Causality-Based Game Solving

Christel Baier[1], Norine Coenen[2], Bernd Finkbeiner[2], Florian Funke[1], Simon Jantsch[1], and Julian Siber[2](✉)

[1] Technische Universität Dresden, Dresden, Germany
{christel.baier,florian.funke, simon.jantsch}@tu-dresden.de

[2] CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
{norine.coenen,finkbeiner,julian.siber}@cispa.de

Abstract. We present a causality-based algorithm for solving two-player reachability games represented by logical constraints. These games are a useful formalism to model a wide array of problems arising, e.g., in program synthesis. Our technique for solving these games is based on the notion of *subgoals*, which are slices of the game that the reachability player necessarily needs to pass through in order to reach the goal. We use Craig interpolation to identify these necessary sets of moves and recursively slice the game along these subgoals. Our approach allows us to infer winning strategies that are structured along the subgoals. If the game is won by the reachability player, this is a strategy that progresses through the subgoals towards the final goal; if the game is won by the safety player, it is a permissive strategy that completely avoids a single subgoal. We evaluate our prototype implementation on a range of different games. On multiple benchmark families, our prototype scales dramatically better than previously available tools.

1 Introduction

Two-player games are a fundamental model in logic and verification due to their connection to a wide range of topics such as decision procedures, synthesis and control [1,2,6,7,11,21]. Algorithmic techniques for *finite-state* two-player games have been studied extensively for many acceptance conditions [20]. For *infinite-state* games most problems are directly undecidable. However, infinite state spaces occur naturally in domains like software synthesis [34] and cyber-physical systems [23], and hence handling such games is of great interest. An elegant classification of infinite-state games that can be algorithmically handled, depending

This work was partially supported by DFG grant 389792660 as part of TRR 248 – CPEC, see https://perspicuous-computing.science, the Cluster of Excellence EXC 2050/1 (CeTI, project ID 390696704, as part of Germany's Excellence Strategy), DFG-projects BA-1679/11-1 and BA-1679/12-1, the Research Training Group QuantLA (GRK 1763), and by the European Research Council (ERC) Grant OSARES (No. 683300).

A. Silva and K. R. M. Leino (Eds.) CAV 2021, LNCS 12759, pp. 894–917, 2021.
https://doi.org/10.1007/978-3-030-81685-8_42

on the acceptance condition of the game, was given in [14]. The authors assume a symbolic encoding of the game in a very general form. More recently, incomplete procedures for solving infinite-state two-player games specified using logical constraints were studied [4,18]. While [4] is based on automated theorem-proving for Horn formulas and handles a wide class of acceptance conditions, the work in [18] focusses on reachability games specified in the theory of linear arithmetic, and uses sophisticated decision procedures for that theory.

In this paper, we present a novel technique for solving logically represented reachability games based on the notion of *subgoals*. A *necessary* subgoal is a transition predicate that is satisfied at least once on every play that reaches the overall goal. It represents an intermediate target that the reachability player must reach in order to win. Subgoals open up game solving to the study of cause-effect relationships in the form of counterfactual reasoning [28]: If a cause (the subgoal) had not occurred, then the effect (reaching the goal) would not have happened. Thus for the safety player, a necessary subgoal provides a chance to win the game based on local information: If they control all states satisfying the pre-condition of the subgoal, then any strategy that in these states picks a transition outside of the subgoal is winning. Finding such a necessary subgoal may let us conclude that the safety player wins without ever having to unroll the transition relation.

On the other hand, passing through a necessary subgoal is in general not enough for the reachability player to win. We call a subgoal *sufficient* if indeed the reachability player has a winning strategy from every state satisfying the post-condition of the subgoal. Dual to the description in the preceding paragraph, sufficient subgoals provide a chance for the reachability player to win the global game as they must merely reach this intermediate target. The two properties differ in one key aspect: While necessity of a subgoal only considers the paths of the game arena, for sufficiency the game structure is crucial.

We show how Craig interpolants can be used to compute necessary subgoals, making our methods applicable to games represented by any logic that supports interpolation. In contrast, determining whether a subgoal is sufficient requires a partial solution of the given game. This motivates the following recursive approach. We slice the game along a necessary subgoal into two parts, the pre-game and the post-game. In order to guarantee these games to be smaller, we solve the post-game under the assumption that the considered subgoal was bridged *for the last time*. We conclude that the safety player wins the overall game if they can avoid all initial states of the post-game that are winning for the reachability player. Otherwise, the pre-game is solved subject to the winning condition given by the sufficient subgoal consisting of these states. This approach does not only determine which player wins from each initial state, but also computes symbolically represented winning strategies with a causal structure. Winning safety player strategies induce necessary subgoals that the reachability player cannot pass, which constitutes a cause for their loss. Winning reachability player strategies represent a sequence of sufficient subgoals that will be passed, providing an explanation for the win. All missing proofs for our theoretical results can be found in the full version of this paper [3].

The Python-based implementation CabPy of our approach was used to compare its performance to SimSynth [18], which is, to the best of our knowledge, the only other available tool for solving linear arithmetic reachability games. Our experiments demonstrate that our algorithm is competitive in many case studies. We can also confirm the expectation that our approach heavily benefits from qualitatively expressive Craig interpolants. It is noteworthy that like SimSynth our approach is fully automated and does not require any input in the form of hints or templates. Our contributions are summarized as follows:

– We introduce the concept of *necessary* and *sufficient subgoals* and show how Craig interpolation can be used to compute necessary subgoals (Sect. 4).
– We describe an algorithm for solving logically represented two-player reachability games using these concepts. We also discuss how to compute representations of winning strategies in our approach (Sect. 5).
– We evaluate our approach experimentally through our Python-based tool CabPy, demonstrating a competitive performance compared to the previously available tool SimSynth on various case studies (Sect. 6).

Related Work. The problem of solving linear arithmetic games is addressed in [18] using an approach that relies on a dedicated decision procedure for quantified linear arithmetic formulas, together with a method to generalize safety strategies from truncated versions of the game that end after a prescribed number of rounds. Other approaches for solving infinite-state games include deductive methods that compute the winning regions of both players using proof rules [4], predicate abstraction where an abstract controlled predecessor operation is used on the abstract game representation [38], and symbolic BDD-based exploration of the state space [15]. Additional techniques are available for finite-state games, e.g., generalizing winning runs into a winning strategy for one of the players [31].

Our notion of subgoal is related to the concept of landmarks as used in planning [22]. Landmarks are milestones that must be true on every successful plan, and they can be used to decompose a planning task into smaller sub-tasks. Landmarks have also been used in a game setting to prevent the opponent from reaching their goal using counter-planning [32]. Whenever a planning task is unsolvable, one method to find out why is checking hierarchical abstractions for solvability and finding the components causing the problem [36].

Causality-based approaches have also been used for model checking of multi-threaded concurrent programs [24,25]. In our approach, we use Craig interpolation to compute the subgoals. Interpolation has already been used in similar contexts before, for example to extract winning strategies from game trees [16] or to compute new predicates to refine the game abstractions [10]. In [18], interpolation is used to synthesize concrete winning strategies from so called *winning strategy skeletons*, which describe a set of strategies of which at least one is winning.

2 Motivating Example

Consider the scenario that an expensive painting is displayed in a large exhibition room of a museum. It is secured with an alarm system that is controlled via a control panel on the opposite side of the room. A security guard is sleeping at the control panel and occasionally wakes up to check whether the alarm is still armed. To steal the painting, a thief first needs to disable the alarm and then reach the painting before the alarm has been reactivated. We model this scenario as a two-player game between a safety player (the guard) and a reachability player (the thief) in the theory of linear arithmetic. The moves of both players, their initial positions, and the goal condition are described by the formulas:

$$\begin{aligned}
\mathit{Init} \equiv\ & \neg\mathbf{r} \wedge x = 0 \wedge y = 0 \wedge p = 0 \wedge a = 1 \wedge t = 0, \\
\mathit{Guard} \equiv\ & \neg\mathbf{r} \wedge \mathbf{r}' \wedge x' = x \wedge y' = y \wedge p' = p \\
& \wedge ((t' = t - 1 \wedge a' = a) \vee (t \leq 0 \wedge t' = 2)), && \text{(sleep or wake up)} \\
\mathit{Thief} \equiv\ & \mathbf{r} \wedge \neg\mathbf{r}' \wedge t' = t \\
& \wedge x + 1 \geq x' \geq x - 1 \wedge y + 1 \geq y' \geq y - 1 && \text{(move)} \\
& \wedge (x' \neq 0 \vee y' \neq 10 \implies a' = a) && \text{(alarm off)} \\
& \wedge (x' \neq 10 \vee y' \neq 5 \vee a = 1 \implies p' = p), && \text{(steal)} \\
\mathit{Goal} \equiv\ & \neg\mathbf{r} \wedge p = 1.
\end{aligned}$$

The thief's position in the room is modeled by two coordinates $x, y \in \mathbb{R}$ with initial value $(0, 0)$, and with every transition the thief can move some bounded distance. Note that we use primed variables to represent the value of variables after taking a transition. The control panel is located at $(0, 10)$ and the painting at $(10, 5)$. The status of the alarm and the painting are described by two boolean variables $a, p \in \{0, 1\}$. The guard wakes up every two time units, modeled by the variable $t \in \mathbb{R}$. The variables x, y are bounded to the interval $[0, 10]$ and t to $[0, 2]$. The boolean variable $\mathbf{r}$ encodes who makes the next move. In the presented configuration, the thief needs more time to move from the control panel to the painting than the guard will sleep. It follows that there is a winning strategy for the guard, namely, to always reactivate the alarm upon waking up.

