Reachability Games Modulo Theories with a Bounded Safety Player*

Marco Faella[1], **Gennaro Parlato**[2]

[1] University of Naples Federico II, Naples, Italy
[2] University of Molise, Pesche, Italy
m.faella@unina.it, gennaro.parlato@unimol.it

Abstract

Solving reachability games is a fundamental problem for the analysis, verification, and synthesis of reactive systems. We consider *logical* reachability games modulo theories (in short, GMTs), i.e., infinite-state games whose rules are defined by logical formulas over a multi-sorted first-order theory. Our games have an asymmetric constraint: the safety player has at most k possible moves from each game configuration, whereas the reachability player has no such limitation. Even though determining the winner of such a GMT is undecidable, it can be reduced to the well-studied problem of checking the satisfiability of a system of *constrained Horn clauses* (CHCs), for which many off-the-shelf solvers have been developed. Winning strategies for GMTs can also be computed by resorting to suitable CHC queries. We demonstrate that GMTs can model various relevant real-world games, and that our approach can effectively solve several problems from different domains, using Z3 as the backend CHC solver.

1 Introduction

Solving reachability games is a fundamental problem for the analysis, verification, and synthesis of reactive systems, as they are key tools for modeling such systems. Reachability games are turn-based games played between two players, over a finite or infinite directed graph. The nodes of the graph represent the configurations of the system it models, while the edges of the graph represent transitions. REACH, or the *reachability player*, aims to hit a *target* configuration. In contrast, SAFE, or the *safety player*, has the opposite objective, i.e., to prevent REACH from reaching a target configuration, thus keeping the system within a set of safe configurations regardless of the choices of REACH.

In *logical* reachability games, the sets of initial and target configurations, and the players' moves, are defined using logical formulas. If a system has an enormous state space, if not an infinite one, symbolic representations allow for not only a finite but often also a compact representation of the system. Logical formulas, for example, can compactly define the set of transitions of programs, or the transition functions of robots moving on an infinite grid. For this reason, computing the winning strategy of logic games is essential for solving relevant real-world problems, such as those related to software verification and synthesis, robotics, etc. Our paper assumes that the components of the game are multi-sorted first-order formulas. Specifically, we consider quantifier-free formulas that are supported by modern SMT solvers.

The problem of finding the winner of a logical game is undecidable in general. Rice's theorem states that all non-trivial semantic properties of programs are undecidable. Since we can easily model a program with logical formulas and use them as the transition function for one of the players, it is easy to see that even the problem of determining the winner of a logical game is undecidable. It follows that it is not possible to have an algorithmic solution that terminates for all games, and any sound technique, like the one we propose, will fail to terminate on some instances. Among the approaches proposed in the literature, three are closely related to ours, as they also target logical reachability games. Beyene et al. (2014) reduce such games to a system of *constrained Horn clauses (CHCs)* extended with existential quantifiers, with the aid of a user-provided strategy template. Farzan and Kincaid (2018) focus on games whose rules can be expressed in linear arithmetic and present an algorithm that considers increasingly longer bounded-horizon versions of the game, until either the reachability player wins or the winning strategy of the safety player can be generalized to the original infinite-horizon game. Finally, Craig interpolation is used by Baier et al. (2021) to propose a method using subgoals, which are slices of the game a reachability player must traverse to reach the target.

Our study introduces a special form of logical reachability games, which allows us to model various relevant real-world problems, in which the safety player has a *bounded number of moves* from all game configurations, whereas the reachability player has no such limitation. For example, consider the *Cinderella-Stepmother* game, that we use as a running example throughout the paper. Here, the two players share

*This work was partially supported by INDAM-GNCS 2022, AWS 2021 Amazon Research Awards, the MUR project Future Artificial Intelligence Research (FAIR), and the MUR project 'Innovation, digitalisation and sustainability for the diffused economy in Central Italy', Spoke 1 MEGHALITIC, VITALITY Ecosystem.

n buckets with a fixed maximum capacity. The buckets are arranged in a circle, and are initially empty. The game is played in a discrete sequence of turns: when it is Cinderella's turn, she chooses two adjacent buckets and empties them; when it is the Stepmother's turn, she pours a total of 1 unit of water into any subset of buckets. If any of the buckets overflows, Stepmother wins. Otherwise, the game continues forever and Cinderella wins. This game can be described as a logical reachability game with Stepmother acting as the reachability player, and Cinderella as the safety player. The game fits in our framework as Cinderella only has n possible moves from every configuration, i.e., one per bucket. The Cinderella-Stepmother game is a real-world problem relevant to wireless sensor networks despite its simple formulation (Bodlaender et al. 2012; Beyene et al. 2014). Henceforth we refer to the class of logical reachability games we study as *reachability games modulo theories with a bounded safe player* (GMTs for short). Later in the paper, we show that many real-world problems can be cast as GMTs.

The crucial property of GMTs is that a winning strategy for player REACH induces a finite tree of game configurations. This property allows us to reduce the problem of determining the winner of a GMT to solving a system of CHCs. Efficient algorithmic solutions and tools have been developed for CHCs, often leveraging or generalizing techniques developed in the context of automatic program verification (Grebenshchikov et al. 2012; Gurfinkel and Bjørner 2019; Bjørner et al. 2015). As a result, CHCs are often used as an intermediate representation in a variety of verification and synthesis tools. Here, we follow a similar approach to solve the problem of determining the winner in GMTs, obtaining several advantages over the state-of-the-art. First, our reduction to CHCs is simple and direct, unloading the main burden of solving the game to the CHC solver, which we treat as a black box. This allows us to fully exploit present and future improvements in CHC solvers, whose performance keeps improving year-over-year, as witnessed by the *competition on constrained Horn clauses* CHC-COMP (Fedyukovich and Rümmer 2021). Second, the very simplicity of our approach provides significant time savings in those benchmarks that fall within the GMT framework. For example, we can solve the hardest cases of the above-mentioned Cinderella-Stepmother game more than an order of magnitude faster than the state-of-the-art.

