CAUSAL SOLVING OF REACHABILITY GAMES

MASTER'S THESIS

JULIAN SIBER

UNIVERSITÄT
DES
SAARLANDES

Faculty of Mathematics and Computer Science
Department of Computer Science

Saarbrücken, 19 March 2021

SUPERVISOR:
Prof. Bernd Finkbeiner, Ph.D.[1]

ADVISOR:
Norine Coenen, M.Sc.[1]

REVIEWERS:
Prof. Bernd Finkbeiner, Ph.D.[1]
Prof. Dr. Christel Baier[2]

[1]CISPA Helmholtz Center for Information Security
Saarbrücken, Germany
[2]Technische Universität Dresden
Dresden, Germany

SUBMITTED:
19 March 2021

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

STATEMENT IN LIEU OF AN OATH

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

EINVERSTÄNDNISERKLÄRUNG

Ich bin damit einverstanden, dass die (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

DECLARATION OF CONSENT

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken, 19 March 2021

Julian Siber

ABSTRACT

Two-player games are an elegant way of modeling many problems such as synthesis, which requires constructing a correct system implementation from a formal specification. The specification is encoded in a game between the system and a non-cooperative environment. If a correct implementation for the specification exists, solving the game will yield a system strategy that describes it. However, many practically relevant synthesis applications require solving games over infinite graphs. This remains a particularly challenging problem.

In this thesis, we propose a causal technique for solving reachability games represented by logical constraints. We consider games over both finite and infinite graphs. Our method focuses on the cause-effect-relationships of events that need to appear in any play reaching the goal states. To describe the set of possible causes for some effect observed in every such play, we introduce the notion of subgoals for two-player games.

Subgoals provide a way to determine that the safety player wins using counterfactual reasoning: If they avoid all possible causes described by the subgoal, then the effect of reaching the goal states cannot happen. Using this, we may infer a winning safety player strategy based on a small number of key decisions without fully computing the winning regions of both players. Dual to this, we can determine that the reachability player wins by finding a sequence of subgoals that connects some initial state to the goal states.

We implemented a prototype tool called CabPy and evaluated it on a variety of games over both finite and infinite graphs. The evaluation demonstrates that our approach works well in practice. Compared to SimSynth, a state-of-the-art tool for solving reachability games over infinite graphs, CabPy scales considerably better on several benchmark families.

ACKNOWLEDGEMENTS

I am sincerely grateful to my supervisor Bernd Finkbeiner not only for his professional advice regarding this thesis but, in particular, also for his supportive mentorship throughout the recent years.

Further, I want to thank Christel Baier for reviewing this thesis and the expertise she contributed to the countless discussions leading up to this work.

Many thanks go to Norine Coenen for advising me throughout my research, proofreading this thesis, and her outstanding moral support.

I want to thank Florian Funke and Simon Jantsch for the great discussions, their insightful feedback, and the hard work they put into our project.

Last but not least, I would like to thank Frederik Schmitt and Simon Spies for the relaxing evenings and engaging discussions, and my sisters Stefanie and Leonie for everything.

CONTENTS

1 INTRODUCTION

Our modern world is almost unthinkable without the ubiquitous deployment of digital systems. Computers control our power plants, guide space rockets and satellites, which in turn guide a growing number of autonomous vehicles on the earth's surface. Recent advances in fields like medical monitoring and smart home systems promise even further benefit of disseminating digital systems in our everyday life.

However, the increase in complexity and spread of digital control systems also comes with an increase in risk of catastrophic failure. In many instances, computer systems are designed to interact directly with their physical environment. The safe and correct design of these so called *cyber-physical systems* is, therefore, of paramount importance to avoid potential security and safety hazards for their users. It is generally agreed upon that it is surprisingly hard to design safe systems with a straightforward development process. This is because bugs are hard to spot and understanding every facet and possible behavior of a system in detail is incredibly difficult with complex systems.

A promising approach to alleviate the problem is the use of formal methods in the development process. Formal methods allow to prove the correctness of a given system design with respect to a formal and logical specification, or even come up with a correct system design altogether, directly from the specification. The former approaches that check the correctness of a given system model are known as *verification* and have become more and more widespread in the hardware design process [8] and development of device drivers [5]. The latter, more ambitious approaches that aim to construct a correct implementation directly from a specification are known as *synthesis*. While synthesis has recently found some use in domains like software [35] and even cyber-physical systems [17, 21], it remains a particularly challenging and intriguing problem.

Synthesis problems can often be framed in an elegant way as a game between two-players [2, 3, 7, 15]. The digital system that is to be designed constitutes one player, while the environment which interacts with and cannot be controlled by the system constitutes the opposing player. Like in a game of chess, each of the players has a number of possible actions available at any point in the game. Both players choosing a valid action whenever it is their turn results in a (modeled) execution of the overall system. The objective of a synthesis procedure then is to find an action policy, called *strategy*, such that the system *wins* every such execution against all possible strate-

gies of the environment. The condition for winning a game encodes the given formal specification for the system. Winning the game with some system strategy therefore ensures that no interaction with the environment can violate the specification. Hence, the system design associated with this strategy is provably correct.

The main challenge for synthesis procedures aimed at software or cyber-physical systems is that these domains often require modeling a game over an infinite state space. In contrast to finite-state games, where complete decision procedures exist for a wide range of winning conditions [18], infinite-state games make most problems immediately undecidable [14]. Contrary to this discouraging theoretical result, however, there have recently been proposed a number of dedicated semi-algorithms that can solve a large range of practically relevant games on infinite graphs [4, 14]. The goal of this work is to push this development further and provide a (semi-)algorithm for solving logically represented infinite-state games with reachability objectives. Reachability games describe a game structure where one of the players, the *reachability player*, wins by reaching a certain set of goal states after a finite play. The other player, the *safety player*, wins a play if no such goal state is seen on a possibly infinite play. Depending on the problem description and specification at hand, either player can describe the system or the environment. Our method complements existing approaches for solving reachability games, in order to push one step further towards the vision of a fully automatic synthesis for complex digital systems.

This work is inspired by recent advances in verification, particularly by causality-based model checking [23–25]. This promising verification method utilizes the observation that for many digital systems it is surprisingly easy to come up with manual proofs, while automatic verification procedures struggle with both the infinite data domains and the state space explosion arising in concurrent systems.

Causality-based verification has been successfully employed to efficiently verify the termination of multi-threaded programs [25], the safety of binary semaphore programs [24] and Fischer's real-time protocol for mutual exclusion [36].

Causality-based model checking imitates the causal reasoning that is responsible for the easier manual proofs by focusing on the cause-effect-relationships present in a hypothetical counterexample trace, i.e., a trace violating the specification checked on the model.

We employ the same focus on cause-effect-relationships, but solving two-player games requires considering the strategic options of both players which is beyond the reach of causality-based verification methods. Therefore, our novel technique for solving logically represented reachability games is based on the notion of *subgoals* to capture the causal relationships present in a game structure.

A *necessary* subgoal describes a set of transitions that cannot be avoided on a play that satisfies the goal condition of the reachability player. It represents an intermediate target that the reachability player necessarily needs to reach in order to have any chance at winning the game at all. Subgoals allow to solve reachability games utilizing counterfactual reasoning [29]: If a cause (the necessary subgoal) had not occurred, then the effect (reaching the overall goal condition) would not have happened. In this way, a necessary subgoal can be utilized to find a strategy for the safety player based on local information: If, whenever at the "edge" of the subgoal, the safety player can pick an alternative transition that is not part of the necessary subgoal, they can avoid that the play reaches the goal condition altogether. Based on such a *locally avoidable* necessary subgoal we may construct a winning safety player strategy without unrolling the transition relation, which alleviates the problem of infinite state spaces on certain game structures.

While avoiding a necessary subgoal constitutes a winning strategy for the safety player, reaching the necessary subgoal does not guarantee a win for the reachability player. For this purpose, we introduce *sufficient* subgoals. These are sets of transitions such that the reachability player has a winning strategy from the successor state of every transition in the subgoal. Just like necessary subgoals provide a way to construct a winning safety player strategy based on local information, sufficient subgoals provide a more local criterion to identify a winning reachability player strategy based on reaching the sufficient subgoal from the initial states.

However, computationally necessity and sufficiency differ in one key aspect: A necessary subgoal is defined over all paths of the game, but a sufficient subgoal makes a statement about which player has a wininng strategy. This means that necessary subgoals are much easier to compute than sufficient subgoals. Indeed, the whole transition relation trivially qualifies as a necessary subgoal, so the challenge is to find necessary subgoals that guide the search for safety player strategies in a meaningful manner. Given two mutually unsatisfiable logical formulas, a Craig interpolant [12] is a formula over their shared variables that is implied by one formula and mutually unsatisfiable with the other. We show that Craig interpolants for the initial and goal formulas of some game can be used to compute necessary subgoals. This effectively introduces causality to the analysis by focusing necessary subgoal instantiation on the possible causes for some observable effect that occurs on every play reaching the goal states. In contrast to this straightforward computation based on interpolation, computing a sufficient subgoal requires to solve a part of the global game.

Therefore, we propose a recursive algorithm based on solving the game after the subgoal, the *post-game* and the game before the subgoal, the *pre-game*. In order to guarantee these games to be smaller

than the original game, we solve a restricted post-game which requires that the subgoal was seen for the last time. This approximates the sufficient subgoal. Remarkably, we show that this approximation is exact if certain conditions on the game structure are met, and can still be utilized to solve the game if not. As outlined before, we then either use necessity of the subgoal and local criteria to conclude the win of the safety player or sufficiency to solve the global game by solving the game from the initial states to the sufficient subgoal. The following motivating example illustrates this in more detail.

MOTIVATING EXAMPLE

Consider the security system of an art museum that displays an expensive painting (which for illustration purposes we call *Mona Lisa*). The painting is placed in the main room at coordinates $(9, 5)$ and secured by an alarm, which can be controlled with the help of a control panel at the opposite end of the room at coordinates $(1, 9)$. Figure 1.1 illustrates the main room of the museum. A security guard secures the control panel, but sleeps most of the time. They occasionally wake up and check whether the alarm is still armed. A thief wants to steal the painting and enters the main room through the entrance in the lower-left corner at coordinates $(1, 1)$. In order to steal the painting without being noticed, the thief first needs to disable the alarm and then reach the painting before the alarm has been reactivated.

We model this setup as a reachability game between the two-players guard and thief. The status of the alarm and the painting are described by two Boolean variables $a, p \in \{0, 1\}$, where $a = 1$ and $p = 1$ mean the alarm is on and the painting is stolen, respectively. In this game, the objective of the thief (the reachability player) is to move the game into a state satisfying the goal condition (painting stolen, i.e., $p = 1$) after a finite duration. The objective of the guard (the safety player) is to prevent the goal condition from every being reached, which results in an infinite duration game. The transitions of both players, their initial positions and the goal condition for the thief are described by the following formulas:

$$
\begin{aligned}
\mathit{Init} \equiv \quad & \neg \mathbf{r} \wedge x = 1 \wedge y = 1 \wedge p = 0 \wedge a = 1 \wedge t = 0; \\
\mathit{Guard} \equiv \quad & \neg \mathbf{r} \wedge \mathbf{r}' \wedge x' = x \wedge y' = y \wedge p' = p \\
& \wedge ((t' = t - 1 \wedge a' = a) \vee (t \leqslant 0 \wedge t' = 2)); \\
\mathit{Thief} \equiv \quad & \mathbf{r} \wedge \neg \mathbf{r}' \wedge t' = t \\
& \wedge x + 1 \geqslant x' \geqslant x - 1 \wedge y + 1 \geqslant y' \geqslant y - 1 \\
& \wedge (x' \neq 1 \vee y' \neq 9 \implies a' = a) \\
& \wedge (x' \neq 9 \vee y' \neq 5 \vee a = 1 \implies p' = p); \\
\mathit{Goal} \equiv \quad & \neg \mathbf{r} \wedge p = 1.
\end{aligned}
$$

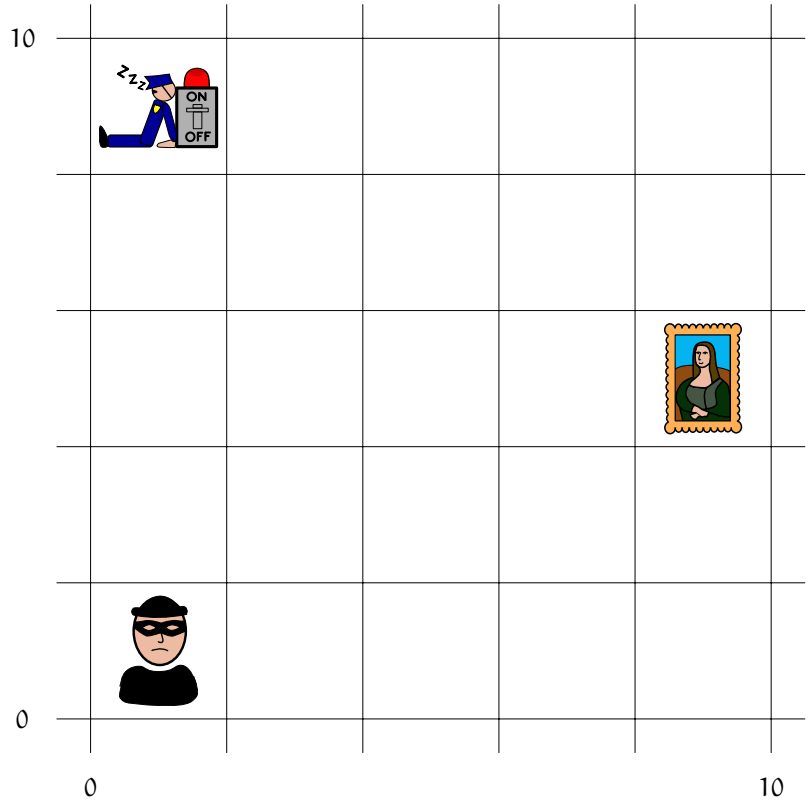

Figure 1.1: The Mona Lisa problem. The painting is secured with an alarm that the thief has to disable first in order to remain undetected. The sleeping guard will occasionally wake up to check whether the alarm is still on.

The Boolean variable $\mathbf{r}$ encodes who makes the next move. We require the guard to make the first move and also require it to be his turn in the goal condition. This is not strictly necessary, but is done to compare our approach with other tools that have this setup fixed.

The thief's position in the room is modeled by two coordinates $x, y \in \mathbb{R}$ with initial values $x = 1 \wedge y = 1$, which can be found as a subformula of the formula *Init* that describes the initial states of the game. With every transition the thief can move some bounded distance, modeled by the subformula

$$x + 1 \geqslant x' \geqslant x - 1 \wedge y + 1 \geqslant y' \geqslant y - 1$$

of the formula *Thief*. The control panel is located at $x = 1 \wedge y = 9$, modeled by the subformula

$$x' \neq 1 \vee y' \neq 9 \implies a' = a$$

of the formula *Thief*. This subformula in particular ensures that the value of the variable a modeling the alarm cannot be changed by the thief as long as the postcondition of the transition satisfies the predicate $x' = 1 \wedge y' = 9$. This means the thief must make a transition

to the location of the control panel to change the alarm. The location of the painting is modeled as just discussed for the control panel in formula *Thief*. Additionally, the formula requires that the alarm is off when the thief steals the painting: Constraint $a = 1$ disqualifies changing the value of p, hence the precondition of the transition to the location of the painting must satisfy $a = 0$.

The guard wakes up every two time units, which is modeled via the variable $t \in \mathbb{R}$ and the subformula

$$((t' = t - 1 \wedge a' = a) \vee (t \leqslant 0 \wedge t' = 2))$$

of formula *Guard*. This formula states that either one time unit is subtracted from t or that t is less or equal to zero and set back to 2. The value of a can only be changed by the guard in the latter case, where it is left unspecified and can therefore be set to either 0 or 1. This means the guard may decide to not reactive alarm even if he wakes up.

The location variables x, y are restricted to the interval $[0, 10]$ to model the room and the time t to $[0, 2]$. This is realized by additional constraints on the formulas *Guard* and *Thief* which we omitted for sake of clarity but will describe further in Section 4.5.

If both player play according to the rules of the reachability game, a play ρ consisting of a sequence of states is produced:

$$\rho = (1, 1, 0, 1, 0, \text{false}) \to (1, 1, 0, 1, 2, \text{true}) \to (3, 7, 0, 1, 2, \text{false}) \to \ldots,$$

where we have that the tuples $(3, 7, 0, 1, 2, \text{false}) = (x, y, p, a, t, \mathbf{r})$ and so on denote the values of the variables in some state (formally, a state is a function from the variables to their domain).

Because the guard only sleeps for 2 time units and the distance from the control panel to the painting takes longer to cover for the thief, the guard has a winning strategy: reactivate the alarm when waking up.

Although the idea of this strategy is very intuitive, it is surprisingly hard for to find fo most game solving tools. The main obstacle is that the state space of this game is infinite, because x, y and t are bounded but real-valued variables.

Our algorithm imitates what we believe is the main reason that it is intuitively easy to construct a strategy for this game: *causal reasoning*. Humans observe that stealing the painting (i.e., $p = 1$) is an effect the thief is required to produce to win. Therefore, something needs to cause this effect. Possible causes are all transitions that move from a configuration where it is not present, i.e., from $p = 0$ to a configuration where the effect is present, i.e., $p = 1$. However, part of the precondition of any such cause is the constraint $a = 0$, i.e., the alarm is off. In order for the effect $a = 0$ to happen, a transition setting a from 1 to 0 must have occurred, and so on. This line of intuitive

reasoning produces a causal chain of necessary transition effects and associated causes, which we aim to reproduce with our approach.

Our algorithm computes observable effects using Craig interpolation between the *Init* and *Goal* formulas. In our example, a possible Craig interpolant is $p = 1$, as it is implied by *Goal* and $\mathit{Init} \wedge p = 1$ is unsatisfiable.

Given such an effect, the algorithm uses *necessary subgoals* to characterize all transitions that could possibly cause it. Necessary subgoals are sets of transitions which are essential milestones that the reachability player has to reach in order have a chance at achieving their overall goal. Subgoals split the state space according to the associated effect into a part that happens before the effect is achieved and a part that happens after the subgoal.

In our example, the possible transitions that could cause the effect $p = 1$ are described by the following necessary subgoal:

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

Clearly, any winning play for the thief has to pass C_1, as on any winning play the painting has to be stolen. Even more, playing C_1 is a sufficient criterion to guarantee the win of the thief, as from any successor state we have $\neg \mathbf{r} \wedge p = 1$. This makes C_1 also a *sufficient subgoal*. Necessity and sufficiency together allow us to solve the global game by solving the *pre-game*, i.e., the game starting in the original initial states with the modified objective to reach those predecessor states of C_1 from which the thief can *enforce* that some concrete transition from C_1 will be the next transition in the play (even if is not the turn of the thief).

Recursively solving this pre-game produces another necessary subgoal

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

which characterizes the set transition that could cause the effect of the alarm being disabled. However, this time C_2 does *not* immediately qualify as sufficient. In order to compute the subset of transitions from C_2 that constitute a sufficient subgoal we need to recursively solve the game from C_2 to C_1. Note that this is, generally, computationally difficult, as this game is not guaranteed to be smaller or easier to solve than the one from *Init* to C_1. Hence, we utilize the following observation: If the thief can produce the effect $a = 0$ on the way to C_1, they in particular can eventually do so *for the last time*. We therefore employ an approximation at this point and assume that after passing C_2, any transition that resets a to 1 is winning for the guard. This under-approximates the sufficient subgoal because it may very well be that after the reactivation the thief is able to turn off the alarm again, and then win the approximated game. But this winning strategy would eventually require them to play through the under-approximated sufficient subgoal we are computing, i.e., there has to

be at least one concrete transition that enforces the effect for the last time.

Remarkably, we show that this approximation is enough to solve the game if either the approximated sufficient subset is empty or if no transition that resets the effect exists. The former case applies to the presented example: Moving through C_2 at any point is losing for the thief because the guard can always reactivate the alarm before the thief reaches the painting. However, at the same time, necessity requires the thief to play through C_2 to reach *Goal* at some point. It follows that the thief loses the global game, and the winning strategy for the guard is to reactivate the alarm whenever possible. Note that this analysis was possible without ever considering the actual size of the room and the distance from the initial location of the thief to the control panel.

We used several configurations of this game in our evaluation in Section 5.1. The results on this and other case studies (Chapter 5) show that our technique works well in practice. Note that while we present this motivation as a toy example (similar to related literature [20]), the discussed problems of controllability and infinite state spaces also occur in synthesis of software or cyber-physical systems. The results in Section 5.1 highlight that indeed the room size has little influence on the time our technique needs to solve the game.

OUTLINE

The rest of this thesis is organized as follows. First, we provide some background on logic and Craig interpolation in Chapter 2. We further introduce some definitions regarding reachability games and discuss approaches to solving these games. Next, we introduce and define the notion of subgoals in Chapter 3. Subgoals constitute the formalism by which we define and analyze causal dependencies in two-player games. We then describe our algorithm that utilizes subgoals to solve reachability games in Chapter 4. We further discuss its correctness and termination behavior. In Chapter 5, we analyze our algorithm on a number of case studies. We compare its performance with SimSynth, a state-of-the-art tool for solving games over infinite graphs. In Chapter 6, we give a detailed overview how our approach relates to causality-based verification, landmarks in planning and other methods for solving infinite-state games. Lastly, we conclude this work and give an outline on promising future avenues of research in Chapter 7.

2

PRELIMINARIES

In this chapter, we give a brief overview over the required background on *satisfiability modulo theories* (SMT) formulas, Craig interpolation and quantifier elimination. Then, we introduce definitions regarding two-player reachability games represented by logical constraints.

2.1 SMT AND CRAIG INTERPOLATION

In this thesis, we encode the semantics of two-player games in (varying) first-order logics. We denote the set formulas from a logic $\mathcal{L}$ over some set of variables $\mathcal{V}$ by $\mathcal{L}(\mathcal{V})$. Formulas from the set $\mathcal{L}(\mathcal{V})$ are called *state predicates*. For some set of variables $\mathcal{V}$, 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 from the set $\mathcal{L}(\mathcal{V} \cup \mathcal{V}')$, called *transition predicates*. For all variables $v \in \mathcal{V}$, we let $\mathcal{D}(v)$ be the domain of v, and we define $\mathcal{D}(\mathcal{V}) = \bigcup\{\mathcal{D}(v) \mid v \in \mathcal{V}\}$.

For the algorithms presented later on in this thesis we rely on a *Satisfiability Modulo Theories* (SMT) solver that has to provide the following functionalities for $\mathcal{L}$-formulas.

We require the satisfiability problem of $\mathcal{L}$-formulas to be decidable and assume the existence of functions that check the satisfiability and unsatisfiability of a given formula $\varphi \in \mathcal{L}(\mathcal{V})$. Formally, we assume there are two functions:

$$\text{Sat} : \mathcal{L}(\mathcal{V}) \to \mathbb{B} \text{ and } \text{Unsat} : \mathcal{L}(\mathcal{V}) \to \mathbb{B}$$

such that $\text{Sat}(\varphi)$ returns true if there is a satisfying model for the formula φ and we have $\text{Unsat}(\varphi) = \neg\,\text{Sat}(\varphi)$ for all $\varphi \in \mathcal{L}(\mathcal{V})$.