Although it is intuitively fairly easy to come up with this strategy for the guard, it is surprisingly hard for game solving tools to find it. The main obstacle is the infinite state space of this game. Our approach for solving games represented in this logical way imitates *causal reasoning*: Humans observe that in order for the thief to steal the painting (i.e., the effect $p = 1$), a transition must have been taken whose source state does not satisfy the pre-condition of (steal) while the target state does. Part of this cause is the condition $a = 0$, i.e., the alarm is off. Recursively, in order for the effect $a = 0$ to happen, a transition setting a from 1 to 0 must have occurred, and so on.

Our approach captures these cause-effect relationships through the notion of *necessary subgoals*, which are essential milestones that the reachability player has to transition through in order to achieve their goal. The first necessary subgoal corresponding to the intuitive description above is

$$C_1 = (\mathit{Guard} \vee \mathit{Thief}) \wedge p \neq 1 \wedge p' = 1.$$

In this case, it easy to see that C_1 is also a *sufficient subgoal*, meaning that all successor states of C_1 are winning for the thief. Therefore, it is enough to solve the game with the modified objective to reach those predecessor states of C_1 from which the thief can *enforce* C_1 being the next move (even if it is not their turn). Doing so recursively produces the necessary subgoal

$$C_2 = (\mathit{Guard} \vee \mathit{Thief}) \wedge a \neq 0 \wedge a' = 0,$$

meaning that some transition must have caused the effect that the alarm is disabled. However, C_2 is *not* sufficient which can be seen by recursively solving the game spanning from successor states of C_2 to C_1. This computation has an important caveat: After passing through C_2, it may happen that a is reset to 1 at a later point (in this particular case, this constitutes precisely the winning strategy of the safety player), which means that there is no canonical way to slice the game along this subgoal into smaller parts. Hence the recursive call solves the game from C_2 to C_1 *subject to* the bold assumption that any move from $a = 0$ to $a' = 1$ is winning for the guard. This generally underapproximates the winning states of the thief. Remarkably, we show that this approximation is enough to build winning strategies for *both* players from their respective winning regions. In this case, it allows us to infer that moving through C_2 is always a losing move for the thief. However, at the same time, any play reaching *Goal* has to move through C_2. It follows that the thief loses the global game.

We evaluated our method on several configurations of this game, which we call *Mona Lisa*. The results in Sect. 6 support our conjecture that the room size has little influence on the time our technique needs to solve the game.

3 Preliminaries

We consider two-player reachability games defined by formulas in a given logic $\mathcal{L}$. We let $\mathcal{L}(\mathcal{V})$ be the $\mathcal{L}$-formulas over a finite set of variables $\mathcal{V}$, also called *state predicates* in the following. We call $\mathcal{V}' = \{v' \mid v \in \mathcal{V}\}$ the set of *primed variables*, which are used to represent the value of variables after taking a transition. Transitions are expressed by formulas in the set $\mathcal{L}(\mathcal{V} \cup \mathcal{V}')$, called *transition predicates*. For some formula $\varphi \in \mathcal{L}(\mathcal{V})$, we denote the substitution of all variables by their primed variant by $\varphi[\mathcal{V}/\mathcal{V}']$. Similarly, we define $\varphi[\mathcal{V}'/\mathcal{V}]$.

For our algorithm we will require the satisfiability problem of $\mathcal{L}$-formulas to be decidable and *Craig interpolants* [13] to exist for any two mutually unsatisfiable formulas. Formally, we assume there is a function $\mathrm{Sat} : \mathcal{L}(\mathcal{V}) \to \mathbb{B}$ that checks the satisfiability of some formula $\varphi \in \mathcal{L}(\mathcal{V})$ and an unsatisfiability check $\mathrm{Unsat} : \mathcal{L}(\mathcal{V}) \to \mathbb{B}$. For interpolation, we assume that there is a function $\mathrm{Interpolate} : \mathcal{L}(\mathcal{V}) \times \mathcal{L}(\mathcal{V}) \to \mathcal{L}(\mathcal{V})$ computing a *Craig interpolant* for mutually unsatisfiable formulas: If $\varphi, \psi \in \mathcal{L}(\mathcal{V})$ are such that $\mathrm{Unsat}(\varphi \wedge \psi)$ holds, then $\psi \implies \mathrm{Interpolate}(\varphi, \psi)$ is valid, $\mathrm{Interpolate}(\varphi, \psi) \wedge \varphi$ is unsatisfiable, and $\mathrm{Interpolate}(\varphi, \psi)$ only contains variables shared by φ and ψ.

These functions are provided by many modern *Satisfiability Modulo Theories* (SMT) solvers, in particular for the theories of linear integer arithmetic and linear real arithmetic, which we will use for all our examples. Note that interpolation is usually only supported for the quantifier-free fragments of these logics, while our algorithm will introduce existential quantifiers. Therefore, we resort to quantifier elimination wherever necessary, for which there are known procedures for both linear integer arithmetic and linear real arithmetic formulas [29,33].

In order to distinguish the two players, we will assume that a Boolean variable called $\mathbf{r} \in \mathcal{V}$ exists, which holds exactly in the states controlled by the reachability player. For all other variables $v \in \mathcal{V}$, we let $\mathcal{D}(v)$ be the domain of v, and we define $\mathcal{D} = \bigcup\{\mathcal{D}(v) \mid v \in \mathcal{V}\}$. In the remainder of the paper, we consider the variables $\mathcal{V}$ and their domains to be fixed.

Definition 1 (Reachability Game). *A reachability game is defined by a tuple* $\mathcal{G} = \langle \mathit{Init}, \mathit{Safe}, \mathit{Reach}, \mathit{Goal} \rangle$*, where* $\mathit{Init} \in \mathcal{L}(\mathcal{V})$ *is the* initial condition*,* $\mathit{Safe} \in \mathcal{L}(\mathcal{V} \cup \mathcal{V}')$ *defines the transitions of player* `SAFE`*,* $\mathit{Reach} \in \mathcal{L}(\mathcal{V} \cup \mathcal{V}')$ *defines the transitions of player* `REACH` *and* $\mathit{Goal} \in \mathcal{L}(\mathcal{V})$ *is the* goal condition*.*

We require the formulas ($\mathit{Safe} \implies \neg\mathbf{r}$) *and* ($\mathit{Reach} \implies \mathbf{r}$) *to be valid.*

A *state* s of $\mathcal{G}$ is a valuation of the variables $\mathcal{V}$, i.e., a function $s \colon \mathcal{V} \to \mathcal{D}$ that satisfies $s(v) \in \mathcal{D}(v)$ for all $v \in \mathcal{V}$. We denote the set of states by S, and we let $S_{\texttt{SAFE}}$ be the states s such that $s(\mathbf{r}) = \texttt{false}$, and $S_{\texttt{REACH}}$ be the states s such that $s(\mathbf{r}) = \texttt{true}$. The variable $\mathbf{r}$ determines whether `REACH` or `SAFE` makes the move out of the current state, and in particular $\mathit{Safe} \wedge \mathit{Reach}$ is unsatisfiable.

Given a state predicate $\varphi \in \mathcal{L}(\mathcal{V})$, we denote by $\varphi(s)$ the closed formula we get by replacing each occurrence of variable $v \in \mathcal{V}$ in φ by $s(v)$. Similarly, given a transition predicate $\tau \in \mathcal{L}(\mathcal{V} \cup \mathcal{V}')$ and states s, s', we let $\tau(s, s')$ be the formula we obtain by replacing all occurrences of $v \in \mathcal{V}$ in τ by $s(v)$, and all occurrences of $v' \in \mathcal{V}'$ in τ by $s'(v)$. For replacing only $v \in \mathcal{V}$ by $s(v)$, we define $\tau(s) \in \mathcal{L}(\mathcal{V}')$. A *trap state* of $\mathcal{G}$ is a state s such that $(\mathit{Safe} \vee \mathit{Reach})(s) \in \mathcal{L}(\mathcal{V}')$ is unsatisfiable (i.e., s has no outgoing transitions).

A *play* of $\mathcal{G}$ starting in state s_0 is a finite or infinite sequence of states $\rho = s_0 s_1 s_2 \ldots \in S^+ \cup S^\omega$ such that for all $i < \mathrm{len}(\rho)$ either $\mathit{Safe}(s_i, s_{i+1})$ or $\mathit{Reach}(s_i, s_{i+1})$ is valid, and if ρ is a finite play, then $s_{\mathrm{len}(\rho)}$ is required to be a trap state. Here, $\mathrm{len}(s_0 \ldots s_n) = n$ for finite plays, and $\mathrm{len}(\rho) = \infty$ if ρ is an infinite play. The set of plays of some game $\mathcal{G} = \langle \mathit{Init}, \mathit{Safe}, \mathit{Reach}, \mathit{Goal} \rangle$ is defined as $\mathrm{Plays}(\mathcal{G}) = \{\rho = s_0 s_1 s_2 \ldots \mid \rho \text{ is a play in } \mathcal{G} \text{ s.t. } \mathit{Init}(s_0) \text{ holds}\}$. `REACH` *wins* some play $\rho = s_0 s_1 \ldots$ if the play reaches a goal state, i.e., if there exists some integer $0 \leq k \leq \mathrm{len}(\rho)$ such that $\mathit{Goal}(s_k)$ is valid. Otherwise, `SAFE` wins play ρ. A *reachability strategy* σ_R is a function $\sigma_R : S^* S_{\texttt{REACH}} \to S$ such that if $\sigma_R(\omega s) = s'$ and s is not a trap state, then $\mathit{Reach}(s, s')$ is valid. We say that a play $\rho = s_0 s_1 s_2 \ldots$ is *consistent* with σ_R if for all i such that $s_i(\mathbf{r}) = \texttt{true}$ we have $s_{i+1} = \sigma_R(s_0 \ldots s_i)$. A reachability strategy σ_R is *winning* from some state s if `REACH` wins every play consistent with σ_R starting in s. We define *safety strategies* σ_S for `SAFE` analogously. We say that a player *wins in or from a state* s if they have a winning strategy from s. Lastly, `REACH` *wins the game* $\mathcal{G}$ if they win from some initial state. Otherwise, `SAFE` wins.