Organization of the paper. Sec. 2 introduces the notation and definitions concerning CHCs. In Sec. 3, we define GMTs and establish some fundamental properties. In Sec. 4, we give an effective linear-time reduction from the problem of determining the winner of our games to the satisfiability problem of a CHC system, and describe some experiments on various standard benchmarks. In Sec. 5, we present our strategy synthesis algorithms. Concluding remarks can be found in Sec. 6.

2 Constrained Horn Clauses

In this section, we introduce the notation and definitions we will use in the paper on Constrained Horn Clauses.

We use $\mathbb{N}$ to denote the set of all natural numbers, $\mathbb{Z}$ to stand for the set of integers, and $\mathbb{B}$ to represent the set $\{0, 1\}$. For $n \in \mathbb{N}$, we write $[n]$ to denote the interval $\{1, \ldots, n\}$.

We use the standard syntax and semantics for first-order logic (FOL) with equality (Manna and Zarba 2002). We deal with formulas of a many-sorted first-order theory $\mathcal{D}$ with sorts $data_1, \ldots, data_n$. For example, each of these theories can be the theory of *arithmetic*, *reals*, *arrays*, etc. From now on, we refer to $\mathcal{D}$ as the *data theory* of our games.

Definition 1. *We fix a set R of uninterpreted fixed-arity relation symbols, which represent the unknowns in the system. A* **Constrained Horn Clause***, or CHC for short, is a formula of the form $H \leftarrow C \wedge B_1 \wedge \cdots \wedge B_n$ where:*

- *C is a formula of the data theory $\mathcal{D}$ that does not contain any relation symbol from R;*
- *for every $i \in [n]$, B_i is an application $p(v_1, \ldots, v_k)$ of a relation symbol $p \in R$ to first-order variables $v_1, \ldots, v_k$;*
- *H is the clause* head *and is either $false$, or it is an application $p(v_1, \ldots, v_k)$ of a relation symbol $p \in R$ to the first-order variables $v_1, \ldots, v_k$.*

A CHC is a fact *if its body has only the C component, and it is a* query *if its head is $false$.* □

A finite set $\mathcal{H}$ of CHCs is a *system*, and it corresponds to the first-order formula obtained by putting all its CHCs in conjunction. We assume that the semantics of constraints is given a priori as a structure. A system $\mathcal{H}$ with relation symbols R is *satisfiable* if there exists an interpretation for every relation symbol in R that makes all clauses in $\mathcal{H}$ valid.

Unsatisfiability of a system of CHCs, or a derivation of $false$, corresponds to a *counterexample* that takes the form of a path or tree representing a resolution derivation. We formalize the notion of counterexamples as follows. For $k \in \mathbb{N}$, a *k-ary tree* T, or simply a *tree*, is a finite and prefix-closed subset of $[k]^*$. We call *node* an element of T, and we refer to the node identified by the empty word ϵ as the *root* of T. The *edge relation* is implicit: for $i \in [k]$, if t and $t.i$ are both nodes of T, then $(t, t.i)$ is an *edge* of T. Further, we say that $t.i$ is the i-th *child* of t, and t is the *parent* of $t.i$. A *leaf* is a node without children, while an *internal node* is a node that is not a leaf. The *height* of T is $\max_{t \in T} |t|$.

Definition 2. *Let $\mathcal{H}$ be a system of CHCs on the set of unknowns R. A* **counterexample** *of $\mathcal{H}$ is a triple (T, chc, val) with the following components:*

- *T is a k-ary tree, for some $k \in \mathbb{N}$;*
- *$chc : T \to \mathcal{H}$ is such that $chc(\varepsilon)$ is a query and $chc(t)$ is a fact for all leaves t;*
- *val labels each node $t \in T$ with an evaluation of the variables occurring in $chc(t)$.*

Moreover, for all $t \in T$, let

$$chc(t) \;=\; H \leftarrow C \wedge B_1 \wedge \cdots \wedge B_n,$$

the following conditions hold:

- *C evaluates to true under the variable assignment $val(t)$;*
- *t has n children in T;*
- *for all children $t.i$ of t, let $p(u_1, \ldots, u_k)$ be the head of $chc(t.i)$, then $B_i = p(v_1, \ldots, v_k)$ and $val(t)(v_j) = val(t.i)(u_j)$ for all $j \in [k]$.* □

Given a counterexample (T, chc, val), for all (non-root) nodes $t \in T$, we denote by $val^H(t)$ the restriction of $val(t)$ to the variables occurring in the head of $chc(t)$.

When $\mathcal{H}$ admits a counterexample (a *derivation* of $false$) the clauses conflict, which means the system cannot be satisfied. On the other hand, the completeness of the first-order resolution implies that the reverse is also true. That is, the absence of a derivation of false implies that $\mathcal{H}$ is satisfied.

Theorem 1. *A system $\mathcal{H}$ of CHCs is unsatisfiable if and only if there exists a counterexample of $\mathcal{H}$.*

In this paper, we solve GMTs by reducing them to the satisfiability of CHCs. Hence, the algorithms that compute winning strategies use a CHC solver as a subprocedure. To this aim, we consider the following API, inspired by the standard language SMT-LIB (Barrett, Fontaine, and Tinelli 2017), which is supported by several state-of-the-art solvers (de Moura and Bjørner 2008). These functions accept a CHC system $\mathcal{H}$ as input, but provide different outputs:

check-sat**:** It verifies the satisfiability of $\mathcal{H}$. The output is one of SAT/UNSAT/UNKNOWN.

get-model**:** Assuming that $\mathcal{H}$ is satisfiable, it returns a satisfying assignment, i.e., an interpretation for the uninterpreted symbols in $\mathcal{H}$ that renders all CHCs in $\mathcal{H}$ true.

get-proof**:** Assuming that $\mathcal{H}$ is unsatisfiable, it returns a counterexample of $\mathcal{H}$.