Further, we assume that for any two formulas $\varphi, \psi \in \mathcal{L}(\mathcal{V})$ that are mutually unsatisfiable we can compute a *Craig interpolant* [12]. Formally, we require a function:

$$\text{Interpolate} : \mathcal{L}(\mathcal{V}_a) \times \mathcal{L}(\mathcal{V}_b) \to \mathcal{L}(\mathcal{V}_a \cap \mathcal{V}_b)$$

such that if $\text{Unsat}(\varphi \wedge \psi)$ and $\delta = \text{Interpolate}(\varphi, \psi)$, then $\psi \implies \delta$ and $\text{Unsat}(\delta \wedge \varphi)$.

A Craig interpolant δ is a formula ranging over the shared variables of two conjunctively unsatisfiable formulas φ and ψ such that δ is implied by ψ but contradictory with φ. In this sense, an interpolant is one partial cause for the unsatisfiability of the conjunction and can be extracted from the unsatisfiability proof of the conjunction produced by an SMT solver [28].

The above functionalities are offered by many modern SMT solvers, in particular for linear integer arithmetic and linear real arithmetic, as well as Boolean formulas. This covers the theories we are concerned with later on in this thesis. We refer to the SMT competition for a comprehensive overview [38] of SMT solvers. However, Craig interpolation is usually only supported for the quantifier-free fragments of the mentioned logics, while our game solving algorithm makes use of existential quantifiers. We therefore further require a quantifier elimination procedure for our logic $\mathcal{L}$. We do not introduce this function explicitly, but assume the quantifier elimination procedure to be applied wherever necessary. There are known procedures for this problem both for linear integer arithmetic and linear real arithmetic formulas [31, 34], as well as quantified boolean formulas [22], again covering all theories we are concerned with later on.

2.2 REACHABILITY GAMES

Based on the above, we now turn to a formal definition of two-player reachability games defined by $\mathcal{L}$-formulas. We first introduce some general definitions regarding these games, before considering two concrete classes in Subsections 2.2.1 and 2.2.2.

DEFINITION 2.1 (Reachability Game). A reachability game is a tuple $\mathcal{G} = \langle \mathcal{V}, \mathit{Init}, \mathit{Safe}, \mathit{Reach}, \mathit{Goal} \rangle$, where

- $\mathcal{V}$ is a finite set of variables;
- $\mathit{Init} \in \mathcal{L}(\mathcal{V})$ is the *initial condition*, a predicate describing the set of initial states;
- $\mathit{Safe} \in \mathcal{L}(\mathcal{V} \cup \mathcal{V}')$ is a transition predicate describing all legal moves of player `SAFE`;
- $\mathit{Reach} \in \mathcal{L}(\mathcal{V} \cup \mathcal{V}')$ is a transition predicate describing all legal moves of player `REACH`;
- $\mathit{Goal} \in \mathcal{L}(\mathcal{V})$ is the *goal condition*, a predicate describing the set of goal states.

We require that a Boolean variable $\mathbf{r} \in \mathcal{V}$ exists, which arbitrates between the two players. This means that the formulas $(\mathit{Safe} \implies \neg\mathbf{r})$ and $(\mathit{Reach} \implies \mathbf{r})$ are valid in every reachability game $\mathcal{G}$, i.e., in every state only one of the players can take a turn.

Next, we introduce some basic definitions regarding reachability games. In the following, let $\mathcal{G} = \langle \mathcal{V}, \mathit{Init}, \mathit{Safe}, \mathit{Reach}, \mathit{Goal} \rangle$ be a reachability game. A *state* of $\mathcal{G}$ is a valuation of the variables $\mathcal{V}$ that assigns each variable a value from its domain, i.e., a function $s \colon \mathcal{V} \to \mathcal{D}(\mathcal{V})$ 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_{SAFE} be the set of states s such that $s(\mathbf{r}) = \mathtt{false}$, and

$S_{\texttt{REACH}}$ be the set of states s such that $s(\mathbf{r}) = \texttt{true}$. We say SAFE *owns* state s if $s \in S_{\texttt{SAFE}}$, similarly we say REACH owns s if $s \in S_{\texttt{REACH}}$.

Given a state predicate $\varphi \in \mathcal{L}(\mathcal{V})$ and state $s \in S$, we denote by $\varphi(s)$ the closed formula one gets 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 two states $s, s' \in S$, we let $\tau(s, s')$ be the closed formula one gets 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 $s'(v)$. We can also only replace the unprimed variables $v \in \mathcal{V}$ with the values of state $s \in S$, which is denoted by $\tau(s) \in \mathcal{L}(\mathcal{V}')$ and describes a (non-closed) formula ranging over $\mathcal{V}'$. Similarly, we denote by $\tau(\mathcal{V}, s) \in \mathcal{L}(\mathcal{V})$ the formula one gets by replacing only the primed variables $v \in \mathcal{V}'$ with the values of state $s \in S$.

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., there is no legal move starting in s for any of the players).

A *play* of $\mathcal{G}$ starting in state s_0 is a finite or infinite sequence of states $\rho = s_0 s_1 s_2 \ldots \in S^\omega \cup S^+$ consisting of legal moves only, i.e., 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 $\mathcal{G}$ 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 $0 \leqslant k \leqslant \mathrm{len}(\rho)$ 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 and a winning strategy σ_R *for* $\mathcal{G}$ is a strategy of REACH that wins from this s. If REACH does not win $\mathcal{G}$, SAFE wins $\mathcal{G}$ and a winning strategy σ_S *for* $\mathcal{G}$ is a strategy of SAFE that wins from all states s that satisfy *Init*.

If we have two reachability games $\mathcal{G} = \langle \mathcal{V}, \mathit{Init}, \mathit{Safe}, \mathit{Reach}, \mathit{Goal} \rangle$ and $\mathcal{G}' = \langle \mathcal{V}, \mathit{Init}', \mathit{Safe}', \mathit{Reach}', \mathit{Goal}' \rangle$ with the same set of variables and domains, such that $(\mathit{Safe} \vee \mathit{Reach}) \implies (\mathit{Safe}' \vee \mathit{Reach}')$ is valid, we say that $\mathcal{G}$ is a *restriction* of $\mathcal{G}'$. Note that for every (reachability or safety player) strategy σ in $\mathcal{G}$ there is a strategy σ^ν that is valid in both $\mathcal{G}$ and $\mathcal{G}'$ and produces the same consistent plays in $\mathcal{G}$ as σ. Such a σ^ν can be constructed easily by taking any strategy σ' that is valid

in $\mathcal{G}'$ and then playing according to σ in all states $s \in S$ such that $(\mathit{Safe} \vee \mathit{Reach})(s)$ is satisfiable and playing according to σ' if not.

Following the argument of the Borel determinancy theorem [27], reachability games are determined: Either REACH wins a game $\mathcal{G}$ or SAFE wins $\mathcal{G}$. This determinancy carries over to single states $s \in S$, i.e., either REACH wins from s or SAFE wins from s.

For some $\varphi \in \mathcal{L}(\mathcal{V})$ and game $\mathcal{G} = \langle \mathcal{V}, \mathit{Init}, \mathit{Safe}, \mathit{Reach}, \mathit{Goal} \rangle$, 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:

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

Further, we define two operations Pre and Post that, for some transition predicate T, abstract the variables of set $\mathcal{V}'$ or $\mathcal{V}$, respectively, and that produce the *pre-condition*, respectively *post-condition*:

$$\text{Pre}(T) = \exists s \in S.\, T(\mathcal{V}, s) \quad \text{and} \quad \text{Post}(T) = \exists s \in S.\, T(s).$$

Based on our general definition of logically represented two-player games with reachability objectives, we now consider several classes of games found in the literature and discuss approaches to computing the winner for these games.

2.2.1 *Boolean Reachability Games*

As a first concrete class of reachability games, we consider a case that has a naturally bounded domain and is particularly pervasive in the literature: *Boolean reachability games*, i.e., reachability games ranging over Boolean variables [1, 6]. These games are well-suited to model synthesis problems within finite domains and allow for the use of particularly efficient formula representations such as binary decision diagrams (BDDs) [10].

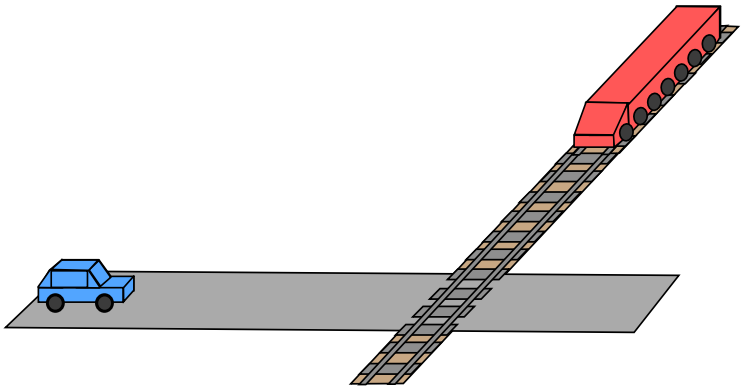

Figure 2.1: A simple train crossing. We want to synthesize a controller for the car such that no crash occurs.

In our notation, $\mathcal{G} = \langle \mathcal{V}, \mathit{Init}, \mathit{Safe}, \mathit{Reach}, \mathit{Goal} \rangle$ is a Boolean reachability game when $\mathcal{D}(\mathcal{V}) = \mathbb{B}$ and the logic $\mathcal{L}$ for describing the game structure is Boolean algebra.

EXAMPLE 2.1. Consider the train crossing depicted in Figure 2.1. We would like to construct a controller for the blue car that avoids a crash. The behavior of the red train, however, cannot be influenced by the controller.

We can model the problem of finding a safe controller as a Boolean reachability game $\mathcal{G} = \langle \mathcal{V}, \mathit{Init}, \mathit{Safe}, \mathit{Reach}, \mathit{Goal} \rangle$, with:

- $\mathcal{V} = \{c, t, \mathbf{r}\}$;
- $\mathit{Init} \equiv c \wedge t \wedge \neg \mathbf{r}$;
- $\mathit{Safe} \equiv (t' \leftrightarrow t) \wedge \neg \mathbf{r} \wedge \mathbf{r}'$;
- $\mathit{Reach} \equiv (c' \leftrightarrow c) \wedge \mathbf{r} \wedge \neg \mathbf{r}'$;
- $\mathit{Goal} \equiv \neg c \wedge \neg t$.

A possible play of $\mathcal{G}$ is

$$\rho = (\text{true}, \text{true}, \text{false}) \to (\text{true}, \text{false}, \text{true}) \to (\text{true}, \text{false}, \text{false}) \to \dots,$$

where we have that a tuple $(\text{true}, \text{true}, \text{true}) = (c, t, \mathbf{r})$ denotes the values of the variables in a state.

In $\mathcal{G}$, SAFE fully controls the variable c, which models whether the car is in the crossing, and REACH controls t, which models the behavior of the train. Here, a variable evaluates to true if the associated vehicle is not in the crossing.

A winning strategy σ_S for $\mathcal{G}$ describes a safe controller for the described problem. It is easy to see that the simple policy of never setting c to false qualifies.

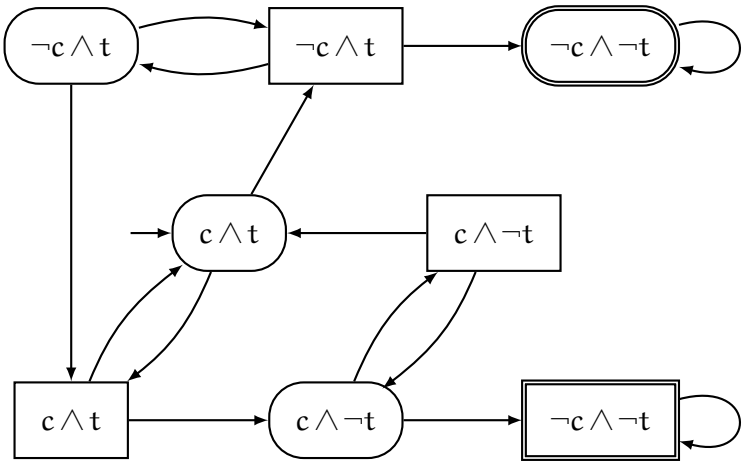

Figure 2.2: Graphical representation of the Boolean reachability game $\mathcal{G}$ described in Example 2.1.

Graphical Notation. Figure 2.2 shows a graphical representation of the Boolean reachability game $\mathcal{G} = \langle \mathcal{V}, \mathit{Init}, \mathit{Safe}, \mathit{Reach}, \mathit{Goal} \rangle$ described in Example 2.1. Each node corresponds to a state of $\mathcal{G}$ and is labeled with a formula characterizing the evaluation of the variables c and t. The arrows represent the transitions characterized by *Safe* and *Reach*. Nodes with rounded corners are states owned by SAFE, while nodes with sharp corners are states owned by REACH. States that satisfy *Goal* are doubly lined. States that satify *Init* have an incoming transition without a source node. We will use a similar graphical notation for all games, but may not depict the full state space in the case of infinite-state games.

The defining feature of Boolean reachability games is their naturally finite game structure, which allows for complete algorithms based on a backwards fixed point computation, commonly called *attractor construction* [37]. The attractor construction is based on repeatedly applying the *controlled predecessor* operation to a set of states winning for REACH, starting with the goal states of the game under consideration, until a fixed point is reached.

In our symbolic game description, we define the controlled predecessor operator $\mathrm{CPre}(\varphi, \mathcal{G})$ for some $\varphi \in \mathcal{L}(\mathcal{V}')$ in the reachability game $\mathcal{G} = \langle \mathcal{V}, \mathit{Init}, \mathit{Safe}, \mathit{Reach}, \mathit{Goal} \rangle$ as:

$$\begin{aligned}\mathrm{CPre}(\varphi, \mathcal{G}) = &(\exists s \in S.\, \mathit{Reach}(\mathcal{V}, s) \wedge \varphi(s)) \\ &\vee (\mathrm{Pre}(\mathit{Safe}) \wedge \forall s \in S.\, \mathit{Safe}(\mathcal{V}, s) \implies \varphi(s)).\end{aligned}$$

The attractor construction then builds a sequence of sets of states until further application of the controlled predecessor operator does not add any new states. In our description, this can be described by the following formulas for some $\mathcal{G} = \langle \mathcal{V}, \mathit{Init}, \mathit{Safe}, \mathit{Reach}, \mathit{Goal} \rangle$:

$$\begin{aligned}\mathit{Attr}_0 &= \mathit{Goal}; \\ \mathit{Attr}_{n+1} &= \mathit{Attr}_n \vee \mathrm{CPre}(\mathit{Attr}_n, \mathcal{G}).\end{aligned}$$

In the spirit of a fixed point computation, the attractor construction continuously computes a new Attr_{n+1} until $\mathit{Attr}_{n+1} \implies \mathit{Attr}_n$ holds. Then, Attr_{n+1} characterizes exactly the set of states winning for the reachability player in $\mathcal{G}$.

2.2.2 *Linear Reachability Games*

As a second class of reachability games, we consider *linear reachability games* [14], which are similar to other infinite-state games considered in the literature [4]. For linear reachability games, the domains of the variables $\mathcal{V}$ may additionally range over either the integers or reals.

We say $\mathcal{G} = \langle \mathcal{V}, \mathit{Init}, \mathit{Safe}, \mathit{Reach}, \mathit{Goal} \rangle$ is a *linear integer reachability game* or a linear reachability game *over* $\mathbb{Z}$ when $\mathcal{D}(\mathcal{V}) = \mathbb{Z} \cup \mathbb{B}$ and the logic $\mathcal{L}$ for describing the game structure is linear integer arithmetic.

Similarly to the above, we say $\mathcal{G} = \langle \mathcal{V}, \mathit{Init}, \mathit{Safe}, \mathit{Reach}, \mathit{Goal} \rangle$ is a *linear real reachability game* or a linear reachability game *over* $\mathbb{R}$ when $\mathcal{D}(\mathcal{V}) = \mathbb{R} \cup \mathbb{B}$ and the logic $\mathcal{L}$ for describing the game structure is linear real arithmetic.

Linear reachability games an interesting formalism for modeling problems that arise in program synthesis, nondeterministic planning and reactive synthesis. This can be demonstrated by revisiting Example 2.1. Clearly, we would like to model the train crossing in much more detail than is possible with Boolean variables. The following example shows that with linear arithmetic games, we can additionally consider the speed and distance of the two vehicles.

EXAMPLE 2.2. We can model the problem of finding a safe controller for the car (see Figure 2.1) as a linear real reachability game $\mathcal{G} = \langle \mathcal{V}, \mathit{Init}, \mathit{Safe}, \mathit{Reach}, \mathit{Goal} \rangle$, with:

- $\mathcal{V} = \{c, t, \mathbf{r}\}$, where $c, t \in \mathbb{R}$;
- $\mathit{Init} \equiv c = 10 \wedge t = 50 \wedge \neg \mathbf{r}$;
- $\mathit{Safe} \equiv c' \leqslant c \wedge t' = t \wedge \neg \mathbf{r} \wedge \mathbf{r}'$;
- $\mathit{Reach} \equiv t' \leqslant t \wedge c' = c \wedge \mathbf{r} \wedge \neg \mathbf{r}'$;
- $\mathit{Goal} \equiv c = 0 \wedge t = 0$.

A possible play of $\mathcal{G}$ is

$$\rho = (10, 50, \text{false}) \to (8.51, 50, \text{true}) \to (8.51, 35.2, \text{false}) \to \dots.$$

As can be seen with this example, linear reachability games allow to model the dynamics of more complex problems that arise when synthesizing cyber-physical systems or programs. However, the infinite game structure induced by the infinite domains of the variables makes the problem of determining the winner of a linear reachability game generally undecidable, as shown by the following proposition.

The general result is widely known, but seldom proved in the literature [14]. We therefore present our own proof based on a reduction from the halting problem of a Minsky machine.

PROPOSITION 2.2. Computing the winner of an arbitrary linear reachability game $\mathcal{G} = \langle \mathcal{V}, \mathit{Init}, \mathit{Safe}, \mathit{Reach}, \mathit{Goal} \rangle$ is undecidable.

Proof. We show that we can reduce the halting problem of an arbitrary two-counter Minsky machine $\mathcal{M} = \langle \{x, y\}, \mathcal{P} \rangle$, which is undecidable [30], to determining the winner of $\mathcal{G} = \langle \mathcal{V}, \mathit{Init}, \mathit{Safe}, \mathit{Reach}, \mathit{Goal} \rangle$. Let x, y be the two counters of the Minsky machine and $\mathcal{P}$ a numbered list of instructions. W.l.o.g. we assume the instruction labels i from $\mathcal{P}$ range over $1 \dots |\mathcal{P}|$. We encode the halting problem in $\mathcal{G}$ as follows:

- $\mathcal{V} = \{pc, halt, x, y, \mathbf{r}\}$, where x, y are variables for the counters, *halt* is set to 1 when the machine halts and *pc* is a program counter, we have $\mathcal{D}(v) = \mathbb{Z}$ for all $v \in \mathcal{V} \setminus \{\mathbf{r}\}$;
- $Init \equiv pc = 1 \wedge halt = 0 \wedge x = 0 \wedge y = 0$;
- *Safe* is the identity relation: $Safe \equiv \neg\mathbf{r} \wedge \mathbf{r}' \wedge \bigwedge_{v \in \mathcal{V} \setminus \{\mathbf{r}\}} (v = v')$;
- *Reach* fully encodes the computations of the Minsky machine, we encode each transition $\mathtt{i} \in \mathcal{P}$ as follows:
  - $\mathtt{i} : \mathtt{INC}(\mathtt{c}, \mathtt{j})$: the instruction increments the counter $\mathtt{c}$ and jumps to $\mathtt{j}$:
  $$pc = \mathtt{i} \wedge pc' = \mathtt{j} \wedge \mathtt{c}' = \mathtt{c} + 1 \wedge \bigwedge_{v \in \{x, y, halt\}} (v = v'),$$
  - $\mathtt{i} : \mathtt{JZDEC}(\mathtt{c}, \mathtt{j}, \mathtt{k})$: the instruction tests whether the counter $\mathtt{c}$ equals 0. If yes, it jumps to $\mathtt{j}$; otherwise it decrements $\mathtt{c}$ and jumps to $\mathtt{k}$:
  $$pc = \mathtt{i} \wedge \Big(\mathtt{c} = 0 \wedge pc' = \mathtt{j} \wedge \bigwedge_{v \in \mathcal{V} \setminus \{pc\}} (v = v') \vee \mathtt{c} \neq 0 \wedge pc' = \mathtt{k} \wedge \mathtt{c}' = \mathtt{c} - 1 \wedge \bigwedge_{v \in \mathcal{V} \setminus \{pc, \mathtt{c}\}} (v = v')\Big),$$
  - $\mathtt{i} : \mathtt{HALT}$: the instruction halts the machine:
  $$pc = \mathtt{i} \wedge halt' = 1 \wedge \bigwedge_{v \in \mathcal{V} \setminus \{halt\}} (v = v'),$$

  Reach is constructed as a disjunction D of all encoded instructions with the added constraint for $\mathbf{r}$: $Reach \equiv D \wedge \mathbf{r} \wedge \neg\mathbf{r}'$;
- *Goal* characterizes that the machine has halted: $Goal \equiv halt = 1$.

It is easy to see that REACH wins $\mathcal{G}$ if and only if $\mathcal{M}$ halts on instruction set $\mathcal{P}$. Hence, we reduced the problem of termination of $\mathcal{M}$ to solving the linear integer reachability game $\mathcal{G}$. Note that the construction works irrespective of whether the variables $\mathcal{V} \setminus \{\mathbf{r}\}$ range over $\mathbb{Z}$ or $\mathbb{R}$. Thus, computing the winner of both linear integer and real reachability games is generally undecidable. □

While this result is far from reassuring in terms of decidability, recent advances have shown that many practically relevant games on infinite graphs can in fact be efficiently solved by dedicated semi-algorithms [4, 14]. This observation motivates our approach and focus on linear reachability games. Ideally, one has a number of methods that work particularly well in different scenarios, in this way covering a large part of the whole class of linear reachability games.

3

SUBGOALS

Our causality-based algorithm for solving reachability games aims to capture the cause-effect-relationships present in every play that is won by the reachability player. Therefore, we analyze sets of transitions that can cause some observable effect in these plays. We call transition predicates for characterizing these causally necessary sets of transitions *subgoals*.

The intuition behind this is that for every observable effect, there might be several concrete transitions that can produce said effect, that is, move the game from a configuration where it is not present to a configuration where the effect is present. Because taking at least one concrete transition from such a subgoal is causally necessary for the global effect of reaching the goal states, subgoals should be seen as essential milestones for the reachability player (just as the goal states are the goal of the reachability player). They can, however, also be utilized to prove the existence of winning safety player strategies, as we discuss in Section 3.2.

In the next section, we introduce the notion of *enforceable transitions*. The idea of these is similar to that of the classic controlled predecessor discussed in Chapter 2: determining from which configurations of the game `REACH` can enforce a desirable and predefined outcome. Later in this chapter, we introduce two important kinds of subgoals: *necessary* and *sufficient* subgoals. We end the chapter by demonstrating how subgoals can be utilized to partition and solve reachability games.

3.1 ENFORCEABLE TRANSITIONS

One central aspect of solving two-player games is computing whether `REACH` can enforce a desirable outcome from a set of states, for instance, computing the set of states from which they can force the play into *Goal* in one turn as realized by the controlled predecessor operator described in Chapter 2.

