R1 and a part of R2. To ensure that Adam makes progress when aiming at the set $dst(v)$ we keep track of pairs of states towards reaching it in $round(v,v')$, see the last conjunct in R2. We note that the proof rule only imposes conditions when $dst(v)$ is not yet reached, as encoded by the second conjunct in R2. Finally, to ensure that $dst(v)$ is eventually reached by each play we require that $round(v,v')$ represents a well-founded relation. Thus, it is impossible to return to $inv(v) \wedge \neg dst(v)$ infinitely many times.

Theorem 2 (Correctness of rule RULEREACH)**.** *The proof rule* RULEREACH *is sound and relatively complete.*

Proof. We split the proof into two parts: soundness and completeness.

Soundness We prove the soundness by contradiction. Assume that there exist assertions $inv(v)$ and $round(v,v')$ that satisfy the premises of RULEREACH, yet the conclusion of RULEREACH does not hold. That is, there is no winning strategy for Eve. Hence, there exists a strategy σ for Adam in which each play never reaches a

Find assertion $inv(v)$ such that:

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

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

Figure 1. Proof rule RULESAFE for a safety game, i.e., the winning condition is given by a formula $G\ safe(v)$.

Find assertions $inv(v)$ and $round(v,v')$ such that:

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

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

Figure 2. Proof rule RULEREACH for a reachability game, i.e., the winning condition is given by a formula $F\ dst(v)$.

state that satisfies $dst(v)$. This strategy σ alternates between existential choices of Adam and universal choices of Eve. Let $aux(v)$ be a set of states for which σ provides existentially chosen successors wrt. Adam. Note that the implication $aux(v) \rightarrow \neg dst(v)$ is valid, since no play determined by σ visits $dst(v)$.

We derive a contradiction by relying on a certain play π that is determined by σ. The play π is constructed iteratively, in a similar way as done in the proof of Theorem 1. We start from some root state s_1 of σ, which satisfies the initial condition $init(v)$. Note that $s_1 \models inv(v)$, due to R1, and $s_1 \models aux(v)$ due to σ. Each iteration round extends the play obtained so far by two states, say s' and s''. We maintain a condition that each such s'' satisfies $inv(v)$ and $aux(v)$. Let s be the last state of the play π constructed so far. Due to our condition, we have $s \models inv(v) \wedge aux(v)$. Then, σ determines a successor state s' such that $(s,s') \models adam(v,v')$, and R2 guarantees that there exists a state s'' such that $(s',s'') \models eve(v,v')$ and $s'' \models inv(v)$. Furthermore, s'' satisfies $aux(v)$ due to σ. Finally, from R2 also follows that $s' \models \neg dst(v)$ and $(s,s'') \models round(v,v')$.

By iteratively constructing $\pi = s_1, s_2, \ldots$ using the above step we obtain a play that satisfies the strategy σ. Thus, there is an infinite sequence of states $s_1, s_3, s_5, \ldots$ that takes states occurring at odd positions in π such that each pair of consecutive states s_{2i-1} and s_{2i+1} is connected by $round(v,v')$, for $i \geq 1$. The existence of such an infinite sequence contradicts the well-foundedness condition imposed by R3.

Completeness Assume that Eve has a winning strategy, say σ, i.e., the conclusion of RULEREACH holds. We prove the completeness claim by showing how to construct $inv(v)$ and $round(v,v')$ that satisfy the premises of RULEREACH.

The strategy σ alternates between universal choices of Adam and existential choices of Eve. Each play $\pi = s_1, s_2, s_3, \ldots$ contributes elements to $inv(v)$ and $round(v,v')$ as follows. Let k be the position of the first occurrence of a state in π that satisfies $dst(v)$, i.e., we have $s_k \models dst(v)$ and $s_i \not\models dst(v)$ for each $i \in 1..k-1$. Such position exists, since the play satisfies $F\,dst(v)$. Then, for each $i \geq 1$ such that $2i-1 \leq k$ we add the state s_{2i-1} to $inv(v)$. Furthermore, for each $i \geq 1$ such that $2i+1 \leq k$ we add the pair of states s_{2i-1} and s_{2i+1} to $round(v,v')$.

We note that the above construction ensures that for each pair of states s and s'' such that $(s,s'') \models round(v,v'')$ holds: i) we have $s \not\models dst(v)$, and ii) there exists a state s' such that $s' \not\models dst(v)$, $(s,s') \models adam(v,v')$, and $(s',s'') \models eve(v,v')$.

We observe that $inv(v)$ satisfies R1, since σ guarantees that Eve wins from every initial state. Now we consider each pair of states s and s' that satisfies the left hand side of R2. σ guarantees that there exists a successor s'' wrt. Eve. Regardless whether $s'' \models dst(v)$ the above construction guarantees that the right-hand side of R2 is satisfied by assigning s, s', and s'' to v, v', and v'', respectively.

Now we show by contradiction that $round(v,v')$ is well-founded. Assume otherwise, i.e., there exists an infinite sequence of states $s_1, s_2, \ldots$ induced by $round(v,v')$. As noted previously, for each pair of consecutive states s_i and s_{i+1} there exists an intermediate state s'_i such that the sequence $s_1, s'_1, s_2, \ldots, s_i, s'_i, s_{i+1}, \ldots$ is a play. Since this play does not visit any state that satisfies $dst(v)$, we obtain a contradiction to the assumption that Eve has a winning strategy. Hence, we conclude that R3 is satisfied. □

4.3 LTL and parity games

Now we show how to solve LTL games and, as a special case, parity games. To state the parity winning condition we assume that the set of all states is partitioned into N subsets that are denoted by the assertions $p_1(v), \ldots, p_N(v)$. Thus, $p_1(v) \vee \cdots \vee p_N(v)$ is valid and for each $1 \leq i < j \leq N$ we have that $p_i(v) \wedge p_j(v)$ is unsatisfiable. Without loss of generality we assume that N is an odd number.

The parity condition states that the system wins the game for a given computation if among the subsets $p_{i_1}(v), \ldots, p_{i_K}(v)$ that are visited infinitely many times by the computation the minimal identifier is odd, i.e., $\min\{i_1, \ldots, i_K\}$ is odd. We can represent

the parity condition by the following LTL formula φ.

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

The first disjunct states that $p_1(v)$ is visited infinitely often, while the second disjunct states that $p_3(v)$ is visited infinitely often and there exists a suffix that neither visits $p_1(v)$ nor $p_2(v)$. The last disjunct states that $p_N(v)$ is visited infinitely often and there is a suffix that visits no other subset.

To solve games where the winning condition is an LTL formula φ, we negate φ and apply a standard technique, e.g., [17], for translating LTL formulas to Büchi automata on the resulting $\neg\varphi$. Let $\mathcal{B}$ be the obtained automaton. We represent $\mathcal{B}$ using assertions over the program counter of the automaton $pc_{\mathcal{B}}$ and the system variables v. Let the initial condition of the automaton be given by $init_{\mathcal{B}}(pc_{\mathcal{B}})$. We represent the transition relation of $\mathcal{B}$ by $next_{\mathcal{B}}(pc_{\mathcal{B}}, v, pc'_{\mathcal{B}})$. This transition relation evolves the value of the program counter of the automaton while taking into consideration the current state of the system given by a valuation of v. Finally, we assume that $acc_{\mathcal{B}}(pc_{\mathcal{B}})$ represents the accepting states of the automaton.

Given a sequence of states $\pi = s_1, s_2, \dots$ we define a run of $\mathcal{B}$ on π to be an infinite sequence of automaton states $q_0, q_1, q_2, \dots$ such that $q_0 \models init_{\mathcal{B}}(pc_{\mathcal{B}})$ and $(q_{i-1}, s_i, q_i) \models next_{\mathcal{B}}(pc_{\mathcal{B}}, v, pc'_{\mathcal{B}})$ for each $i \geq 1$. A run is accepting wrt. the Büchi acceptance condition if it contains infinitely many states that satisfy $acc_{\mathcal{B}}(pc_{\mathcal{B}})$. The automaton $\mathcal{B}$ accepts a play π if there exists an accepting run on π. Note that our construction ensures that if $\mathcal{B}$ accepts π then $\pi \not\models \varphi$.