Representing models and counterexamples. By definition, a counterexample is a finite tree labeled with variable valuations, so we can safely assume that if the solver is able to prove that the system is unsatisfiable, it can present an explicit representation of a counterexample.[1]

On the other hand, a satisfying assignment includes an interpretation for the relations in R, which may not be represented explicitly if some of the data domains are infinite. Hence, we should expect the solver to present a *symbolic* representation of such models, using some logic language (possibly but not necessarily the same logic language in which the game data constraints were presented). In the following (particularly in Sec. 5), we assume that the language used to encode models is *membership-decidable*, i.e., it is decidable whether a given variable valuation belongs to the interpretation of a given relation symbol from R.

3 Games Modulo Theories

In this section, we define *reachability games modulo theories with a bounded* SAFE *player* (GMTs). There are a finite number of variables in a GMT, and each variable can have a finite or infinite domain. An evaluation of all variables represents a game configuration. The game starts from an initial configuration and players move in turns by alternating a move of player REACH with a move of player SAFE. The game specifies which player starts the game. We use formulas on a combination of theories to define each player's moves, and the target configurations for player REACH. Unlike REACH, which has an unrestricted transition function, we limit with a constant the number of moves that SAFE can make from any given game configuration.

To define the configurations of our games, we consider a finite and ordered set of first-order variables $V = \{v_1, \ldots, v_h\}$. We assume that the sort $data_v$ of each variable $v \in V$ coincides with one of the domains of the game data theory $\mathcal{D}$. We denote by $Val(V)$ the set of all total functions that map each variable $v \in V$ into a value of sort $data_v$. For our purposes, $Val(V)$ represents the set of configurations of any game having V as its set of variables. Given a valuation $val \in Val(V)$ and a formula φ of $\mathcal{D}$ with free variables in V, we write $\varphi(val) = \varphi[val(v_1)/v_1, \ldots, val(v_h)/v_h]$ to denote the Boolean value resulting from the evaluation of φ when each variable $v \in V$ is replaced by $val(v)$. We also define a new set of first-order variables $V' = \{v' \mid v \in V\}$ associated to V, called the set of *primed variables* of V, where the sorts of variable v and v' coincide, i.e., $data_v = data_{v'}$, for every $v \in V$. Furthermore, we define a map $prime : Val(V) \to Val(V')$ that takes a valuation on V and converts it into a valuation on V' in a natural way, i.e., if $val \in Val(V)$ then $prime(val) = val'$ where $val'(v') = val(v)$, for every $v \in V$. Finally, let V_1 and V_2 be two disjoint sets of variables, and let $val_i \in Val(V_i)$, for $i \in \{1, 2\}$. We denote by $(val_1 \cup val_2)$ the function with domain $(V_1 \cup V_2)$ that combines val_1 and val_2 in the natural way: if $v \in V_i$ with $i \in \{1, 2\}$, then $(val_1 \cup val_2)(v) = val_i(v)$.

Definition 3 (GAMES MODULO THEORIES)**.** *Let $\mathcal{D}$ be a first-order theory with sorts $data_1, \ldots, data_n$. A* reachability game $\mathcal{G}$ modulo $\mathcal{D}$ with a bounded SAFE player (GMT) *is a tuple $(V, c_0, p_0, target, T_{\text{REACH}}, \{T^i_{\text{SAFE}}\}_{i \in [k]})$ where:*

- *V is a finite set of first-order variables where $data_v$ is a sort of $\mathcal{D}$, for each $v \in V$. We call V the* set of game variables. *A* configuration *of $\mathcal{G}$ is an element in $\mathcal{C} = Val(V)$.*
- *$c_0 \in \mathcal{C}$ is the initial configuration, and $p_0 \in \{\text{REACH}, \text{SAFE}\}$ is the* initial player.
- *$target$ is a quantifier-free formula of $\mathcal{D}$ whose free variables are contained in V. A configuration c is* target *if $target(c)$ holds true.*
- *T_{REACH} and $T^1_{\text{SAFE}}, \ldots, T^k_{\text{SAFE}}$ are each a $\mathcal{D}$ formula with free variables in $(V \cup V')$, which collectively define the moves of the game. Furthermore, for each configuration c, the following holds:*
  - *there is at least a configuration c' such that $T_{\text{REACH}}(c, prime(c')) = true$ (i.e., T_{REACH} is total);*
  - *for every $i \in [k]$, there is a unique configuration c' such that $T^i_{\text{SAFE}}(c, prime(c')) = true$ (i.e., T^i_{SAFE} is total and functional).* □

Example 1 (CINDERELLA-STEPMOTHER GAME AS A GMT)**.** *Given values for n (number of buckets) and cap (bucket capacity), we define V as the set of real-valued variables $\{b_1, \ldots, b_n\}$ where b_i represents the content of the bucket i, for $i \in [n]$. Since all buckets are initially empty, the initial configuration c_0 assigns zero to all variables b_i. Stepmother is player* REACH*, because her objective is to reach an overflow configuration, and consequently Cinderella is player* SAFE*. The game is started by Stepmother,*

[1] If (quantifier-free) data theory is decidable, we can obtain a counterexample by logically describing a counterexample of height h and then checking the satisfiability for increasing values of h.

so $p_0 = \textsc{Reach}$. *We define the formula* $target$ *to capture an overflow in at least one bucket:*

$$target(b_1, \ldots, b_n) \stackrel{\text{def}}{=} \bigvee_{i=1}^{n} \left(b_i > cap\right).$$

Then, we encode Stepmother's ability to pour 1 *unit of water into any subset of buckets.* $T_{\textsc{Reach}}(b_1, \ldots, b_n, b'_1, \ldots, b'_n)$ *holds iff:*

$$\left(\sum_{i=1}^{n} b'_i = 1 + \sum_{i=1}^{n} b_i \right) \wedge \left(\bigwedge_{i=1}^{n} \left(b'_i \geq b_i\right) \right).$$

Note that the number of possible successors following a move of Stepmother is infinite since any partition of the water unit is allowed.