For our approach, focusing on transitions over states carries over to this computation and we define a different operation, which we call the *enforceable transitions* operator.

DEFINITION 3.1 (Enforceable transitions). The set of *enforceable transitions* relative to a transition predicate $T \in \mathcal{L}(\mathcal{V} \cup \mathcal{V}')$ is defined as

$$\mathrm{Enf}(T, \mathcal{G}) = (\mathit{Safe} \vee \mathit{Reach}) \wedge T \wedge \neg\exists s \in S.\, \big(\mathit{Safe}(\mathcal{V}, s) \wedge \neg T(\mathcal{V}, s)\big)$$

For some transition predicate T and a game $\mathcal{G}$, we say that $\mathrm{Enf}(T, \mathcal{G})$ characterizes the *enforceable subset of* T.

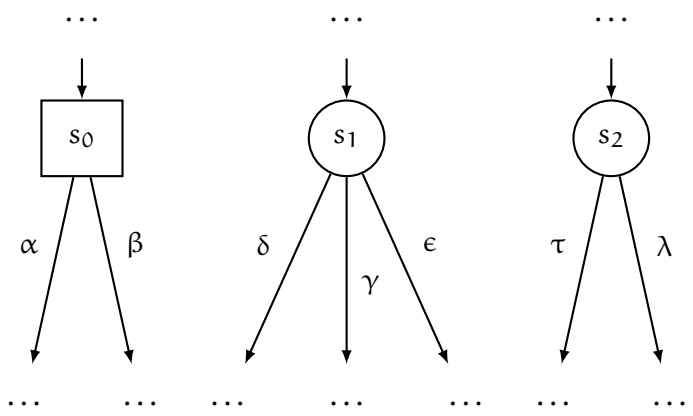

Figure 3.1: Visualization of several possible scenarios of some transition predicates as described in Example 3.1, leading to different enforceable subsets. Recall that the state with sharp corners, s_0, is owned by REACH and the states with round corners, s_1 and s_2, are owned by SAFE.

Intuitively, the definition of this operator states that to qualify as a enforceable transition, a concrete transition has to be a valid transition in $\mathcal{G}$, must be from the set described by T and either start in a state owned by REACH or in a state s owned by SAFE such that there is no outgoing transition from s that does not satisfy T.

The function of this operator is the same as that of the controlled predecessors operator $\mathrm{CPre}(\varphi, \mathcal{G})$, which is commonly used for the attractor construction, as presented in Section 2. Both operations are meant to resolve the partial control of the two players REACH and SAFE by identifying from which part of the game structure REACH can enforce some outcome.

In contrast to $\mathrm{CPre}(\varphi, \mathcal{G})$, we do not consider a state predicate φ but a transition predicate T. This lets us restrict the analysis to some predefined set of transitions and check from the predecessor states of which transitions *taking a transition from the set* can be enforced by REACH. These include all transitions in T that start in a state $s \in S_{\mathsf{REACH}}$, i.e., s is owned by REACH, and additionally transitions in T that start in a state $s' \in S_{\mathsf{SAFE}}$, i.e., s' is owned by SAFE, such that all other transitions from s' also satisfy T.

EXAMPLE 3.1. Consider the (partial) game structure $\mathcal{G}$ as depicted in Figure 3.1. It visualizes the different scenarios that may lead to different enforceable subsets $\mathrm{Enf}(T, \mathcal{G})$ of some transition predicate T. Let us first consider T_1 characterizing the set of transitions $\{\alpha, \delta\}$. We have that $\mathrm{Enf}(T_1, \mathcal{G})$ characterizes the set $\{\alpha\}$, because α starts in s_0, which is owned by REACH, so the alternative transition β does not change whether REACH can ensure taking a transition form T_1 from $\mathrm{Pre}(\mathrm{Enf}(T_1, \mathcal{G}))$. On the other hand, δ starts in s_1, which is owned

by SAFE and has the alternative transitions γ and ϵ, which are not characterized by T_1. This means REACH cannot enforce that a transition from T_1 is taken. It follows that δ is not in the enforceable subset of T_1.

If we instead consider T_2 characterizing $\{\alpha, \tau, \lambda\}$, the situation is different. This is because even though s_2, the source state of transitions τ and λ is owned by SAFE, there is no legal alternative transition satisfying $\neg T_2$ from s_2. Therefore, there in particular is none that satisfies $\mathit{Safe} \wedge \neg T_2$. Hence, the enforceable subset characterized by $\mathrm{Enf}(T_2, \mathcal{G})$ is also $\{\alpha, \tau, \lambda\}$.

In order to formally state the similarities between the controlled predecessor $\mathrm{CPre}(\varphi, \mathcal{G})$ and the enforceable transitions $\mathrm{Enf}(T, \mathcal{G})$, we show that we can use the latter for characterizing a set of predecessor states such that REACH can enforce that the play moves to a state satisfying φ next.

LEMMA 3.2. 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. Any state $s \in S_{\texttt{REACH}}$ that satisfies $\mathrm{Pre}(\mathrm{Enf}(T, \mathcal{G}))$ is winning for REACH. This is because we have $\mathrm{Enf}(T, \mathcal{G}) \implies T$ and, therefore, $T(s)$ is satisfiable. Hence, there is some $s' \in S$ such that $T(s, s')$ holds. Then, we know that $\mathrm{Post}(T)[\mathcal{V}'/\mathcal{V}](s')$ is valid, so there is a winning reachability player strategy σ'_R from s'. It follows that REACH wins from s with the strategy of playing to s' and then playing according to σ'_R.

Any state $s \in S_{\texttt{SAFE}}$ that satisfies $\mathrm{Pre}(\mathrm{Enf}(T, \mathcal{G}))$, we know that s has a transition that satisfies T, so s is not a trap state. We also know that there is no $s' \in S$ such that $\mathit{Safe}(s, s') \wedge \neg T(s, s')$ holds. This means there is no transition starting in s that does not end in $\mathrm{Post}(T)$. Since we know that REACH wins from every state s' satisfying $\mathrm{Post}(T)[\mathcal{V}'/\mathcal{V}]$, it is clear that SAFE has no strategy that does not play into a losing state. Hence, REACH wins from s. □

This result effectively means that if we have some reachability game $\mathcal{G} = \langle \mathcal{V}, \mathit{Init}, \mathit{Safe}, \mathit{Reach}, \mathit{Goal} \rangle$ and a state predicate φ such that REACH wins from every $s \in S$ with $\varphi(s)$, then the formulas $\mathrm{Pre}(\mathrm{Enf}((\mathit{Safe} \vee \mathit{Reach}) \wedge \varphi[\mathcal{V}/\mathcal{V}'], \mathcal{G}))$ and $\mathrm{CPre}(\varphi, \mathcal{G})$ describe the same set of states. Consequently, we could use $\mathrm{Pre}(\mathrm{Enf}((\mathit{Safe} \vee \mathit{Reach}) \wedge \mathit{Attr}_n[\mathcal{V}/\mathcal{V}'], \mathcal{G})))$ to replace $\mathrm{CPre}(\mathit{Attr}_n, \mathcal{G})$ in the computation of the attractor Attr_{n+1} as outlined in Section 2.

Our recursive algorithm presented later on in this thesis splits the game into two subgames and as a result receives two subgoals which get combined into a larger one for the global game. This means we will often have two transition predicates F and T such that each of their enforceable subsets is empty in a subgame produced by restricting the transition relation of the global game $\mathcal{G}$. We would like to

make a statement about whether the subset is also empty if we consider the disjunction $F \vee T$ in the global game $\mathcal{G}$, so that we can construct the global subgoal with this reuqired property.

A general and strong statement about this is, however, difficult. This is because, clearly, the transition that witnesses $\neg F$, i.e., is the "alternative" for some transition in F, might be from T. In the case where this is the only alternative, there is a non-empty enforceable subset in $F \vee T$, meaning we cannot generally state that empty enforceable subsets of two transitions predicates carry over to the disjunction of the two predicates.

We fix this by strictly requiring the alternative transitions in safety player states that may possibly cause the non-enforceability of a concrete transition from F to not satisfy T. In order to do this, we assume that T is not valid in the subgame where F has an empty enforceable subset. Hence, the alternative transitions for F cannot be from T and remain as valid alternative transitions for the disjunction $F \vee T$. Note that, of course, F and T can be instantiated arbitrarily, so the symmetric case is automatically covered as well by the following lemma.

LEMMA 3.3. Let F, T be transition predicates and $\mathcal{G}$, $\mathcal{G}_F, \mathcal{G}_T$ be reachability games such that $\mathrm{Unsat}(\mathrm{Enf}(F, \mathcal{G}_F))$ and $\mathrm{Unsat}(\mathrm{Enf}(T, \mathcal{G}_T))$ both evaluate to true, and we have

1. $F \implies (\mathit{Safe}_F \vee \mathit{Reach}_F) \implies (\mathit{Safe} \vee \mathit{Reach})$;
2. $T \implies (\mathit{Safe}_T \vee \mathit{Reach}_T) \implies (\mathit{Safe} \vee \mathit{Reach})$;
3. $\mathrm{Unsat}((\mathit{Safe}_F \vee \mathit{Reach}_F) \wedge T)$.

Then it follows that $\mathrm{Unsat}(\mathrm{Enf}(F \vee T, \mathcal{G}))$ is also true.

Proof. Before treating the above claim, we show: (*) Under the conditions of the lemma, we have that if there are $s, v \in S$ such that $(\mathit{Safe} \wedge F)(s, v)$ is satisfiable, then there is some $u \in S$ such that $(\mathit{Safe} \wedge \neg F \wedge \neg T))(s, u)$ is satisfiable. For this, consider that $s \in S_{\mathsf{SAFE}}$ and with $\mathrm{Unsat}(\mathrm{Enf}(F, \mathcal{G}_F))$ we know there is some $u \in S$ such that $(\mathit{Safe}_F \wedge \neg F)(s, u)$. Since $\mathrm{Unsat}((\mathit{Safe}_F \vee \mathit{Reach}_F) \wedge T)$, we can further infer that $(\mathit{Safe}_F \wedge \neg F \wedge \neg T)(s, u)$. With $(\mathit{Safe}_F \vee \mathit{Reach}_F) \implies (\mathit{Safe} \vee \mathit{Reach})$, we can conclude that $(\mathit{Safe} \wedge \neg F \wedge \neg T))(s, u)$ is satisfiable.

We now show the original claim of the lemma by contradiction. Suppose there were $s, v \in S$ such that $\mathrm{Enf}(F \vee T, \mathcal{G})(s, v)$ holds. It follows that $(F \vee T)(s, v)$ must hold. We can immediately conclude that $s \in S_{\mathsf{SAFE}}$, because if not we have $\mathrm{Sat}(\mathrm{Enf}(F, \mathcal{G}_F))$ or $\mathrm{Sat}(\mathrm{Enf}(T, \mathcal{G}_T))$, a contradiction with our assumption that both $\mathrm{Unsat}(\mathrm{Enf}(F, \mathcal{G}_F))$ and $\mathrm{Unsat}(\mathrm{Enf}(T, \mathcal{G}_T))$ evaluate to true.

From $F \implies (\mathit{Safe}_F \vee \mathit{Reach}_F)$ and $\mathrm{Unsat}((\mathit{Safe}_F \vee \mathit{Reach}_F) \wedge T)$ we can infer that $\mathrm{Unsat}(F \wedge T)$. It follows that either (1) $F(s, v)$ or (2) $T(s, v)$ holds, but both cannot be true at the same time.

Consider case (1). With (*) we can infer that there is some $u \in S$ such that $(\mathit{Safe} \wedge \neg F \wedge \neg T))(s, u)$ holds, which is in contradiction with $\mathrm{Enf}(F \vee T, \mathcal{G})(s, v)$.

Consider case (2). Since $\mathrm{Unsat}(\mathrm{Enf}(T, \mathcal{G}_T))$, we know there is some $u \in S$ such that $(\mathit{Safe}_T \wedge \neg T)(s, u)$.
With $(\mathit{Safe}_T \vee \mathit{Reach}_T) \implies (\mathit{Safe} \vee \mathit{Reach})$, it follows that $(\mathit{Safe} \wedge \neg T)(s, u)$ holds. Now, consider either: $F(s, u)$ or $\neg F(s, u)$. In the latter case, we immediately have a contradiction with $\mathrm{Enf}(F \vee T, \mathcal{G})(s, v)$. In the former case, we can apply (*) again to get the same contradiction as in case (1). □

As shown in Lemma 3.3, $F \vee T$ has an empty enforceable subset in particular if F and T have an empty enforceable subset in games $\mathcal{G}_F$ and $\mathcal{G}_T$ that are more or equally constrained (with respect to the transition relations of both players) than the global game $\mathcal{G}$ (Conditions 1 and 2). $\mathcal{G}_F$ and $\mathcal{G}_T$ can be seen as describing partial game structures of the original game $\mathcal{G}$. Intuitively, this is because there can only be *more* possible alternative transitions satisfying $\neg F$ and $\neg T$ in the less or equally constrained game $\mathcal{G}$. Of course, this result holds only as long as T describes no valid transitions in $\mathcal{G}_F$ (Condition 3).

EXAMPLE 3.2. Let us again consider the partial game structure $\mathcal{G}$ depicted in Figure 3.1, this time with two transition predicates T_3 and T_4, characterizing the sets $\{\delta\}$ and $\{\gamma, \epsilon\}$, respectively. Observe that we indeed have that $\mathrm{Unsat}(\mathrm{Enf}(T_3, \mathcal{G}))$ and $\mathrm{Unsat}(\mathrm{Enf}(T_4, \mathcal{G}))$ both evaluate to true. However, if we consider $T_3 \vee T_4$, which characterizes the set $\{\delta, \gamma, \epsilon\}$, we have that $\mathrm{Sat}(\mathrm{Enf}(T_3 \vee T_4, \mathcal{G}))$ evaluates to true. This result matches with Lemma 3.3, because T_3 and T_4 each describe valid transitions in $\mathcal{G}$, which do not satisfy Condition 3, so we cannot conclude $\mathrm{Unsat}(\mathrm{Enf}(T_3 \vee T_4, \mathcal{G}))$ with the lemma.

However, consider T_5 and T_6, characterizing the sets $\{\delta\}$ and $\{\tau\}$, respectively. We again have for both transition predicates that the enforceable subsets, which are characterized by $\mathrm{Enf}(T_5, \mathcal{G})$ and $\mathrm{Enf}(T_6, \mathcal{G})$, are empty. But this time we can define two partial game structures $\mathcal{G}_5$, consisting only of s_1 and successor states of δ, γ and ϵ on one hand, and $\mathcal{G}_6$, consisting of s_3 and successor states of τ and λ on the other. Note that we still have $\mathrm{Unsat}(\mathrm{Enf}(T_5, \mathcal{G}_5))$ and $\mathrm{Unsat}(\mathrm{Enf}(T_6, \mathcal{G}_6))$. But this time, we are allowed to apply Lemma 3.3 to infer that we have $\mathrm{Unsat}(\mathrm{Enf}(T_5 \vee T_6, \mathcal{G}))$ evaluating to true. This is because T_5 is not valid in $\mathcal{G}_6$. This ensures that the alternative transition for T_5 cannot satisfy T_6, leaving at least this one alternative transition even if the transition predicates shared predecessor states (which they do not, in this case).

In this section, we considered the enforceable transitions operator. Given a transition predicate, this operator characterizes the subset of transitions such that SAFE has no alternatives available in their pre-

decessor states. Intuitively, this means that REACH can enforce these transitions from their predecessor states.

The transition predicates on which we will apply this operation are *subgoals*, i.e., causally necessary sets of transitions which we introduce in more detail in the following section.

3.2 NECESSARY SUBGOALS

Next, we present a formal definition of *subgoals*. They denote sets of transitions and are meant to express causal dependencies between the transitions on some play from *Init* to *Goal*. This means that if we can observe some *effect* on every play winning for REACH, there is a number of possible *causes* that may be (partially) responsible for this effect and, in particular, it is strictly *necessary* that one transition from this set of possible causes appears in the play. We capture this notion of necessity in the following definition.

DEFINITION 3.4 (Necessary subgoal). A *necessary subgoal* $C \in \mathcal{L}(\mathcal{V} \cup \mathcal{V}')$ in $\mathcal{G}$ is a transition predicate such that for every play $\rho = s_0 s_1 \ldots$ of $\mathcal{G}$, if there exists a $n \in \mathbb{N}$ such that $\mathit{Goal}(s_n)$ is valid, then there exists some $k < n$ such that $C(s_k, s_{k+1})$ is valid.

Per definition, REACH has to reach a necessary subgoal C to have a shot at winning a play ρ. Note that this does not mean, however, that reaching C guarantees a win for REACH, because the definition makes no statement about which player has a winning strategy from some arbitrary state in $\mathrm{Post}(C)$. This gets clearer if one considers that for some game $\mathcal{G} = \langle \mathcal{V}, \mathit{Init}, \mathit{Safe}, \mathit{Reach}, \mathit{Goal} \rangle$, the whole transition relation $(\mathit{Safe} \vee \mathit{Reach})$ trivially qualifies as a necessary subgoal, completely irrespective of who actually wins the game. In a way, the necessary sugboal C is not *sufficient* to guarantee a win for REACH (and we introduce sufficient subgoals for this reason in the next section). On the other hand, *avoiding* a necessary subgoal C *does* indeed guarantee a win for the safety player. An even stronger statement holds, as the following Lemma 3.5 states: There is an avoidable necessary subgoal for every winning safety player strategy and vice versa.

LEMMA 3.5. A safety strategy σ_S is winning in $\mathcal{G}$ if and only if there exists a necessary subgoal C in $\mathcal{G}$ 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. " $\Longrightarrow$ ". We show that all transitions entering the goal states constitute a necessary subgoal avoided by any winning safety player strategy. Consider the transition predicate $\mathit{Goal}[\mathcal{V}/\mathcal{V}']$, which describes all transitions with successor states satisfying *Goal*. This predicate clearly qualifies as a necessary subgoal: For every play $\rho = s_0 s_1 \ldots$ such that there is some $n \in \mathbb{N}$ with $\mathit{Goal}(s_n)$ we have that $\neg \mathit{Goal}(s_0)$ is valid (or else SAFE would not win in $\mathcal{G}$) and, therefore, $0 < n$ and

also $Goal[\mathcal{V}/\mathcal{V}'](s_{n-1}, s_n)$.
"$\Longleftarrow$". Let C be a necessary subgoal such that no play consistent with σ_S contains a transition of C. Then by Definition 3.4 no play consistent with σ_S reaches a state satisfying *Goal*. Hence σ_S is a winning strategy for SAFE. □

Of course, the question remains how to compute non-trivial necessary subgoals. Indeed, using *Goal* as outlined in the proof above provides no further benefit over a simple backwards exploration as realized by the attractor construction (see Section 2). Another candidate, the full set of transitions $(Safe \vee Reach)$ shows equally little promise as one would not expect the safety player to be able avoid this subgoal.

Ideally, a necessary subgoal describes some interesting key decision than can be used to focus the strategy search. Therefore, we turn to causality to compute interesting necessary subgoals. 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 a concrete transition that bridges the interpolant *causes* this effect.

PROPOSITION 3.6. 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 a Craig interpolant for *Init* and *Goal*, 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 $0 \leqslant i < n$ 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. □

EXAMPLE 3.3. To illustrate this discussion on necessary subgoals, we turn to the Boolean reachability game $\mathcal{G} = \langle \mathcal{V}, Init, Safe, Reach, Goal \rangle$ described in Example 2.1. The game models a train crossing, and the goal states describe a situation where a crash between the train t and the car c occurs. Clearly, it is causally necessary for the *effect*, e.g., the crash, that the train has entered the crossing. In this game, we modeled the train via variable t and the state predicate $\neg t$ means the train is in the crossing. Hence, it is causally necessary for some transition from t to $\neg t$ to occur on any play winning for REACH. Note that, indeed, $\neg t$ is a valid Craig interpolant for *Init* and *Goal*. Hence, $C_1 = \text{Instantiate}(\neg t, \mathcal{G})$ is a necessary subgoal in $\mathcal{G}$ following Proposition 3.6. Figure 3.2 illustrates C_1 and shows how the state space is

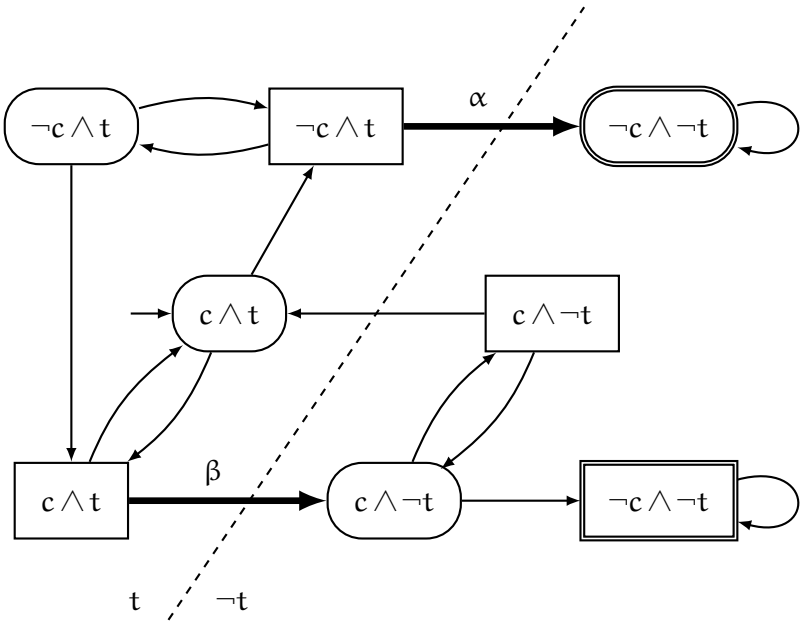

Figure 3.2: The necessary subgoal $C_1 = \text{Instantiate}(\neg t, \mathcal{G})$, in bold and labeled with α and β, for the Boolean reachability game described in Example 2.1.

partitioned by the interpolant $\neg t$, indicated by the dashed line. The transitions characterized by C_1 are drawn bold and labeled with α and β. Note that every play reaching *Goal* has to move through C_1, but REACH does not win from moving through the subgoal alone. This is because taking the transition labeled with β ends in a state where SAFE has a winning strategy.

Lemma 3.5 ensures that there is some winning safety strategy if a necessary subgoal is avoidable, but it makes no statement about how that strategy looks like. In fact, SAFE may be able to avoid the subgoal by some key decision long before ever reaching the predecessor states of the subgoal. Therefore, Lemma 3.5 is generally not useful enough for computing actual strategies, or even as a simple criterion for determining the winner of a game.

The following lemma strengthens the statement of Lemma 3.5 and shows that a necessary subgoal C directly yields a strategy for SAFE in $\mathcal{G}$, if the subgoal is, in a certain sense, *locally avoidable* by SAFE, that is, if $\text{Unsat}(\text{Enf}(C, \mathcal{G}))$ holds.

LEMMA 3.7. Let C be a necessary subgoal for $\mathcal{G}$ and suppose that $\text{Unsat}(\text{Enf}(C, \mathcal{G}))$ holds. Then, there is a winning strategy σ_S for SAFE in $\mathcal{G}$.