A proof rule BÜCHITERM for LTL games based on the above automata-theoretic approach [44] is presented in Figure 3. BÜCHITERM requires that the negation of the winning condition is translated into a Büchi automaton $\mathcal{B}$, which together with the system description appears in the proof rule. An interesting property of this proof rule is that it relies on a non-deterministic Büchi automaton representation of the negated winning condition, and does not require any determinization via Rabin, Muller, or Streett acceptance conditions.

We consider a synchronous parallel product of the transition relations of the players and the transition relation of the Büchi automaton, which is expressed in the proof rule by appropriate conjunctions. We use $w = (v, pc_{\mathcal{B}})$ to refer to the vector of the system variables and the program counter of the automaton.

The existence of a winning strategy for Eve depends on the identification of auxiliary assertions $inv(w)$, $aux(w, w', v'')$, $round(w, w', w'')$, and $fair(w, w')$ as follows. $inv(w)$ keeps track of the system states reached by Adam, similarly to RULESAFE and RULEREACH. To deal with the non-determinism in the transition relation of the automaton, we introduce an intermediate bookkeeping assertion $aux(w, w', v'')$, which allows us to decouple the treatment of the automaton state q'' from the selection of s''. $round(w, w', w'')$ contains all triples of adjacent program states occurring in plays. Here, it is more fine-grained than the counterpart in RULEREACH, as we keep track of intermediate states visited by Eve instead of only considering the combined steps (visited by Adam). For keeping track of acceptance $fair(w, w')$ contains all pairs of program states that describe play segments visiting Büchi accepting states at least once. We derive $fair(w, w')$ from $round(w, w', w'')$ using transitive closure-like conditions B4 and B5. Finally, the well-foundedness condition B6 shows that accepting states cannot be visited infinitely many times.

Theorem 3 (Correctness of rule BÜCHITERM)**.** *The proof rule* BÜCHITERM *is sound and relatively complete.*

Proof. We split the proof into two parts: soundness and completeness.

Soundness We prove the soundness by contradiction. Assume that there exist assertions $inv(w)$, $aux(w, w', v'')$, $round(w, w', w'')$, and $fair(w, w')$ that satisfy the premises of BÜCHITERM, yet the conclusion of BÜCHITERM does not hold. That is, there is no winning strategy for Eve. Hence, there exists a strategy σ for Adam in which each play violates φ. This strategy σ alternates between existential choices of Adam and universal choices of Eve. We derive a contradiction by relying on a certain set of trees whose branches are sequences $(s_1, q_1), (s_2, q_2), \dots$ that are jointly determined by σ and the assumed assertions (via BÜCHITERM).

The requisite branches are constructed iteratively, in a similar way as the play construction is done in the proof of Theorem 1. We start from some root state s_1 of σ, which satisfies the initial condition $init(v)$. Then the play is extended from a state s by considering an existential choice s' offered by σ that is followed by an existential choice s'' offered by B2. We obtain appropriate runs $q_0, q_1, \dots$ by applying B1, B2, and B3 for values of v, v', and v'' determined by currently considered s, s', and s'', respectively. Since the automaton $\mathcal{B}$ is non-deterministic, for each s, s', and s'' there is a set of appropriate automaton states. Considering each choice leads to a tree construction, as described below.

First, we consider s_1 and B1, and for each q_1 such that there exists q_0 with $init(s_1) \wedge init_{\mathcal{B}}(q_0) \wedge next_{\mathcal{B}}(q_0, s_1, q_1)$ we add a (s_1, q_1) as a root to our tree. We remember the state q_0 that was used to create each (s_1, q_1). Then, for each tree leaf (s, q) we perform the following tree expansion. First, we consider the state s' that σ provides as a successor of s. Then, we rely on B2, and for each q' such that $next_{\mathcal{B}}(q, s', q')$ holds we add (s', q') as a child node of (s, q). Furthermore, for given s and q, and each s' and q' we take s'' such that $eve(s', s'') \wedge aux(s, q, s', q', s'')$. Now we rely on B3, and for each q'' such that $next_{\mathcal{B}}(q', s'', q'')$ holds we add (s'', q'') as a child node of the corresponding (s', q').

By applying the above tree expansion steps we construct a set of trees where every branch is a sequence $(s_1, q_1), (s_2, q_2), \dots$ that comes with the corresponding initial automaton state q_0. Note that $s_1, s_2, \dots$ is a play determined by σ, hence it violates φ. Thus, there exists a branch for which the sequence $q_0, q_1, \dots$ is an accepting run of $\mathcal{B}$ for the corresponding play determined by the branch. Let $(s_{i_1}, q_{i_1}), (s_{i_2}, q_{i_2}), \dots$ be a subsequence such that $q_{i_j} \models acc_{\mathcal{B}}(pc_{\mathcal{B}})$ for each $j \geq 1$. Then, each pair (s_{i_j}, q_{i_j}) contributes

$$((s_{i_j}, q_{i_j}), (s_{i_{j+1}}, q_{i_{j+1}}))$$

(or its closest neighbour visited by Adam) to $fair(w, w')$. Thus, the condition B6 is violated.

Completeness Assume that Eve has a winning strategy, say σ, i.e., the conclusion of BÜCHITERM holds. We prove the completeness claim by showing how to construct $inv(w)$, $aux(v, w', v'')$, $round(v, w', w'')$, and $fair(w, w')$ that satisfy the premises of BÜCHITERM.

The strategy σ alternates between universal choices of Adam and existential choices of Eve. Each play $\pi = s_1, s_2, s_3, \dots$ contributes elements to $inv(w)$, $aux(v, w', v'')$, and $round(w, w', w'')$ in the following way through an appropriate sequence of automaton states $q_0, q_1, q_2, \dots$. Since $\pi \models \varphi$, we note that π is not accepted by $\mathcal{B}$. Hence, either there is an infinite run $q_0, q_1, q_2, \dots$ that is not accepting, or the exists a finite run $q_0, \dots, q_n$ that cannot be extended (i.e., there is no automaton

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

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

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

Figure 3. Proof rule BÜCHITERM for an LTL game, i.e., the winning condition is given by an LTL formula φ. From $\neg\varphi$ we obtain a Büchi automaton $\mathcal{B}$ with an initial condition $init_{\mathcal{B}}(pc_{\mathcal{B}})$, a transition relation $next_{\mathcal{B}}(pc_{\mathcal{B}}, v, pc'_{\mathcal{B}})$, and accepting states $acc_{\mathcal{B}}(pc_{\mathcal{B}})$.

state q_{n+1} such that $next_{\mathcal{B}}(q_n, s_n, s_{n+1})$). In either case, for each $i \geq 0$ (and $2i + 1 \leq n$ if the run is finite) we let (s_{2i-1}, q_{2i-1}) be an element of $inv(w)$, $(s_{2i-1}, q_{2i-1}, s_{2i}, q_{2i}, s_{2i+1})$ be an element of $aux(w, w', v'')$, and $(s_{2i-1}, q_{2i-1}, s_{2i}, q_{2i}, s_{2i+1}, q_{2i+1})$ be an element of $round(w, w', w'')$. Then we define $fair(w, w')$ for the obtained $round(w, w', w'')$ as the least solution of B4 and B5.

Since the run is not accepting, it visits accepting states only finitely many times. Hence, $fair(w, w')$ is well-founded. □

5. Case study: Cinderella-Stepmother games

In this section we illustrate our constraint-based approach to solving games applying it to the Cinderella-Stepmother game introduced in Section 2. We consider five variants of this game corresponding to different winning conditions. In Section 7 we report on running times required for solving these games using CONSYNTH.

5.1 Games with Cinderella's safety objective

In these games, we attempt to obtain winning strategies for Cinderella in her attempt to keep the buckets from overflowing. The winning condition for Cinderella is $G\ \neg overflow(v)$. As mentioned in Section 2, the Cinderella player has simple winning strategies for bucket capacity $c \geq 3$. For values $2 \leq c \leq 3$, the strategies are more involved. (For values $c \leq 2$ there are no strategies for the player Cinderella to win the game).