Cinderella's possible moves as the SAFE *player are as follows. For all* $i \in [n]$, *she can empty bucket* i *and the next one. So,* $T^i_{\textsc{Safe}}(b_1, \ldots, b_n, b'_1, \ldots, b'_n)$ *holds iff:*

$$\left(\bigwedge_{j \in \{i, succ(i)\}} \left(b'_j = 0\right) \right) \wedge \left(\bigwedge_{j \in [n] \setminus \{i, succ(i)\}} \left(b'_j = b_j\right) \right),$$

where $succ(i)$ *is* $i + 1$ *if* $i < n$, *and* 1 *otherwise.* $T^i_{\textsc{Safe}}$ *thus defined is total and functional, so the whole game is a* GMT. □

Next, we define the semantics of a GMT in terms of moves, plays, and strategies. In the remainder of the paper, unless otherwise specified, we refer to a fixed GMT with components $(V, c_0, p_0, target, T_{\textsc{Reach}}, \{T^i_{\textsc{Safe}}\}_{i \in [k]})$.

Let $c, c' \in \mathcal{C}$ be two configurations. There is a **move** of player REACH from c to c', denoted by $c \rightarrow_{\textsc{Reach}} c'$, if $T_{\textsc{Reach}}(c, prime(c'))$ holds true. Instead, there is a move of player SAFE from c to c', denoted by $c \rightarrow_{\textsc{Safe}} c'$, if there is an index $i \in [k]$ such that $T^i_{\textsc{Safe}}(c, prime(c'))$ holds true.

A **play** is a possibly infinite sequence of configurations $\pi = c_0 c_1 \ldots$ with the following properties. Let $len(\pi) \in \mathbb{N} \cup \{\infty\}$ be the length of π, and $last(\pi)$ be its last configuration (if any). Moreover, let $p_i \in \{\textsc{Reach}, \textsc{Safe}\}$ be the player whose turn it is to move at step i: p_0 is one of the game parameters, and the two players strictly alternate after that. Then, the following properties hold:

- the sequence conforms to the moves of the two players: for all $i \in \mathbb{N}$, if $len(\pi) > i$, then $c_i \rightarrow_{p_i} c_{i+1}$;
- all configurations c along π except the last one (if it exists), are such that $target(c) = false$, whereas the last configuration $last(\pi)$ (if it exists) is such that $target(last(\pi)) = true$.

We say that REACH wins all plays of finite length and loses all those of infinite length. The opposite applies to SAFE.

We now describe the concept of **strategy**, which intuitively determines the next move to be made based on the sequence of configurations encountered so far. A *history* is any finite prefix of a play. Denoting by Π the set of all histories, a *strategy* for player p is a function $f : \Pi \rightarrow \mathcal{C}$ that maps every history π to a configuration $f(\pi)$ such that $last(\pi) \rightarrow_p f(\pi)$. A play $\pi = c_0 c_1 \ldots$ *conforms* to a strategy f of p, if for any proper prefix π_i of π of the form $c_0 c_1 \ldots c_i$, if $p_i = p$ then $c_{i+1} = f(\pi_i)$. A strategy f is **memoryless** if its value depends only on the last configuration of the history, i.e., for all histories π and π', if $last(\pi) = last(\pi')$ then $f(\pi) = f(\pi')$. Clearly, a memoryless strategy can be represented as a partial function $f : \mathcal{C} \rightharpoonup \mathcal{C}$.

A **winning strategy** for a player is a strategy that guarantees victory for the player regardless of the moves made by the other player. Formally, a strategy f for player p is *winning* if p wins all plays that conform to f. We also say that player p wins the game $\mathcal{G}$ if there exists a winning strategy for p. A winning strategy f for player REACH is always associated with a **strategy tree**, which is a finite tree labeled with the configurations obtained by playing the game according to f, and which includes all possible moves of player SAFE: the root is labeled with c_0, all internal nodes where it is REACH's turn to play have a single child labeled with the configuration chosen by f, and all internal nodes where it is SAFE's turn to play have k children corresponding to the k possible moves of player SAFE.

We say that a family of games is **determined** when exactly one of the players has a winning strategy in each game. The classic Borel determinacy result by Martin (Martin 1975) implies that GMTs are determined. Moreover, it is well known that in reachability games both players have memoryless winning strategies (Mazala 2002).

Theorem 2. GMT*s are determined. Moreover, the winner has memoryless winning strategy.*

We will now demonstrate that while GMTs are determined, establishing which player wins the game is an **undecidable problem** as soon as we consider theories of linear integer (or real) arithmetic.

Theorem 3. *The problem of deciding which player wins a given* GMT *is undecidable.*

Proof sketch. Observe that we can encode the transition relation of a given two-counter machine $\mathcal{M}$ as a formula in the theory of linear integer (or real) arithmetic, say $\Delta_{\mathcal{M}}$. We reduce the halting problem of $\mathcal{M}$, which is a well-known undecidable problem (Minsky 1967), into the problem of determining the winner of a GMT $\mathcal{G}$. The game $\mathcal{G}$ uses three variables to encode the configurations of $\mathcal{M}$: two for the counters and one for the program counter. The initial configuration of $\mathcal{G}$ corresponds to the initial configuration of $\mathcal{M}$. We take $\Delta_{\mathcal{M}}$ to be the transition relation of player REACH, and the identity relation as the transition function of player SAFE. Thus, $\mathcal{M}$ halts if and only if REACH wins $\mathcal{G}$. □

4 Finding the Winner of a GMT Using CHCs

In this section, we show that the problem of determining the winner of a GMT can be reduced, in linear time, to the satisfiability problem of CHC systems.