Proof. Consider a function σ_S such that for all $s \in S_{\text{SAFE}}$ such that $C(s)$ is satisfiable, we have $\sigma_S(\omega s) = s'$ with $\neg C(s, s')$ for all $\omega \in S^*$. For

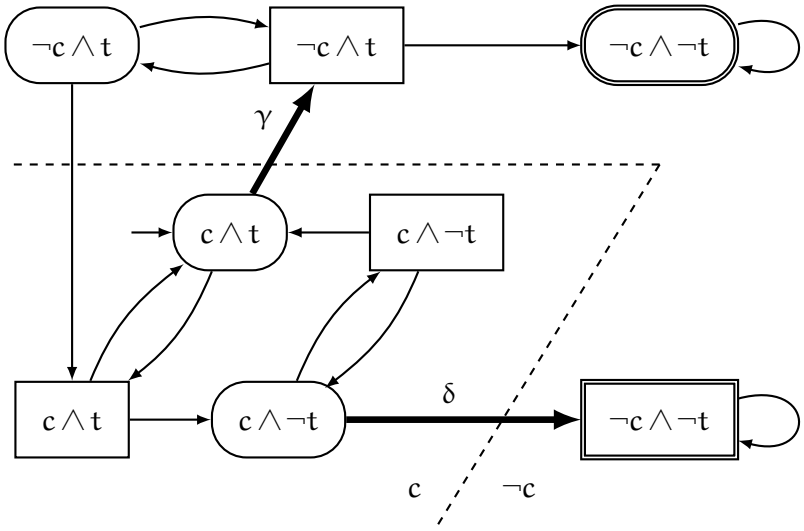

Figure 3.3: The necessary subgoal $C_2 = \text{Instantiate}(\neg c, \mathcal{G})$, in bold and labeled with γ and δ, for the Boolean reachability game described in Example 2.1.

all $s \in S_{\mathsf{SAFE}}$ such that $C(s)$ is unsatisfiable, we require $\sigma_S(\omega s) = s'$ with $\mathit{Safe}(s, s')$ for all $\omega \in S^*$ or that s is a trap state. We know the function σ_S describes a strategy in $\mathcal{G}$ because $\text{Unsat}(\text{Enf}(C, \mathcal{G}))$ means for all states s satisfying $\text{Pre}(C)$: (1) we have $s \in S_{\mathsf{SAFE}}$ and (2) there is some $s' \in S$ such that $\mathit{Safe} \wedge \neg C(s, s')$ holds (s cannot be a trap state because it satisfies $\text{Pre}(C)$).

Now, consider some play $\rho = s_0 s_1 \ldots$ consistent with σ_S. By the construction above, we know that for any $k \in \mathbb{N}$ such that $\text{Pre}(C)(s_k)$ holds that $\neg C(s_k, s_{k+1})$. Hence, there is no $n \in \mathbb{N}$ such that $C(s_n, s_{n+1})$ holds. With Lemma 3.6, we can immediately conclude that σ_S is winning in $\mathcal{G}$. □

EXAMPLE 3.4. Note that the necessary subgoal $C_1 = \text{Instantiate}(\neg t, \mathcal{G})$ as described in Example 3.3 does *not* qualify for the local strategy as given by Lemma 3.7. This is because the enforceable subset of C_1 is not empty. In fact, we have $\text{Enf}(C_1, \mathcal{G}) \equiv C_1$, as both predecessor states of C_1 are owned by REACH.

On the other hand, let us consider the necessary subgoal $C_2 = \text{Instantiate}(\neg c, \mathcal{G})$ as illustrated in Figure 3.3. Here, C_2 is drawn bold and labeled with γ and δ. Craig interpolants are not unique and $\neg c$ is also a valid Craig interpolant for *Init* and *Goal*. Therefore, Proposition 3.6 can again be used to show that C_2 is a necessary subgoal in $\mathcal{G}$. For C_2, we have that $\text{Unsat}(\text{Enf}(C_2, \mathcal{G}))$ evaluates to true. Intuitively, this is because both transitions γ and δ that constitute the subgoal

start in states owned by SAFE, and from each state there is a legal alternative transition that does not satisfy C_2. Consequently, a safety player strategy σ_S that avoids C_2 locally by taking these alternatives whenever possible never passes the subgoal C_2. This means any play consistent with σ_S never leaves the region delimited by the dashed line. It is easy to see that any such play is winning for SAFE.

3.3 SUFFICIENT SUBGOALS

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. As an illustration of this discussion, consider again the game and necessary subgoal shown in Figure 3.2 and described in more detail in Example 3.3.

For the purpose of describing a set of transitions that is guaranteed to be winning for the reachability player, we introduce *sufficient subgoals*.

DEFINITION 3.8 (Sufficient subgoal). We call a transition predicate $F \in \mathcal{L}(\mathcal{V} \cup \mathcal{V}')$ a *sufficient subgoal* if REACH wins from every state satisfying $\mathrm{Post}(F)$.

If the set of transitions characterized by the necessary subgoal C, that leads 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.

EXAMPLE 3.5. Let us first again consider the necessary subgoal C_1 obtained as a instantiation of the Craig interpolant $\neg t$, as depicted in Figure 3.2. As mentioned before, REACH does not win after taking transition β. Hence, the largest sufficient subgoal F_1 included in C_1 only characterizes transition α, as α ends in a goal state.

On the other hand, if we consider the necessary subgoal C_2 obtained as a instantiation of the Craig interpolant $\neg c$, as depicted in Figure 3.3, we observe that the largest sufficient subgoal F_2 included in F_2 characterizes both transitions γ and δ, as REACH has a winning strategy from both successor states.

Note that a sufficient subgoal is by no means tied to some Craig interpolant like the necessary subgoals C_1 and C_2 might suggest. This can be observed by considering the subgoal F depicted in Figure 3.4. F is drawn bold and labeled with δ. It only consists of a single transition, but qualifies as sufficient solely because δ ends in a goal state.

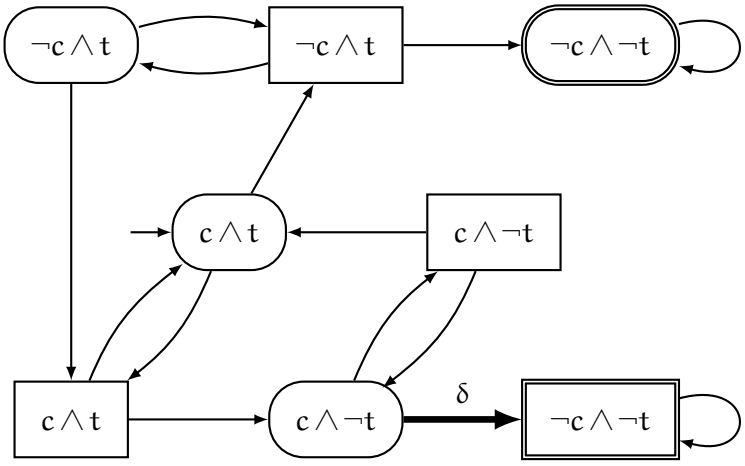

Figure 3.4: A sufficient subgoal F, in bold and labeled with δ, for the Boolean reachability game described in Example 2.1.

However, there is no valid Craig interpolant for *Init* and *Goal* which *only* describes δ, because the transition only considers one of the goal states and there are winning plays that reach the other goal state without ever seeing δ.

If we have a necessary subgoal C and F is the largest sufficient subgoal included in C, any winning strategy of REACH has to either play a transition in F or be able to force SAFE to play a transition in F. This is because any play to the goal states has to see C, but a transition through C is winning for REACH if and only if it is in the largest sufficient subgoal F included in C. We use this observation to partition the game along C into a game from F to the goal states and one from the initial states to F. The following proposition formalizes that if we know for some game $\mathcal{G}$ a necessary subgoal C and the largest sufficient subgoal included in F, we can solve $\mathcal{G}$ by solving the game from the initial states to F.

PROPOSITION 3.9. 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

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

Proof. "$\Longrightarrow$". Let σ_R be a winning strategy of REACH from s in $\mathcal{G}$. We show the claim of the proposition by contradiction. For this, assume that SAFE wins from s in $\mathcal{G}_{\text{pre}}$. Since $\mathcal{G}_{\text{pre}}$ is a restriction of $\mathcal{G}$, there is a strategy σ_S that is valid in $\mathcal{G}$ and $\mathcal{G}_{\text{pre}}$ and winning for SAFE from s in $\mathcal{G}_{\text{pre}}$. Let $\rho = s_0 s_1 \ldots$ be the play in $\mathcal{G}$ consistent with both σ_R and σ_S.

Because σ_R is winning for REACH from s in $\mathcal{G}$ and $\mathit{Init}(s)$ holds, we know there exists a $n \in \mathbb{N}$ such that $\mathit{Goal}(s_n)$ holds. C is a necessary subgoal in $\mathcal{G}$, so there is some $m \in \mathbb{N}$ such that $C(s_m, s_{m+1})$ evaluates to true. Let m be the smallest such index.

We make a case distinction between (1) $F(s_m, s_{m+1})$ holds and $\neg F(s_m, s_{m+1})$ holds, i.e., we consider whether the play moves through the largest sufficient subgoal included in C or not.

Case (1): So we have that $F(s_m, s_{m+1})$ is valid. If s_m was a safety player trap state in $\mathcal{G}_{\text{pre}}$, we would have $\text{Pre}(\text{Enf}(F, \mathcal{G}))(s_m)$, a contradiction with σ_S winning in $\mathcal{G}_{\text{pre}}$. Therefore, we know that $(\mathit{Reach} \wedge F)(s_m, s_{m+1})$ must hold: This is because σ_S is a valid strategy in $\mathcal{G}_{\text{pre}}$, F is not a valid transition predicate in $\mathcal{G}_{\text{pre}}$ and s_m is not a trap state in $\mathcal{G}_{\text{pre}}$. Since m was chosen as the smallest index that moves through C and F characterizes a subset of C, we know there is no index before that would satisfy as a transition in F. This means that $s_0 \dots s_m$ is a valid sequence of moves in $\mathcal{G}_{\text{pre}}$ and since σ_S is winning in $\mathcal{G}_{\text{pre}}$, no goal state of $\mathcal{G}_{\text{pre}}$ is encountered in this sequence. Hence, we have $\neg \text{Pre}(\text{Enf}(F, \mathcal{G}))(s_i)$ for all $0 \leqslant i \leqslant m$. However, from $(\mathit{Reach} \wedge F)(s_m, s_{m+1})$ it directly follows that $\text{Pre}(\text{Enf}(F, \mathcal{G}))(s_m)$ holds, which is a contradiction to the assumption that σ_S is a winning strategy for SAFE from s in $\mathcal{G}_{\text{pre}}$.

Case (2): If the play does not move through the largest sufficient subgoal included in C, we have that $(\neg F)(s_m, s_{m+1})$ is valid. Consider state s_{m+1}. Because σ_R is a winning strategy for REACH and ρ is consistent with σ_R, we know every that state in the play must be winning for REACH in $\mathcal{G}$. However, this would mean that F does not characterize the largest sufficient subgoal included in C, because there is the sufficient subgoal $F' = F \vee \left(\bigwedge_{v \in \mathcal{V}} v = s_m(v) \wedge v' = s_{m+1}(v)\right)$. We have that $F \implies F'$ but $F' \not\Longrightarrow \mathsf{F}$, which means that F is not the largest sufficient subgoal included in C, a contradiction.

"$\Longleftarrow$". Since F is a sufficient subgoal, we know that every state satisfying $\text{Post}(F)$ is winning for REACH by definition. With Lemma 3.2 we can infer that every state s' satisfying $\text{Pre}(\text{Enf}(F, \mathcal{G}))$ is also winning for REACH. We know that REACH wins in $\mathcal{G}_{\text{pre}}$, so there is some s such that $\mathit{Init}(s)$ evaluates to true and a strategy σ_R such that every play starting in s and consistent with σ_R is winning for REACH in $\mathcal{G}_{\text{pre}}$, i.e., the play reaches some state s' satisfying $\text{Pre}(\text{Enf}(F, \mathcal{G}))$. We know that s' is winning for REACH, so there is a strategy σ'_R such that any play starting in s' reaches *Goal*. Playing according to σ_R until seeing some s' that satisfies $\text{Pre}(\text{Enf}(F, \mathcal{G}))$ and then playing according to σ'_R yields a winning strategy σ'' for REACH from s in $\mathcal{G}$. □

Proposition 3.9 hints at how our causality-based game solving algorithm, which we treat in the following chapter, determines the winner of some reachability game $\mathcal{G} = \langle \mathcal{V}, \mathit{Init}, \mathit{Safe}, \mathit{Reach}, \mathit{Goal} \rangle$. We compute a necessary subgoal C using a Craig interpolant between *Init* and *Goal*, and then compute the largest sufficient subgoal F included in C. This then allows us to solve the global game by solving $\mathcal{G}_{\text{pre}}$, i.e., the game from the global initial states to the sufficient subgoal F.

4

CAUSALITY-BASED GAME SOLVING

In this chapter, we describe our algorithms which utilize necessary and sufficient subgoals to decompose and solve two-player reachability games (Algorithms 1 and 2). In the first section, we discuss an approximate technique we use to compute sufficient subgoals included in a necessary subgoal. Then, we provide an intuitive explanation and the pseudo-code of the algorithms in Section 4.2. After that, we proceed by illustrating an example execution of the algorithms in Section 4.3. The algorithms are necessarily incomplete for linear reachability games, as solving them is an undecidable problem. However, they are partially correct and return a correct result whenever terminating. Section 4.4 discusses partial correctness, while Section 4.5 analyzes the termination properties of the algorithms, in particular highlighting a class of games where termination is guaranteed.

4.1 APPROXIMATING SUFFICIENCY

The high-level idea of our recursive algorithm (Algorithm 1) for solving a reachability game $\mathcal{G} = \langle \mathcal{V}, \mathit{Init}, \mathit{Safe}, \mathit{Reach}, \mathit{Goal} \rangle$ is as follows: We use a Craig interpolant for *Init* and *Goal* to instantiate a necessary subgoal C as shown by Proposition 3.6. We then check whether C already yields a safety player strategy by locally avoiding C, following Lemma 3.7. If this check does not succeed, we proceed to partially solve the game from the successor states of C to the goal, in order to compute the largest sufficient subgoal F included in C. This we can then use to solve the global game as outlined in Proposition 3.9.

This approach effectively slices the game into a *pre-game* and a *post-game* along the necessary subgoal C, but there is an important catch: As mentioned before, it is easy to compute necessary subgoals as they consider all paths through the game structure regardless of actual strategies. Computing the largest sufficient subgoal included in C, however, requires a solution of the post-game.

Naively applying our algorithm to the post-game would require us to solve a game that is possibly as large as the original game. We therefore use the observation that if `REACH` can pass the necessary subgoal C before reaching *Goal*, they eventually have to be able to do so *for the last time*. If the necessary subgoal C is the instantiation of some Craig interpolant φ, this effectively means that `REACH` has to be able to win the following *restricted* post-game:

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

Its initial states are characterized by the post-condition of the necessary subgoal C, that is, describe a situation after just moving through the subgoal. In order to guarantee that $\mathcal{G}_{\text{post}}$ is easier to solve than $\mathcal{G}$, we restrict the transitions in $\mathcal{G}_{\text{post}}$ to the one starting in states satisfying the interpolant φ associated with subgoal C. This effectively makes every state satisfying $\neg\varphi$ a trap state by removing all outgoing transitions. In particular, this restriction is guaranteed to remove at least one concrete transition from $\mathcal{G}$, which we will show in Section 4.5.

The restriction makes it easier for SAFE to win the post-game, as all plays ending in these new trap states without seeing *Goal* before are immediately winning for SAFE in $\mathcal{G}_{\text{post}}$. However, it is harder to win in $\mathcal{G}_{\text{post}}$ for REACH, as the states turned trap states in $\mathcal{G}_{\text{post}}$ may actually be winning for REACH in $\mathcal{G}$. Following this discussion, we can conclude that every state determined as winning for REACH in $\mathcal{G}_{\text{post}}$ is also winning for REACH in $\mathcal{G}$. We utilize this observation in our algorithms and can make the following formal statement.

LEMMA 4.1. If REACH wins from s in $\mathcal{G}_{\text{post}}$, they also win from s in $\mathcal{G}$.

Proof. Since $\mathcal{G}_{\text{post}}$ is a restriction of $\mathcal{G}$, we have that there is a strategy σ_R that is valid in both $\mathcal{G}$ and $\mathcal{G}_{\text{post}}$ and winning for REACH from s in $\mathcal{G}_{\text{post}}$. We show that σ_R is also winning for REACH from s in $\mathcal{G}$ by contradiction. Let $\rho = s_0 s_1 \ldots$ with $s_0 = s$ be a play consistent with σ_R in $\mathcal{G}$ but winning for SAFE who plays according to the winning strategy σ_S. Hence, we have for all $m \in \mathbb{N}$ that $\neg \mathit{Goal}(s_m)$ is valid.

The play has to move out of $\mathcal{G}_{\text{post}}$ eventually, otherwise the play would be winning for REACH in $\mathcal{G}$ by assumption, leading to a contradiction. Hence, there is some $i \in \mathbb{N}$ such that $\neg\varphi(s_i)$ holds. Let i be the first such index in ρ, i.e., we have for all $k < i$ that $\varphi(s_k)$ holds. Note that s_i has no outgoing transitions in $\mathcal{G}_{\text{post}}$ and is therefore won by SAFE.

Playing according to σ_S is a valid strategy for σ_S in $\mathcal{G}_{\text{post}}$ and the partial play $\rho' = s_0 \ldots s_i$ is consistent with σ_S and σ_R. However, observe that ρ' is, in fact, won by SAFE in $\mathcal{G}_{\text{post}}$ as no goal state is seen on the play and it ends in a trap state. This is a contradiction to σ_R being a winning strategy for REACH in $\mathcal{G}_{\text{post}}$. □

However, there may be successor states of C that are in fact winning for REACH, but do not get classified as winning by the above restricted post-game approximation. We illustrate this with the following example.

EXAMPLE 4.1. Consider the Boolean reachability game $\mathcal{G}$ shown in Figure 4.1, played over variables x, y. Note that while we have only seen example games with strict alternation between the players so far, this is not necessarily required by the general definitions in Chapter 2. We discard alternation to have a more succinct example in this case.

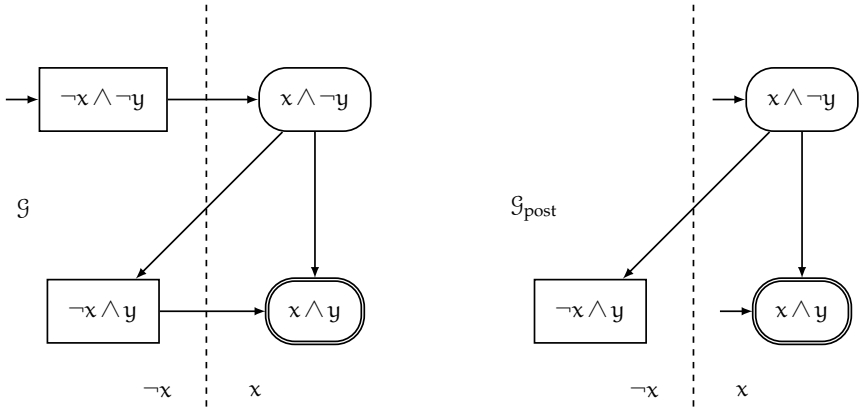

Figure 4.1: The Boolean reachability game $\mathcal{G}$ considered in Example 4.1 with necessary subgoal $C = \text{Instantiate}(x, \mathcal{G})$, and the restricted post-game $\mathcal{G}_{\text{post}}$.

It is easy to see and formally proven with Proposition 3.6 that $C = \text{Instantiate}(x, \mathcal{G})$ is a necessary subgoal in $\mathcal{G}$. The restricted post-game $\mathcal{G}_{\text{post}}$ of $\mathcal{G}$ obtained from this subgoal is shown on the right. From Lemma 4.1, we know that any state winning for REACH in $\mathcal{G}_{\text{post}}$ is also winning for REACH in $\mathcal{G}$. The question we want to answer now is whether the same holds for SAFE.

Consider the state labeled with $x \wedge y$ is winning for REACH in $\mathcal{G}_{\text{post}}$, hence we can conclude with Lemma 4.1 that it is also winning in $\mathcal{G}$.

However, consider now the state labeled with $x \wedge \neg y$. It is a state owned by SAFE that has a transition that moves out of the region of the state space that satisfies x. In the restricted post-game, such transitions are approximated to be winning for SAFE. Indeed, it is easy to see that taking this transition ends the play in the trap state $\neg x \wedge y$. This yields a winning strategy for SAFE, it follows that $x \wedge \neg y$ is won by SAFE in $\mathcal{G}_{\text{post}}$. But if we now consider the global game $\mathcal{G}$, we quickly see that $x \wedge \neg y$ is, in fact, not winning for SAFE. This is because $\neg x \wedge y$ is not a trap state in $\mathcal{G}$, but owned by REACH with a transition directly into a goal state.

This observation, together with Lemma 4.1, effectively means that when we use the restricted post-game for computing the largest sufficient subgoal included in C, we generally have to assume that it returns an under-approximation of the actual largest included sufficient subgoal.

While this does not seem particularly useful for our recursive algorithm at first, we can show that under certain, easily verifiable conditions the approximation is exact. 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. Intuitively, this predicate states that there is no transition "out of" the restricted

post-game $\mathcal{G}_{\text{post}}$. If this holds, then the restricted post-game $\mathcal{G}_{\text{post}}$ is not really restricted at all, as no concrete outgoing transitions from states satisfying φ are actually removed and both players can play the same transitions as in $\mathcal{G}$. The intuition that this condition results in an exact computation of the largest sufficient subgoal included in C is formalized by the following lemma.

LEMMA 4.2. Assume that φ *perfectly partitions* $\mathcal{G}$, and let C be the instantiation of φ and s be a state satisfying $\text{Post}(C)[\mathcal{V}'/\mathcal{V}]$. Then REACH wins from s in $\mathcal{G}_{\text{post}}$ if and only if REACH wins from s in $\mathcal{G}$.

Proof. "$\Longrightarrow$". This direction directly follows from Lemma 4.1.
"$\Longleftarrow$". We assume that there is a reachability player strategy σ_R that is winning from s in $\mathcal{G}$. We show that REACH wins from s in $\mathcal{G}_{\text{post}}$ by contradiction. Therefore, assume there is a safety player strategy σ_S that is winning from s in $\mathcal{G}_{\text{post}}$. The strategy σ_S is also valid in $\mathcal{G}$ because $\mathcal{G}_{\text{post}}$ is a restriction of $\mathcal{G}$.

Let $\rho = s_0 s_1 \ldots$ with $s_0 = s$ be the unique play in $\mathcal{G}$ consistent with σ_R and σ_S. Since this play is consistent with the reachability player strategy σ_R that is winning in $\mathcal{G}$, there is some $n \in \mathbb{N}$ such that $\mathit{Goal}(s_n)$ is valid.

However, we have that σ_S is winning in $\mathcal{G}_{\text{post}}$. Hence, the play must leave $\mathcal{G}_{\text{post}}$ before the goal states are reached or else we immediately have a contradiction. So we can conclude that there is some $0 \leqslant m < n$ such that $\neg\varphi(s_m)$ holds. Let m and n be the first indices with the above properties. Since $s \in \text{Post}(C)[\mathcal{V}'/\mathcal{V}]$ and $C = \text{Instantiate}(\varphi, \mathcal{G}) = (\mathit{Safe} \vee \mathit{Reach}) \wedge \neg\varphi \wedge \varphi'$, we know that $\varphi(s)$ is valid. This means we can conclude that $0 < m$ and that $(\varphi \wedge \neg\varphi')(s_{m-1}, s_m)$ holds. Since ρ is a valid play in $\mathcal{G}$ and φ perfectly partitions $\mathcal{G}$, this is a contradiction. □

The results discussed in this section mean that we have to compute the largest sufficient subgoal included in some necessary subgoal in different ways depending on the game structure and the chosen interpolant. We will describe this process in more detail in the following section when discussing our algorithm. In particular, we will highlight how the result of the approximation can still be used for further computation in the cases where it is not exact.