Round strategy We define the first game using the value $c = 3$ for the bucket capacity. A winning strategy might follow an alternation of consecutive buckets that are emptied. Accordingly, we use an auxiliary variable r for a pair of buckets to be emptied, to remember the previous choice made by the Cinderella player. The tuple of game variables contains the five bucket variables from v and is extended with the round variable r as follows: $w = (b_1, b_2, b_3, b_4, b_5, r)$. The initial states assertion sets the round variable to $r = 1$. We let Adam play the role of Stepmother and therefore the transition relation of Adam is based on the assertion $stepmother(v, v')$, while the transition relation of Eve is given by $cinderella(v, v')$. (Both constraints are given in Section 2.)

$$
\begin{aligned}
init(w) &= (b_1 = 0 \wedge \cdots \wedge b_5 = 0 \wedge r = 1) \\
eve(w, w') &= cinderella(v, v') \\
adam(w, w') &= (stepmother(v, v') \wedge r' = r)
\end{aligned}
$$

Considering the safety condition $obj(w) = G(\neg overflow(v))$, we instantiate the proof rule from Figure 1 as follows.

$$(init(w), adam(w, w'), eve(w, w')) \models obj(w)$$

There exists a strategy for Eve provided that the premises of the proof rule are satisfied. These premises are Horn clauses over the auxiliary assertion $inv(w)$. We apply a solver, e.g., CONSYNTH, to find a solution for the auxiliary assertion. The clauses S1 and S3 are universally quantified over the game variables, while the existentially quantified clause S2 is skolemized in the CONSYNTH approach. We use the skolem relation $sk(w, w', w'')$ to denote the witness constraint corresponding to the existentially quantified variables w''.

$$
\begin{aligned}
inv(w) \wedge stepmother(v, v') \wedge r' = r \wedge sk(w, w', w'') \rightarrow \\
cinderella(v', v'') \wedge inv(w'')
\end{aligned}
$$

CONSYNTH requires a template for the skolem relation and we present below the intuition behind this constraint. For each of the five disjuncts from $cinderella(v, v')$ transition relation, we add guards (one guard exclusive to the others) and update the value of the round variable. We use $c_1(v, v')$ to $c_5(v, v')$ to denote the five disjuncts from the transition relation of the Cinderella player introduced in Section 2. We obtain the following template constraint.

$$
\begin{aligned}
\text{TEMPL}(sk)(w, w', w'') = (r' = 1 \wedge r'' = ?_1 \wedge c_1(v', v'') \vee \\
r' = 2 \wedge r'' = ?_2 \wedge c_2(v', v'') \vee \\
r' = 3 \wedge r'' = ?_3 \wedge c_3(v', v'') \vee \\
r' = 4 \wedge r'' = ?_4 \wedge c_4(v', v'') \vee \\
r' = 5 \wedge r'' = ?_5 \wedge c_5(v', v''))
\end{aligned}
$$

The template parameters are denoted by "?"-variables and different subscripts indicate distinct template parameters.

Our approach is able to synthesize automatically the values used to update the round and implicitly the order in which the Cinderella player should alternate emptying the buckets. CONSYNTH returns the solution $?_1 = 4, ?_2 = 1, ?_3 = 1, ?_4 = 3, ?_5 = 1$. Corresponding to this solution, the strategy for the Cinderella player consists of a repeating sequence of three player moves:

1. Since initially $r = 1$ and the first disjunct is enabled, decide to empty buckets 1 and 2 and update the round variable $r'' = 4$.
2. Since the disjunct $r' = 4 \wedge r'' = ?_4$ is enabled, decide to empty buckets 4 and 5 and update the round variable to $r'' = 3$.
3. Since the disjunct $r' = 3 \wedge r'' = ?_3$ is enabled, decide to empty buckets 3 and 4 and update the round variable to $r'' = 1$.

After these three moves, r has value 1, the first disjunct is again enabled and the strategy will continue with the first move/decision above. This strategy ensures that the Cinderella player empties

often enough all the buckets and therefore the Stepmother player cannot enforce an overflow. This game is won by the Cinderella player based on the round strategy described above.

Second strategy We show how our approach can be used to obtain a strategy for the case of the game that is more difficult for Cinderella to win, i.e., $c = 2$.

We fix the roles of the two players similar to the previous paragraph: $eve(v, v') = cinderella(v, v')$ and $adam(v, v') = stepmother(v, v')$. To explain the rationale behind the Cinderella's decisions for this case, we refer to the proof rule from Figure 1. We repeat the second clause S2 instantiated for the two players of the C-S game:

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

To change the state of the system from v' to v'', the strategy for the Cinderella player takes into consideration her previous move (reflected in variables v) and the reply by Stepmother (reflected in variables v'). Therefore the template for the strategy considers five cases depending on which buckets the Cinderella player may have emptied in the previous turn:

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

The T_{ij} conjuncts refer to non-obvious knowledge and relate to an invariant stating that each pair of non-adjacent buckets should have total contents at most 1 [28]. The first part of the template, i.e., T_{12}, is based on the intuition that if in the previous round Cinderella emptied buckets 1 and 2 ($b_1 = 0 \wedge b_2 = 0$), then during the next round she will decide to empty another pair of buckets. That is, either the pair of buckets 3 and 4 ($b_3'' = 0 \wedge b_4'' = 0$) or the pair of buckets 4 and 5 ($b_4'' = 0 \wedge b_5'' = 0$) will be emptied. However, the condition on which to decide if to empty buckets 3 and 4 or buckets 4 and 5 is not straightforward. We use template parameters and leave the decision to be automated by our game solving approach. The formula T_{12} is provided as follows.

$$\begin{aligned}T_{12}(v', v'') = (&?_5 * b_5' + ?_2 * b_2' \leq 1 \wedge b_3'' = 0 \wedge b_4'' = 0 \vee \\ &?_1 * b_1' + ?_3 * b_3' \leq 1 \wedge b_4'' = 0 \wedge b_5'' = 0)\end{aligned}$$

Following a similar argument, we obtain the formulas that complete the definition of the template $\textsc{Templ}(sk)(v, v', v'')$:

$$\begin{aligned}T_{23}(v', v'') = (&?_1 * b_1' + ?_3 * b_3' \leq 1 \wedge b_4'' = 0 \wedge b_5'' = 0 \vee \\ &?_2 * b_2' + ?_4 * b_4' \leq 1 \wedge b_5'' = 0 \wedge b_1'' = 0) \\ T_{34}(v', v'') = (&?_2 * b_2' + ?_4 * b_4' \leq 1 \wedge b_5'' = 0 \wedge b_1'' = 0 \vee \\ &?_3 * b_3' + ?_5 * b_5' \leq 1 \wedge b_1'' = 0 \wedge b_2'' = 0) \\ T_{45}(v', v'') = (&?_3 * b_3' + ?_5 * b_5' \leq 1 \wedge b_1'' = 0 \wedge b_2'' = 0 \vee \\ &?_4 * b_4' + ?_1 * b_1' \leq 1 \wedge b_2'' = 0 \wedge b_3'' = 0) \\ T_{51}(v', v'') = (&?_4 * b_4' + ?_1 * b_1' \leq 1 \wedge b_2'' = 0 \wedge b_3'' = 0 \vee \\ &?_5 * b_5' + ?_2 * b_2' \leq 1 \wedge b_3'' = 0 \wedge b_4'' = 0).\end{aligned}$$

The template parameters are marked as before by "?"-variables and we aim to obtain solutions for the five template parameters $?_1, ?_2, ?_3, ?_4, ?_5$. Our approach is indeed able to synthesize automatically values for these parameters. The tool ConSynth returns the solutions $?_1 = 1, ?_2 = 1, ?_3 = 1, ?_4 = 1, ?_5 = 1$. The resulting strategy for the Cinderella player guarantees that no state with overflow can be reached. For a different perspective, we refer the interested reader to an article on (non-automated) reasoning and invariants needed to establish strategies for the Cinderella-Stepmother game similar to the ones we synthesize [28].