We often project a transition predicate T onto the source or target states of transitions satisfying T, which is taken care of by the formulas $\mathrm{Pre}(T) = \exists \mathcal{V}'.\, T$ and $\mathrm{Post}(T) = \exists \mathcal{V}.\, T$. The notation $\exists \mathcal{V}$ (resp. $\exists \mathcal{V}'$) represents the existential quantification over all variables in the corresponding set. Given $\varphi \in \mathcal{L}(\mathcal{V})$, we call the set of transitions in $\mathcal{G}$ that move from states not satisfying φ, to states satisfying φ, the *instantiation* of φ, formally:

$$\mathrm{Instantiate}(\varphi, \mathcal{G}) = (\mathit{Safe} \vee \mathit{Reach}) \wedge \neg\varphi \wedge \varphi'.$$

4 Subgoals

We formally define the notion of subgoals. Let $\mathcal{G} = \langle \mathit{Init}, \mathit{Safe}, \mathit{Reach}, \mathit{Goal} \rangle$ be a fixed reachability game throughout this section, where we assume that $\mathit{Init} \wedge \mathit{Goal}$ is unsatisfiable. Whenever this assumption is not satisfied in our algorithm, we will instead consider the game $\mathcal{G}' = \langle \mathit{Init} \wedge \neg \mathit{Goal}, \mathit{Safe}, \mathit{Reach}, \mathit{Goal} \rangle$ which does satisfy it. As states in $\mathit{Init} \wedge \mathit{Goal}$ are immediately winning for REACH, this is not a real restriction.

Definition 2 (Enforceable transitions). *The set of* enforceable transitions *relative to a transition predicate* $T \in \mathcal{L}(\mathcal{V} \cup \mathcal{V}')$ *is defined by the formula*

$$\mathrm{Enf}(T, \mathcal{G}) = (\mathit{Safe} \vee \mathit{Reach}) \wedge T \wedge \neg\exists \mathcal{V}'.\, \big(\mathit{Safe} \wedge \neg T\big).$$

The enforceable transitions operator serves a purpose similar to the *controlled predecessors* operator commonly known in the literature, which is often used in a backwards fixed point computation, called *attractor construction* [37]. For both operations, the idea is to determine controllability by REACH. The main difference is that we do not consider the whole transition relation, but only a predetermined set of transitions and check from which predecessor states the post-condition of the set can be enforced by REACH. These include all transitions in T controlled by REACH and additionally transitions in T controlled by SAFE such that *all other transitions* in the origin state of the transition also satisfy T. The similarity with the controlled predecessor is exemplified by the following lemma:

Lemma 3. *Let T be a transition predicate, and suppose that all states satisfying* $\mathrm{Post}(T)[\mathcal{V}'/\mathcal{V}]$ *are winning for REACH in $\mathcal{G}$. Then all states in* $\mathrm{Pre}(\mathrm{Enf}(T, \mathcal{G}))$ *are winning for REACH in $\mathcal{G}$.*

Proof. Clearly, all states in $\mathrm{Pre}(\mathrm{Enf}(T, \mathcal{G}))$ that are under the control of REACH are winning for REACH, as in any such state they have a transition satisfying T (observe that $\mathrm{Enf}(T, \mathcal{G}) \implies T$ is valid), which leads to a winning state by assumption.

So let s be a state satisfying $\mathrm{Pre}(\mathrm{Enf}(T, \mathcal{G}))$ that is under the control of SAFE. As $\mathrm{Pre}(\mathrm{Enf}(T, \mathcal{G}))(s)$ is valid, s has a transition that satisfies T (in particular, s is not a trap state). Furthermore, we know that there is no $s' \in S$ such that $\mathit{Safe}(s, s') \wedge \neg T(s, s')$ holds, and hence there is no transition satisfying $\neg T$ from s. Since $\mathrm{Post}(T)[\mathcal{V}'/\mathcal{V}]$ is winning for REACH, it follows that from s player SAFE cannot avoid playing into a winning state of REACH. $\square$

We now turn to a formal definition of *necessary subgoals*, which intuitively are sets of transitions that appear on every play that is winning for `REACH`.

Definition 4 (Necessary subgoal). *A* necessary subgoal $C \in \mathcal{L}(\mathcal{V} \cup \mathcal{V}')$ *for* $\mathcal{G}$ *is a transition predicate such that for every play* $\rho = s_0 s_1 \ldots$ *of* $\mathcal{G}$ *and* $n \in \mathbb{N}$ *such that* $Goal(s_n)$ *is valid, there exists some* $k < n$ *such that* $C(s_k, s_{k+1})$ *is valid.*

Necessary subgoals provide a means by which winning safety player strategies can be identified, as formalized in the following lemma.

Lemma 5. *A safety strategy* σ_S *is winning in* $\mathcal{G}$ *if and only if there exists a necessary subgoal* C *for* $\mathcal{G}$ *such that for all plays* $\rho = s_0 s_1 \ldots$ *of* $\mathcal{G}$ *consistent with* σ_S *there is no* $n \in \mathbb{N}$ *such that* $C(s_n, s_{n+1})$ *holds.*

Proof. "$\Longrightarrow$". The transition predicate $Goal[\mathcal{V}/\mathcal{V}']$ (i.e., transitions with endpoints satisfying *Goal*) is clearly a necessary subgoal. If σ_S is winning for `SAFE`, then no play consistent with σ_S contains a transition in this necessary subgoal. "$\Longleftarrow$". Let C be a necessary subgoal such that no play consistent with σ_S contains a transition of C. Then by Definition 4 no play consistent with σ_S contains a state satisfying *Goal*. Hence σ_S is a winning strategy for `SAFE`. $\square$

Of course, the question remains how to compute non-trivial subgoals. Indeed, using *Goal* as outlined in the proof above provides no further benefit over a simple backwards exploration (see Remark 15 in the following section).

Ideally, a subgoal should represent an interesting key decision to focus the strategy search. As we show next, Craig interpolation allows to extract partial causes for the mutual unsatisfiability of *Init* and *Goal* and can in this way provide necessary subgoals. Recall that a Craig interpolant φ between *Init* and *Goal* is a state predicate that is implied by *Goal*, and unsatisfiable in conjunction with *Init*. In this sense, φ describes an observable *effect* that must occur if `REACH` wins, and the concrete transition that instantiates the interpolant *causes* this effect.

Proposition 6. *Let* φ *be a Craig interpolant for Init and Goal. Then the transition predicate* $\text{Instantiate}(\varphi, \mathcal{G})$ *is a necessary subgoal.*

Proof. As φ is an interpolant, it holds that $Goal \implies \varphi$ is valid and $Init \wedge \varphi$ is unsatisfiable. Consider any play $\rho = s_0 s_1 \ldots$ of $\mathcal{G}$ such that $Goal(s_n)$ is valid for some $n \in \mathbb{N}$. It follows that $\neg\varphi(s_0)$ and $\varphi(s_n)$ are both valid. Consequently, there is some $0 \leq i < n$ such that $\neg\varphi(s_i)$ and $\varphi(s_{i+1})$ are both valid. As all pairs (s_k, s_{k+1}) satisfy either *Safe* or *Reach*, it follows that $\big(\text{Instantiate}(\varphi, \mathcal{G})\big)(s_i, s_{i+1})$ is valid. Hence, $\text{Instantiate}(\varphi, \mathcal{G})$ is a necessary subgoal. $\square$

While avoiding a necessary subgoal is a winning strategy for `SAFE`, reaching a necessary subgoal is in general not sufficient to guarantee a win for `REACH`. This is because there might be some transitions in the necessary subgoal that produce the desired effect described by the Craig interpolant, but that trap `REACH` in a region of the state space where they cannot enforce some other necessary effect to reach goal. For the purpose of describing a set of transitions that is guaranteed to be winning for the reachability player, we introduce *sufficient subgoals.*

Definition 7 (Sufficient subgoal). *A transition predicate $F \in \mathcal{L}(\mathcal{V} \cup \mathcal{V}')$ is called a* sufficient subgoal *if `REACH` wins from every state satisfying* $\mathrm{Post}(F)[\mathcal{V}'/\mathcal{V}]$.

Example 8. Consider the Mona Lisa game $\mathcal{G}$ described in Sect. 2.

$$C_1 = (\mathit{Guard} \vee \mathit{Thief}) \wedge p \neq 1 \wedge p' = 1$$

qualifies as sufficient subgoal, because `REACH` wins from every successor state as all those states satisfy *Goal*. Also, every play reaching *Goal* eventually passes C_1, and hence C_1 is also necessary. On the other hand,

$$C_2 = (\mathit{Guard} \vee \mathit{Thief}) \wedge a \neq 0 \wedge a' = 0$$

is only a necessary subgoal in $\mathcal{G}$, because `SAFE` wins from some (in fact all) states satisfying $\mathrm{Post}(C_2)$.