4.2 A RECURSIVE ALGORITHM

Our technique solves a reachability game in two steps. First, a recursive algorithm (Algorithm 1) using subgoals computes the winning regions of the players in the initial states. Then Algorithm 2 evaluates the returned predicate from Algorithm 1. We begin this section with a presentation and explanation of these algorithms. An example execution of Algorithm 1 can be found in the next section. We provide a formal proof of correctness in Section 4.4. Algorithm 1 is incomplete

in the sense that it does not return on every reachability game over an infinite graph. We discuss the termination of Algorithm 1 in more detail in Section 4.5 and characterize a class of games for which termination is guaranteed. The algorithm therefore is complete on that class.

Algorithm 2 determines the winner of a reachability game $\mathcal{G} = \langle \mathcal{V}, \mathit{Init}, \mathit{Safe}, \mathit{Reach}, \mathit{Goal} \rangle$. Recall that by definition from Chapter 2, $\mathcal{G}$ is won by REACH if and only if they win from at least one initial state. To compute the subset of initial states winning for REACH, Algorithm 2 in turn applies Algorithm 1 to $\mathcal{G}$. Algorithm 1 recursively slices $\mathcal{G}$ along necessary subgoals and will be the focus of our following description.

To solve a game $\mathcal{G} = \langle \mathcal{V}, \mathit{Init}, \mathit{Safe}, \mathit{Reach}, \mathit{Goal} \rangle$, we compute (R, T) using Algorithm 1:

- R is a state predicate that characterizes the largest subset of the initial states *Init* that are winning for REACH in $\mathcal{G}$;
- T is a transition predicate s.t.
  1. T is a necessary subgoal in the game $\mathcal{G}^{\setminus R} = \langle \mathcal{V}, \mathit{Init} \wedge \neg R, \mathit{Safe}, \mathit{Reach}, \mathit{Goal} \rangle$, where we only consider initial states not winning for REACH and therefore winning for SAFE and
  2. T is locally avoidable in $\mathcal{G}$, i.e., we have $\mathrm{Unsat}(\mathrm{Enf}(T, \mathcal{G}))$.

Using Lemma 3.7 and the local avoidability of T, this necessary subgoal can be seen as a witness for a safety player strategy showing that the initial states not characterized by R are indeed winning for SAFE. It also ensures that R does, in fact, describe the largest subset of *Init* winning for REACH.

The computation of R and T proceeds as follows. Initially, we can directly classify states satisfying $\mathit{Init} \wedge \mathit{Goal}$ as winning for REACH and they are added to the set characterized by R (line 2). The algorithm proceeds with the remaining initial states $I = \mathit{Init} \wedge \neg \mathit{Goal}$ (line 3).

If there is no initial state that is not a goal state, i.e., the check $\mathrm{Unsat}(I)$ succeeds, we have of course directly classified all initial states and can return R immediately (line 5). In this case, the set of initial states in $\mathcal{G}^{\setminus R}$ is empty, which means there are no valid plays that could start in a concrete initial state. Hence, with Definition 3.4 any transition predicate would qualify as a necessary subgoal in $\mathcal{G}^{\setminus R}$, but to ensure $\mathrm{Unsat}(\mathrm{Enf}(T, \mathcal{G}))$, we return $T = \texttt{false}$.

In case there are remaining initial states in I that we need to classify, i.e., $\mathrm{Unsat}(I)$ is false, we compute a Craig interpolant φ between I and *Goal* (line 6) and use its instantiation as a necessary subgoal C (line 7).

After these initial base cases are checked, our algorithm proceeds to analyze the necessary subgoal C and partition the game along C into a post- and pre-game. There are four distinct cases that can occur

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

In : reachability game $\mathcal{G} = \langle \mathcal{V}, \mathit{Init}, \mathit{Safe}, \mathit{Reach}, \mathit{Goal} \rangle$
Out : (R,T) s.t. R characterizes the largest subset of *Init* won by `REACH`, Unsat(Enf($T, \mathcal{G}$)) is true and T is a necessary subgoal in $\mathcal{G}^{\setminus R} = \langle \mathit{Init} \wedge \neg R, \mathit{Safe}, \mathit{Reach}, \mathit{Goal} \rangle$.

begin
  $R \leftarrow \mathit{Init} \wedge \mathit{Goal}$
  $I \leftarrow \mathit{Init} \wedge \neg \mathit{Goal}$
  if Unsat(I) **then**
    return R, `false`
  $\varphi \leftarrow$ Interpolate(I, Goal)
  $C \leftarrow$ Instantiate($\varphi, \mathcal{G}$)
  if Unsat(Enf($C, \mathcal{G}$)) **then**
    return R, C
  $\mathcal{G}_{\text{post}} \leftarrow \langle \text{Post}(C)[\mathcal{V}'/\mathcal{V}], \mathit{Safe} \wedge \varphi, \mathit{Reach} \wedge \varphi, \mathit{Goal} \rangle$
  $R_{\text{post}}, T_{\text{post}} \leftarrow \text{Reach}\left(\mathcal{G}_{\text{post}}\right)$
  $F \leftarrow C \wedge R_{\text{post}}[\mathcal{V}/\mathcal{V}']$
  if Unsat(Enf($F, \mathcal{G}$)) **then**
    return $R, F \vee T_{\text{post}}$
  if Sat($(\mathit{Reach} \vee \mathit{Safe}) \wedge \varphi \wedge \neg\varphi' \wedge \neg \mathit{Goal}$) **then**
    $F \leftarrow F \vee \mathit{Goal}[\mathcal{V}/\mathcal{V}']$
    $T_{\text{post}}, \varphi \leftarrow$ `false`, `false`
  $\mathcal{G}_{\text{pre}} \leftarrow \langle I, \mathit{Safe} \wedge \neg F, \mathit{Reach} \wedge \neg F, \text{Pre}(\text{Enf}(F, \mathcal{G})) \rangle$
  $R_{\text{pre}}, T_{\text{pre}} \leftarrow \text{Reach}\left(\mathcal{G}_{\text{pre}}\right)$
  return $R \vee R_{\text{pre}}$, $(\neg\varphi \wedge T_{\text{pre}}) \vee T_{\text{post}} \vee (F \wedge \neg \text{Enf}(F, \mathcal{G}))$

Algorithm 2 : Solve($\mathcal{G}$)

In : reachability game $\mathcal{G} = \langle \mathcal{V}, \mathit{Init}, \mathit{Safe}, \mathit{Reach}, \mathit{Goal} \rangle$
Out : winning player $p \in \{$`REACH`, `SAFE`$\}$

begin
  $R, T \leftarrow$ Reach($\mathcal{G}$)
  if Sat(R) **then**
    return 'REACH'
  else
    return 'SAFE'

based on the properties of φ, C and $\mathcal{G}$. They correspond to the main conditional cases and associated return statements in our algorithm and we proceed to explain the four cases separately in the following description.

Case 1: *SAFE can locally avoid the necessary subgoal C (line 9).*

Recall that if the check $\text{Unsat}(\text{Enf}(C, \mathcal{G}))$ in line 8 succeeds, then SAFE can locally avoid the necessary subgoal C. This effectively means that no transition satisfying C starts in a reachability player state and in the predecessor states owned by SAFE, there is an alternative transitions available not characterized by C (see also Section 3.1). Following Lemma 3.7, we can directly infer that SAFE wins from all states statisfying I (lines 9) by picking the alternative transition whenever at the "edge" of the subgoal C. Note that Case 1 trivially applies when the instantiation of φ describes an empty set.

If Case 1 does not apply, the general idea going forward is to compute the largest sufficient subgoal F that is included in C. We do this by constructing the restricted post-game

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

in line 10. Recursively solving this game (line 11) yields a state predicate R_{post} characterizing a set of initial states in $\mathcal{G}_{\text{post}}$ winning for REACH and a locally avoidable necessary subgoal T_{post} in $\mathcal{G}_{\text{post}}$ from its initial states without R_{post}.

We know that all transitions satisfying C and ending in a state satisfying R_{post} are part of the sufficient subgoal F (line 12), but following the discussion in Section 4.1, we have to assume this F to be an under-approximation. Hence, we will have to proceed differently depending on whether the game structure is perfectly partitioned by φ or not. However, before checking for perfect partitioning and applying Proposition 3.9, we can exploit another special case.

Case 2: *The approximated sufficient subgoal F is locally avoidable (line 13).*

If checking $\text{Unsat}(\text{Enf}(F, \mathcal{G}))$ in line 13 returns true, we get a particularly interesting scenario. Clearly, if $F = C \wedge R_{\text{post}}[\mathcal{V}/\mathcal{V}']$ (line 12) was the largest sufficient subgoal included in C and locally avoidable, SAFE would trivially win the game. This is because every play to *Goal* has to "hit" a transition in the necessary subgoal C and SAFE can locally avoid all transitions of C that are part of F. Hence, at best C is passed through outside of F, into a state in $\text{Post}(C)[\mathcal{V}'/\mathcal{V}] \wedge \neg R_{\text{post}}$, which would be winning for SAFE. However, R_{post} was computed in the restricted post-game, so F generally is an under-approximation of the actual largest sufficient subgoal included in C. Therefore, we do not know whether passing into a state in $\text{Post}(C)[\mathcal{V}'/\mathcal{V}] \wedge \neg R_{\text{post}}$ is in

fact winning for SAFE in $\mathcal{G}$. Remarkably, the argument from above can be salvaged in the case where our approximated F is locally avoidable.

Central to this is the fact that $\mathcal{G}_{\text{post}}$ is an approximation because any move out of $\mathcal{G}_{\text{post}}$ into a state satisfying $\neg\varphi$ is assumed to be winning for SAFE. From such a state, it is necessary to move through the subgoal C again. Intuitively, this means while we do not know whether all states satisfying $\text{Post}(C)[\mathcal{V}'/\mathcal{V}] \wedge \neg R_{\text{post}}$ are winning for SAFE, from every such state they either have a strategy to move to a state satisfying $\neg\varphi$ or do in fact win in $\mathcal{G}_{\text{post}}$. So moving through C at a point that is not the approximated sufficient subgoal F either requires moving through C again at a later point or results in a direct win for SAFE. But moving through C at the approximated sufficient subgoal F can be locally avoided by SAFE. Hence, SAFE has a winning strategy that continuously forces the play to move through C again, but can avoid any move through C that is losing.

The locally avoidable, necessary subgoal for this strategy can be constructed with T_{post}, the locally avoidable necessary subgoal that witnesses the partial safety player strategy for moving back from φ to $\neg\varphi$. Following the discussion above, any path to the goal states that does not move through F has to move through T_{post}. Both are locally avoidable and F is not a valid transition predicate in $\mathcal{G}_{\text{post}}$, because F starts in states satisfying $\neg\varphi$. This allows us to apply Lemma 3.3 to construct the necessary and locally avoidable subgoal $F \vee T_{\text{post}}$.

If there are transitions in F that cannot be locally avoided by SAFE, we need to consider the *pre-game* (line 18):

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

This is the game from the initial states to the largest sufficient subgoal included in C and qualifies for Proposition 3.9. Recursively solving the game in line 19 therefore yields the largest subset of initial states winning in both $\mathcal{G}_{\text{pre}}$ and $\mathcal{G}$.

Recall, however, that we do not yet know whether F is an approximation or not and therefore cannot directly apply the proposition. The algorithm therefore checks whether the interpolant φ perfectly partitions $\mathcal{G}$ (line 15). If it does not, we restore necessary and sufficiency of C and F, respectively, by additionally considering transitions that move into the goal states (lines 16 and 17).

Case 3: *The necessary subgoal C perfectly partitions $\mathcal{G}$ (line 15).*

Recall that φ *perfectly partitions* $\mathcal{G}$ if $(\mathit{Reach} \vee \mathit{Safe}) \wedge \varphi \wedge \neg\varphi' \wedge \neg \mathit{Goal}$ is unsatisfiable, which we check in line 15. If this holds, then there is actually no transition "out of" $\mathcal{G}_{\text{post}}$ and the two players can play in $\mathcal{G}_{\text{post}}$ just like they can play in $\mathcal{G}$. Following Lemma 4.2, we know that in this case R_{post} characterizes the largest set of states satisfying

$\mathrm{Post}(C)[\mathcal{V}'/\mathcal{V}]$ and winning for REACH in $\mathcal{G}$. From this, we can condude that $F = C \wedge R_{\text{post}}[\mathcal{V}/\mathcal{V}']$ is the largest sufficient subgoal included in C. As outlined above, we can then use Proposition 3.9 to solve the global game $\mathcal{G}$ by solving the pre-game $\mathcal{G}_{\text{pre}}$ from the initial states to this largest sufficient subgoal F included in C.

Case 4: *The subgoal does not perfectly partition $\mathcal{G}$ (line 15).*

If φ does not perfectly partition $\mathcal{G}$ we have that $\mathrm{Sat}((\mathit{Reach} \vee \mathit{Safe}) \wedge \varphi \wedge \neg\varphi' \wedge \neg \mathit{Goal})$ in line 15 evaluates to true. In this case, there is a transitions "out of" the subgame $\mathcal{G}_{\text{post}}$ and following the discussion in Example 4.1, this may lead to states winning for SAFE in $\mathcal{G}_{\text{post}}$ but losing for SAFE in $\mathcal{G}$. This is because moving out of the subgame is a winning move for SAFE in $\mathcal{G}_{\text{post}}$, but may lead to a state that is in fact winning for REACH in $\mathcal{G}$.

Hence, we can only assume that R_{post} is an under-approximation of the set of initial states in $\mathcal{G}_{\text{post}}$ which are winning for REACH. This means that while F is still sufficient (following from Lemma 4.1), it is not guaranteed to be the largest sufficient subgoal included in C, which means we cannot directly apply Proposition 3.9.

In order to apply the proposition, we extend F by all transitions that move directly into *Goal* (line 16). This preserves sufficiency of the subgoal, because every transition ending in *Goal* is winning for REACH. Since these transitions trivially appear on any play reaching *Goal*, we also make $C = F$ a necessary subgoal in this way. Taken together, we can then apply Proposition 3.9 to $\mathcal{G}_{\text{pre}}$ (line 18).

Since all successor states of the necessary and sufficient subgoal $(F \vee \mathit{Goal}[\mathcal{V}/\mathcal{V}'])$ are winning for REACH, there is no necessary subgoal from these successor states. We can therefore set the necessary subgoal T_{post} to `false` (line 17). The necessity of the subgoal $(F \vee \mathit{Goal}[\mathcal{V}/\mathcal{V}'])$ is also not directly tied to the interpolant φ anymore. We set it to `false` as well (line 17) to ignore it in further computations.

It remains to construct the locally avoidable necessary subgoal T in both Case 3 and Case 4. Inductively, we have that the same properties as desired for T hold for the returned predicates T_{pre} and T_{post} in the pre-game and the post-game, respectively. We also have the locally avoidable transitions at the edge of the subgoal characterized by $F \wedge \neg\,\mathrm{Enf}(F, \mathcal{G})$. Together, these three locally avoidable transition predicates can be combined to a global necessary subgoal. This is because intuitively, there are three parts of the necessary subgoal C any play to the goal states could pass. If it passes through $\neg R_{\text{post}}$, then it will pass T_{post}. Otherwise, the play passes through F, which can be split into enforceable and non-enforceable subsets. In the former case, the predecessors are the goal states of $\mathcal{G}_{\text{pre}}$, hence the play passes T_{pre}.

If we pick the first time C is passed through, we can even strengthen the last case to $\neg\varphi \wedge T_{\text{pre}}$, because reaching $\varphi \wedge T_{\text{pre}}$ would require

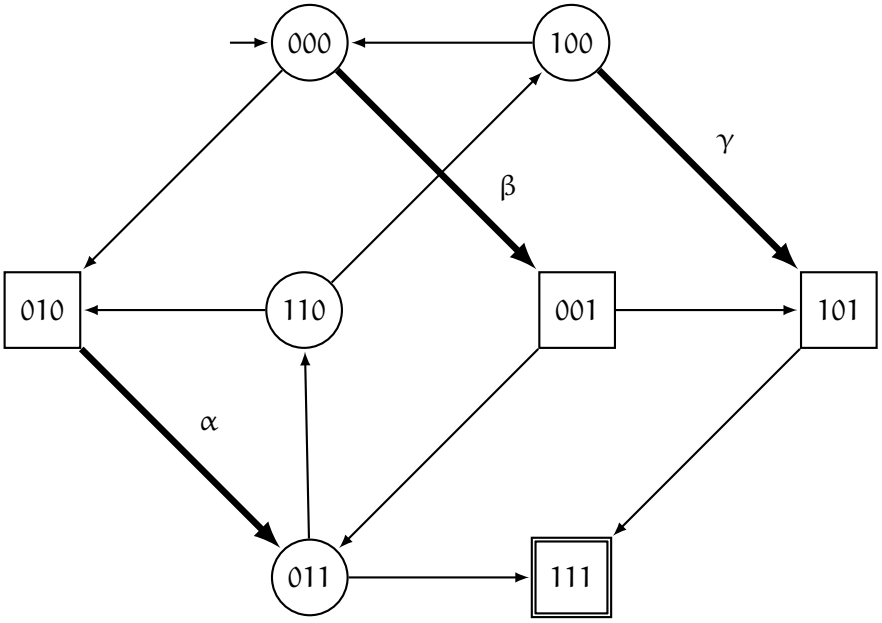

Figure 4.2: Reachability game $\mathcal{G} = \langle \mathcal{V}, \mathit{Init}, \mathit{Safe}, \mathit{Reach}, \mathit{Goal} \rangle$ with $\mathit{Init} \equiv 000$ and $\mathit{Final} \equiv 111$. The necessary subgoal C_1 in $\mathcal{G}$ is highlighted.

passing C at some other point first. This strengthening allows us to apply Lemma 3.3 twice to establish the local avoidability of the necessary subgoal

$$(\neg\varphi \wedge T_{\text{pre}}) \vee T_{\text{post}} \vee (F \wedge \neg \operatorname{Enf}(F, \mathcal{G}))$$

returned in line 20. This follows the observation that F is not valid in $\mathcal{G}_{\text{pre}}$, and both $(\neg\varphi \wedge T_{\text{pre}})$ and F are not valid in $\mathcal{G}_{\text{post}}$.

4.3 EXAMPLE

In this section, we describe an example execution of Algorithm 1 on the Boolean reachability game $\mathcal{G}$ depicted in Figure 4.2. We provide a recursion tree outlining which subgames were constructed in which global game in Figure 4.11 at the end of the section, but the figure can be consulted during reading for further clarity. We deviate from our usual labeling of states with full state predicates to a short-hand notation where the label abc is a stand-in for the state predicate

$$x_0 = a \wedge x_1 = b \wedge x_2 = c.$$

For instance, the initial state which is labeled with 000 is characterized by the formula $\mathit{Init} \equiv x_0 = 0 \wedge x_1 = 0 \wedge x_2 = 0$ and so on.

Let us consider the global call of $\text{Reach}(\mathcal{G})$. Note that the exact execution of the algorithm depends on the chosen Craig inteprolants, which are not unique. For instance, $x_0 = 1 \wedge x_1 = 1$, $x_2 = 1$ and $x_1 = 1$ are all valid interpolants for *Init* and *Goal*. For our example

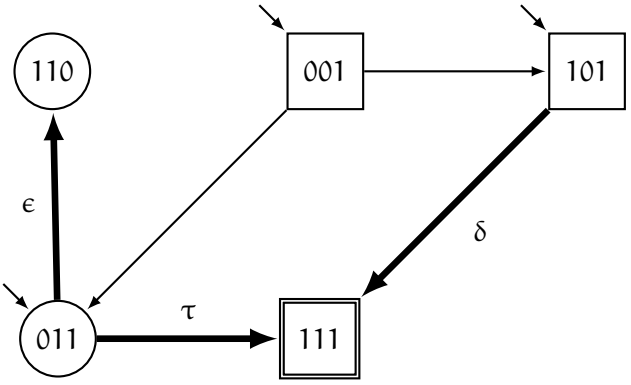

Figure 4.3: The post-game $\mathcal{G}_{\text{post}}$ constructed in the call Reach($\mathcal{G}$) by slicing $\mathcal{G}$ along C_1 associated with interpolant $x_2 = 1$. The necessary subgoal C_2 in $\mathcal{G}_{\text{post}}$ is highlighted.

execution, we assume that the interpolation procedure returns $x_2 = 1$ as the first Craig interpolant.

This yields C_1 as a first necessary subgoal, where

$$C_1 \equiv (\mathit{Safe} \vee \mathit{Reach}) \wedge x_2 \neq 1 \wedge x_2' = 1.$$

C_1 is highlighted in Figure 4.2. It is satisfiable and not locally avoidable by SAFE (because transition α cannot be avoided in state 010 owned by REACH), therefore the computation continues by recursively solving the post-game

$$\mathcal{G}_{\text{post}} = \langle \text{Post}(C_1)[\mathcal{V}'/\mathcal{V}], \mathit{Safe} \wedge x_2 = 1, \mathit{Reach} \wedge x_2 = 1, \mathit{Goal} \rangle$$

starting in the successor states of C_1. Note that $\mathcal{G}_{\text{post}}$ is restricted to transitions starting in the region satisfying $x_2 = 1$. The resulting game structure is illustrated in Figure 4.3. In $\mathcal{G}_{\text{post}}$, we have no initial states that are also goal states and proceed to compute a new necessary subgoal C_2 with the Craig interpolant $x_0 = 1 \wedge x_1 = 1$:

$$C_2 \equiv ((\mathit{Safe} \vee \mathit{Reach}) \wedge x_2 = 1) \wedge \neg(x_0 = 1 \wedge x_1 = 1) \wedge x_0' = 1 \wedge x_1' = 1.$$

SAFE is again unable to locally avoid C_2, because of transition δ. This means we have to solve the post-game $\mathcal{G}'_{\text{post}}$ starting after C_2, depicted in Figure 4.4.

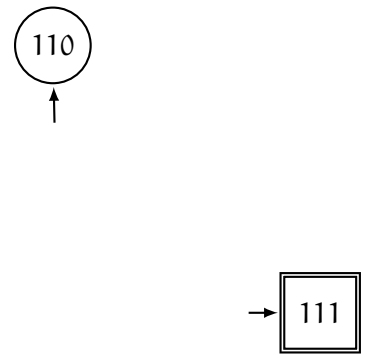

Figure 4.4: The post-game $\mathcal{G}'_{\text{post}}$ constructed in the call Reach($\mathcal{G}_{\text{post}}$) by slicing $\mathcal{G}_{\text{post}}$ along C_2 associated with interpolant $x_0 = 1 \wedge x_1 = 1$.

Game $\mathcal{G}'_{\text{post}}$ is solved quickly, as no transitions can be instantiated in this game. This means Case 1 from Algorithm 1 as described in Section 4.2 applies. It follows that only the initial states that are also goal states are winning for REACH. This is only the state 111.