5.2 Game with Stepmother's reachability objective

We continue illustrating our approach with the Cinderella-Stepmother game, this time based on a reachability objective: the winning condition for the Stepmother player requires that a state with overflow is reached, $obj(v) = F\ overflow(v)$. For this game, we use the bucket capacity $c = 1.4$, a value for which the Stepmother has indeed a winning strategy. To derive this strategy, we instantiate the proof rule for the reachability game as follows.

$$\begin{aligned}init(v) &= (b_1 = 0 \wedge \cdots \wedge b_5 = 0) \\ eve(v, v') &= stepmother(v, v') \\ adam(v, v') &= cinderella(v, v')\end{aligned}$$

Next we provide a template corresponding to the existentially quantified clause. The insight behind the template is that the quantity of water from each bucket increases during the turn of Stepmother, but without specifying the amount, i.e., $(b_i'' = b_i' + ?_i \wedge ?_i \geq 0)$.

$$\begin{aligned}\textsc{Templ}(sk)(v, v', v'') = (&?_1 + \cdots + ?_5 = 1 \wedge \\ &\bigwedge_{i \in \{1..5\}} (b_i'' = b_i' + ?_i \wedge ?_i \geq 0)\end{aligned}$$

Our approach computes the auxiliary assertions that are required by the reachability proof rule and a witness for the existential quantifier. The witness instantiates the template parameters and represents the Stepmother's strategy to ensure that the buckets eventually overflow no matter what moves are made by the Cinderella player.

$$\begin{aligned}sk(v, v') = (&b_1' = b_1 + 0.8 \wedge b_2' = b_2 \\ &\wedge b_3' = b_3 + 0.1 \wedge b_4' = b_4 \wedge b_5' = b_5 + 0.1)\end{aligned}$$

In this case, since the addition of water is done in non-adjacent buckets, e.g., b_1 and b_3, eventually the game reaches an overflow state, and the Stepmother is the player to win this game.

5.3 Games with Cinderella's LTL objectives

Apart from games with safety and reachability objectives, our approach is able to handle games with more general LTL objectives. For this game, we use the following player roles.

$$\begin{aligned}init(v) &= (b_1 = 0 \wedge \cdots \wedge b_5 = 0) \\ eve(v, v') &= cinderella(v, v') \\ adam(v, v') &= stepmother(v, v')\end{aligned}$$

We use the value $c = 1.4$ for the bucket capacity, similar to Section 5.2. As already explained, with this value Stepmother has a strategy to win the game with the objective $\varphi(v) = F overflow(v)$. Consequently, Cinderella does not have a strategy to win the game with the objective set to the complement formula, i.e., $\neg\varphi(v) = G\ \neg overflow(v)$. For this section, we formalize a winning condition that is a weaker logical formula than $\neg\varphi(v)$ for which Cinderella has a winning strategy. The objective constraint $GF\ \neg overflow(v)$ states that an overflow state does not occur infinitely often in the plays of the game.

More generally, we use $color$ to indicate the most significant bucket for which an overflow occurs.

- A state without overflow: $(color = 0)$.
- A state with overflow such that i is the smallest index of those that correspond to buckets that have overflown: $(color = i)$.

We group the states of the system based on the truth value of the predicates $color = i$ as follows.

$$p_0(v) = (color = 0) = (b_1 \leq 1.4 \wedge \cdots \wedge b_5 \leq 1.4)$$
$$p_1(v) = (color = 1) = (b_1 > 1.4)$$
$$p_2(v) = (color = 2) = (b_1 \leq 1.4 \wedge b_2 > 1.4)$$
$$p_3(v) = (color = 3 \vee color = 4 \vee color = 5) = \ldots$$

A winning condition corresponding to a value i ensures that states from $p_i(v)$ occur infinitely often in the plays of the system, and that i is the smallest value for which states occur infinitely often.

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

Our approach for solving games with LTL objectives proceeds in three steps: 1) complement the LTL formula φ representing the winning condition; 2) construct a Büchi automaton corresponding to the complemented formula $\neg\varphi$; 3) instantiate the proof rule from Figure 3 using the Büchi automaton representation.

LTL game 1 For the first LTL game, we define the objective for the Cinderella player $obj(v) = win(v, 0) = GF\ p_0(v)$. We complement the objective formula to obtain $FG\ \neg p_0(v) = FG\ overflow(v)$, then construct the Büchi automaton corresponding to the complemented formula as follows.

$$\begin{aligned} init_{\mathcal{B}}(pc_{\mathcal{B}}) &= (pc_{\mathcal{B}} = 0) \\ next_{\mathcal{B}}(pc_{\mathcal{B}}, v, pc'_{\mathcal{B}}) &= (pc_{\mathcal{B}} = 0 \wedge pc'_{\mathcal{B}} = 0 \vee \\ &\quad pc_{\mathcal{B}} = 0 \wedge pc'_{\mathcal{B}} = 1 \wedge overflow(v) \vee \\ &\quad pc_{\mathcal{B}} = 1 \wedge pc'_{\mathcal{B}} = 1 \wedge overflow(v)) \\ acc_{\mathcal{B}}(pc_{\mathcal{B}}) &= (pc_{\mathcal{B}} = 1) \end{aligned}$$

We instantiate the proof rule from Figure 3 as follows.

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

There exists a strategy for Eve provided that the premises of the proof rule are satisfied. These premises are Horn clauses over the auxiliary assertion $inv(w)$, $aux(w, w', v'')$, $round(w, w', w'')$, and $fair(w, w')$. We apply ConSynth to find a solution for the auxiliary assertions. The clause B2 is an existentially quantified clause. By skolemization of the existential clause B2 we obtain the following.

$$\begin{aligned} inv(w)\ &\wedge stepmother(v, v') \wedge next_{\mathcal{B}}(pc_{\mathcal{B}}, v, pc'_{\mathcal{B}}) \\ &\wedge sk(w, w', v'') \rightarrow cinderella(v', v'') \wedge aux(w, w', v'') \end{aligned}$$

Using the template described in the paragraph "Second strategy" from Section 5.1, our approach is able to derive solutions for the auxiliary assertions and the following template parameters $?_1 = ?_2 = ?_3 = ?_4 = ?_5 = 1$. We conclude that Cinderella is the player to win this game, and that her strategy ensures that states without overflow occur infinitely often in the plays of the game.

LTL game 2 For the second LTL game, we define the objective for the Cinderella player $win(v, 0) \vee win(v, 2)$. The objective for the Stepmother player is $win(v, 1) \vee win(v, 3)$. The formula corresponding to the Cinderella's objective:

$$\varphi = (GF\ p_0(v) \vee (GF\ p_2(v) \wedge FG\ \neg p_1(v) \wedge FG\ \neg p_0(v))).$$

The complemented formula is

$$\neg\varphi = (FG\ \neg p_0(v) \wedge (FG\ \neg p_2(v) \vee GF\ p_1(v) \vee GF\ p_0(v))).$$

The Büchi automaton corresponding to the complemented formula contains 10 distinct control states, from which two are accepting states. Using our proof rule, we are able to compute automatically auxiliary assertions and obtain that the same second strategy is winning for the Cinderella player.

Note that for the player Cinderella, the LTL game 2 (with objective $win(v, 0) \vee win(v, 2)$) is easier to win than the LTL game 1 (with objective $win(v, 0)$). However, the relation between the two objectives is not immediately usable in a deductive approach like ours. We presented both LTL games 1 and 2, since our approach based on the proof rule from Figure 3 constructs different Buechi automata and different auxiliary assertions for the two objectives.

6. Case study: program repair/synthesis games

In this section we illustrate how our constraint-based approach to solving games applies to the synthesis of reactive programs from temporal specifications. We consider synthesis problems obtained from program repair questions, see Section 6.1 and Section 6.2, as well as inference of thread synchronization, see Section 6.3. In Section 7 we report on running times required for solving these games using ConSynth.

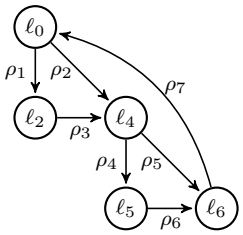
6.1 Program repair game with safety objective

We model program repair as a game following [29]. That is, given a set of suspect statements, we look for a modification of those program statements such that the modified program satisfies its specification. For the first repair game, we assume that a program is given by a tuple $(init(v), next(v, v'), error(v))$ that represents initial states, a transition relation, and error states, respectively.

As an example we consider the program shown in Figure 3 in [29]. The program has three program variables $v = (l, gl, pc)$. The variable l models a lock, the variable gl is used to keep track of the status of the lock, while the variable pc is the program counter variable. The initial states of the program are

$$init(v) = (gl = 0 \wedge l = 0 \wedge pc = \ell_0).$$

We show the control-flow graph of the program below.