If the set of transitions in the necessary subgoal C that lead to winning states of `REACH` is definable in $\mathcal{L}$ then we call the transition predicate F that defines it the *largest sufficient subgoal* included in C. It is characterized by the properties (1) $F \implies C$ is valid, and (2) if F' is such that $F \implies F'$ is valid, then either $F \equiv F'$, or F' is not a sufficient subgoal. Since C is a necessary subgoal and F is maximal with the properties above, `REACH` needs to see a transition in F eventually in order to win. This balance of necessity and sufficiency allows us to partition the game along F into a game that happens after the subgoal and one that happens before.

Proposition 9. *Let C be a necessary subgoal, and F be the largest sufficient subgoal included in C. Then `REACH` wins from an initial state s in $\mathcal{G}$ if and only if `REACH` wins from s in the pre-game*

$$\mathcal{G}_{pre} = \langle \mathit{Init}, \mathit{Safe} \wedge \neg F, \mathit{Reach} \wedge \neg F, \mathrm{Pre}(\mathrm{Enf}(F, \mathcal{G}))\rangle.$$

Proof. "$\implies$". Suppose that `REACH` wins in $\mathcal{G}$ from s using strategy σ_R. Assume for a contradiction that `SAFE` wins in $\mathcal{G}_{pre}$ from s using strategy σ_S. Consider strategy σ'_S such that $\sigma'_S(\omega s') = \sigma_S(\omega s')$ if $(\mathit{Safe} \wedge \neg F)(s')$ is satisfiable, and else $\sigma'_S(\omega s') = \sigma''_S(\omega s')$, where σ''_S is an arbitrary safety player strategy in $\mathcal{G}$. Let $\rho = s_0 s_1 \ldots$ be the (unique) play of $\mathcal{G}$ consistent with both σ_R and σ'_S, where $s_0 = s$. Since σ_R is winning in $\mathcal{G}$ and C is a necessary subgoal in $\mathcal{G}$, there must exist some $m \in \mathbb{N}$ such that $C(s_m, s_{m+1})$ is valid. Let m be the smallest such index. Since $F \implies C$, we know for all $0 \leq k < m$ that $\neg F(s_k, s_{k+1})$ holds. Hence, there is the play $\rho' = s_0 s_1 \ldots s_m \ldots$ in $\mathcal{G}_{pre}$ consistent with σ_S. The state s_{m+1} is winning for `REACH` in $\mathcal{G}$, as it is reached on a play consistent with the

winning strategy σ_R. Hence, we know that $F(s_m, s_{m+1})$ holds, because F is the largest sufficient subgoal included in C. If $(\mathit{Reach} \wedge F)(s_m, s_{m+1})$ held, we would have that $\mathrm{Pre}(\mathrm{Enf}(F, \mathcal{G})(s_m)$ holds: a contradiction with ρ' being consistent with σ_S, which we assumed to be winning in $\mathcal{G}_{pre}$. It follows that $(\mathit{Safe} \wedge F)(s_m, s_{m+1})$ holds. We can conclude that $(\mathit{Safe} \wedge \neg F)(s_m)$ is unsatisfiable (i.e., s_m is a trap state in $\mathcal{G}_{pre}$), because in all other cases `SAFE` plays according to σ_S, which cannot choose a transition satisfying F. However, this implies that $\mathrm{Pre}(\mathrm{Enf}(F, \mathcal{G})(s_m)$ holds, again a contradiction with ρ' being consistent with winning strategy σ_S.
"$\Longleftarrow$". If `REACH` wins in $\mathcal{G}_{pre}$ they have a strategy σ_R such that every play consistent with σ_R reaches the set $\mathrm{Pre}(\mathrm{Enf}(F, \mathcal{G}))$. As F is a sufficient subgoal, the states $\mathrm{Post}(F)$ are winning for `REACH` by definition. It follows by Lemma 3 that all states satisfying $\mathrm{Pre}(\mathrm{Enf}(F, \mathcal{G}))$ are winning in $\mathcal{G}$. Combining σ_R with a strategy that wins in all these states yields a winning strategy for `REACH` in $\mathcal{G}$. $\square$

5 Causality-Based Game Solving

Lemma 9 in the preceding section foreshadows how subgoals can be employed in building a recursive approach for the solution of reachability games. Before turning to our actual algorithm, we describe a way to symbolically represent nondeterministic memoryless strategies. As discussed in [18], there is no ideal strategy description language for the class of games we consider. Our approach allows us to describe sets of concrete strategies as defined in Sect. 3 with linear arithmetic formulas. This framework will prove convenient for *strategy synthesis*, i.e., the computation of winning strategies instead of simply determining the winner of the game.

5.1 Symbolically Represented Strategies

We will represent strategies for both players using transition predicates $\mathfrak{S} \in \mathcal{L}(\mathcal{V} \cup \mathcal{V}')$, henceforth called *symbolic strategies*, where we only require that $(\mathfrak{S} \implies (\mathit{Safe} \vee \mathit{Reach}))$ is valid. A sequence $s_0 \dots s_n \in S^+$ is called a *play prefix* if it is a prefix of some play in $\mathcal{G}$, $(\neg \mathit{Goal})(s_j)$ holds for all $0 \leq j \leq n$, and s_n is not a trap state. We say that a play prefix $\rho = s_0 \dots s_n$ *conforms to* a symbolic reachability strategy $\mathfrak{S}$ if for all $j < n$ we have that $\mathfrak{S}(s_j, s_{j+1})$ holds whenever $s_j \in S_{\texttt{REACH}}$ (and analogously for safety strategies). A play conforms to $\mathfrak{S}$ if all its play prefixes conform to $\mathfrak{S}$. We say that $\mathfrak{S}$ is winning for `REACH` in s if all plays from s that conform to $\mathfrak{S}$ are winning for `REACH` and all play prefixes $s_0 \dots s_n \in S^* S_{\texttt{REACH}}$ from s that conform to $\mathfrak{S}$ are such that $(\mathfrak{S} \wedge \mathit{Reach})(s_n)$ is satisfiable (and analogously for `SAFE`). The second condition ensures that the player cannot be forced to play a transition outside of $\mathfrak{S}$ by their opponent while the play has not reached a trap state or *Goal*, and in particular guarantees the existence of a concrete strategy (as defined in Sect. 3) conforming to $\mathfrak{S}$.

Lemma 10. *If `REACH` (`SAFE`) has a winning symbolic strategy in s, then `REACH` (`SAFE`) has a concrete winning strategy in s.*

Proof. Let $\mathfrak{S}$ by a symbolic winning strategy for REACH. Let σ_R be any reachability strategy such that for all play prefixes $\omega s \in S^* S_{\texttt{REACH}}$ that conform to $\mathfrak{S}$ the formula $\mathfrak{S}(s, \sigma_R(\omega s))$ is valid. Such a function is guaranteed to exist, as $(\mathfrak{S} \wedge \mathit{Reach})(s)$ is satisfiable for all such play prefixes by definition. Furthermore, σ_R is winning as all play prefixes of plays consistent with σ_R conform to $\mathfrak{S}$, and hence all these plays are winning by assumption. The proof for SAFE is analogous. $\square$

This representation allows us to specify nondeterministic strategies, but classical memoryless strategies on finite arenas (specified as a function $\sigma \colon S_{\texttt{REACH}} \to S$ or $S_{\texttt{SAFE}} \to S$) can also be represented in this form using a disjunction over formulas $\bigwedge_{v \in \mathcal{V}} (v = s(v) \wedge v' = \sigma(s)(v))$ for varying $s \in S$.

The following lemma shows that a necessary subgoal directly yields a symbolic strategy for SAFE if the subgoal is, in a certain sense, *locally avoidable* by SAFE. It will be our main tool for synthesizing safety player strategies.

Lemma 11. *Let C be a necessary subgoal for $\mathcal{G}$ and suppose that* $\mathrm{Unsat}(\mathrm{Enf}(C, \mathcal{G}))$ *holds. Then, Safe $\wedge \neg C$ is a winning symbolic strategy for SAFE in $\mathcal{G}$.*

5.2 A Recursive Algorithm

We now describe our algorithm which utilizes necessary subgoals to decompose and solve two-player reachability games (Algorithm 1). It is incomplete in the sense that it does not return on every input (Sect. 5.3 discusses special cases with guaranteed termination). If the algorithm returns on input $\mathcal{G}$, it returns a triple $(R, \mathfrak{S}_{\texttt{REACH}}, \mathfrak{S}_{\texttt{SAFE}})$, where (1) R is a state predicate characterizing the initial states that are winning for REACH in $\mathcal{G}$, (2) $\mathfrak{S}_{\texttt{REACH}}$ is a symbolic strategy for REACH that wins in all initial states satisfying R, and (3) $\mathfrak{S}_{\texttt{SAFE}}$ is a symbolic strategy for SAFE that wins in all initial states satisfying $\mathit{Init} \wedge \neg R$. The returned safety strategy $\mathfrak{S}_{\texttt{SAFE}}$ is such that $\neg \mathfrak{S}_{\texttt{SAFE}}$ is a necessary subgoal that SAFE can avoid locally in the game $\mathcal{G}$ restricted to intial states $\mathit{Init} \wedge \neg R$ (see Lemma 11).

Algorithm 1 works as follows. States satisfying *Init* and *Goal* are immediately winning for REACH and thus always part of the returned formula R. Following the discussion at the beginning of Sect. 4, further analysis considers the game starting in the remaining initial states $I = \mathit{Init} \wedge \neg \mathit{Goal}$. If there is no such state, we may return that all initial states are winning (line 5). Here, REACH wins from R without playing any move, and hence $\mathfrak{S}_{\texttt{REACH}} = \texttt{false}$ is a valid winning symbolic strategy (winning symbolic strategies are only required to provide moves in prefixes that have not seen *Goal* so far). We may choose $\mathfrak{S}_{\texttt{SAFE}}$ arbitrarily as there is no initial state winning for SAFE.