We define the set of CHCs $\mathcal{H}(\mathcal{G})$ that, roughly speaking, uses two uninterpreted relations s and r to characterize the game configurations from which player REACH wins the game. In detail, $r(c)$ (resp., $s(c)$) is meant to be true if REACH wins when the game starts from configuration c and it is REACH (resp., SAFE) turn to play. The set of CHCs defining $\mathcal{H}(\mathcal{G})$ is shown in Figure 1. Rules (I) and (II) are

(I)	$r(c)$	$\leftarrow$	$target(c)$	
(II)	$s(c)$	$\leftarrow$	$target(c)$	
(III)	$r(c)$	$\leftarrow$	$T_{\text{REACH}}(c, c') \wedge s(c')$	
(IV)	$s(c)$	$\leftarrow$	$\bigwedge_{i\in[k]} T^i_{\text{SAFE}}(c, c'_i) \wedge \bigwedge_{i\in[k]} r(c'_i)$	
(V)	$false$	$\leftarrow$	$r(c_0)$	if $p_0 = \text{REACH}$
	$false$	$\leftarrow$	$s(c_0)$	if $p_0 = \text{SAFE}$

Figure 1: The system of CHCs $\mathcal{H}(\mathcal{G})$. Each of c, c', and c'_i is a vector of h distinct variables. Variables from different vectors are disjoint. In particular, $c = (v_1, \ldots, v_h)$.

two facts which allow us to initialize both s and r so that $s(c)$ and $r(c)$ are both set to $true$ for all target configurations c of $\mathcal{G}$. We also add a configuration c to the set defined by r whenever there exists a configuration c' such that REACH has a move from c to c' and $s(c')$ holds true (Rule (III)). Rule (IV) says that a configuration c must belong to the set defined by s if all configurations c'_i where player SAFE can transition from c belong to the set defined by r. In addition, Rule (V) states that the system will become unsatisfiable the moment an initial configuration must be included in s (resp., r) when $p_0 = \text{SAFE}$ (resp., $p_0 = \text{REACH}$).

Before stating the main results of this section, we provide two technical lemmas. Given a configuration c of $\mathcal{G}$ and a player $p \in \{\text{REACH}, \text{SAFE}\}$, let us denote by $\mathcal{G}(c, p)$ the GMT that has the same components as $\mathcal{G}$, except for the initial configuration and the initial player that are replaced by c and p, resp., i.e., $\mathcal{G}(c, p) = (V, c, p, target, T_{\text{REACH}}, \{T^i_{\text{SAFE}}\}_{i\in[k]})$. We also denote by $head(z)$ the relation symbol that appears in the head of the CHC z, provided that z is not a query.

Lemma 1. *Let $\mathcal{G}$ be a* GMT *such that $\mathcal{H}(\mathcal{G})$ is unsatisfiable, and let (T, chc, val) be a counterexample of $\mathcal{H}(\mathcal{G})$. Then, for every $t \in (T \setminus \{\epsilon\})$,* REACH *wins $\mathcal{G}(val^H(t), p)$, where $p =$* REACH *if $head(chc(t)) = r$, and $p =$* SAFE *if $head(chc(t)) = s$.*

Proof. The proof is by structural induction on T, starting from the leaves and ending in the root. Henceforth, we denote by c the game configuration $val^H(t)$.

The base case is when t is a leaf. By definition of counterexample, $chc(t)$ is either Rule (I) or (II) from Figure 1. In both cases, it follows that $target(c) = true$ and thus REACH wins $\mathcal{G}(c, p)$, with a play that requires no moves.

For the inductive step, t must be an internal node. We distinguish two cases based on the relation symbol $head(chc(t))$. If that symbol is s, from the definition of $\mathcal{H}(\mathcal{G})$ it follows that $chc(t)$ must be Rule (IV) and therefore t has exactly k children, $t.1, \ldots, t.k$, corresponding to the k possible moves of SAFE from c. Further, $head(chc(t.i)) = r$, for every $i \in [k]$. By inductive hypothesis, REACH wins $\mathcal{G}(val^H(t.i), \text{REACH})$, for all $i \in [k]$. Thus, no matter which move SAFE takes from c, REACH wins $\mathcal{G}(c, \text{REACH})$.

If instead that symbol is r, it follows that $chc(t)$ is Rule (III) and t has only $t.1$ as a child, and $head(chc(t.1)) = s$. Let $c_1 = val^H(t.1)$. By inductive hypothesis, REACH wins $\mathcal{G}(c_1, \text{SAFE})$. When playing in $\mathcal{G}(c, \text{REACH})$, REACH wins with a strategy f that starts with $f(c) = c_1$ and then plays according to the game $\mathcal{G}(c_1, \text{SAFE})$. This concludes the proof. □

Lemma 2. *If player* REACH *wins a* GMT *$\mathcal{G}$ then $\mathcal{H}(\mathcal{G})$ is unsatisfiable.*

Proof. Since REACH wins in $\mathcal{G}$, from Theorem 2, there is a memoryless winning strategy for REACH, say $f(\cdot)$, and a strategy tree $\mathcal{S} = (T, \lambda)$ derived from f where T is a k-ary tree and $\lambda : T \to \mathcal{C}$ labels each node of T with a game configuration. We define a counterexample $\mathcal{T}$ of $\mathcal{H}(\mathcal{G})$, obtained from $\mathcal{S}$, which demonstrates that $\mathcal{H}(\mathcal{G})$ is unsatisfiable.

We define $\mathcal{T} = (T', chc, val)$ as follows. T' is the result of adding a new node to T as its root, and T's root becomes its only child. As for the CHC associated to each node through chc, we set the following associations:

- the newly added node to the unique query, i.e., Rule (V);
- every internal node different from the root with an even (resp., odd) depth to Rule (III) if p_0 is SAFE (resp., REACH), otherwise to Rule (IV); and
- each leaf to either Rule (I) or (II), depending on its depth.