The transition relation is defined as follows.

$$\begin{aligned} \rho_1(v, v') &= (pc = \ell_0 \wedge pc' = \ell_2 \wedge l' = l \wedge gl' = gl) \\ \rho_2(v, v') &= (pc = \ell_0 \wedge pc' = \ell_4 \wedge l' = l \wedge gl' = gl) \\ \rho_3(v, v') &= (pc = \ell_2 \wedge pc' = \ell_4 \wedge l \leq 0 \wedge l' = 1) \\ \rho_4(v, v') &= (pc = \ell_4 \wedge pc' = \ell_5 \wedge gl \neq 0 \wedge l' = l \wedge gl' = gl) \\ \rho_5(v, v') &= (pc = \ell_4 \wedge pc' = \ell_6 \wedge gl = 0 \wedge l' = l \wedge gl' = gl) \\ \rho_6(v, v') &= (pc = \ell_5 \wedge pc' = \ell_6 \wedge l \geq 1 \wedge l' = 0 \wedge gl' = gl) \\ \rho_7(v, v') &= (pc = \ell_6 \wedge pc' = \ell_0 \wedge l' = l) \\ next(v, v') &= (\rho_1(v, v') \vee \cdots \vee \rho_7(v, v')) \end{aligned}$$

Note that in $\rho_3(v, v')$ and $\rho_7(v, v')$ the variable gl is assigned a non-deterministic value, since it is not constrained by the corresponding assertions. The execution of the program enters an error state at location ℓ_2 if the lock variable l is held, and at location ℓ_5 if the lock variable l is not held, i.e., we have

$$error(v) = (pc = \ell_2 \wedge l = 1 \vee pc = \ell_5 \wedge l = 0).$$

We instantiate the safety game proof rule such that the system role is played by the program transition relation and the environment role is to provide inputs to the program (in this case, the pro-

gram does not expect any inputs).

$$\begin{aligned} eve(v,v') &= next(v,v') \\ adam(v,v') &= skip(v,v') \\ obj(v) &= G\,\neg error(v) \end{aligned}$$

A repair of the program restricts the transition relation of the program such that $G\ \neg error(v)$ holds. To this end, we provide a template corresponding to the existentially quantified clause of the proof rule:

$$\begin{aligned} \textsc{Templ}(sk)(v,v',v'') = (pc' = \ell_2 \wedge pc'' = \ell_4 \wedge gl'' = ?_1 \\ \vee\ pc' = \ell_6 \wedge pc'' = \ell_0 \wedge gl'' = ?_2) \end{aligned}$$

Our algorithm returns the witness for the existential quantifier clause that instantiates the template parameters $?_1 = 1$ and $?_2 = 0$. This corresponds to a repaired program that assigns the value 1 to gl at location ℓ_2, and assigns the value 0 to gl at location ℓ_6. We obtain the same program repair as the solution originally presented in [29].

6.2 Concurrent program repair games with safety and response objectives

We illustrate how our approach can be applied to concurrent program repair problems, and in particular to repair problems under fairness assumptions. We use the CRITICAL SECTION example from Figure 5 in [29] for this purpose. In this example, the assignment $turn1B = false$ at location ℓ_2 is faulty. The goal is to repair this assignment, and hence, the entire program by checking if there exists an assignment to the variable $turn1B$ from its domain $\{true,\ false\}$ such that the resulting program satisfies certain temporal properties. These properties are used in directing the repair process towards the correct version of the program.

Let $(f1a, f1b, t1b, f2a, f2b, t2b)$ be abbreviations of original variable names (*flag1A*, *flag1B*, *turn1B*, *flag2A*, *flag2B*, *turn2B*). We encode the original program over variables $v = (pc_1, pc_2, x, y, f1a, f1b, t1b, f2a, f2b, t2b)$ using an initial condition $init(v)$ such that

$$\begin{aligned} init(v) = (f1a = 0 \wedge f1b = 0 \wedge t1b = 0 \wedge f2a = 0 \\ \wedge\ f2b = 0 \wedge t2b = 0 \wedge pc_1 = \ell_1 \wedge pc_2 = \ell_1), \end{aligned}$$

and a transition relation $next(v,v')$. Since the program is multi-threaded with two threads, we give $next(v,v')$ as a disjunction of transition relations of individual threads

$$next_1(v,v') \wedge pc_2' = pc_2 \vee next_2(v,v') \wedge pc_1' = pc_1.$$

For the first thread we define (note that we explicate assignments of a non-deterministic value to a variable z by $z' = ND$ and we omit equalities for variables that do not change, hence, each variable z that does not appear in $z' = \ldots$ is constrained by $z' = z$):

$$\begin{aligned} next_1(v,v') = (&pc_1 = \ell_1 \wedge pc_1' = \ell_2 \wedge f1a' = 1 \vee \\ &pc_1 = \ell_2 \wedge pc_1' = \ell_3 \wedge t1b' = ND \vee \\ &pc_1 = \ell_3 \wedge f1b = 1 \wedge t1b = 1 \wedge pc_1' = \ell_3 \vee \\ &pc_1 = \ell_3 \wedge (f1b = 0 \vee t1b = 0) \wedge pc_1' = \ell_4 \vee \\ &pc_1 = \ell_4 \wedge pc_1' = \ell_5 \wedge f1a' = 0 \vee \\ &pc_1 = \ell_5 \wedge t1b = 1 \wedge pc_1' = \ell_6 \wedge f2a' = 1 \vee \\ &pc_1 = \ell_5 \wedge t1b = 1 \wedge pc_1' = \ell_9 \vee \\ &pc_1 = \ell_6 \wedge pc_1' = \ell_7 \wedge t2b' = 1 \vee \\ &pc_1 = \ell_7 \wedge f2b = 1 \wedge t2b = 1 \wedge pc_1' = \ell_7 \vee \\ &pc_1 = \ell_7 \wedge (f2b = 0 \vee t2b = 0) \wedge pc_1' = \ell_8 \vee \\ &pc_1 = \ell_8 \wedge pc_1' = \ell_9 \wedge f2a' = 0 \vee \\ &pc_1 = \ell_9 \wedge pc_1' = \ell_1). \end{aligned}$$

Note that the second disjunct above leaves the value of $t1b$ unrestricted, as denoted by $t1b' = ND$. For the second thread we define:

$$\begin{aligned} next_2(v,v') = (&pc_2 = \ell_1 \wedge pc_2' = \ell_2 \wedge f1b' = 1 \vee \\ &pc_2 = \ell_2 \wedge pc_2' = \ell_3 \wedge t1b' = 0 \vee \\ &pc_2 = \ell_3 \wedge f1a = 1 \wedge t1b = 0 \wedge pc_2' = \ell_3 \vee \\ &pc_2 = \ell_3 \wedge (f1a = 0 \vee t1b = 1) \wedge pc_2' = \ell_4 \vee \\ &pc_2 = \ell_4 \wedge pc_2' = \ell_5 \wedge f2b' = 1 \vee \\ &pc_2 = \ell_5 \wedge pc_2' = \ell_6 \wedge t2b' = 0 \vee \\ &pc_2 = \ell_6 \wedge f2a = 1 \wedge t2b = 0 \wedge pc_2' = \ell_6 \vee \\ &pc_2 = \ell_6 \wedge (f2a = 0 \vee t2b = 1) \wedge pc_2' = \ell_7 \vee \\ &pc_2 = \ell_7 \wedge pc_2' = \ell_8 \wedge f2b' = 0 \vee \\ &pc_2 = \ell_8 \wedge pc_2' = \ell_9 \wedge f1b' = 0 \vee \\ &pc_2 = \ell_9 \wedge pc_2' = \ell_1). \end{aligned}$$

In the original problem there are two properties directed by which the program should be repaired. The first property requires that the two threads do not enter their critical sections at the same time. This property is specified with the following LTL formula.

$$\begin{aligned} \varphi_1(v) = G\,((pc_1 = \ell_4 \rightarrow pc_2 \neq \ell_4) \wedge \\ (pc_1 = \ell_8 \rightarrow pc_2 \neq \ell_7)) \end{aligned}$$

The second property requires that neither of the threads can be in a deadlock state. This property is specified as follows.

$$\begin{aligned} \varphi_2(v) = G\,((pc_1 = \ell_3 \rightarrow F\,(pc_1 = \ell_4)) \wedge \\ (pc_1 = \ell_7 \rightarrow F\,(pc_1 = \ell_8)) \wedge \\ (pc_2 = \ell_3 \rightarrow F\,(pc_2 = \ell_4)) \wedge \\ (pc_2 = \ell_6 \rightarrow F\,(pc_2 = \ell_7))) \end{aligned}$$

Safety game Doing the program repair using the first property amounts to applying the safety proof rule from Figure 1 to find $inv(v)$. Since there is no interaction with the environment, $adam(v,v')$ will simply be equivalent with $skip(v,v')$. To apply our proof rule, we use $next(v,v')$ for $eve(v,v')$ and $\varphi_1(v)$ as the winning condition obj. We use the following template for the existential clause:

$$\textsc{Templ}(sk)(v,v',v'') = (t1b'' = ?_1).$$

CONSYNTH computes the solution $?_1 = 1$, and correspondingly we obtain a modified version of $next_1(v,v')$ where the non-deterministic assignment $t1b' = ND$ from the second disjunct is replaced by $t1b' = 1$.