If the algorithm does not return in line 5, a necessary subgoal C between I and *Goal* is computed by instantiating a Craig interpolant φ for the two predicates (lines 6 and 7, see also Proposition 6). We break up the remaining description of the algorithm into three parts, which correspond to the main cases that occur when splitting the game along the subgoal C.

Algorithm 1: Reach($\mathcal{G}$)

```
In  : reachability game $\mathcal{G} = \langle \mathit{Init}, \mathit{Safe}, \mathit{Reach}, \mathit{Goal} \rangle$
Out: triple $(R, \mathfrak{S}_{\texttt{REACH}}, \mathfrak{S}_{\texttt{SAFE}})$ s.t.
  – $R \in \mathcal{L}(\mathcal{V})$ represents the set of initial states winning for REACH;
  – $\mathfrak{S}_{\texttt{REACH}}$ is a winning symbolic reachability strategy for states in $R$;
  – $\mathfrak{S}_{\texttt{SAFE}}$ is a winning symbolic safety strategy for states in $\mathit{Init} \wedge \neg R$.

1  begin
2      $R \leftarrow \mathit{Init} \wedge \mathit{Goal}$
3      $I \leftarrow \mathit{Init} \wedge \neg \mathit{Goal}$
4      if Unsat($I$) then
5          return $R$, false, false
6      $\varphi \leftarrow$ Interpolate($I$, $\mathit{Goal}$)
7      $C \leftarrow$ Instantiate($\varphi, \mathcal{G}$)
8      if Unsat(Enf($C, \mathcal{G}$)) then
9          return $R$, false, $\mathit{Safe} \wedge \neg C$
10     $\mathcal{G}_{post} \leftarrow \langle \mathrm{Post}(C)[\mathcal{V}'/\mathcal{V}], \mathit{Safe} \wedge \varphi, \mathit{Reach} \wedge \varphi, \mathit{Goal} \rangle$
11     $R_{post}, \mathfrak{S}^{post}_{\texttt{REACH}}, \mathfrak{S}^{post}_{\texttt{SAFE}} \leftarrow$ Reach $(\mathcal{G}_{post})$
12     $F \leftarrow C \wedge R_{post}[\mathcal{V}/\mathcal{V}']$
13     if Unsat(Enf($F, \mathcal{G}$)) then
14         return $R$, false, $\mathit{Safe} \wedge \neg F \wedge (\varphi \implies \mathfrak{S}^{post}_{\mathit{SAFE}})$
15     if Sat($(\mathit{Reach} \vee \mathit{Safe}) \wedge \varphi \wedge \neg \varphi' \wedge \neg \mathit{Goal}$) then
16         $F \leftarrow F \vee \mathit{Goal}[\mathcal{V}/\mathcal{V}']$
17         $\varphi \leftarrow$ false
18     $\mathcal{G}_{pre} \leftarrow \langle I, \mathit{Safe} \wedge \neg F, \mathit{Reach} \wedge \neg F, \mathrm{Pre}(\mathrm{Enf}(F, \mathcal{G})) \rangle$
19     $R_{pre}, \mathfrak{S}^{pre}_{\texttt{REACH}}, \mathfrak{S}^{pre}_{\texttt{SAFE}} \leftarrow$ Reach $(\mathcal{G}_{pre})$
20     return $R \vee R_{pre}$,
21             combine($\mathfrak{S}^{pre}_{\mathit{REACH}}, F, \mathfrak{S}^{post}_{\mathit{REACH}}$),
22             $(\neg \varphi \implies \mathfrak{S}^{pre}_{\mathit{SAFE}}) \wedge (\varphi \implies \mathfrak{S}^{post}_{\mathit{SAFE}})$
```

Case 1: SAFE can avoid the subgoal C. If the necessary subgoal C qualifies for Lemma 11, we can immediately conclude that SAFE is winning for all states statisfying I (lines 8 and 9). An instance of this case occurs if the interpolant describes a bottleneck in the game which is fully controlled by SAFE. The winning symbolic reachability strategy is $\mathit{Safe} \wedge \neg C$ in this case (line 9), and we will assume that safety strategies returned by recursive calls of the algorithm are essentially negations of necessary subgoals that can be avoided by SAFE.

If Lemma 11 is not applicable, we next find those transitions in C that move into a winning state for the safety player. This is achieved by analyzing the *post-game* (line 10):

$$\mathcal{G}_{post} = \langle \mathrm{Post}(C)[\mathcal{V}'/\mathcal{V}], \mathit{Safe} \wedge \varphi, \mathit{Reach} \wedge \varphi, \mathit{Goal} \rangle.$$

Its initial states are exactly the states one sees after bridging the subgoal C. In order to make sure that $\mathcal{G}_{post}$ is, in some sense, easier to solve than $\mathcal{G}$, we restrict both *Safe* and *Reach* to φ, which is the interpolant used to compute the subgoal C. This has the effect of removing all transitions in states *not* satisfying φ, making them trap states. For the safety player this makes $\mathcal{G}_{post}$ easier to win than $\mathcal{G}$ as all plays ending in such a trap state without seeing *Goal* before are winning for `SAFE` in $\mathcal{G}_{post}$. Hence we formally have:

Lemma 12. *If $\mathfrak{S}$ is a winning symbolic reachability strategy from s in $\mathcal{G}_{post}$, then $\mathfrak{S}$ is also winning from s in $\mathcal{G}$.*

Due to the restriction to φ, intuitively `REACH` wins from a state s in $\mathcal{G}_{post}$ if they can win from s in $\mathcal{G}$ *while staying inside the interpolant* φ. In other words, `REACH` must guarantee that the necessary subgoal C is not visited again in the play. Still, the set R_{post}, as returned in line 11 by the recursive call to Algorithm 1 on $\mathcal{G}_{post}$, is a sufficient subgoal in $\mathcal{G}$, by the above lemma. Furthermore, if `SAFE` can avoid all states satisfying R_{post} (see line 13), then this also implies a winning strategy from all initial states in I. The reason is that `REACH` can only win by eventually visiting a state from which they can win without leaving φ again, as $(\mathit{Goal} \implies \varphi)$ is valid. This is not possible if `SAFE` can avoid all states in R_{post}.

In this case we construct $\mathfrak{S}_{\mathtt{SAFE}}$ as follows. We assume that $\neg\mathfrak{S}^{post}_{\mathtt{SAFE}}$ is a necessary subgoal that can be locally avoided in $\mathcal{G}_{post}$ from all states satisfying $\mathrm{Post}(C)[\mathcal{V}'/\mathcal{V}] \wedge \neg R_{post}$, and furthermore, we know that $F := C \wedge R_{post}[\mathcal{V}/\mathcal{V}']$ can be locally avoided in $\mathcal{G}$ (line 13). Intuitively, playing according to $\mathfrak{S}^{post}_{\mathtt{SAFE}}$ in $\mathcal{G}_{post}$ yields a strategy for `SAFE` which avoids *Goal* and may move back into a state satisfying $\neg\varphi$, which forces `REACH` to bridge the subgoal C again in order to win. It follows that $F \vee (\varphi \wedge \neg\mathfrak{S}^{post}_{\mathtt{SAFE}})$ is a necessary subgoal from I that can be locally avoided by `SAFE` in $\mathcal{G}$, and the corresponding symbolic strategy is $\mathit{Safe} \wedge \neg F \wedge (\varphi \implies \mathfrak{S}^{post}_{\mathtt{SAFE}})$ (we additionally intersect the negated necessary subgoal with *Safe* to ensure that the symbolic strategy only includes legal transitions).

So far, the subgoal was such that `SAFE` could avoid it entirely, or at least avoid all states from which `REACH` would win when forced to remain inside the post-game. If this is not the case, then we also need to consider the *pre-game* (line 18):

$$\mathcal{G}_{pre} = \langle I, \mathit{Safe} \wedge \neg F, \mathit{Reach} \wedge \neg F, \mathrm{Pre}(\mathrm{Enf}(F, \mathcal{G}))\rangle.$$

which intuitively describes the game before bridging the interpolant C for the last time. The exact definition of F will depend on whether C *perfectly partitions* the game or not. In both cases F will be the largest sufficient subgoal contained in a necessary subgoal, which lets us apply Proposition 9 to conclude that the initial winning regions of $\mathcal{G}$ and $\mathcal{G}_{pre}$ coincide.

Case 2: The Subgoal Perfectly Partitions $\mathcal{G}$. We say that φ *perfectly partitions $\mathcal{G}$* if $(\mathit{Reach} \vee \mathit{Safe}) \wedge \varphi \wedge \neg\varphi' \wedge \neg\mathit{Goal}$ is unsatisfiable (cf. line 15). Intuitively, this means that there is no transition that "undoes" the effect of the subgoal C. If this holds, then the restriction of $\mathcal{G}_{post}$ to states satisfying φ is *de facto* no longer a restriction, as no play can reach such a state anyway after passing through the subgoal. This intuition is formalized by the following lemma.