Finally, we obtain val by naturally expanding λ, i.e., for each node, we combine the label of the corresponding node in T with those of its children.

It is easy to see that $\mathcal{T}$ defined above, that is, by straightforwardly deriving it from a strategy tree of REACH, forms a counterexample of $\mathcal{H}(\mathcal{G})$. □

We can now state the main results of this section, as a direct consequence of Lemmas 1 and 2.

Theorem 4. *Let $\mathcal{G}$ be a* GMT*, player* REACH *wins $\mathcal{G}$ if and only if $\mathcal{H}(\mathcal{G})$ is unsatisfiable.*

When coupled with the fact that GMTs are determined (see Theorem 2), we have the following.

Corollary 1. *Let $\mathcal{G}$ be a* GMT*, player* SAFE *wins $\mathcal{G}$ if and only if $\mathcal{H}(\mathcal{G})$ is satisfiable.*

Experiments

We now report a selection of experiments comparing our approach to the state-of-the-art. Somewhat surprisingly, our direct encoding into CHCs manages to significantly outperform those custom techniques, often by orders of magnitude.

All experiments were performed on the virtual machine provided by Baier et al. (2021), running on an AMD Ryzen 2700X with 16GB of RAM. The VM was allotted 4 CPUs and 6GB of RAM. The CHC solver was Z3 v.4.8.7 64bit (de Moura and Bjørner 2008), whose CHC engine is based on SPACER (Komuravelli et al. 2013).

Example 2. *We applied the reduction described in this section to the* **Cinderella-Stepmother** *game described in Example 1, with $n = 5$ buckets and various values for the bucket capacity. The experiment consisted in designing a*

system of CHC*s in the shape of Figure 1 in the language SMT-LIB and feeding it to the SMT-solver Z3 (see (de Moura and Bjørner 2008)). As shown in Table 1, the running times are significantly lower compared to implementations of the existing approaches of Faella and Parlato (2022) (column MSO-D), Baier et al. (2021) (column CabPy), and Farzan and Kincaid (2018) (column SimSynth).*

Cap.	Winner	**GMT**	MSO-D	CabPy	SimSynth
1.0	Stepmother	**0.1**	253.4	16.8	0.3
1.5	Stepmother	**0.9**	423.0	timeout	3.0
1.8	Stepmother	**1.2**	timeout	timeout	timeout
2.0	Cinderella	**8.0**	timeout	timeout	162.0
3.0	Cinderella	**2.6**	timeout	timeout	3.6
4.0	Cinderella	2.2	timeout	timeout	**0.8**

Table 1: Results of the Cinderella-Stepmother experiments. Times in seconds, timeout in 10 minutes.

Example 3. *We consider the* **program synthesis** *example from Beyene et al. (2014); Farzan and Kincaid (2018), where the synthesized program must keep a real-valued temperature within given bounds. Due to space constraints, we refer to the cited papers for details on the model.*
This problem can be framed as a GMT *because the temperature controller is the safety player and it can only make a binary choice at each iteration. With such an encoding, Z3 could prove victory for the safety player in* 0.01 *sec, in line with the state-of-the-art (we do not report exact times as they are hardly significant at this scale).*

Example 4. **Nim** *is a board-game based on removing tokens from 3 distinct heaps. On their turn, players must remove at least one token and at most all tokens from a single heap. The player who is able to take the last token from the board wins. The game can start with any number of tokens in the three heaps. When Nim is started from a fixed configuration, it is a finite-state game and therefore it can be cast as a GMT. Table 2 reports the results, with the first column denoting the initial heap sizes. On most instances, our approach performs one or two orders of magnitude faster than competing tools.*

Heap Sizes	Winner	**GMT**	CabPy	SimSynth
(1,2,3)	REACH	**0.36**	1.6	3.1
(1,4,5)	REACH	**0.71**	4.7	210.3
(3,5,6)	REACH	**4.50**	30.9	timeout
(4,4,4)	SAFE	**3.30**	27.2	16.1
(5,5,5)	SAFE	**0.90**	80.6	92.5
(5,5,6)	SAFE	**0.93**	212.3	timeout

Table 2: Results of the Nim experiments. Times in seconds, timeout in 10 minutes.

5 Synthesis of Winning Strategies

In this section, we present algorithmic techniques for computing a winning strategy for both types of players. Since strategies may not be finitely representable, let us first clarify what we mean by computing them. We focus on memoryless strategies, thanks to Theorem 2, and distinguish two types of strategy synthesis, *plain* and *strong*, which we will discuss below. In both cases, we seek a procedure that returns the move chosen by the strategy, given a configuration that conforms to the strategy itself. The difference is whether the procedure *recognizes* such conformance (strong version), or it just assumes it as a pre-condition (plain version). We now formally define these two notions.

Assume that the game is won by player p and that $f : \mathcal{C} \rightharpoonup \mathcal{C}$ is a memoryless winning strategy for p. We say that a configuration c conforms to f if there exists a history $\pi \in \Pi$ such that: π conforms to f, the last configuration of π is c, and it is p's turn to move. In particular, c must not be a target configuration (otherwise the game is over).

Plain synthesis of f. An algorithm that, given a game configuration c that conforms to f, returns the configuration $f(c)$.

Strong synthesis of f. An algorithm that, given a game configuration c, returns the configuration $f(c)$ if c conforms to f, and UNDEF otherwise.

Next, we describe the strategy synthesis algorithms for the two players.