Fair LTL game The second property relies on fairness assumptions. To deal with the fairness, we apply a transformation technique from [36] that reduces fair parallelism semantics to the usual parallelism semantics. The idea is to use the equivalence $P \models_{fair} \Phi$ if and only $T_{fair}(P) \models \Phi$, where P is the original program, T_{fair} is the fair transformation function, $T_{fair}(P)$ is the transformed program with embedded tracking of fairness, and Φ is the property to check. The transformation does not change the initial states of the program, but it significantly modifies the semantics of the transition relation of the program. A counter variable is introduced for each thread from the program, and the first statement of each loop is strengthened by adding a guard and an update involving the counter variables. See [36] for details. For our example program, the transformation:

- introduces the counters k_1 and k_2,
- adds the guard $k_1 \leq k_2$ and the update constraint $(k_1' = ND \wedge k_2' = k_2 - 1)$ to the first, third and ninth disjunct from $next_1(v,v')$,

- adds the guard $k_2 \leq k_1$ and the update constraint $(k_1' = k_1 - 1 \wedge k_2' = ND)$ to the first, third and sixth disjunct in $next_2(v, v')$.

Let $initT(v, k_1, k_2) = init(v)$ be the initial condition of the transformed program. We refer to the transformed transition relations as $nextT_1(v, k_1, k_2, v', k_1', k_2')$ and $nextT_2(v, k_1, k_2, v', k_1', k_2')$, and present them below.

$$
\begin{aligned}
&nextT_1(v, k_1, k_2, v', k_1', k_2') = \\
&\quad (pc_1 = \ell_1 \wedge pc_1' = \ell_2 \wedge f1a' = 1 \\
&\quad \wedge k_1 \leq k_2 \wedge k_1' = ND \wedge k_2' = k_2 - 1 \vee \\
&\quad \dots \vee \\
&\quad pc_1 = \ell_3 \wedge f1b = 1 \wedge t1b = 1 \wedge pc_1' = \ell_3 \\
&\quad \wedge k_1 \leq k_2 \wedge k_1' = ND \wedge k_2' = k_2 - 1 \vee \\
&\quad \dots \vee \\
&\quad pc_1 = \ell_7 \wedge f2b = 1 \wedge t2b = 1 \wedge pc_1' = \ell_7 \\
&\quad \wedge k_1 \leq k_2 \wedge k_1' = ND \wedge k_2' = k_2 - 1 \vee \\
&\quad \dots)
\end{aligned}
$$

$$
\begin{aligned}
&nextT_2(v, k_1, k_2, v', k_1', k_2') = \\
&\quad (pc_2 = \ell_1 \wedge pc_2' = \ell_2 \wedge f1b' = 1 \\
&\quad \wedge k_2 \leq k_1 \wedge k_1' = k_1 - 1 \wedge k_2' = ND \vee \\
&\quad \dots \vee \\
&\quad pc_2 = \ell_3 \wedge f1a = 1 \wedge t1b = 0 \wedge pc_2' = \ell_3 \\
&\quad \wedge k_2 \leq k_1 \wedge k_1' = k_1 - 1 \wedge k_2' = ND \vee \\
&\quad \dots \vee \\
&\quad pc_2 = \ell_6 \wedge f2a = 1 \wedge t2b = 0 \wedge pc_2' = \ell_6 \\
&\quad \wedge k_2 \leq k_1 \wedge k_1' = k_1 - 1 \wedge k_2' = ND \vee \\
&\quad \dots)
\end{aligned}
$$

The second property is more complicated than the first property since it involves nesting of temporal operators. Like the case for the first property, we assume $adam(v, v')$ to be equivalent with $skip(v, v')$. We make $nextT(v, v')$ to play the role of $eve(v, v')$, and $G\ ((pc_1 = \ell_3 \rightarrow F\ pc_1 = \ell_4) \wedge (pc_1 = \ell_7 \rightarrow F\ pc_1 = \ell_8) \wedge (pc_2 = \ell_3 \rightarrow F\ pc_2 = \ell_4) \wedge (pc_2 = \ell_6 \rightarrow F\ pc_2 = \ell_7))$ is now a winning condition obj.

We reuse the template used for the previous game and we get exactly the same solution. That is, we determine $t1b' = ND$ to $t1b' = 1$ in the second disjunct of $nextT_1(v, k_1, k_2, v', k_1', k_2')$.

6.3 Synthesis of synchronization game with safety objective

Synthesis of synchronization in multi-threaded programs [45] can be automated using our approach. For illustration, we use the example program from Figure 1 in [45] and represent it using a tuple $(init(v), next(v, v'), error(v))$ for the case when three threads are involved in computation. The program variables are $v = (x, y_1, y_2, z, pc_1, pc_2, pc_3)$, the initial states are described by $init(v) = (x = 0 \wedge z = 0 \wedge pc_1 = \ell_1 \wedge pc_2 = \ell_1 \wedge pc_3 = \ell_1)$. The transition relation of the program is

$$
\begin{aligned}
next(v, v') = (&next_1(v, v') \wedge pc_2' = pc_2 \wedge pc_3' = pc_3 \vee \\
&next_2(v, v') \wedge pc_1' = pc_1 \wedge pc_3' = pc_3 \vee \\
&next_3(v, v') \wedge pc_1' = pc_1 \wedge pc_2' = pc_2)
\end{aligned}
$$

such that

$$
\begin{aligned}
&next_1(v, v') = \\
&\quad (pc_1 = \ell_1 \wedge pc_1' = \ell_2 \wedge x' = x + z \wedge skip(y_1, y_2, z) \vee \\
&\quad pc_1 = \ell_2 \wedge pc_1' = \ell_3 \wedge x' = x + z \wedge skip(y_1, y_2, z)) \\
&next_2(v, v') = \\
&\quad (pc_2 = \ell_1 \wedge pc_2' = \ell_2 \wedge z' = z + 1 \wedge skip(x, y_1, y_2) \vee \\
&\quad pc_2 = \ell_2 \wedge pc_2' = \ell_3 \wedge z' = z + 1 \wedge skip(x, y_1, y_2))
\end{aligned}
$$

$$
\begin{aligned}
&next_3(v, v') = \\
&\quad (pc_3 = \ell_1 \wedge pc_3' = \ell_2 \wedge x = 1 \wedge y_1' = 3 \wedge skip(x, y_2, z) \vee \\
&\quad pc_3 = \ell_1 \wedge pc_3' = \ell_2 \wedge x = 2 \wedge y_1' = 6 \wedge skip(x, y_2, z) \vee \\
&\quad pc_3 = \ell_1 \wedge pc_3' = \ell_2 \wedge (x \leq 0 \vee x \geq 3) \wedge y_1' = 5 \wedge \\
&\quad skip(x, y_2, z) \vee \\
&\quad pc_3 = \ell_2 \wedge pc_3' = \ell_3 \wedge y_2' = x \wedge skip(x, y_1, z) \vee \\
&\quad pc_3 = \ell_3 \wedge pc_3' = \ell_4 \wedge y_1 \neq y_2 \wedge skip(x, y_1, y_2, z)).
\end{aligned}
$$