Lemma 13. *Assume that* φ perfectly partitions $\mathcal{G}$*, and let* s *be a state satisfying* $\mathrm{Post}(C)[\mathcal{V}'/\mathcal{V}]$*. Then* `REACH` *wins from* s *in* $\mathcal{G}_{post}$ *if and only if* `REACH` *wins from* s *in* $\mathcal{G}$*.*

It follows that $F = C \wedge R_{post}[\mathcal{V}/\mathcal{V}']$ is the largest sufficient subgoal included in C. By Proposition 9, the same initial states are winning for `REACH` in $\mathcal{G}_{pre}$ and in $\mathcal{G}$. In this case, we construct the desired safety strategy (line 22) as

$$\mathfrak{S}_{\texttt{SAFE}} = (\neg\varphi \implies \mathfrak{S}^{pre}_{\texttt{SAFE}}) \wedge (\varphi \implies \mathfrak{S}^{post}_{\texttt{SAFE}}),$$

where $\neg\mathfrak{S}^{pre/post}_{\texttt{SAFE}}$ are assumed to be necessary subgoals avoidable by `SAFE` in the corresponding subgames. Intuitively, the combined strategy consists of following $\mathfrak{S}^{pre}_{\texttt{SAFE}}$ as long as one remains in the pre-game, which, by induction hypothesis, allows `SAFE` to avoid all transitions from F if starting in R_{pre}. If the play crosses $C \wedge \neg F$, the strategy is to play according to the winning strategy of the post-game.

A symbolic strategy for `REACH` can be given by combining pre- and post-strategies as follows (line 21):

$$\begin{aligned}
\mathrm{combine}(\mathfrak{S}^{pre}_{\texttt{REACH}}, F, \mathfrak{S}^{post}_{\texttt{REACH}}) := \ & (\mathrm{Pre}(\mathfrak{S}^{post}_{\texttt{REACH}}) \implies \mathfrak{S}^{post}_{\texttt{REACH}}) \\
\wedge \ & ((\neg \mathrm{Pre}(\mathfrak{S}^{post}_{\texttt{REACH}}) \wedge \mathrm{Pre}(F)) \implies F) \\
\wedge \ & ((\neg \mathrm{Pre}(\mathfrak{S}^{post}_{\texttt{REACH}}) \wedge \neg \mathrm{Pre}(F)) \implies \mathfrak{S}^{pre}_{\texttt{REACH}}) \\
\wedge \ & (\mathrm{Pre}(\mathfrak{S}^{post}_{\texttt{REACH}}) \vee \mathrm{Pre}(F) \vee \mathrm{Pre}(\mathfrak{S}^{pre}_{\texttt{REACH}})).
\end{aligned}$$

This represents a nested conditional strategy that prefers the strategies of the subgames in the priority order $\mathfrak{S}^{post}_{\texttt{REACH}}, F$, and finally $\mathfrak{S}^{pre}_{\texttt{REACH}}$. The reason for this order is that the winning condition in the post-game coincides with the global winning objective (to reach *Goal*), while in the pre-game `REACH` tries to reach a winning state in the post-game. The set F is exactly the bridge between these two. The last condition makes sure that the strategy only includes transitions of states in which it is winning.

Case 3: The subgoal does not perfectly partition $\mathcal{G}$**.** If $\mathrm{Sat}((\mathit{Reach} \vee \mathit{Safe}) \wedge \varphi \wedge \neg\varphi' \wedge \neg \mathit{Goal})$ is true in line 15, we can no longer assume that F is the largest sufficient subgoal in C. The reason is that `SAFE` may win in $\mathcal{G}_{post}$ by moving out of the subgame, but if this move leads to a winning state for `REACH` in $\mathcal{G}$, then such a strategy is winning in $\mathcal{G}_{post}$, but not in $\mathcal{G}$. So we can only assume that F is sufficient (this follows by Lemma 12). In order to apply Proposition 9 we extend F by all transitions that move directly into *Goal* (line 16). This immediately yields a necessary and sufficient subgoal, and so again Proposition 9 applies to $\mathcal{G}_{pre}$ (line 18). We could have also added *Goal*-states to F in Case

2, but we have observed that not doing so improves the performance of our procedure considerably.

The reachability strategy is composed of $\mathfrak{S}^{pre}_{\texttt{REACH}}, F$, and $\mathfrak{S}^{post}_{\texttt{REACH}}$ exactly as in Case 2 (line 21). As all transitions in F are losing for SAFE, and these are the only ones that are removed in $\mathcal{G}_{pre}$, essentially SAFE can play using the same strategies in $\mathcal{G}$ and $\mathcal{G}_{pre}$. We implement this by setting φ to `false` (line 17), in which case $\mathfrak{S}_{\texttt{SAFE}}$ (line 22) equals $(\texttt{true} \implies \mathfrak{S}^{pre}_{\texttt{SAFE}}) \wedge (\texttt{false} \implies \mathfrak{S}^{post}_{\texttt{SAFE}}) \equiv \mathfrak{S}^{pre}_{\texttt{SAFE}}$.

Finally, we formally state the partial correctness of the algorithm, using the ideas from above.

Theorem 14 (Partial correctness). *If* Reach($\mathcal{G}$) *returns* $(R, \mathfrak{S}_{\texttt{REACH}}, \mathfrak{S}_{\texttt{SAFE}})$*, then*

- *R characterizes the set of initial states that are winning for REACH in $\mathcal{G}$,*
- *$\mathfrak{S}_{\texttt{REACH}}$ is a winning symbolic reachability strategy from R,*
- *$\mathfrak{S}_{\texttt{SAFE}}$ is a winning symbolic safety strategy from $Init \wedge \neg R$.*

Remark 15 (Simulating the attractor). Note that Craig interpolants are by no means unique. If we choose the interpolation function so that Interpolate($I, Goal$) always returns $Goal$ (this is a valid interpolant), Algorithm 1 essentially simulates the attractor. In this case the subgoal C (line 7) contains exactly the transitions that move directly into $Goal$. All states in Post$(C)[\mathcal{V}'/\mathcal{V}]$ are then winning for REACH and hence R_{post} would be equivalent to Post$(C)[\mathcal{V}'/\mathcal{V}]$, which implies that $C \equiv F$ holds in this case. The new goal states in $\mathcal{G}_{pre}$ are set to Pre(Enf($F, \mathcal{G}$)), which are exactly the states in Pre(C) that either are controlled by REACH, or such that all their transitions are included in F. Hence the set Pre(Enf($F, \mathcal{G}$)) is exactly the classical controlled predecessor.

One effect of slicing the game along general subgoals is that the initial predicate of the post-game (which describes all states satisfying the post-condition of the subgoal) may be satisfied by many states that do not necessarily need to be considered in order to decide who wins from the initial states of $\mathcal{G}$ (for example, because they are not reachable from any initial state, or cannot reach *Goal*). This can be a drawback if the (superfluous) size of the subgames makes them hard to solve. Notably, this is in general less of an issue for approaches based on unrolling of the transition relation: The method of solving increasingly large step-bounded games [18] will only consider states that are reachable from *Init*, while backwards fixpoint computations will not explore states that do not reach *Goal*. A way of coping with this is to provide additional information on the domains of variables, whenever this is available (we discuss the effect of bounding variable domains in Sect. 6). Indeed, in the case where all variable domains are finite, Algorithm 1 is guaranteed to terminate, as shown in the next subsection.

5.3 Special Cases with Guaranteed Termination

Deciding the winner in the types of games we consider is generally undecidable (see [18] for the case that $\mathcal{L}$ is linear real arithmetic). Since Algorithm 1 returns a correct result whenever it terminates, this implies that it cannot always terminate. In this section, we give two important cases in which we can prove termination.

Theorem 16. *If the domains of all variables in $\mathcal{G}$ are finite, then* Reach($\mathcal{G}$) *terminates.*

Remark 17 (Time complexity). The termination argument in the proof yields a single-exponential upper bound on the runtime of the algorithm, where the input size is measured in the number of concrete transitions of the game. This is because in both recursive calls the subgames may be "almost" as large as the input – they are only guaranteed to have at least one concrete transition less.

We now show that, under certain assumptions, our algorithm also terminates for games that have a finite bisimulation quotient. To this end, we first clarify what bisimilarity means in our setting. A relation $R \subseteq S \times S$ over the states of $\mathcal{G}$ is called a *bisimulation* on $\mathcal{G}$, if

- for all $(s_1, s_2) \in R$ the formulas $\mathit{Goal}(s_1) \iff \mathit{Goal}(s_2)$, $\mathit{Init}(s_1) \iff \mathit{Init}(s_2)$ and $\mathbf{r}(s_1) \iff \mathbf{r}(s_2)$ are valid (recall that $\mathbf{r}$ holds exactly in states controlled by REACH).
- for all $(s_1, s_2) \in R$ and $s_1' \in S$ such that $(\mathit{Safe} \vee \mathit{Reach})(s_1, s_1')$ holds, there exists $s_2' \in S$ such that $(\mathit{Safe} \vee \mathit{Reach})(s_2, s_2')$ holds, and $(s_1', s_2') \in R$.
- for all $(s_1, s_2) \in R$ and $s_2' \in S$ such that $(\mathit{Safe} \vee \mathit{Reach})(s_2, s_2')$ holds, there exists $s_1' \in S$ such that $(\mathit{Safe} \vee \mathit{Reach})(s_1, s_1')$ holds, and $(s_1', s_2') \in R$.

We say that s_1 and s_2 are *bisimilar* (denoted by $s_1 \sim s_2$) if there exists a bisimulation R such that $(s_1, s_2) \in R$. Bisimilarity is an equivalence relation, and it is the coarsest bisimulation on $\mathcal{G}$. The equivalence classes are called *bisimulation classes*. As the winning region of any player can be expressed in the μ-calculus [39] and the μ-calculus is invariant under bisimulation [9], it follows that bisimilar states are won by the same player.

Lemma 18. *Let R be a bisimulation on $\mathcal{G}$. If $(s_1, s_2) \in R$, then REACH wins from s_1 in $\mathcal{G}$ if and only if REACH wins from s_2 in $\mathcal{G}$.*

We will assume that for each bisimulation class S_i there exists a formula $\psi_i \in \mathcal{L}(\mathcal{V})$ that *defines* S_i, formally: For all $s \in S$, $\psi_i(s)$ holds if and only if $s \in S_i$. Furthermore, we will assume that the interpolation procedure *respects* $\sim$, formally: Interpolate(φ, ψ) is equivalent to a disjunction of formulas ψ_i. Such an interpolant exists if ψ or φ already satisfy this assumption.