By solving $\mathcal{G}'_{\text{post}}$, we have computed the largest sufficient subgoal F_2 included in C_2 consisting of both δ and τ (see Figure 4.3). This is because our interpolant $x_0 = 1 \wedge x_1 = 1$ perfectly partitioned the game $\mathcal{G}_{\text{post}}$, which means we can apply Case 3. At this point, we therefore construct the pre-game $\mathcal{G}'_{\text{pre}}$ (with respect to $\mathcal{G}_{\text{post}}$), which is shown in Figure 4.5.

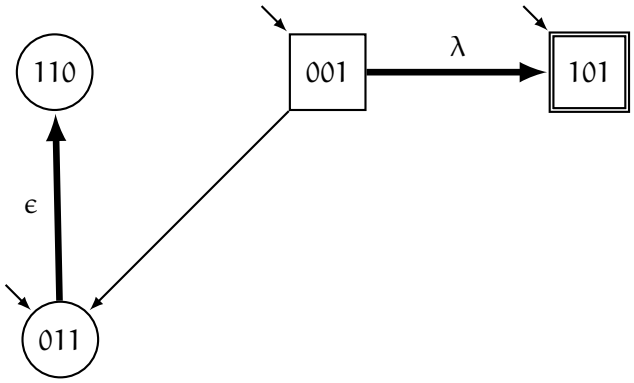

Figure 4.5: The pre-game $\mathcal{G}'_{\text{pre}}$ constructed in the call Reach($\mathcal{G}_{\text{post}}$) by slicing $\mathcal{G}_{\text{post}}$ along C_2. The necessary subgoal C_3 in $\mathcal{G}'_{\text{pre}}$ is highlighted.

The predecessor state of τ, 011 (see Figure 4.3), is not marked as a goal state in $\mathcal{G}'_{\text{pre}}$ because SAFE can avoid this losing transition by playing to 110 (of which we do not yet know whether it is won by SAFE).

A possible interpolant in $\mathcal{G}'_{\text{pre}}$ is $x_0 = 1$. The associated necessary subgoal C_3 is highlighted in Figure 4.5. Slicing along C_3 leads to the recursive solving of the post-game $\mathcal{G}''_{\text{post}}$ shown in Figure 4.6.

This is again a case that is solved quickly, as no transitions can be instantiated and Case 1 applies again. We have that only λ is part of the largest sufficient subgoal F_3 included in C_3.

We proceed to solve the pre-game $\mathcal{G}''_{\text{pre}}$ from the initial states of $\mathcal{G}'_{\text{pre}}$ to the predecessor of λ, which is depicted in Figure 4.7. A possible Craig interpolant from the initial states without goal states to the goal states is $x_1 = 0$. The instantiation of this interpolant, however,

Figure 4.6: The post-game $\mathcal{G}''_{\text{post}}$ constructed in the call Reach($\mathcal{G}'_{\text{pre}}$) by slicing $\mathcal{G}'_{\text{pre}}$ along C_3 associated with interpolant $x_0 = 1$.

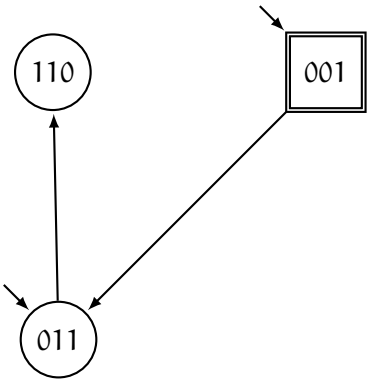

Figure 4.7: The pre-game $\mathcal{G}''_{\text{pre}}$ constructed in the call Reach($\mathcal{G}'_{\text{pre}}$) by slicing $\mathcal{G}'_{\text{pre}}$ along C_3.

describes an empty set. This allows us to apply Case 1 to infer that SAFE wins from state 011 in $\mathcal{G}''_{\text{pre}}$.

We now have determined who wins in $\mathcal{G}'_{\text{pre}}$ from all its initial states: SAFE wins from 011 and REACH from 001 and 101. Since $\mathcal{G}'_{\text{pre}}$ is the pre-game of $\mathcal{G}_{\text{post}}$, the players win from these states in $\mathcal{G}_{\text{post}}$ as well. Recall that we use $\mathcal{G}_{\text{post}}$ to compute the largest sufficient subgoal F_1 included in the necessary subgoal C_1 in $\mathcal{G}$ (see Figure 4.2). F_1 are all transitions in C_1 that end in a state winning for REACH. Hence, the transitions β and γ that end in 001 and 101, respectively, are part of the largest included sufficient subgoal in C_1.

However, in this case, our interpolant $x_2 = 1$ associated with C_1 *does not perfectly partition* $\mathcal{G}$. This effectively means that we do not know whether α should also be part of F_1. Hence, we have to resort

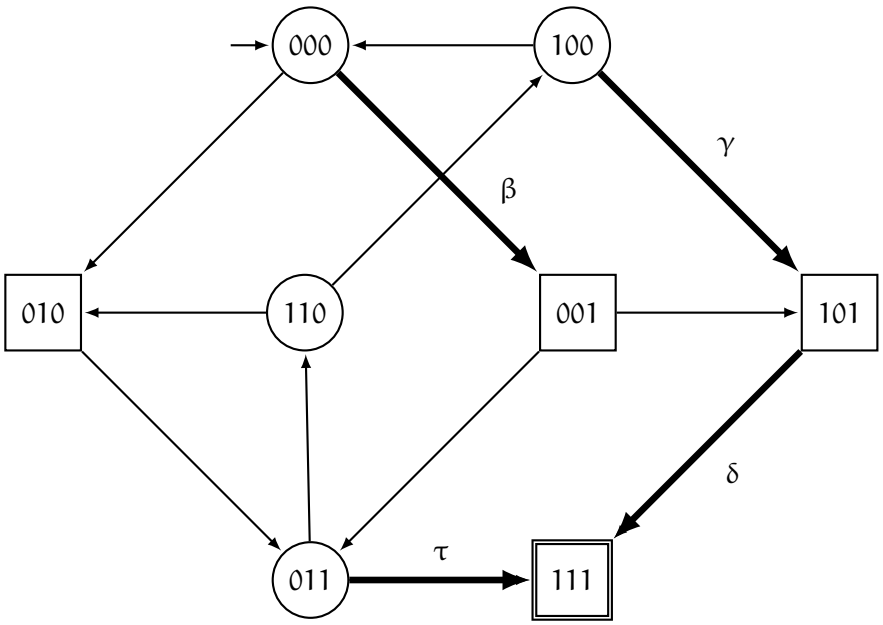

Figure 4.8: The global game $\mathcal{G}$. The necessary subgoal $C_4 \equiv \mathit{Goal}[\mathcal{V}/\mathcal{V}'] \vee F_1$ in $\mathcal{G}$ is highlighted.

to Case 4 of Algorithm 1 for further computation. We therefore proceed to add all transitions ending in the goal state to F_1 to construct a new necessary and sufficient subgoal C_4, which characterizes the set $\{\beta, \gamma, \delta, \tau\}$. This new subgoal is depicted in Figure 4.8.

As before, we now construct the pre-game $\mathcal{G}_{\text{pre}}$, as shown in Figure 4.9, that in this case is played from the initial states of $\mathcal{G}$ to the predecessors of the enforcebale subset of C_4 (recall that C_4 qualifies as sufficient by the above discussion). The predecessor states of β, γ and τ are all owned by SAFE and have an alternative outgoing transition. Hence, they are not marked as goal states in $\mathcal{G}_{\text{pre}}$. The predecessor state of δ, however, is owned by REACH. Following this, δ is the only transition in the enforceable subset of C_4 and 101, the predecessor state of δ is the only goal state in $\mathcal{G}_{\text{pre}}$.

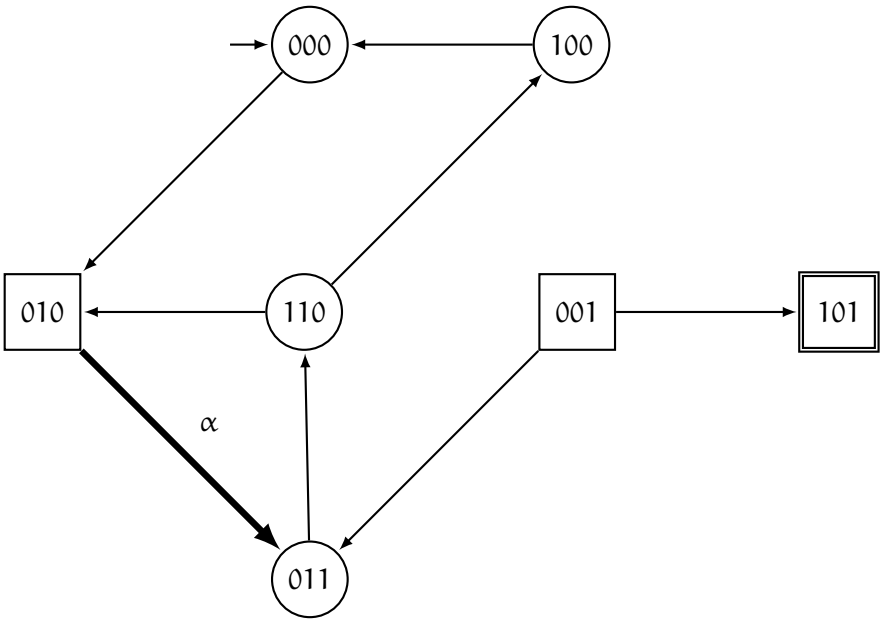

Figure 4.9: The pre-game $\mathcal{G}_{\text{pre}}$ constructed in the call Reach($\mathcal{G}$) by slicing $\mathcal{G}$ along C_4. The necessary subgoal C_5 in $\mathcal{G}_{\text{pre}}$ is highlighted.

Again, a possible interpolant between the initial state and goal state of this game is $x_2 = 1$. The associated necessary subgoal C_5 consisting only of transition α is again not locally avoidable by SAFE, leading to the recursive solving of the post-game $\mathcal{G}'''_{\text{post}}$, shown in Figure 4.10. In $\mathcal{G}'''_{\text{post}}$, we have $x_1 = 0$ as a possible interpolant for the initial state and goal state, which cannot be instantiated. This lets us again apply Case 1 to infer that SAFE wins from 011 in the restricted post-game $\mathcal{G}'''_{\text{post}}$.

Note that again, the restricted post-game $\mathcal{G}'''_{\text{post}}$ simply approximates the transition from 011 to 110 that goes "out of" the subgame to be winning for SAFE. In this case, however, we can still compute the largest sufficient subgoal included in C_5, the necessary subgoal in $\mathcal{G}_{\text{pre}}$, following the argument of Case 2 of Algorithm 1: To reach the

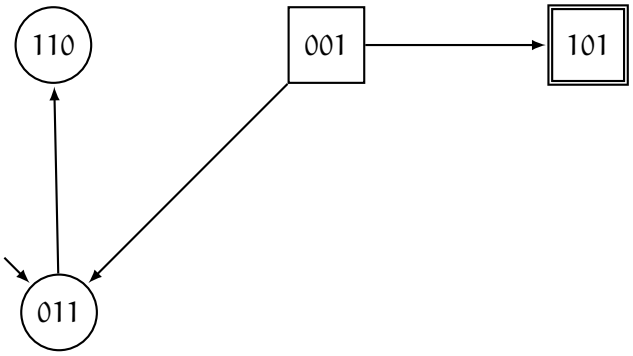

Figure 4.10: The post-game $\mathcal{G}'''_{\text{post}}$ constructed in the call Reach($\mathcal{G}_{\text{pre}}$) by slicing $\mathcal{G}_{\text{pre}}$ along C_5 associated with interpolant $x_2 = 1$.

goal state 101, REACH has to be able move through C_5 for the last time eventually. However, if SAFE plays out of the restricted post-game, REACH has to move through C_5 again, since it is a necessary subgoal. Since there is no transition in C_5 after which SAFE cannot force REACH to move through C_5 again, SAFE has a strategy in $\mathcal{G}_{\text{pre}}$ (see Figure 4.9) to win from 011 by enforcing this loop through subgoal C_5.

We now know that no initial states of $\mathcal{G}_{\text{pre}}$ are winning for REACH. Since $\mathcal{G}_{\text{pre}}$ is the pre-game of $\mathcal{G}$, we also know that no initial states of $\mathcal{G}$ are winning for REACH. This concludes our example execution of Algorithm 1 on the game $\mathcal{G}$. Figure 4.11 shows the recursion tree of this execution, highlighting further which pre-games and post-games were constructed in which global games.

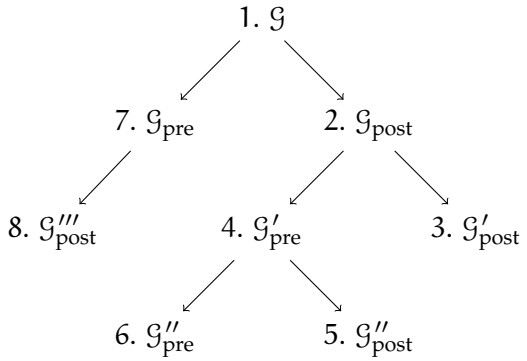

Figure 4.11: The recursion tree for the execution of Reach($\mathcal{G}$).

Remark 4.3. This example was constructed to highlight an execution where every case of the algorithm is encountered and does not show the best-case outcome. Consider, for instance, if the first generated Craig interpolant was $(x_0 = 1 \vee x_1 = 0) \wedge x_2 = 1$. This predicate describes the three states on the lower right owned by REACH. Every transition into these states starts in a state owned by SAFE that has a valid

alternative. In this case, our analysis would terminate immediately after instantiating the interpolant, because the associated necessary subgoal would be locally avoidable.

Generally, we observe that the behavior and runtime of our algorithm highly depend on the generated Craig interpolants as well as the game structure and the execution can be quite fast if these two are amicable, as shown in Chapter 5. However, the example in this section shows that the worst-case runtime of our algorithm on Boolean reachability games is worse than that of classic approaches based on the attractor construction, which scale linearly in the number of transitions. We discuss this in more detail in Section 4.5.

4.4 CORRECTNESS

Following the intuitive description of Algorithms 1 and 2 in Section 4.2, we now provide a formal proof of correctness for the latter algorithm. This in turn requires us to prove that Algorithm 1 is correct as well. Since deciding the winner for many classes of reachability games is an undecidable problem, we prove partial correctness only. In the next section, we will discuss the termination behavior of Algorithm 1 in more detail.

THEOREM 4.4 (Partial correctness). Let $\mathcal{G} = \langle \mathcal{V}, \mathit{Init}, \mathit{Safe}, \mathit{Reach}, \mathit{Goal} \rangle$ be a reachability game. If Solve($\mathcal{G}$) returns 'REACH', then REACH wins $\mathcal{G}$. If it returns 'SAFE', then SAFE wins $\mathcal{G}$.

Proof. In order to show the correctness of Solve($\mathcal{G}$), we prove the following properties about Reach($\mathcal{G}$). We show by induction on the recursion depth that if Reach($\mathcal{G}$) returns (R, T), then

(a) REACH wins in $\mathcal{G}$ from all states $s \in S$ that satisfy R;

(b) T is a necessary subgoal in $\mathcal{G}^{\setminus R} = \langle \mathit{Init} \wedge \neg R, \mathit{Safe}, \mathit{Reach}, \mathit{Goal} \rangle$ and Unsat(Enf($T, \mathcal{G}$)) evaluates to true.

Lemma 3.7 states that if we have a necessary subgoal in $\mathcal{G}^{\setminus R}$ that is locally avoidable, then SAFE wins from all initial states. Hence, (b) implies the existence of a winning strategy σ_S for SAFE in $\mathcal{G}^{\setminus R}$. Together, the two properties in particular ensure that R characterizes the largest set of initial states that is winning for REACH in $\mathcal{G}$.

If the above properties of Reach($\mathcal{G}$) are proven, then clearly Solve($\mathcal{G}$) determines the winner of $\mathcal{G}$ correctly: If Sat(R) in line 3 evaluates to true, then by definition REACH wins $\mathcal{G}$. If not, SAFE wins $\mathcal{G}$.

We proceed to prove the properties of Reach($\mathcal{G}$) by considering five different cases. Each corresponds to one of the four return statements, with a case distinction for the return statement in line 20 based on the

outcome of the conditional in line 15.

Case 1: Reach($\mathcal{G}$) returns in line 5. We have that Unsat($\mathit{Init} \wedge \neg \mathit{Goal}$) evaluates to true, which means that the formula ($\mathit{Init} \implies \mathit{Goal}$) is valid and all initial states are goal states. The algorithm returns $R = \mathit{Init} \wedge \mathit{Goal}$, which in this case is logically equivalent to *Init*. Trivially, any strategy is winning for `REACH` from states $s \in S$ satisfying this R, proving (a).

$\mathcal{G}^{\setminus R}$ has no initial states, and therefore the set of plays of $\mathcal{G}^{\setminus R}$ is empty. Hence, $T = \texttt{false}$ qualifies as a necessary subgoal in $\mathcal{G}^{\setminus R}$. Note also that Unsat(Enf($\texttt{false}, \mathcal{G}$)) evaluates to true. It follows that (b) holds.

Case 2: Reach($\mathcal{G}$) returns in line 9. We can conclude that (a) holds with the same argument as in Case 1.

We have that Enf($C, \mathcal{G}$) is unsatisfiable, where C is the instantiation of the Craig interpolant φ (lines 6 and 7). Proposition 3.6 states that C is a necessary subgoal in $\mathcal{G}^{\setminus R}$. Together with Unsat(Enf($C, \mathcal{G}$)), we can conclude that statement (b) holds.

Case 3: Reach($\mathcal{G}$) returns in line 14. We can again immediately conclude that (a) holds with the same argument as in Case 1.

By induction hypothesis, we assume that the recursive call in line 11 returned a tuple $(R_{\text{post}}, T_{\text{post}})$ satisfying properties (a) and (b) for the subgame $\mathcal{G}_{\text{post}}$. We proceed to show property (b) for $T = F \vee T_{\text{post}}$ in the global game $\mathcal{G}$.

First, we show that T is a necessary subgoal in $\mathcal{G}^{\setminus R}$. Let $\rho = s_0 s_1 \ldots$ be a play such that $(\mathit{Init} \wedge \neg R)(s_0)$ and $\mathit{Goal}(s_n)$ for some $n \in \mathbb{N}$. As C is a necessary subgoal in $\mathcal{G}$, there exists $m \in \mathbb{N}$ with $m < n$ such that $C(s_m, s_{m+1})$ holds. Take m to be the last index with this property. We make a case distinction based on whether $F(s_m, s_{m+1})$ holds. If it does, then we in particular also have $T(s_m, s_{m+1})$. If it does not hold, i.e., we have $\neg F(s_m, s_{m+1})$, we know that $(\text{Post}(C)[\mathcal{V}'/\mathcal{V}] \wedge \neg R_{\text{post}})(s_{m+1})$ evaluates to true. By induction hypothesis, for every play $\rho' = s'_0 s'_1 \ldots$ of $\mathcal{G}^{\setminus R}_{\text{post}} = \langle \text{Post}(C)[\mathcal{V}'/\mathcal{V}] \wedge \neg R_{\text{post}}, \mathit{Safe} \wedge \varphi, \mathit{Reach} \wedge \varphi, \mathit{Goal} \rangle$ such that there is a $l \in \mathbb{N}$ with $\mathit{Goal}(s'_l)$ there is some $j < l$ such that $T_{\text{post}}(s'_j, s'_{j+1})$ holds. $\rho'' = s_{m+1} \ldots s_n$ is a play in $\mathcal{G}^{\setminus R}_{\text{post}}$ because we chose m to be the last index that satisfies $C(s_m, s_{m+1})$, hence we know that for all $m < k \leqslant n$, $\varphi(s_k)$ holds. For ρ, it follows that there is some $n > j > m$ such that $T_{\text{post}}(s_j, s_{j+1})$. This means in particular that $T(s_j, s_{j+1})$ holds. Taken together, we can conclude that T is a necessary subgoal in $\mathcal{G}$.

It remains to show that Unsat(Enf($T, \mathcal{G}$)) evaluates to true. We have Unsat(Enf($F, \mathcal{G}$)), Unsat(Enf($T_{\text{post}}, \mathcal{G}_{\text{post}}$)) and:

1. Trivially, ($\mathit{Safe} \vee \mathit{Reach}$) $\implies$ ($\mathit{Safe} \vee \mathit{Reach}$);

2. $((\mathit{Safe} \wedge \varphi) \vee (\mathit{Reach} \wedge \varphi)) \implies (\mathit{Safe} \vee \mathit{Reach})$;
3. Unsat$(((\mathit{Safe} \wedge \varphi) \vee (\mathit{Reach} \wedge \varphi)) \wedge F)$, because $F \implies \neg\varphi$.

This lets us apply Lemma 3.3 to conclude that Unsat$(\mathrm{Enf}(F \vee T_{\mathrm{post}}, \mathcal{G}))$ evaluates to true.

Case 4: Reach$(\mathcal{G})$ returns in line 20, the conditional in line 15 is false, so we know that Unsat$((\mathit{Reach} \vee \mathit{Safe}) \wedge \varphi \wedge \neg\varphi' \wedge \neg\mathit{Goal})$ holds. By induction hypothesis, we assume that the recursive calls in line 11 and in line 19 returned tuples $(R_{\mathrm{post}}, T_{\mathrm{post}})$ and $(R_{\mathrm{pre}}, T_{\mathrm{pre}})$ that satisfy the properties (a) and (b) for the subgames $\mathcal{G}_{\mathrm{post}}$ and $\mathcal{G}_{\mathrm{pre}}$, respectively. We proceed to prove that the same properties hold in $\mathcal{G}$ for $R = \mathit{Init} \wedge \mathit{Goal} \vee R_{\mathrm{pre}}$ and $T = (\neg\varphi \wedge T_{\mathrm{pre}}) \vee T_{\mathrm{post}} \vee (F \wedge \neg\,\mathrm{Enf}(F, \mathcal{G}))$.

For (a), note that REACH wins from any state s satisfying $(\mathit{Init} \wedge \mathit{Goal})$ following the argument from Case 1. Since we have Unsat$((\mathit{Reach} \vee \mathit{Safe}) \wedge \varphi \wedge \neg\varphi' \wedge \neg\mathit{Goal})$, i.e., φ perfectly partitions $\mathcal{G}$, we know that F is the largest sufficient subgoal included in the necessary subgoal C by Lemma 4.2. This lets us apply Proposition 3.9 to conclude that REACH wins from all states s satisfying R_{pre}, hence also from all states satisfying R.