Different interleavings of the three threads lead to different values of y_1 and y_2. An assertion in the third thread at location ℓ_3 requires that the values given to y_1 and y_2 are not equal, i.e., we have

$$error(v) = (pc_3 = \ell_3 \wedge y_1 = y_2).$$

For the given program, some interleavings lead to the values of y_1 and y_2 being equal, while other interleavings lead to distinct values for the two variables. The goal of [45] is to add synchronization to the program such that the assertion holds on all executions. To apply our proof rule to this problem, we encode the choice between executing a single step and executing an atomic section using auxiliary variables. The transition relation of the program is augmented with guards deciding a single step or an atomic section based on the values of the auxiliary variables. For our example, we use four auxiliary variables $(c_{11}, c_{21}, c_{31}, c_{32})$. We obtain an extended tuple of variables $w = (v, c_{11}, c_{21}, c_{31}, c_{32})$. A constraint $c_{ij} = \ell$ is used in thread i to decide that the control flows from location ℓ_j to location ℓ. Correspondingly, the transition relation of the first thread is augmented to:

$$
\begin{aligned}
&next_1(w, w') = \\
&(pc_1 = \ell_1 \wedge c_{11} = \ell_2 \wedge pc_1' = \ell_2 \wedge x' = x + z \wedge skip(\dots) \vee \\
&\ pc_1 = \ell_1 \wedge c_{11} = \ell_3 \wedge pc_1' = \ell_3 \wedge x' = x + 2 * z \wedge skip(\dots) \vee \\
&\ pc_1 = \ell_2 \wedge pc_1' = \ell_3 \wedge x' = x + z \wedge skip(\dots))
\end{aligned}
$$

The transition relations of the second and third thread are instrumented similarly. We instantiate the safety game proof rule such that the system role is played by the instrumented program transition relation and the environment role is to provide inputs to the program (similar to the previous case, this example program does not expect any inputs).

$$
\begin{aligned}
init(w) &= init(v) \\
eve(w, w') &= (next_1(w, w') \vee next_2(w, w') \vee next_3(w, w')) \\
adam(w, w') &= skip(w, w')
\end{aligned}
$$

Furthermore, we represent the search of initial parameter values using a strengthening of the original initial condition with an assertion $mid(w, t)$ such that:

$$
\begin{aligned}
&init(w) \rightarrow \exists t : mid(w, t) \\
&(mid(w, t), eve(w, w'), adam(w, w')) \models G\ \neg error(w)
\end{aligned}
$$

We use ConSynth and provide the following template for the existential clause involving the initial states.

$$
\begin{aligned}
\text{TEMPL}(sk)(w, t) = (&c_{11} = ?_{11} \wedge c_{21} = ?_{21} \\
&\wedge c_{31} = ?_{31} \wedge c_{32} = ?_{32}).
\end{aligned}
$$

Name	Game	Player p	Objective for player p	Result	Time (z3)	Time (Barcelogic)
G1	Cinderella ($c = 3$)	Cinderella	$G\,\neg overflow$	✓	3.2s	**1.2s**
G2	Cinderella ($c = 2$)	Cinderella	$G\,\neg overflow$	✓	**1m52s**	**1m52s**
G3	Cinderella ($c = 1.4$)	Stepmother	$F\ overflow$	✓	**18s**	1m14s
G4	Cinderella ($c = 1.4$)	Cinderella	$win(0)$	✓	**7m16s**	SysError
G5	Cinderella ($c = 1.4$)	Cinderella	$win(0) \vee win(2)$	✓	**4.7s**	**4.7s**
G6	Repair-Lock	Program	$G\,\neg error$	✓	**0.3s**	**0.3s**
G7	Repair-Critical	Program	$G\,\neg error$	✓	17.7s	**16.9s**
G8	Repair-Critical	Program	$G\,(at\_p \rightarrow F\,\neg at\_p)$	✓	**53.3s**	3m6s
G9	Synth-Synchronization	Program	$G\,\neg error$	✓	T/O	**1s**

Table 1. Statistics for case studies. A T/O-mark stands for time out after 15 minutes.

A solution to $mid(w, t)$ sets the auxiliary variables to target program locations so that the objective of the game is satisfied, i.e., the error states are not reachable. ConSynth returns the following witness for the existential quantifier clause:

$$sk(w, t) = (c_{11} = 3 \wedge c_{21} = 3 \wedge c_{31} = 4 \wedge c_{32} = 3).$$

We note that our proof rule does not represent an optimization problem. The solutions we obtain correspond to a synthesized program that is not necessarily the most efficient one, i.e., the longest atomic sections may be picked instead of smaller steps. Dealing with optimality is a subject for future work.

7. Experimental results

In this section we describe how we used ConSynth as a proof-of-concept implementation of our proposed approach to solving infinite-state games. ConSynth is implemented in SICStus Prolog and is a solver for Horn clauses over linear inequalities. The implementation uses two SMT solvers for handling non-linear constraints: the Z3 solver [13] and the Barcelogic solver [5, 6].

For our experiments we used a computer with an Intel Core 2 Duo 2.53 GHz CPU and 4 GB of RAM. Table 1 shows the results corresponding to the case studies described in the paper: Cinderella-Stepmother games with safety objectives (G1 and G2), with reachability objectives (G3) and with more general LTL objectives (G4 and G5). Lastly, we show results on the program repair/synthesis games (G6, G7, G8 and G9).

For each game we report the player and the objective for which we synthesize strategies (see Columns 3 and 4). Column 5 shows the result obtained from our tool: an ✓-mark stands for a strategy successfully synthesized by our tool using either Z3 or Barcelogic as solving back-ends. In one case (G5), due to the general LTL objective we obtain a large Büchi automaton. Our normal encoding times-out for both Z3 and Barcelogic. However, ConSynth can synthesize a winning strategy quickly if we exploit an optimization where we treat infinite datatypes symbolically using a decision procedure, and finite domain datatypes explicitly without abstraction. Because the control locations of the Büchi automaton $pc_{\mathcal{B}}$ range over a finite domain, this optimization allows the tool to track the states of $pc_{\mathcal{B}}$ explicitly, and this simplifies the proof obligations.

We believe our approach will benefit from future improvements in constraint solving. The cases when either Z3 or Barcelogic times out are challenging SMT problems and of potential interest to the SMT-solving community.

8. Related work

There is a rich literature on decision procedures for graph games with application to formal methods [14, 32, 38, 43]. In particular, many techniques, both explicit-state [43] and symbolic [25, 37], are known for games on finite graphs. Decidability results are also known for games on certain restricted classes of infinite graphs, such as pushdown graphs [8, 46] and prefix-recognizable graphs [9].

Known approaches to games on graphs that represent state spaces of general infinite-state programs can be divided into two categories: those based on symbolic execution and those based on abstraction-refinement. De Alfaro et al [12] offer an example of the first kind of approach. In this work, a symbolic semi-algorithm is used to explore the state space of the game directly. In contrast, we reduce the problem of solving a game to Horn constraint solving, leaving the constraints to be solved by a dedicated solver relying on CEGAR and interpolation.

The second category of methods [2, 15, 16, 22–24] lift predicate abstraction and CEGAR, originally proposed for safety verification, to games. The core idea here is the use of abstract transition systems where overapproximate ("may") and underapproximate ("must") transitions are permitted, and which are model-checked against properties with 3-valued semantics. In contrast to existing approaches of this sort, we do not directly construct a program abstraction with must-transitions or 3-valued semantics. Instead, we use a Skolem relation that is iteratively refined. Moreover, our backend solver uses disjunctively well-founded transition invariants [39] to resolve liveness goals for players, which (so far as we are aware) existing approaches do not do. This algorithmic difference has a significant impact in practice.

Our approach is perhaps more closely related to a recent paper by Cook and Koskinen [11], which uses a combination of CEGAR with a form of Skolemization for verification of branching-time properties of programs. However, this method only studies verification of CTL — the class of properties that we handle is significantly larger (e.g., it includes the full μ-calculus).

Games have a particularly close connection to program synthesis and repair, areas that have seen a flurry of activity in the last few years. However, in recent as well as classical algorithms in these areas, the focus is tends to be either on finite-state systems [29, 38, 41], or on functional, rather than reactive, programs [31, 42, 45]. In contrast, the natural application of our approach is in the repair and synthesis of infinite-state reactive programs.