Theorem 19. *Let $\mathcal{G}$ be a reachability game with finite bisimulation quotient under $\sim$ and assume that all bisimulation classes of $\mathcal{G}$ are definable in $\mathcal{L}$. Furthermore, assume that* Interpolate *respects $\sim$. Then,* Reach($\mathcal{G}$) *terminates.*

6 Case Studies

In this section we evaluate our approach on a number of case studies. Our prototype CabPy[2] is written in Python and implements the game solving part of the presented algorithm. Extending it to returning a symbolic strategy using the ideas outlined above is straightforward. We compared our prototype with SimSynth [18], the only other readily available tool for solving linear arithmetic games. The evaluation was carried out with Ubuntu 20.04, a 4-core Intel® Core™ i5 2.30 GHz processor, as well as 8 GB of memory. CabPy uses the PySMT [19] library as an interface to the MathSAT5 [12] and Z3 [30] SMT solvers. On all benchmarks, the timeout was set to 10 min. In addition to the winner, we report the runtime and the number of subgames our algorithm visits. Both may vary with different SMT solvers or in different environments.

6.1 Game of Nim

Game of Nim is a classic game from the literature [8] and played on a number of heaps of stones. Both players take turns of choosing a single heap and removing at least one stone from it. We consider the version where the player that removes the last stone wins. Our results are shown in Fig. 1. In instances with three heaps or more we bounded the domains of the variables in the instance description, by specifying that no heap exceeds its initial size and does not go below zero.

Following the discussion in Sect. 5.3, we need to bound the domains to ensure the termination of our tool on these instances. Remarkably, bounding the variables was not necessary for instances with only two heaps, where our tool CabPy scales to considerably larger instances than SimSynth. We did not add the same constraints to the input of SimSynth, as for SimSynth this resulted in longer runtimes rather than shorter. In Game of Nim, there are no natural necessary subgoals that the safety player can locally control.

The results (see Fig. 1) demonstrate that our approach is not completely dependent on finding the right interpolants and is in particular also competitive when the reachability player wins the game. We suspect that SimSynth performs worse in these cases because the safety player has a large range of possible moves in most states, and inferring the win of the reachability player requires the tool to backtrack and try our all of them.

6.2 Corridor

We now consider an example that demonstrates the potential of our method in case the game structure contains natural bottlenecks. Consider a corridor of 100 rooms arranged in sequence, i.e., each room i with $0 \leq i < 100$ is connected to room $i + 1$ with a door. The objective of the reachability player is to reach

[2] The source code of CabPy and our experimental data are both available at https://github.com/reactive-systems/cabpy. We provide a virtual machine image with CabPy already installed for reproducing our evaluation [35].

	CabPy		SimSynth	
Heaps	Subgames	Time(s)	Time(s)	Winner
(4,4)	19	1.50	10.44	`REACH`
(4,5)	23	1.92	12.74	`SAFE`
(5,5)	23	1.99	85.75	`REACH`
(5,6)	27	2.90	91.66	`SAFE`
(6,6)	28	3.04	Timeout	`REACH`
(6,7)	31	3.76	Timeout	`SAFE`
(20,20)	88	94.85	Timeout	`REACH`
(20,21)	94	113.04	Timeout	`SAFE`
(30,30)	128	364.13	Timeout	`REACH`
(30,31)	135	404.02	Timeout	`SAFE`
(3,3,3)b	23	13.63	2.85	`SAFE`
(1,4,5)b	32	7.00	289.85	`REACH`
(4,4,4)b	33	50.55	24.39	`SAFE`
(2,4,6)b	38	19.77	Timeout	`REACH`
(5,5,5)b	33	127.89	162.50	`SAFE`
(3,5,6)b	40	86.56	Timeout	`REACH`
(2,2,2,2)b	39	84.79	213.79	`REACH`
(2,2,2,3)b	41	102.01	Timeout	`SAFE`

Fig. 1. Experimental results for the Game of Nim. The notation $(h_1, \ldots, h_n)$ denotes the instance played on n heaps, each of which consists of h_i stones. Instances marked with b indicate that the variable domains were explicitly bounded in the input for CabPy.

	CabPy		SimSynth	
r	Subgames	Time(s)	Time(s)	Winner
10	10	0.57	3.93	`SAFE`
20	20	1.23	20.48	`SAFE`
40	40	3.42	121.96	`SAFE`
60	60	7.36	Timeout	`SAFE`
80	80	17.72	Timeout	`SAFE`
100	100	26.36	Timeout	`SAFE`

Fig. 2. Experimental results for the Corridor game. The safety player controls the door between rooms $r - 1$ and r.

room 100 and they are free to choose valid values from $\mathbb{R}^2$ for the position in each room at every other turn. The safety player controls some door to a room $r \leq 100$. Naturally, a winning strategy is to prevent the reachability player from passing that door, which is a natural bottleneck and necessary subgoal on the way to the last room.

The experimental results are summarized in Fig. 2. We evaluated several versions of this game, increasing the length from the start to the controlled door.

The results confirm that our causal synthesis algorithm finds the trivial strategy of closing the door quickly. This is because Craig interpolation focuses the subgoals on the room number variable while ignoring the movement in the rooms in between, as can be seen by the number of considered subgames. SimSynth, which tries to generalize a strategy obtained from a step-bounded game, struggles because the tool solves the games that happen between each of the doors before reaching the controlled one.

6.3 Mona Lisa

The game described in Sect. 2 between a thief and a security guard is very well suited to further assess the strength and limitations of both our approach as well as of SimSynth. We ran several experiments with this scenario, scaling the size of the room and the sleep time of the guard, as well as trying a scenario where the guard does not sleep at all. Scaling the size of the room makes it harder for SimSynth to solve this game with a forward unrolling approach, while our approach extracts the necessary subgoals irrespective of the room size. However, scaling the guard's sleep time makes it harder to solve the subgame between the two necessary subgoals, while it only has a minor effect on the length of the unrolling needed to stabilize the play in a safe region, as done by SimSynth.

The results in Fig. 3 support this conjecture. The size of the room has *almost no effect at all* on both the runtime of CabPy and the number of considered subgames. However, as the results for a sleep value of 4 show, the employed combination of quantifier elimination and interpolation introduces some instability in the produced formulas. This means we may get different Craig interpolants and slice the game with more or less subgoals. Therefore, we see a lot of potential in optimizing the interplay between the employed tools for quantifier elimination and interpolation. The phenomenon of the runtime being sensitive to these small changes in values is also seen with SimSynth, where a longer sleep time sometimes means a faster execution.

6.4 Program Synthesis

Lastly, we study two benchmarks that are directly related to program synthesis. The first problem is to synthesize a controller for a thermostat by filling out an incomplete program, as described in [4]. A range of possible initial values of the room temperature c is given, e.g., $20.8 \leq c \leq 23.5$, together with the temperature dynamics which depend on whether the heater is on (variable $o \in \mathbb{B}$). The objective for `SAFE` is to control the value of o in every round such that c stays between 20 and 25. This is a common benchmark for program synthesis tools and both CabPy and SimSynth solve it quickly (see Fig. 4). The other problem relates to Lamport's bakery algorithm [26]. We consider two processes using this protocol to ensure mutually exclusive access to a shared resource. The game describes the task of synthesizing a scheduler that violates the mutual exclusion. This essentially is a model checking problem, and we study it to see how well the tools can infer a safety invariant that is out of control of the safety player.

		CabPy		SimSynth	
Size	Sleep	Subgames	Time(s)	Time(s)	Winner
10×10	-	7	0.61	4.79	SAFE
20×20	-	7	0.60	25.26	SAFE
40×40	-	7	0.61	157.62	SAFE
10×10	1	10	4.22	20.31	SAFE
20×20	1	11	4.34	36.44	SAFE
40×40	1	11	4.65	226.14	SAFE
10×10	2	13	5.88	7.40	SAFE
20×20	2	14	5.98	60.00	SAFE
40×40	2	13	5.92	270.48	SAFE
10×10	3	18	26.58	13.94	SAFE
20×20	3	17	26.19	115.53	SAFE
40×40	3	18	27.85	290.12	SAFE
10×10	4	30	175.27	13.96	SAFE
20×20	4	22	204.79	60.08	SAFE
40×40	4	27	123.95	319.47	SAFE

Fig. 3. Experimental results for the Mona Lisa game.

	CabPy		SimSynth	
Name	Subgames	Time(s)	Time(s)	Winner
Thermostat	6	0.44	0.39	SAFE
Bakery	46	18.25	Timeout	SAFE

Fig. 4. Experimental results for program synthesis problems.

For our approach, this makes no difference, as both players may play through a subgoal and the framework is well suited to find a safety invariant. The forward unrolling approach of SimSynth, however, seems to explore the whole state space before inferring safety, and fails to find an invariant before a timeout.

7 Conclusion

Our work is a step towards the fully automated synthesis of software. It targets symbolically represented reachability games which are expressive enough to model a variety of problems, from common game benchmarks to program synthesis problems. The presented approach exploits causal information in the form of *subgoals*, which are parts of the game that the reachability player needs to pass through in order to win. Having computed a subgoal, which can be done using Craig interpolation, the game is split along the subgoal and solved recursively. At the same time, the algorithm infers a structured symbolic strategy for the winning player. The evaluation of our prototype implementation CabPy shows that our approach is practically applicable and scales much better than previously available tools on several benchmarks. While termination is only guaranteed for games with finite bisimulation quotient, the experiments demonstrate that several infinite games can be solved as well.