For (b), consider the necessary subgoal C. For every play $\rho = s_0 s_1 \ldots$ with some $n \in \mathbb{N}$ such that $\mathit{Goal}(s_n)$ there is some $k \in \mathbb{N}$ with $k < n$ and $C(s_k, s_{k+1})$. Take k to be the last index with this property. As F characterizes a subset of C, we observe three mutually exclusive cases: Either (1) $\neg F(s_k, s_{k+1})$, (2) $(F \wedge \neg\,\mathrm{Enf}(F, \mathcal{G}))(s_k, s_{k+1})$ or (3) $(F \wedge \mathrm{Enf}(F, \mathcal{G}))(s_k, s_{k+1})$. In case (1), we have $\neg R_{\mathrm{post}}(s_{k+1})$ and because of our assumption for all $j \in \mathbb{N}$ with $k \leqslant j < n : \varphi(s_j)$. It follows that there is some $l \in \mathbb{N}$ such that $T_{\mathrm{post}}(s_l, s_{l+1})$. Case (2) immediately qualifies for passing T. In case (3), we know that $\mathrm{Pre}(\mathrm{Enf}(F, \mathcal{G}))(s_k)$ holds. From our induction hypothesis, we know that there is some $h \in \mathbb{N}$ such that $h < k$ and $T_{\mathrm{pre}}(s_h, s_{h+1})$. With our assumption, we know that for all $m \in \mathbb{N}$ with $0 \leqslant m < k : \neg\varphi(s_m)$. It follows that $(\neg\varphi \wedge T_{\mathrm{pre}})(s_h, s_{h+1})$ holds. In all three cases, we pass T. Hence, T qualifies as a necessary subgoal in $\mathcal{G}$.

It remains to show that Unsat$(\mathrm{Enf}(T, \mathcal{G}))$ evaluates to true. We have Unsat$(\mathrm{Enf}((\neg\varphi \wedge T_{\mathrm{pre}}), \mathcal{G}_{\mathrm{pre}}))$, Unsat$(\mathrm{Enf}(T_{\mathrm{post}}, \mathcal{G}_{\mathrm{post}}))$, Unsat$(\mathrm{Enf}(F \wedge \neg\,\mathrm{Enf}(F, \mathcal{G}), \mathcal{G}))$ and:

1. Trivially, $(\mathit{Safe} \vee \mathit{Reach}) \implies (\mathit{Safe} \vee \mathit{Reach})$;
2. $((\mathit{Safe} \wedge \varphi) \vee (\mathit{Reach} \wedge \varphi)) \implies (\mathit{Safe} \vee \mathit{Reach})$;
3. $((\mathit{Safe} \wedge \neg F) \vee (\mathit{Reach} \wedge \neg F)) \implies (\mathit{Safe} \vee \mathit{Reach})$;
4. Unsat$(((\mathit{Safe} \wedge \neg F) \vee (\mathit{Reach} \wedge \neg F)) \wedge F)$;
5. Unsat$(((\mathit{Safe} \wedge \varphi) \vee (\mathit{Reach} \wedge \varphi)) \wedge (F \vee (\neg\varphi \wedge T_{\mathrm{pre}}))$, because $(F \vee (\neg\varphi \wedge T_{\mathrm{pre}})) \implies \neg\varphi$.

This lets us apply Lemma 3.3 twice (first, T_{pre} with F, then the result with T_{post}) to conclude that $\text{Unsat}(\text{Enf}(T, \mathcal{G}))$ evaluates to true.

Case 5: $\text{Reach}(\mathcal{G})$ returns in line 20, the conditional in line 15 is true. By induction hypothesis, we assume that the recursive calls in line 11 and in line 19 returned tuples $(R_{\text{post}}, T_{\text{post}})$ and $(R_{\text{pre}}, T_{\text{pre}})$ that satisfy the properties (a) and (b) for the subgames $\mathcal{G}_{\text{post}}$ and $\mathcal{G}_{\text{pre}}$, respectively. We proceed to prove that the same properties hold in $\mathcal{G}$ for $R = \mathit{Init} \wedge \mathit{Goal} \vee R_{\text{pre}}$ and $T \equiv (T_{\text{pre}}) \vee (F \wedge \neg\, \text{Enf}(F, \mathcal{G}))$.

For (a), note again that REACH wins from any state s satisfying $(\mathit{Init} \wedge \mathit{Goal})$ following the argument from Case 1. F is the largest sufficient subgoal included in the necessary subgoal F. Necessity of F follows from $\mathit{Goal}[\mathcal{V}/\mathcal{V}'] \implies F$. Sufficiency because for any s such that $\text{Post}(F)(s)$ is true we have either $R_{\text{post}}(s)$, which means s is won by REACH by induction hypothesis or $\mathit{Goal}(s)$. Necessity and sufficiency let us apply Proposition 3.9 to conclude that REACH wins from all states s satisfying R_{pre}, hence also from all states satisfying R.

For (b), consider the necessary subgoal F, newly constructed as outlined above. For every play $\rho = s_0 s_1 \ldots$ with some $n \in \mathbb{N}$ such that $\mathit{Goal}(s_n)$ there is some $k \in \mathbb{N}$ with $k < n$ and $F(s_k, s_{k+1})$. We observe two mutually exclusive cases: Either (1) $(F \wedge \neg\, \text{Enf}(F, \mathcal{G}))(s_k, s_{k+1})$ or (2) $(F \wedge \text{Enf}(F, \mathcal{G}))(s_k, s_{k+1})$. Case (1) immediately qualifies for passing T. In case (2), we know that $\text{Pre}(\text{Enf}(F, \mathcal{G}))(s_k)$ holds. From our induction hypothesis, we know that there is some $h \in \mathbb{N}$ such that $h < k$ and $T_{\text{pre}}(s_h, s_{h+1})$. In both cases, we pass T. Hence, T qualifies as necessary subgoal in $\mathcal{G}$.

It remains to show that $\text{Unsat}(\text{Enf}(T, \mathcal{G}))$ evaluates to true. We have $\text{Unsat}(\text{Enf}(F \wedge \neg\, \text{Enf}(F, \mathcal{G}), \mathcal{G}))$, $\text{Unsat}(\text{Enf}(T_{\text{pre}}, \mathcal{G}_{\text{pre}}))$ and:

1. Trivially, $(\mathit{Safe} \vee \mathit{Reach}) \implies (\mathit{Safe} \vee \mathit{Reach})$;
2. $((\mathit{Safe} \wedge \neg F) \vee (\mathit{Reach} \wedge \neg F)) \implies (\mathit{Safe} \vee \mathit{Reach})$;
3. $\text{Unsat}(((\mathit{Safe} \wedge \neg F) \vee (\mathit{Reach} \wedge \neg F)) \wedge F)$.

This lets us apply Lemma 3.3 to conclude that $\text{Unsat}(\text{Enf}(T, \mathcal{G}))$ evaluates to true.

We have now proved that properties (a) and (b) hold in all five cases and whenever Algorithm 1 terminates. This means that REACH wins in $\mathcal{G}$ if and only if the returned state predicate R is satisfiable, i.e., if there is at least some initial state in $\mathcal{G}$ that is winning for REACH. Hence, we have also shown that Algorithm 2 is correct. □

4.5 TERMINATION

As discussed and proven in Chapter 2, deciding the winner for linear arithmetic games is generally undecidable. Since Algorithm 2 returns

a correct result whenever it terminates, this implies that it cannot always terminate on linear arithmetic games. On the other hand, there are complete decision procedures for Boolean reachability games and other games on finite graphs. In this section, we prove that Algorithm 2 is guaranteed to terminate on finite game structures such as Boolean reachability games. We also detail the main challenge that arises with termination on infinite game structures, as well as discuss a way to alleviate the problem.

4.5.1 *Finite Graphs*

We now first consider the behavior of Algorithm 2 on finite game structures such as Boolean reachability games. As mentioned, there are many known decision procedures for determining the winner of these games. Our algorithm is guaranteed to terminate on a reachability game if all variables range over a finite domain, as stated by the following theorem. This particularly includes Boolean reachability games. The theorem was originally proved by Simon Jantsch for a different version of Algorithm 1. In this work, we adapt his proof for the presented version of the algorithm.

THEOREM 4.5. If the domains of all variables in $\mathcal{G}$ are finite, then Solve($\mathcal{G}$) terminates.

Proof. This proof follows the intuition that, if we remove at least one concrete transition from $(\mathit{Safe} \vee \mathit{Reach})$ in every recursive call and have a finite number of concrete transitions in $\mathcal{G}$, then Reach($\mathcal{G}$) terminates. This is enough to show the termination of Solve($\mathcal{G}$).

The size of a game $\mathcal{G}$, denoted by size($\mathcal{G}$), is defined as the number of concrete transitions in $\mathcal{G}$. Formally we have:

$$\text{size}(\mathcal{G}) = |\{(s_1, s_2) \in S \times S \mid (\mathit{Safe} \vee \mathit{Reach})(s_1, s_2) \text{ is valid}\}|.$$

Note that if the domains of all variables are finite as assumed in this theorem, then so is size($\mathcal{G}$). We proceed by showing that the subgames considered in every recursive call have a strictly smaller size than the global game from which they originate. This is enough to show that Reach($\mathcal{G}$) terminates.

First, observe that adding an additional constraint to the transition relation such as in both $\mathcal{G}_{\text{post}}$ or $\mathcal{G}_{\text{pre}}$ can never produce additional transitions. Therefore, we only need to show that at least one concrete transition is removed in both subgames.

In Reach($\mathcal{G}$) we first recurse on the post-game:

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

The post-game restricts the transition relation of both players to predecessor states satisfying φ. However, to show that size($\mathcal{G}_{\text{post}}$) is in

fact strictly smaller than size($\mathcal{G}$), we need to show that at least one concrete transition exists in $\mathcal{G}$ that is not a valid transition in $\mathcal{G}_{\text{post}}$. As the execution has passed line 9, we know that Enf($C, \mathcal{G}$) evaluates to true whenever the algorithm constructs $\mathcal{G}_{\text{post}}$, where C is the instantiation of interpolant φ and we have $C = (\mathit{Safe} \vee \mathit{Reach}) \wedge \neg\varphi \wedge \varphi'$ (line 7). From Definition 3.1 it follows that $(\mathit{Safe} \vee \mathit{Reach}) \wedge C$ is satisfiable. Hence, there exist $s, s' \in S$ such that $((\mathit{Safe} \vee \mathit{Reach}) \wedge C)(s, s')$, $\neg\varphi(s)$ and $\varphi(s')$ are all valid. We now have that $(\mathit{Safe} \vee \mathit{Reach})(s, s')$ is valid but $((\mathit{Safe} \vee \mathit{Reach}) \wedge \varphi)(s, s')$ is not valid. We can conclude that size($\mathcal{G}_{\text{post}}$) < size($\mathcal{G}$).

The second recursive call in line 18 is called on the pre-game:

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

If the algorithm reaches this line, we have passed line 13, so we can conclude that Sat(Enf($F, \mathcal{G}$)) holds. This means there exist $s, s' \in S$ such that $((\mathit{Safe} \vee \mathit{Reach}) \wedge F)(s, s')$ is valid. Trivially, we have that $(((\mathit{Safe} \vee \mathit{Reach}) \wedge \neg F) \wedge F)(s, s')$ is not valid. Therefore, it immediately follows that size($\mathcal{G}_{\text{pre}}$) < size($\mathcal{G}$).

This concludes the argument. We have shown that Reach($\mathcal{G}$) terminates if the variables in $\mathcal{G}$ have a finite domain, which directly implies that Solve($\mathcal{G}$) terminates in this case as well. □

Remark 4.6 (Time complexity). Note that we only ensure that at least one concrete transition is removed from the global game when constructing the pre-game and the post-game. Following the above termination argument therefore results in a single-exponential upper bound on the runtime of the algorithm in the number of concrete transitions of the input game.

If we consider finite game structures, this is a discouraging result, because the algorithms based on the attractor construction [37] (as presented in Chapter 2) scale linearly in the number of concrete transitions of the input game.

The exponential time complexity is less of a concern for our goal of solving games on infinite graphs as this is an undecidable problem without any efficient and generally applicable algorithm. As we show in our experimental evaluation in Chapter 5, our approach does scale significantly better than the only other readily available tool for solving linear arithmetic games on several benchmark families.

4.5.2 *Infinite Graphs*

We now discuss the termination behavior of our algorithm on games over infinite graphs. As mentioned before, there is no algorithm for this problem that terminates on all inputs. Here, we try to exemplify the kinds of game structures which result in non-termination of our algorithm.

The main problem our algorithm faces in terms of termination on infinite game graphs is that the post-game behind a necesary subgoal may have an initial predicate that is satisfied by infinitely many states. Depending on the transition relation, solving the post-game recursively might not terminate in these cases, if only a finite number of initial states can be classified with every recursive call. The Example 4.2 illustrates the described behavior.

If some of the initial states of the post-game are not in the reachable fragment of the game structure or would not be explored in a backwards exploration starting in the goal states, they are actually not needed to determine the winner of the game. This is a disadvantage of our approach compared to techniques based on forward or backward unrolling.

EXAMPLE 4.2. Consider the linear integer arithmetic game $\mathcal{G}$ described by the following formulas

$$\begin{aligned}
\mathit{Init} \equiv \quad & \neg \mathbf{r} \wedge x = 0 \wedge y = 0; \\
\mathit{Safe} \equiv \quad & \neg \mathbf{r} \wedge \mathbf{r}' \wedge ((x' \leqslant 1 \wedge x' = x + 1 \wedge y' = y) \\
& \vee (x' = x \wedge y' = y - 1)); \\
\mathit{Reach} \equiv \quad & \mathbf{r} \wedge \neg \mathbf{r}' \wedge ((x' \leqslant 1 \wedge x' = x + 1 \wedge y' = y) \\
& \vee (x' = x \wedge y' = y - 1)); \\
\mathit{Goal} \equiv \quad & \neg \mathbf{r} \wedge x = 1 \wedge y = 0.
\end{aligned}$$

The game is played over variables x and y and both are initially equal to 0. Both `REACH` and `SAFE` can either set x from 0 to 1 (but not back) or decrement y, each without changing the value of the other variable. The goal condition states that x has to be set to 1, but y is to remain at 0.

If we consider an execution of Algorithm 1 on $\mathcal{G}$, the only valid initial Craig interpolant would be $x = 1$. Hence, the first necessary subgoal computed is:

$$C_1 = (\mathit{Safe} \vee \mathit{Reach}) \wedge x \neq 1 \wedge x = 1.$$

Neither *Safe* nor *Reach* specify a concrete value of y for the case where x is incremented, so we have that $\mathrm{Post}(C_1)[\mathcal{V}'/\mathcal{V}] \implies x = 1$. Intuitively, this means that all states with value $x = 1$ are valid successors of C_1. These states are the initial states Init' of our post-game $\mathcal{G}_{\text{post}}$ and describe an infinitely large set of states, because any possible value of y is allowed.

This would not necessarily be a problem if we could find a sufficient subgoal that also captures an infinite set of winning states in its precondition, but in $\mathcal{G}$ the transition relation only allows to decrement and increment.

Hence, we proceed with computing the largest sufficient subgoal included in C_1 in the following way: We first conclude that all states

satisfying $R = \mathit{Init}' \wedge \mathit{Goal}$ are winning for REACH and proceed with the set of states characterized by $\mathit{Init}' \wedge \neg \mathit{Goal}$. This in turn yields the interpolant and associated subgoal

$$C_2 = (\mathit{Safe}' \vee \mathit{Reach}') \wedge y \neq 0 \wedge y = 0.$$

Note that Safe' and Reach' are restricted to $x = 1$ since we restrict the subgame to the last time the effect is bridged. Hence, the post-game after C_2 immediately terminates.

We proceed with $\mathrm{Pre}(C_2)$ as the goal in the pre-game (with respect to $\mathcal{G}_{\mathrm{post}}$) $\mathcal{G}_{\mathrm{pre}}$ starting in $\mathrm{Post}(C_1)[\mathcal{V}'/\mathcal{V}]$. We have $\mathrm{Pre}(C_2) \implies y = 1$ and now continue the process of computing the subset of winning states of $\mathrm{Post}(C_1)[\mathcal{V}'/\mathcal{V}]$ with interpolant $y = 1$ and associated necessary subgoal C_3. Since we can only decrement y again, the precondition of the largest sufficient subgoal F_3 included in C_3 implies $y = 2$. This can then be chosen as a interpolant in the pre-game from $\mathrm{Post}(C_1)[\mathcal{V}'/\mathcal{V}]$ to $\mathrm{Pre}(\mathrm{Enf}(F_3, \mathcal{G}_{\mathrm{pre}}))$.

Every subgoal instantiated in this way will only classify a single state of the infinite set characterized by $\mathrm{Post}(C_1)[\mathcal{V}'/\mathcal{V}]$. Consequently, this process continues ad infinitum and Algorithm 1 does not terminate.

In a way, the problem in the example above is that we lose information about the initial value of y in the post-game. This can make the post-games much harder to solve than necessary. Approaches based on an unrolling of the transition relation, such as [14], do not encounter this problem in the same way, because the information about the initial values of the variables gets propagated in the forward unrolling.

We propose the following practical solution to improve the termination of our algorithm: restricting the values of the variables of the game to some predefined interval. We realize this by adding constraints of the following form to the formulas of the game $\mathcal{G}$ for every variable y bounded to some interval $[a, b]$:

$$\begin{aligned}
\mathit{Init} = \quad & \mathit{Init} \wedge a \leqslant y \wedge y \leqslant b; \\
\mathit{Safe} = \quad & \mathit{Safe} \wedge a \leqslant y \wedge y \leqslant b \wedge a' \leqslant y' \wedge y' \leqslant b'; \\
\mathit{Reach} = \quad & \mathit{Reach} \wedge a \leqslant y \wedge y \leqslant b \wedge a' \leqslant y' \wedge y' \leqslant b'; \\
\mathit{Goal} = \quad & \mathit{Goal} \wedge a \leqslant y \wedge y \leqslant b.
\end{aligned}$$

Note that this does not necessarily make $\mathcal{G}$ finite state, depending on whether y ranges over $\mathbb{N}$ or $\mathbb{R}$ and also depending on the other variables of the game. However, in practice these additional constraints let our algorithm terminate on several game instances on which it would not terminate within the timeout of 10 minutes otherwise, as we discuss in more detail in the next chapter.

5 CASE STUDIES

In this chapter, we report on the evaluation of our approach on a number of case studies. We consider multiple benchmarks from the literature and also some that we constructed for this work.

Our prototype implementation, named CabPy, is written in Python and uses the PySMT [16] library as an interface to the MathSAT5 [11] and Microsoft's Z3 [32] SMT solvers.

In CabPy, we have implemented the *game solving* part of Algorithm 1 and do not keep track of the necessary subgoal that witnesses a strategy for the safety player and could be used for strategy synthesis. However, it would be straightforward to use the intermediate results of the algorithm to compute this predicate, as outlined in the correctness proof in Section 4.4.

We compared our prototype with SimSynth [14], the only other readily available tool for solving linear arithmetic games. SimSynth does synthesize a strategy in the case where the safety player wins the game, but does not provide a strategy if the reachability player wins the game.

Following the discussion in Section 4.5, we sometimes need to bound the domains of the variables to ensure the termination of our tool CabPy. In these cases, we specify this in the description of the case study. Note that depending on the domain of the variables this still does not make the game structures finite. This, together with our ability to treat highly symbolic game descriptions, leaves SimSynth as the most applicable tool for comparison even in these cases.

The evaluation was carried out on a computer running Ubuntu 20.04 and equipped with a 4-core Intel(R) Core(TM) i5 2.30GHz processor, as well as 8GB of memory. On all benchmarks, the timeout was set to 10 minutes.

Besides the runtimes of the programs, we have also recorded the number of subgames considered by Algorithm 1, which corresponds to the size of the recursion tree and lets us make some statements about the structure of the subgoal-based exploration.

5.1 MONA LISA

The game described as the motivating example in Chapter 1, played between a thief and a security guard, is very well suited to assess the strength and limitations of both our approach as well as 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

		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

Table 5.1: Experimental results for the Mona Lisa game.

where the guard does not sleep at all. The results can be found in Table 5.1.

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. This is a clear disadvantage of our approach, demonstrating a trade-off between the two tools.

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.

5.2 GAME OF NIM

The Game of Nim is a classic game from the literature [9]. It is played on a number of heaps of stones, where both players remove stones from the heaps. There exist several different versions of "Game of Nim" that have other setups. We consider the version where both players take turns choosing a single heap and removing at least one stone from it.

For this version of Game of Nim, two different winning conditions can be found in the literature: *misère* and *normal play* Game of Nim. In misère play, the player who takes the last stone loses, while this player wins in normal play. We consider normal play Game of Nim.

We model this game as a linear integer reachability game, as shown in the following for an example configuration. We have two heaps modeled by variables x and y that start with 6 and 7 stones, respectively. Technically, there is no actual difference in the moves for both players. We still call them `REACH` and `SAFE`, and fix that `SAFE` is the player that is allowed to make the first move:

$$
\begin{aligned}
\mathit{Init} \equiv \quad & \neg \mathbf{r} \wedge x = 6 \wedge y = 7; \\
\mathit{Safe} \equiv \quad & \neg \mathbf{r} \wedge \mathbf{r}' \wedge (0 \leqslant x' \leqslant x - 1 \wedge y' = y \\
& \vee 0 \leqslant y' \leqslant y - 1 \wedge x' = x); \\
\mathit{Reach} \equiv \quad & \mathbf{r} \wedge \neg \mathbf{r}' \wedge (0 \leqslant x' \leqslant x - 1 \wedge y' = y \\
& \vee 0 \leqslant y' \leqslant y - 1 \wedge x' = x); \\
\mathit{Goal} \equiv \quad & \neg \mathbf{r} \wedge x = 0 \wedge y = 0.
\end{aligned}
$$

The runtimes of our experiments are shown in Table 5.2. In all instances with three heaps or more we have bounded the domains of the heap variables in the instance description, by specifying that no heap exceeds its initial size and does not go below zero.

Following the discussion in Section 4.5, 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 ones.

The special characteristics of the Game of Nim mean that for our approach, there are no natural necessary subgoals that the safety player can locally control. This is because the game structure ensures a sort of natural progress towards a winning state for either of the players. There are also no infinite plays in this configuration. Either *Goal* is reached as a trap state of `SAFE` and `REACH` wins. Or a trap state of `REACH` is reached and `SAFE` wins because the goal condition was not satisfied before.

	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

Table 5.2: Experimental results for the Game of Nim. The notation $(h_1, \ldots, h_n)$ denotes an instance played on n heaps, each of which consists of h_i stones. Instances marked with b indicate that the variable domains were restricted for CabPy (cf. Section 4.5).

Our results in Table 5.2 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 all of them out.

5.3 CORRIDOR

We now consider an example that demonstrates the potential of our method in case the game structure contains natural bottlenecks that are found by Craig interpolation. Consider a corridor of 100 rooms arranged in sequence, i.e., each room i with $0 \leqslant i < 100$ connects

	CabPy		SimSynth	
Door	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

Table 5.3: Experimental results for the Corridor game.

to room $i+1$ with a door. The objective of the reachability player is to reach room 100 and they are free to choose valid values from $\mathbb{R}^2$ for the position in each room at every other turn, but must reach the door at $x = 5 \wedge y = 5$ in every room to pass to the next one, where they start at $x = -5 \wedge y = -5$. The safety player controls the door to some room $i < 100$. We evaluated this game with controlled doors at different places in the corridor as seen in Table 5.3.