Finally, our work here was inspired by the rich tradition of on deductive program synthesis [34, 38, 40]. The main difference between this line of work and ours lies in our focus on automation. For example, Slanina [40] also offers a deductive rule for games with response objectives. However, the proof rule demands global ranking functions and justice and compassion assumptions, which are known to be difficult to discharge automatically.

9. Conclusion

We have presented a constraint based approach to computing winning strategies in infinite-state games. The approach consists of: (1) a set of sound and relatively complete proof rules for solving such games, and (2) automation of the rules on top of an existing auto-

mated deduction engine. We demonstrate the practical promise of our approach through several case studies using examples derived from prior work on program repair and synthesis.

Many avenues for future work remain open. The system we have presented is a prototype. Much more remains to be done on engineering it for greater scalability. In particular, we are especially interested in applying the system to reactive synthesis questions arising out of embedded systems and robotics. On the theoretical end, exploring opportunities of synergy between our approach and abstraction-based [22, 24] and automata-theoretic approaches to games [43] remains a fascinating open question.

Acknowledgements

This research was supported in part by the ERC project 308125, by the DFG Graduiertenkolleg 1480 (PUMA), and by NSF Awards #1162076 and #1156059.

References

[1] R. Alur, P. Cerný, P. Madhusudan, and W. Nam. Synthesis of interface specifications for java classes. In *POPL*, pages 98–109, 2005.

[2] T. Ball and O. Kupferman. An abstraction-refinement framework for multi-agent systems. In *LICS*, pages 379–388. IEEE, 2006.

[3] T. Beyene, C. Popeea, and A. Rybalchenko. Solving existentially quantified Horn clauses. In *CAV*, 2013.

[4] M. Bodlaender, C. Hurkens, V. Kusters, F. Staals, G. Woeginger, and H. Zantema. Cinderella versus the Wicked Stepmother. In *IFIP TCS*, pages 57–71, 2012.

[5] M. Bofill, R. Nieuwenhuis, A. Oliveras, E. Rodríguez-Carbonell, and A. Rubio. The Barcelogic SMT solver. In *CAV*, pages 294–298, 2008.

[6] C. Borralleras, S. Lucas, A. Oliveras, E. Rodríguez-Carbonell, and A. Rubio. SAT modulo linear arithmetic for solving polynomial constraints. *J. Autom. Reasoning*, 48(1):107–131, 2012.

[7] J. R. Büchi and L. Landweber. Solving sequential conditions by finite-state strategies. *Trans. Amer. Math. Soc.*, 138:295–311, 1969.

[8] T. Cachat. Symbolic strategy synthesis for games on pushdown graphs. In *ICALP*, pages 704–715. 2002.

[9] T. Cachat. Uniform solution of parity games on prefix-recognizable graphs. *Electronic Notes in Theoretical Computer Science*, 68(6):71–84, 2003.

[10] K. Chatterjee and L. Doyen. Energy parity games. *TCS*, 2012.

[11] B. Cook and E. Koskinen. Reasoning about nondeterminism in programs. In *PLDI*, 2013.

[12] L. De Alfaro, T. Henzinger, and R. Majumdar. Symbolic algorithms for infinite-state games. In *CONCUR*, pages 536–550. Springer, 2001.

[13] L. M. de Moura and N. Bjørner. Z3: An efficient SMT solver. In *TACAS*, 2008.

[14] E. A. Emerson and C. Jutla. Tree automata, mu-calculus and determinacy. In *FOCS*, pages 368–377. IEEE, 1991.

[15] H. Fecher and M. Huth. Ranked predicate abstraction for branching time: Complete, incremental, and precise. In *ATVA*, pages 322–336. Springer, 2006.

[16] H. Fecher and S. Shoham. Local abstraction–refinement for the μ-calculus. *STTT*, 13(4):289–306, 2011.

[17] P. Gastin and D. Oddoux. Fast LTL to Büchi automata translation. In *CAV*, pages 53–65, 2001.

[18] E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research*, 2002.

[19] S. Grebenshchikov, A. Gupta, N. P. Lopes, C. Popeea, and A. Rybalchenko. HSF(C): A software verifier based on Horn clauses - (competition contribution). In *TACAS*, 2012.

[20] S. Grebenshchikov, N. P. Lopes, C. Popeea, and A. Rybalchenko. Synthesizing software verifiers from proof rules. In *PLDI*, 2012.

[21] A. Griesmayer, R. Bloem, and B. Cook. Repair of boolean programs with an application to C. In *CAV*, pages 358–371. Springer, 2006.

[22] O. Grumberg, M. Lange, M. Leucker, and S. Shoham. Dont know in the μ-calculus. In *VMCAI*, pages 233–249, 2005.

[23] O. Grumberg, M. Lange, M. Leucker, and S. Shoham. When not losing is better than winning: Abstraction and refinement for the full μ-calculus. *Information and Computation*, 205(8):1130–1148, 2007.

[24] A. Gurfinkel and M. Chechik. Why waste a perfectly good abstraction? In *TACAS*, pages 212–226. 2006.

[25] A. Harding, M. Ryan, and P.-Y. Schobbens. A new algorithm for strategy synthesis in LTL games. In *TACASs*, pages 477–492. Springer, 2005.

[26] T. A. Henzinger, R. Jhala, and R. Majumdar. Counterexample-guided control. In *ICALP*, pages 886–902, 2003.

[27] K. Hoder, N. Bjørner, and L. M. de Moura. *Z*- an efficient engine for fixed points with constraints. In *CAV*, pages 457–462, 2011.

[28] A. J. C. Hurkens, C. A. J. Hurkens, and G. J. Woeginger. How Cinderella won the bucket game (and lived happily ever after). *Mathematics Magazine*, 84(4):pp. 278–283, 2011.

[29] B. Jobstmann, A. Griesmayer, and R. Bloem. Program repair as a game. In *CAV*, pages 226–238, 2005.

[30] M. Jurdziński. Small progress measures for solving parity games. In *STACS*, pages 290–301, 2000.

[31] V. Kuncak, M. Mayer, R. Piskac, and P. Suter. Complete functional synthesis. In *PLDI*, 2010.

[32] O. Kupferman and M. Y. Vardi. Robust satisfaction. In *CONCUR*, pages 383–398, 1999.

[33] Z. Manna and A. Pnueli. Completing the temporal picture. *Theor. Comput. Sci.*, 83(1):91–130, 1991.

[34] Z. Manna and R. Waldinger. A deductive approach to program synthesis. *TOPLAS*, 2(1):90–121, 1980.

[35] D. Martin. Borel determinacy. *The Annals of Mathematics*, 102(2): 363–371, 1975.

[36] E.-R. Olderog and K. R. Apt. Fairness in parallel programs: The transformational approach. *TOPLAS*, 10(3), 1988.

[37] N. Piterman, A. Pnueli, and Y. Saar. Synthesis of reactive(1) designs. In *VMCAI*, pages 364–380, 2006.

[38] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *POPL*, pages 179–190. ACM, 1989.

[39] A. Podelski and A. Rybalchenko. Transition invariants. In *LICS*, 2004.

[40] M. Slanina. Control rules for reactive system games. In *AAAI Spring Symposium on Logic-Based Program Synthesis*, 2002.

[41] A. Solar-Lezama, L. Tancau, R. Bodík, S. A. Seshia, and V. A. Saraswat. Combinatorial sketching for finite programs. In *ASPLOS*, pages 404–415, 2006.

[42] S. Srivastava, S. Gulwani, and J. S. Foster. From program verification to program synthesis. In *POPL*, pages 313–326, 2010.

[43] W. Thomas. On the synthesis of strategies in infinite games. In *STACS*, pages 1–13, 1995.

[44] M. Y. Vardi. Verification of concurrent programs: The automata-theoretic framework. *Ann. Pure Appl. Logic*, 51(1-2):79–98, 1991.

[45] M. T. Vechev, E. Yahav, and G. Yorsh. Abstraction-guided synthesis of synchronization. In *POPL*, 2010.

[46] I. Walukiewicz. Pushdown processes: Games and model-checking. *Information and computation*, 164(2):234–263, 2001.

[47] W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200 (1):135–183, 1998.