This work opens up several interesting questions for further research. One concerns the quality of the returned strategies. Due to its compositional nature, at first sight it seems that our approach is not well-suited to handle global optimization criteria, such as reaching the goal in fewest possible steps. On the other hand, the returned strategies often involve only a few key decisions and we believe that therefore the strategies are often very sparse, although this has to be further investigated. We also plan to automatically extract deterministic strategies from the symbolic ones [5,17] we currently consider.

Another question regards the computation of subgoals. The performance of our algorithm is highly influenced by which interpolant is returned by the solver. In particular this affects the number of subgames that have to be solved, and how complex they are. We believe that template-based interpolation [27] is a promising candidate to explore for computing good interpolants. This could be combined with the possibility for the user to provide templates or expressive interpolants directly, thereby benefiting from the user's domain knowledge.

References

1. Alur, R., Moarref, S., Topcu, U.: Pattern-based refinement of assume-guarantee specifications in reactive synthesis. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 501–516. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46681-0_49
2. Alur, R., Moarref, S., Topcu, U.: Compositional synthesis of reactive controllers for multi-agent systems. In: Chaudhuri, S., Farzan, A. (eds.) CAV 2016. LNCS, vol. 9780, pp. 251–269. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-41540-6_14
3. Baier, C., Coenen, N., Finkbeiner, B., Funke, F., Jantsch, S., Siber, J.: Causality-based game solving. CoRR (2021). https://arxiv.org/abs/2105.14247, long version with appendix
4. Beyene, T., Chaudhuri, S., Popeea, C., Rybalchenko, A.: A constraint-based approach to solving games on infinite graphs. In: Principles of Programming Languages (POPL). ACM, New York (2014). https://doi.org/10.1145/2535838.2535860
5. Bloem, R., Egly, U., Klampfl, P., Könighofer, R., Lonsing, F.: SAT-based methods for circuit synthesis. In: Formal Methods in Computer-Aided Design (FMCAD). IEEE (2014). https://doi.org/10.1109/FMCAD.2014.6987592
6. Bloem, R., Galler, S., Jobstmann, B., Piterman, N., Pnueli, A., Weiglhofer, M.: Specify, compile, run: hardware from PSL. Electron. Notes Theor. Comput. Sci. **190**(4), 3–16 (2007). https://doi.org/10.1016/j.entcs.2007.09.004
7. Bloem, R., Jobstmann, B., Piterman, N., Pnueli, A., Sa'ar, Y.: Synthesis of reactive(1) designs. J. Comput. Syst. Sci. **78**(3), 911–938 (2012). https://doi.org/10.1016/j.jcss.2011.08.007. In Commemoration of Amir Pnueli
8. Bouton, C.L.: Nim, a game with a complete mathematical theory. Ann. Math. **3**(1/4), 35–39 (1901). https://doi.org/10.2307/1967631

9. Bradfield, J.C., Stirling, C.: Modal mu-calculi. In: Blackburn, P., van Benthem, J.F.A.K., Wolter, F. (eds.) Handbook of Modal Logic, Studies in Logic and Practical Reasoning, vol. 3, pp. 721–756, North-Holland (2007). https://doi.org/10.1016/s1570-2464(07)80015-2
10. Brückner, I., Dräger, K., Finkbeiner, B., Wehrheim, H.: Slicing abstractions. In: Arbab, F., Sirjani, M. (eds.) FSEN 2007. LNCS, vol. 4767, pp. 17–32. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75698-9_2
11. Chen, Y., Tåmovů, J., Belta, C.: LTL Robot Motion Control based on Automata Learning of Environmental Dynamics. In: International Conference on Robotics and Automation. IEEE (2012). https://doi.org/10.1109/ICRA.2012.6225075
12. Cimatti, A., Griggio, A., Schaafsma, B.J., Sebastiani, R.: The MathSAT5 SMT solver. In: Piterman, N., Smolka, S.A. (eds.) TACAS 2013. LNCS, vol. 7795, pp. 93–107. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36742-7_7
13. Craig, W.: Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory. J. Symbolic Logic **22**(3), 269–285 (1957). https://doi.org/10.2307/2963594
14. de Alfaro, L., Henzinger, T.A., Majumdar, R.: Symbolic algorithms for infinite-state games. In: Larsen, K.G., Nielsen, M. (eds.) CONCUR 2001. LNCS, vol. 2154, pp. 536–550. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44685-0_36
15. Edelkamp, S.: Symbolic exploration in two-player games: preliminary results. In: The International Conference on AI Planning & Scheduling (AIPS), Workshop on Model Checking (2002)
16. Eén, N., Legg, A., Narodytska, N., Ryzhyk, L.: SAT-based strategy extraction in reachability games. In: Conference on Artificial Intelligence (AAAI) (2015). https://ojs.aaai.org/index.php/AAAI/article/view/9752
17. Ehlers, R., Moldovan, D.: Sparse positional strategies for safety games. In: Workshop on Synthesis (SYNT), EPTCS (2012). https://doi.org/10.4204/EPTCS.84.1
18. Farzan, A., Kincaid, Z.: Strategy synthesis for linear arithmetic games. Proc. ACM Program. Lang. **2**(POPL) (2017). https://doi.org/10.1145/3158149
19. Gario, M., Micheli, A.: PySMT: a solver-agnostic library for fast prototyping of SMT-based algorithms. In: SMT Workshop 2015 (2015)
20. Grädel, E., Thomas, W., Wilke, T. (eds.): Automata Logics, and Infinite Games. LNCS, vol. 2500. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-36387-4
21. Harding, A., Ryan, M., Schobbens, P.-Y.: A new algorithm for strategy synthesis in LTL games. In: Halbwachs, N., Zuck, L.D. (eds.) TACAS 2005. LNCS, vol. 3440, pp. 477–492. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-31980-1_31
22. Hoffmann, J., Porteous, J., Sebastia, L.: Ordered landmarks in planning. J. Artif. Intell. Res. **22**(1), 215–278 (2004)
23. Jessen, J.J., Rasmussen, J.I., Larsen, K.G., David, A.: Guided controller synthesis for climate controller using Uppaal Tiga. In: Raskin, J.-F., Thiagarajan, P.S. (eds.) FORMATS 2007. LNCS, vol. 4763, pp. 227–240. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75454-1_17
24. Kupriyanov, A., Finkbeiner, B.: Causality-based verification of multi-threaded programs. Concur. Theory (CONCUR) (2013). https://doi.org/10.1007/978-3-642-40184-8_19
25. Kupriyanov, A., Finkbeiner, B.: Causal termination of multi-threaded programs. Comput. Aided Verification (CAV) (2014). https://doi.org/10.1007/978-3-319-08867-9_54

26. Lamport, L.: A new solution of Dijkstra's concurrent programming problem. Commun. ACM **17**(8), 453–455 (1974). https://doi.org/10.1145/361082.361093
27. Leroux, J., Rümmer, P., Subotić, P.: Guiding Craig interpolation with domain-specific abstractions. Acta Informatica **53**(4), 387–424 (2016). https://doi.org/10.1007/s00236-015-0236-z
28. Menzies, P., Beebee, H.: Counterfactual theories of causation. In: Zalta, E.N. (ed.) The Stanford Encyclopedia of Philosophy. Stanford University, Metaphysics Research Lab (2020)
29. Monniaux, D.: A quantifier elimination algorithm for linear real arithmetic. In: Cervesato, I., Veith, H., Voronkov, A. (eds.) LPAR 2008. LNCS (LNAI), vol. 5330, pp. 243–257. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-89439-1_18
30. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_24
31. Narodytska, N., Legg, A., Bacchus, F., Ryzhyk, L., Walker, A.: Solving games without controllable predecessor. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 533–540. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08867-9_35
32. Pozanco, A., E-Martín, Y., Fernández, S., Borrajo, D.: Counterplanning using goal recognition and landmarks. In: International Joint Conference on Artificial Intelligence (IJCAI) (2018). https://doi.org/10.24963/ijcai.2018/668
33. Presburger, M.: Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. Comptes Rendus du I congres de Mathématiciens des Pays Slaves (1929)
34. Ryzhyk, L., Chubb, P., Kuz, I., Le Sueur, E., Heiser, G.: Automatic device driver synthesis with termite. In: Symposium on Operating Systems Principles (SOSP). Association for Computing Machinery (ACM) (2009). https://doi.org/10.1145/1629575.1629583
35. Siber, J.: The Virtual Machine containing CabPy (2021). https://doi.org/10.6084/m9.figshare.14493804.v3
36. Sreedharan, S., Srivastava, S., Smith, D.E., Kambhampati, S.: Why can't you do that HAL? Explaining unsolvability of planning tasks. In: International Joint Conference on Artificial Intelligence (IJCAI) (2019). https://doi.org/10.24963/ijcai.2019/197
37. Thomas, W.: On the synthesis of strategies in infinite games. In: Mayr, E.W., Puech, C. (eds.) STACS 1995. LNCS, vol. 900, pp. 1–13. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-59042-0_57
38. Walker, A., Ryzhyk, L.: Predicate abstraction for reactive synthesis. Formal Meth. Comput. Aided Des. (FMCAD) (2014). https://doi.org/10.1109/FMCAD.2014.6987617
39. Zappe, J.: Modal μ-calculus and alternating tree automata. In: Grädel, E., Thomas, W., Wilke, T. (eds.) Automata Logics, and Infinite Games. LNCS, vol. 2500, pp. 171–184. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-36387-4_10