We model this scenario as a linear reachability game played over $\mathbb{R}$. The variable c models the current room, the variables x and y the current position in that room. In this example, SAFE controls the door between rooms 59 and 60:

$$
\begin{aligned}
\mathit{Init} \equiv\ & \neg \mathbf{r} \wedge x = 0 \wedge y = 0 \wedge c = 0; \\
\mathit{Safe} \equiv\ & \neg \mathbf{r} \wedge \mathbf{r}' \wedge \\
& ((x = 5 \wedge y = 5 \wedge x' = -5 \wedge y' = -5 \wedge c' = c + 1) \\
& \vee (c < 60 \wedge c \geqslant 59 \wedge x' = x \wedge y' = y \wedge c' = c) \\
& \vee (\neg(x = 5 \wedge y = 5) \wedge x' = x \wedge y' = y \wedge c' = c)); \\
\mathit{Reach} \equiv\ & \mathbf{r} \wedge \neg \mathbf{r}' \wedge x' \leqslant 5 \wedge x' \geqslant -5 \wedge y' \leqslant 5 \wedge y' \geqslant -5 \wedge c' = c; \\
\mathit{Goal} \equiv\ & \neg \mathbf{r} \wedge c = 100.
\end{aligned}
$$

Naturally, a winning strategy for SAFE is to prevent the reachability player from passing the door under their control, which is a natural bottleneck and a necessary subgoal for REACH on the way to the last room.

The experimental results are summarized in Table 5.3. 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 within the rooms, as can be seen by the number of considered subgames. SimSynth, which

tries to generalize a strategy obtained from a step-bounded game in a forward exploration approach, struggles because the tool solves the games that happen between each of the doors before reaching the controlled one.

5.4 HARE AND HEDGEHOG

Like similar efforts in the literature [20], we look to the realm of tales and stories to inspire our next case study. We consider the classic tale of the hare and the hedgehog, who decide to race over a furrow. In the tale, the hedgehog outsmarts the hare by placing his partner at the end of the furrow. The partner appears there shortly after the hare has reached the halfway point, thereby winning the race. This way, the two hedgehogs are able to defeat the hare in every race. The hare demands revenge until it is finally exhausted after 73 races.

The following linear real reachability game models the above scenario. The hare is modeled by the reachability player, the hedgehogs by the safety player. Two variables represent the position of the hare (a) and one hedgehog (e) on the furrow, which has a length of 100 in the example below. We model only one hedgehog on the furrow at any time. The hare can move twice as fast over the furrow as the hedgehog, but once the hare reaches the halfway point the hedgehogs are allowed to execute their ploy and immediately win the current race. x and y are used to count the wins of the hare and hedgehogs, respectively. Variable p counts the number of races. We assume that the hare is a bad winner and will claim victory as soon as overall gaining more victories than the hedgehog, therefore the goal condition is $x > y$, but this has to be reached before the hare is exhausted, therefore we add $p \leqslant 73$:

$$
\begin{aligned}
\mathit{Init} &\equiv \neg\mathbf{r} \wedge a = 0 \wedge e = 0 \wedge y = 0 \wedge x = y \wedge p = 0;\\
\mathit{Safe} &\equiv \neg\mathbf{r} \wedge \mathbf{r}'\\
&\quad \wedge ((e' \leqslant e+1 \wedge e' \geqslant e-1 \wedge a' = a \wedge p' = p \wedge x' = x\\
&\quad \wedge y' = y) \vee (a \geqslant 50 \wedge a' = 0 \wedge e' = 0 \wedge p' = p+1\\
&\quad \wedge x' = x \wedge y = y+1));\\
\mathit{Reach} &\equiv \mathbf{r} \wedge \neg\mathbf{r}'\\
&\quad \wedge ((a < 100 \wedge a' \leqslant a+2 \wedge a' \geqslant a-2 \wedge e' = e \wedge p' = p\\
&\quad \wedge x' = x \wedge y' = y) \vee (a \geqslant 100 \wedge a' = 0 \wedge e' = 0 \wedge p' = p+1\\
&\quad \wedge x' = x+1 \wedge y = y));\\
\mathit{Goal} &\equiv \neg\mathbf{r} \wedge x > y \wedge p \leqslant 73.
\end{aligned}
$$

This case study highlights some interesting points about the two approaches. Consider the experimental results in Table 5.4. As one would expect, SimSynth scales with the distance of the furrow. This

	CabPy		SimSynth	
Distance	Subgames	Time(s)	Time(s)	Winner
10	8	0.65	5.02	SAFE
100	8	0.63	Timeout	SAFE
1000	7	0.56	Timeout	SAFE

Table 5.4: Experimental results for the Hare (REACH) and Hedgehog (SAFE) game.

is because the key winning transition for the hedgehog (appearance of the partner at the finish line) only appears after the halfway point of the furrow is reached. This is not the case for CabPy, because our approach does not necessarily depend on a forward exploration.

Additionally, the initial and goal descriptions of this game are amicable to producing a very strong first interpolant, e.g., $x > y$. This means as a first necessary subgoal the algorithm considers whether the hare can ever transition from having won a less or equal number of races than the hedgehog to having won more races. The algorithm then quickly infers that any predecessor states of this subgoal are in a configuration where the hedgehogs would have the option of winning before. This is because the hare has to be past the halfway point to win. The case study therefore highlights that our approach can benefit a lot from optimizing interpolant generation.

5.5 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, 14].

```
//Start between 20.8 and 23.5
assume(20.8 <= temp <= 23.5);
while(*) {
     //Objective: Stay between 20 and 25
     assert(20 <= temp <= 25);
     if (isOn == 1) {
          temp = temp + 1 - (1/10 * (temp - 19));
     } else {
          temp = temp - (1/10 * (temp - 19));
     }
     //Control the value of ?? such that objective is realized
     isOn = ??;
}
```

Figure 5.1: The thermostat program synthesis problem.

Thread 1

```
while(*) {
     //Noncritical
     y = x + 1
     await (x = 0 || y <= x)
     //Critical
     y = 0
}
```

Thread 2

```
while(*) {
     //Noncritical
     x = y + 1
     await (y = 0 || x < y)
     //Critical
     x = 0
}
```

Figure 5.2: The Bakery program synthesis problem considers the parallel execution of Threads 1 and 2. Initially, we have `x = 0` and `y = 0`.

Figure 5.1 shows a partial program that has to be filled by the synthesis procedure to produce a safe controller. A range of possible initial values of the room temperature `temp` is given, in our example, $20.8 \leqslant$ `temp` $\leqslant 23.5$, together with the temperature dynamics wich depend on whether the heater is on or (Boolean variable `isOn`). The objective for `SAFE` is to control the value of `isOn` in every round such that `temp` stays between 20 and 25 degrees. This is a common benchmark for program synthesis tools and both CabPy and SimSynth solve it quickly, as shown by the results in Table 5.5.

The other problem relates to Lamport's Bakery algorithm [26]. We consider two threads using this algorithm to ensure mutually exclusive access to a shared resource. The threads are shown in Figure 5.5 and the line that accesses the shared resource is denoted by the comment "`//Critical`". The game we consider for this scenario describes the task of synthesizing a scheduler that violates mutual exclusion. We model this by defining the full system dynamics in the reachability player transitions, with the safety player having no option but to set all variables to their previous value. Note that if such a scheduler existed, the protocol would be incorrect. This essentially makes this a model checking problem, and we study it to see how well the tools can infer a safety invariant that does not depend on a particular safety player strategy. Our approach solves this problem rather quickly as seen in Table 5.5. The forward unrolling approach of SimSynth, however, seems to explore the whole state space and fails to find an invariant before the timeout of 10 minutes.

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

Table 5.5: Experimental results for program synthesis problems.

6

RELATED WORK

Our focus on cause-effect-relationships for solving reachability games is related to and inspired by causality-based verification [23–25]. This method is based on the data structure of concurrent traces, which allow to succinctly model sets of traces in concurrent scenarios. This is realized by constructing a partial relation over events, e.g., transitions that happen in the trace. Proof rules capturing causal inference are defined over concurrent traces and allow to expand them with new, causally necessary events. One method that is used to infer these causally necessary events is Craig interpolation. The global exploration is organized in a trace tableau. With this trace tableau, it can be inferred whether there exists a counterexample trace which violates the property that is checked on the model.

The main similarities between this causality-based model checking approach and our causal game solving algorithm are that we focus on realizing a finite play that reaches the goal condition. This is conceptually similar to trying to realize a finite counterexample trace. We also use Craig interpolation both for observing effects and instantiating causes that have to happen on this play.

The main difference between the approaches is that for model checking, concurrent traces allow refinement and instantiation of events regardless of temporal order. This means it is sound to insert new events at any point in a given concurrent trace as long as the other conditions of the proof rules are met. The same refinement regardless of temporal order is, however, not possible in a game setting. Intuitively, this is because model checking simply requires to find a counterexample trace. Game solving, on the other hand, requires not only a single play reaching the goal condition, but a strategy that wins against all opposing strategies. Our approach therefore uses subgoals instead of events, and a recursive game solving algorithm instead of a trace tableau.

The notion of subgoals is related to the concept of landmarks in planning [19]. Planning describes the task of finding an action sequence such that the corresponding solution path moves some planning instance from an initial configuration to the goal configuration. In the deterministic setting, this relates more to model checking than game solving, as no opposing player has to be considered. A landmark is a configuration that must necessarily be true at some point on any solution path. It is therefore quite comparable to the necessary subgoals as considered in this work.

There is a significant difference in how landmarks and necessary subgoals are computed. Computing landmarks for planning can be done in a two way process. First, one constructs a set of landmark candidates with a backchaining process starting in the goal states and collecting new candidates by analyzing the preconditions of actions achieving the current candidates. In the second step, each candidate is checked by evaluating a sufficient condition for being a true landmark. The condition is checked by solving a relaxed planning task that misses all actions that could produce the landmark candidate configuration. In contrast to this two step computation, we use Craig interpolation between the initial and goal states of our games to find necessary subgoals. Another difference between the approaches is that landmarks are commonly used for heuristics that guide the search for a solution plan, while we use necessary subgoals to partition and solve the reachability game recursively.

Closer related to our two-player setting, there are techniques that utilize landmarks for counterplanning in non-cooperative multi agent systems [33]. The goal there is to find a plan for one agent that prevents another agent from reaching their goal. The approach is based on identifying the goal of the opposing agent and computing landmarks for this goal. Preventing the opposing agent from reaching such a landmark is sufficient to prevent them from reaching the overall goal. This is quite similar to the way we identify possible families of safety player strategies based on avoiding some necessary subgoal.

There are several other approaches to solving games over infinite graphs. Following the discussion in Chapter 2, they are all necessarily incomplete. One promising method [14] for solving linear arithmetic games is based on a dedicated decision procedure for quantified linear arithmetic formulas. This procedure is used to solve step-bounded plays of the game that end after a certain number of rounds. Non-losing moves for the safety player are explored in a tree-like structure until a safety strategy that stabilizes all consistent plays in a safe region is found. The main similarity to our approach is that they both aim at solving linear arithmetic games without additional knowledge provided by the user. The approach itself is, however, quite different from ours. In particular it is strictly based on a forward exploration of the game, while our method's behavior in this respect depends on the generated Craig interpolants. In Chapter 5 we evaluated the different strengths and weaknesses of the two approaches in more detail.

Another approach to solve infinite-state games uses deductive methods that compute the winning regions of both players using proof rules [4]. This method, however, relies on user-provided templates that capture the high-level intuitions about possible strategies. In contrast, our algorithm solves games without any additional user input.

7

CONCLUSION & FUTURE WORK

This thesis presented a causality-based approach for solving reachability games described by logical constraints. These games are an expressive formalism that allows to model a variety of problem arising for instance in non-deterministic planning and synthesis of programs or cyber-physical systems.

Our approach analyzes the cause-effect-relationships present in all plays satisfying the goal condition. To define these causal relationships, we introduced the notion of necessary and sufficient subgoals for two-player games.

Subgoals provide local criteria to determine the winner of a game. Avoiding a necessary subgoal yields a safety player strategy based on counterfactual reasoning: Since it defines the whole set of possible causes for some effect on the way to the goal condition, avoiding these causes guarantees that the effect of reaching the goal cannot happen. Dual to this, reaching a sufficient subgoal ensures a win for the reachability player.

Based on the framework of subgoals, we presented an algorithm for solving reachability games that slices a game along necessary subgoals computed from Craig interpolants. Recursively solving the subgame starting from some necessary subgoal allows us to compute the largest sufficient subgoal included in this necessary subgoal. The properties of necessity and sufficiency can then be used to solve the global game by another recursive call on the subgame from the initial states to the subgoal.

Our prototype implementation called CabPy realizes our algorithm. The experimental evaluation and comparison with the state-of-the-art tool SimSynth demonstrates that our approach is practically applicable and scales much better on several benchmarks. In particular, the two tools scale in different ways depending on the game structure: SimSynth unrolls the game from the beginning and is in this way highly dependent on the length of the path to key decisions which may yield a strategy of the safety player. Our approach, however, is less dependent on the size of the game structure and more dependent on the quality and amicability of the instantiated interpolants with the game structure. This leads us to conclude that the two tools complement each other and can be effectively utilized together for solving a large variety of games on infinite graphs.

An interesting topic for future work is strategy synthesis based on our causal game solving approach. The intermediate results com-

puted by our game solving algorithm can be used to describe symbolic strategies, i.e., predicates that define a family of strategies adhering to certain policies. In the case of the safety player, this corresponds directly to the necessary and locally avoidable subgoal that is returned by Algorithm 1. Any concrete strategy that avoids this set of moves is winning for the safety player, and the local avoidability criterion ensures that such a concrete strategy exists. Strategies for the reachability player, on the other hand, can be extracted by considering the sequence of sufficient subgoals between the initial states and the goal states of the analyzed reachability game.

However, even when constructing transition predicates that characterize families of strategies, it is unclear how to optimally extract a deterministic strategy from this relation [6, 13]. An interesting question on this front is whether our causality-based approach can be used to construct particularly sparse strategies. Sparse strategies, and therefore ideally small implementations of the system, can lead to more efficient and understandable synthesis outputs. Given that our safety player strategies can often be described by a necessary subgoal, irrespective of an actual unrolling of the reachability game, we are hopeful that our approach can help in synthesizing efficient and understandable system implementations.

Another possible topic to be explored in future work is the observation that approximate methods for determining the winning region of subgames can be effectively used to compute the winner of the global game. In this work, we considered an approximated post-game that restricts the reachability player by making any transition out of the post-game immediately winning for the safety player. This leads to an under-approximation of the reachability player's initial winning region. Dual to this, it is possible to restrict the safety player by forbidding transitions out of the post-game. This would lead to an under-approximation of the safety player's initial winning region. It is an interesting question whether a an approach based on both of these approximations could optimize our causality-based game solving algorithm.

BIBLIOGRAPHY

[1] Rajeev Alur, P. Madhusudan, and Wonhong Nam. "Symbolic Computational Techniques for Solving Games." In: *International Journal on Software Tools for Technology Transfer* 7 (Apr. 2005), pp. 118–128.

[2] Rajeev Alur, Salar Moarref, and Ufuk Topcu. "Pattern-Based Refinement of Assume-Guarantee Specifications in Reactive Synthesis." In: *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2015)*. Ed. by Christel Baier and Cesare Tinelli. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 501–516.

[3] Rajeev Alur, Salar Moarref, and Ufuk Topcu. "Compositional Synthesis of Reactive Controllers for Multi-agent Systems." In: *Computer Aided Verification (CAV 2016)*. Ed. by Swarat Chaudhuri and Azadeh Farzan. Cham: Springer International Publishing, 2016, pp. 251–269.

[4] Tewodros Beyene, Swarat Chaudhuri, Corneliu Popeea, and Andrey Rybalchenko. "A Constraint-Based Approach to Solving Games on Infinite Graphs." In: *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL '14. San Diego, California, USA: Association for Computing Machinery, 2014, 221–233. ISBN: 9781450325448.

[5] Dirk Beyer and Alexander K. Petrenko. "Linux Driver Verification." In: *Leveraging Applications of Formal Methods, Verification and Validation. Applications and Case Studies*. Ed. by Tiziana Margaria and Bernhard Steffen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 1–6. ISBN: 978-3-642-34032-1.

[6] Roderick Bloem, Uwe Egly, Patrick Klampfl, Robert Könighofer, and Florian Lonsing. "SAT-based methods for circuit synthesis." In: *Formal Methods in Computer-Aided Design, FMCAD 2014, Lausanne, Switzerland, October 21-24, 2014*. IEEE, 2014, pp. 31–34.

[7] Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. "Synthesis of Reactive(1) designs." In: *Journal of Computer and System Sciences* 78.3 (2012). In Commemoration of Amir Pnueli, pp. 911–938. ISSN: 0022-0000.

[8] Jörg Bormann, Jörg Lohse, Michael Payer, and Gerd Venzl. "Model Checking in Industrial Hardware Design." In: *Proceedings of the 32nd Annual ACM/IEEE Design Automation Conference*. DAC '95. San Francisco, California, USA: Association for Computing Machinery, 1995, 298–303. ISBN: 0897917251.

[9] Charles L. Bouton. "Nim, A Game with a Complete Mathematical Theory." In: *Annals of Mathematics* 3.1/4 (1901), pp. 35–39. ISSN: 0003486X.

[10] Randal E. Bryant. "Graph-Based Algorithms for Boolean Function Manipulation." In: *IEEE Trans. Comput.* 35.8 (Aug. 1986), 677–691. ISSN: 0018-9340.

[11] Alessandro Cimatti, Alberto Griggio, Bastiaan Schaafsma, and Roberto Sebastiani. "The MathSAT5 SMT Solver." In: *Proceedings of TACAS*. Ed. by Nir Piterman and Scott Smolka. Vol. 7795. LNCS. Springer, 2013.

[12] William Craig. "Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory." In: *Journal of Symbolic Logic* 22.3 (1957), 269–285.

[13] Rüdiger Ehlers and Daniela Moldovan. "Sparse Positional Strategies for Safety Games." In: *Proceedings First Workshop on Synthesis, SYNT 2012, Berkeley, California, USA, 7th and 8th July 2012*. Ed. by Doron A. Peled and Sven Schewe. Vol. 84. EPTCS. 2012, pp. 1–16.

[14] Azadeh Farzan and Zachary Kincaid. "Strategy Synthesis for Linear Arithmetic Games." In: *Proc. ACM Program. Lang.* 2.POPL (Dec. 2017).

[15] B. Finkbeiner. "Synthesis of reactive systems." In: Apr. 2016, pp. 72–98.

[16] Marco Gario and Andrea Micheli. "PySMT: a solver-agnostic library for fast prototyping of SMT-based algorithms." In: *SMT Workshop 2015*. 2015.

[17] G. Geier, P. Heim, F. Klein, and B. Finkbeiner. "Syntroids: Synthesizing a Game for FPGAs using Temporal Logic Specifications." In: *2019 Formal Methods in Computer Aided Design (FMCAD)*. 2019, pp. 138–146.

[18] Erich Grädel, Wolfgang Thomas, and Thomas Wilke, eds. *Automata, Logics, and Infinite Games: A Guide to Current Research*. Vol. 2500. Lecture Notes in Computer Science. Springer, 2002. ISBN: 3-540-00388-6.

[19] Jörg Hoffmann, Julie Porteous, and Laura Sebastia. "Ordered Landmarks in Planning." In: *J. Artif. Int. Res.* 22.1 (2004), 215–278.

[20] Antonius J. C. Hurkens, Cor A. J. Hurkens, and Gerhard J. Woeginger. "How Cinderella Won the Bucket Game (and Lived Happily Ever After)." In: *Mathematics Magazine* 84.4 (2011), pp. 278–283. ISSN: 0025570X, 19300980.

[21] Jan Jakoh Jessen, Jacob Illum Rasmussen, Kim G. Larsen, and Alexandre David. "Guided Controller Synthesis for Climate Controller Using Uppaal Tiga." In: *Formal Modeling and Analysis of Timed Systems (FORMATS 2007)*. Ed. by Jean-François Raskin and P. S. Thiagarajan. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 227–240. ISBN: 978-3-540-75454-1.

[22] Jie-Hong R. Jiang. "Quantifier Elimination via Functional Composition." In: *Computer Aided Verification*. Ed. by Ahmed Bouajjani and Oded Maler. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 383–397. ISBN: 978-3-642-02658-4.

[23] Andrey Kupriyanov. "Causality-based Verification." PhD thesis. Universität des Saarlandes, Saarbrücken, Germany, 2016.

[24] Andrey Kupriyanov and Bernd Finkbeiner. "Causality-Based Verification of Multi-threaded Programs." In: *CONCUR 2013 - Concurrency Theory - 24th International Conference, CONCUR 2013, Buenos Aires, Argentina, August 27-30, 2013. Proceedings*. Ed. by Pedro R. D'Argenio and Hernán C. Melgratti. Vol. 8052. Lecture Notes in Computer Science. Springer, 2013, pp. 257–272.

[25] Andrey Kupriyanov and Bernd Finkbeiner. "Causal Termination of Multi-Threaded Programs." In: *Proceedings of the 16th International Conference on Computer Aided Verification - Volume 8559*. Berlin, Heidelberg: Springer-Verlag, 2014, 814–830. ISBN: 9783319088662.

[26] Leslie Lamport. "A New Solution of Dijkstra's Concurrent Programming Problem." In: *Commun. ACM* 17.8 (1974), 453–455.

[27] Donald A. Martin. "Borel Determinacy." In: *Annals of Mathematics* 102.2 (1975), pp. 363–371. ISSN: 0003486X.

[28] K. L. McMillan. "Interpolants from Z3 proofs." In: *2011 Formal Methods in Computer-Aided Design (FMCAD)*. 2011, pp. 19–27.

[29] Peter Menzies and Helen Beebee. "Counterfactual Theories of Causation." In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta. Winter 2020. Metaphysics Research Lab, Stanford University, 2020.

[30] M. Minsky. "Recursive Unsolvability of Post's Problem of "Tag" and other Topics in Theory of Turing Machines." In: *Annals of Mathematics* 74 (1961), p. 437.

[31] David Monniaux. "A Quantifier Elimination Algorithm for Linear Real Arithmetic." en. In: *Logic for Programming, Artificial Intelligence, and Reasoning*. Ed. by Iliano Cervesato, Helmut Veith, and Andrei Voronkov. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2008, pp. 243–257. ISBN: 978-3-540-89439-1.

[32] Leonardo de Moura and Nikolaj Bjørner. "Z3: An Efficient SMT Solver." In: *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2008)*. Ed. by C. R. Ramakrishnan and Jakob Rehof. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 337–340. ISBN: 978-3-540-78800-3.

[33] Alberto Pozanco, Yolanda E-Martín, Susana Fernández, and Daniel Borrajo. "Counterplanning using Goal Recognition and Landmarks." In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018*. Ed. by Jérôme Lang. ijcai.org, 2018, pp. 4808–4814. ISBN: 978-0-9992411-2-7.

[34] Mojżesz Presburger. "Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt." In: *Comptes Rendus du I congres de Mathématiciens des Pays Slaves* (1929).

[35] Leonid Ryzhyk, Peter Chubb, Ihor Kuz, Etienne Le Sueur, and Gernot Heiser. "Automatic Device Driver Synthesis with Termite." In: *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*. SOSP '09. Big Sky, Montana, USA: Association for Computing Machinery, 2009, pp. 73–86. ISBN: 9781605587523.

[36] Julian Siber. "Causality-based Model Checking for Real-time Systems." Bachelor's Thesis. Universität des Saarlandes, Saarbrücken, Germany, 2019.

[37] Wolfgang Thomas. "On the synthesis of strategies in infinite games." In: *STACS 95*. Ed. by Ernst W. Mayr and Claude Puech. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 1–13. ISBN: 978-3-540-49175-0.

[38] Tjark Weber, Sylvain Conchon, David Déharbe, Matthias Heizmann, Aina Niemetz, and Giles Reger. "The SMT Competition 2015-2018." In: *J. Satisf. Boolean Model. Comput.* 11.1 (2019), pp. 221–259.