Strategy Synthesis for Player REACH

When player REACH wins the games, by Theorem 4 the CHC system $\mathcal{H}(\mathcal{G})$ is unsatisfiable. As discussed in Sec. 2, we can assume that the solver supports the get-proof command, which provides a counterexample (T, chc, val) of $\mathcal{H}(\mathcal{G})$. We now show how to use the counterexample to solve the strong synthesis problem of a memoryless winning strategy f for REACH. Given a configuration c, check if there is a node $t \in T$ such that $chc(t)$ is Rule (III) from Figure 1 and $val^H(t) = c$. If there is no such node, return UNDEF. Otherwise, by construction t has a single child $t.1$. Then, return the configuration corresponding to $val^H(t.1)$.

Theorem 5. *Let $\mathcal{G}$ be a* GMT*, and assume that* REACH *wins $\mathcal{G}$. Then, Algorithm 1 realizes the strong synthesis of a memoryless winning strategy for player* REACH.

Proof. By Theorem 1, let $\mathcal{T} = (T, chc, val)$ be a counterexample of $\mathcal{H}(\mathcal{G})$. We can show that we can assume w.l.o.g. that $\mathcal{T}$ corresponds to a memoryless strategy of REACH. I.e., for all nodes $t_1, t_2 \in T$ if $chc(t_i)$ is Rule (III) for $i = 1, 2$ and $val^H(t_1) = val^H(t_2)$ then $val^H(t_1.1) = val^H(t_2.1)$.

Algorithm 1: Synthesizing a winning strategy for REACH if a counterexample is available.

```
input  : A configuration c and a counterexample
         (T, chc, val) of H(G)
output: A configuration or UNDEF
for t ∈ T do
    if chc(t) = (III) and val^H(t) = c then
        return val^H(t.1)
return UNDEF
```

We extract from $\mathcal{T}$ a memoryless winning strategy f for player REACH, and prove that Algorithm 1 applied to $\mathcal{T}$ solves the strong synthesis problem for f. Assume that the initial player p_0 is REACH since the other case is analogous. We define f only for the game configurations corresponding to the internal nodes t in the counterexample, such that $chc(t)$ is Rule (III), while we leave the f undefined elsewhere. Specifically, any such node t has a single child $t.1$ and we set $f(val^H(t)) = val^H(t.1)$.

The counterexample itself proves that f is winning. First, each leaf $t' \in T$ is labeled with a target configuration $val^H(t')$. Then, starting with the initial configuration c_0, f guarantees that one of those target configurations will be reached, regardless of the moves chosen by SAFE.

Finally, it is direct to see that Algorithm 1 indeed solves the strong synthesis problem for the above strategy f. □

Strategy Synthesis for Player SAFE

When player SAFE is the winner of $\mathcal{G}$, we have that the CHC system $\mathcal{H}(\mathcal{G})$ is satisfiable. We show two ways to compute a winning strategy for player SAFE, depending on whether or not the underlying CHC solver can provide a satisfying assignment for $\mathcal{H}(\mathcal{G})$.

Synthesis With a Model. Assume that the CHC solver supports the $\mathrm{get\text{-}model}$ command, and let $\mathbb{I}$ be an interpretation of the uninterpreted relations of $\mathcal{H}(\mathcal{G})$ that makes all the CHCs of the system true. We should be aware that $\mathcal{H}(\mathcal{G})$ may have more than one model. We shall see that our approach is correct regardless of the choice of $\mathbb{I}$.

Given a configuration c, for all $i \in [k]$ we find the unique successor configuration c_i corresponding to the i-th move of player SAFE from c. This can be obtained by solving w.r.t. c' the satisfiability problem for $T^i_{\text{SAFE}}(c, c')$ in the data constraint logic. We then evaluate the interpretation $\mathbb{I}(r)$ on c_i. If $\mathbb{I}(r)(c_i)$ is *false*, we return c_i. If $\mathbb{I}(r)(c_i)$ is *true* for all i, we return UNDEF. Algorithm 2 implements the procedure we have just described.

We can prove that Algorithm 2 synthesizes a winning strategy for player SAFE, even if $\mathbb{I}$ is an over-approximation of the set of configurations where REACH wins.

Theorem 6. *Let $\mathcal{G}$ be a* GMT*, and assume that* SAFE *wins $\mathcal{G}$. Then, Algorithm 2 realizes the plain synthesis of a memoryless winning strategy for* SAFE*.*

Algorithm 2: Synthesizing a winning strategy for SAFE if a model is available.

```
input  : A configuration c and a satisfying
         assignment 𝕀 for H(G)
output: A configuration or UNDEF
for i ∈ [k] do
    c_i ← get-model(T^i_SAFE(c, c'))
    if 𝕀(r)(c_i) = false then
        return c_i
return UNDEF
```

Algorithm 3: Synthesizing a winning strategy for SAFE if no model is available.

```
input  : A configuration c
output: A configuration or UNDEF
for i ∈ [k] do
    c_i ← get-model(T^i_SAFE(c, c'))
    x ← check-sat(H( G(c_i, REACH) ))
    if x = SAT then
        return c_i
return UNDEF
```

Synthesis Without a Model. Unlike the solution presented earlier, here we show how to compute a winning strategy for SAFE using a CHC solver that tells us only whether a given CHC system is solvable or not, without the solver providing us with a solution for the system's uninterpreted relations.

Algorithm 3 illustrates the approach. The idea is similar to the one implemented in Algorithm 2, where the main difference is that instead of querying the interpretation $\mathbb{I}$, we check the satisfiability of at most k CHC systems. In particular, we repeat the following procedure for each of the k successors of configuration c obtained as a result of a move of player SAFE, i.e., all configurations c_i such that $T^i_{\text{SAFE}}(c, prime(c_i))$ holds true, for $i \in [k]$. For each $i \in [k]$, we solve the game $\mathcal{G}(c_i, \text{REACH})$, interrupting this process as soon as player SAFE wins one of these games, say $\mathcal{G}(c_j, \text{REACH})$. At that point, we return c_j. The correctness of the above procedure is stated in the following result, whose proof can be found in the supplemental material.

Theorem 7. *Let $\mathcal{G}$ be a* GMT*, and assume that* SAFE *wins $\mathcal{G}$. Then, Algorithm 3 realizes the plain synthesis of a memoryless winning strategy for* SAFE*.*

6 Conclusions

In this paper, we identified a class of logical reachability games in which the safety player is allowed only a constant number of moves from each game configuration, while the reachability player is left unrestricted. These games, while simpler than unrestricted games, can model many standard benchmarks and real-world problems. We propose a direct reduction to determine the winner of a GMT and a winning strategy using only CHC solvers, which enables us to take advantage of existing efficient technologies and future algorithms for CHCs. Furthermore, our approach empowers us to use a variety of logical theories, unlike existing solutions that are mainly limited to the theory of linear arithmetic.

In the future, it would be interesting to develop a similar approach for logical games in which the bound on the number of moves affects player REACH instead of SAFE. That would be useful to model synthesis scenarios where a discrete controller tries to reach a target region in the face of continuous disturbances (Benerecetti and Faella 2017). We also intend to apply our approach to *impartial combinatorial games* (Wu et al. 2020), a class of infinite-state games where both players have the same finite set of actions.

References

Baier, C.; Coenen, N.; Finkbeiner, B.; Funke, F.; Jantsch, S.; and Siber, J. 2021. Causality-Based Game Solving. In Silva, A.; and Leino, K. R. M., eds., *Computer Aided Verification - 33rd International Conference, CAV 2021, Virtual Event, July 20-23, 2021, Proceedings, Part I*, volume 12759 of *Lecture Notes in Computer Science*, 894–917. Springer.

Barrett, C.; Fontaine, P.; and Tinelli, C. 2017. The SMT-LIB Standard: Version 2.6. Technical report, Department of Computer Science, The University of Iowa. Available at `www.SMT-LIB.org`.

Benerecetti, M.; and Faella, M. 2017. Automatic Synthesis of Switching Controllers for Linear Hybrid Systems: Reachability Control. *ACM Trans. Embed. Comput. Syst.*, 16(4): 104:1–104:27.

Beyene, T.; Chaudhuri, S.; Popeea, C.; and Rybalchenko, A. 2014. A constraint-based approach to solving games on infinite graphs. In *POPL'14, Proc. of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 221–233.

Bjørner, N.; Gurfinkel, A.; McMillan, K. L.; and Rybalchenko, A. 2015. Horn Clause Solvers for Program Verification. In *Fields of Logic and Computation II - Essays Dedicated to Yuri Gurevich on the Occasion of His 75th Birthday*, volume 9300 of *LNCS*, 24–51. Springer.

Bodlaender, M. H. L.; Hurkens, C. A.; Kusters, V. J.; Staals, F.; Woeginger, G. J.; and Zantema, H. 2012. Cinderella versus the wicked stepmother. In *IFIP International Conference on Theoretical Computer Science*, 57–71. Springer.

de Moura, L. M.; and Bjørner, N. S. 2008. Z3: An Efficient SMT Solver. In Ramakrishnan, C. R.; and Rehof, J., eds., *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4963 of *Lecture Notes in Computer Science*, 337–340. Springer.

Faella, M.; and Parlato, G. 2022. Reasoning About Data Trees Using CHCs. In Shoham, S.; and Vizel, Y., eds., *Computer Aided Verification - 34th International Conference, CAV 2022, Haifa, Israel, August 7-10, 2022, Proceedings, Part II*, volume 13372 of *Lecture Notes in Computer Science*, 249–271. Springer.

Farzan, A.; and Kincaid, Z. 2018. Strategy synthesis for linear arithmetic games. In *POPL*, volume 2 of *Proc. ACM Program. Lang.*, 61:1–61:30.

Fedyukovich, G.; and Rümmer, P. 2021. Competition Report: CHC-COMP-21. In *Proceedings 8th Workshop on Horn Clauses for Verification and Synthesis, HCVS@ETAPS 2021, Virtual, 28th March 2021*, volume 344 of *EPTCS*, 91–108.

Grebenshchikov, S.; Lopes, N. P.; Popeea, C.; and Rybalchenko, A. 2012. Synthesizing software verifiers from proof rules. In *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '12, Beijing, China - June 11 - 16, 2012*, 405–416. ACM.

Gurfinkel, A.; and Bjørner, N. 2019. The Science, Art, and Magic of Constrained Horn Clauses. In *21st International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2019, Timisoara, Romania, September 4-7, 2019*, 6–10. IEEE.

Komuravelli, A.; Gurfinkel, A.; Chaki, S.; and Clarke, E. M. 2013. Automatic Abstraction in SMT-Based Unbounded Software Model Checking. In Sharygina, N.; and Veith, H., eds., *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*, 846–862. Springer.

Manna, Z.; and Zarba, C. G. 2002. Combining Decision Procedures. In *Formal Methods at the Crossroads. From Panacea to Foundational Support, 10th Anniversary Colloquium of UNU/IIST, the International Institute for Software Technology of The United Nations University, Lisbon, Portugal, March 18-20, 2002, Revised Papers*, volume 2757 of *LNCS*, 381–422. Springer.

Martin, D. A. 1975. Borel Determinacy. *Annals of Mathematics*, 102(2): 363–371.

Mazala, R. 2002. Infinite games. In *Automata logics, and infinite games*, 23–38. Springer.

Minsky, M. L. 1967. *Computation: Finite and Infinite Machines*. USA: Prentice-Hall, Inc. ISBN 0131655639.

Wu, K.; Fang, L.; Xiong, L.; Lai, Z.-R.; Qiao, Y.; Chen, K.; and Rong, F. 2020. Automatic Synthesis of Generalized Winning Strategies of Impartial Combinatorial Games Using SMT Solvers. In *IJCAI*, 1703–1711.