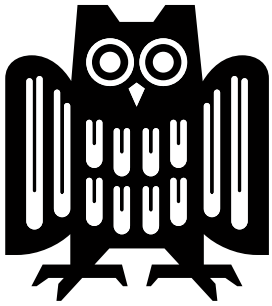
Synthesis and Control of Infinite-State Systems with Partial Observability



Rayna Dimitrova

Fachrichtung Informatik, Fakultät 6

Universität des Saarlandes

Dissertation zur Erlangung des Grades des Doctors der Naturwissenschaften der Naturwissenschaftlich-Technischen Fakultäten der Universität des Saarlandes

Saarbrücken 2013

Tag des Kolloquiums	12.06.2014
Dekan	Prof. Dr. Markus Bläser

Prüfungsausschuss

Vorsitzender	Prof. Dr. Christoph Weidenbach
Berichterstattende	Prof. Bernd Finkbeiner, Ph.D.
	Prof. Rupak Majumdar, Ph.D.
	Prof. Ufuk Topcu, Ph.D.
Akademischer Beisitzer	Dr. Martin Zimmermann

Abstract

Complex computer systems play an important role in every part of everyday life and their correctness is often vital to human safety. In light of the recent advances in the area of formal methods and the increasing availability and maturity of tools and techniques, the use of verification techniques to show that a system satisfies a specified property is about to become an integral part of the development process. To minimize the development costs, formal methods must be applied as early as possible, before the entire system is fully developed, or even at the stage when only its specification is available. The goal of synthesis is to automatically construct an implementation guaranteed to fulfill the provided specification, and, if no implementation exists, to report that the given requirements cannot be realized. When synthesizing an individual component within a system and its external environment, the synthesis procedure must take into account the component's interface and deliver implementations that comply with it. For example, what a component can observe about its environment may be restricted by imprecise sensors or inaccessible communication channels. In addition, sufficiently precise models of a component's environment are typically infinite-state, for example due to modeling real time or unbounded communication buffers. This thesis presents novel synthesis methods that respect the given interface limitations of the synthesized system components and are applicable to infinite-state models.

The studied computational model is that of infinite-state two-player games under incomplete information. The contributions are structured into three parts, corresponding to a classification of such games according to the interface between the synthesized component and its environment. In the first part, we obtain decidability results for a class of game structures where the

player corresponding to the synthesized component has a given finite set of possible observations and a finite set of possible actions. A prominent type of systems for which the interface of a component naturally defines a finite set of observations are Lossy Channel Systems. We provide symbolic game solving and strategy synthesis algorithms for lossy channel games under incomplete information with safety and reachability winning conditions. Our second contribution is a counterexample-guided abstraction refinement scheme for solving infinite-state under incomplete information in which the actions available to the component are still finitely many, but no finite set of possible observations is given. This situation is common, for example, in the synthesis of mutex protocols or robot controllers. In this setting, the observations correspond to observation predicates, which are logical formulas, and their computation is an integral part of our synthesis procedure. The resulting game solving method is applicable to games that are out of the scope of other available techniques. Last we study systems in which, in addition to the possibly infinite set of observation predicates, the component can choose between infinitely many possible actions. Timed games under incomplete information are a fundamental class of games for which this is the case. We extend the abstraction-refinement procedure to develop the first systematic method for the synthesis of observation predicates for timed control. Automatically refining the set of candidate observations based on counterexamples demonstrates better potential than brute-force enumeration of observation sets, in particular for systems where fine granularity of the observations is necessary.

Zusammenfassung

Komplexe Computer Systeme spielen eine wichtige Rolle in jedem Teil des Alltags und ihre Korrektheit ist oft entscheidend für die menschliche Sicherheit. Angesichts der neuesten Fortschritte auf dem Gebiet der formalen Methoden und die zunehmende Verfügbarkeit und Reife von Tools und Verfahren, wird die Verwendung von Techniken zur Prüfung, dass ein System eine bestimmte Eigenschaft erfüllt, zu einem integralen Bestandteil des Entwicklungsprozesses. Um die Entwicklungskosten zu minimieren, sollen formale Methoden so früh wie möglich angewendet werden, bevor das System vollständig entwickelt ist, oder sogar in der Phase, wenn nur seine Spezifikation zur Verfügung steht. Das Ziel von Synthese ist, automatisch eine Implementierung zu konstruieren, die garantiert die gegebene Spezifikation erfüllt. Falls keine solche Implementierung existiert, soll die Unrealisierbarkeit der Spezifikation ausgewiesen werden. Bei der Synthese einer einzelnen Komponente innerhalb eines Systems und seiner äußeren Umgebung müssen synthetisierte Implementierungen die Schnittstelle der Komponente berücksichtigen. Beispielsweise kann eine Komponente ihre Umgebung nur über wenige, unpräzise Sensoren beobachten. Darüber hinaus haben präzise Modelle einer Umgebung einer Komponente normalerweise einen unendlichen Zustandsraum, z.B. durch die Modellierung von Realzeit oder durch unbegrenzte Kommunikationspuffer. Diese Dissertation stellt neuartige Syntheseverfahren für Modelle mit unendlichem Zustandsraum vor, die die Einschränkungen berücksichtigen, die durch die Schnittstelle der synthetisierten Systemkomponenten gegeben sind.

Das grundlegende Berechnungsmodell sind Spiele mit zwei Spielern und einem unendlichen Zustandsraum. Der Beitrag der Dissertation ist in drei

Teile gegliedert. Der erste Teil der Dissertation liefert Entscheidbarkeitsresultate für eine Klasse von Spielen, in der der Spieler, der die Systemkomponente repräsentiert, eine endliche Menge von Beobachtungen und Aktionen hat. Ein prominenter Repräsentant dieser Klasse sind Lossy Channel Systeme. Es werden symbolische Algorithmen zur Strategiesynthese für Lossy Channel Spiele unter unvollständiger Information mit Sicherheits- und Erreichbarkeits-Gewinnzielen präsentiert.

Der zweite Beitrag besteht aus einem Gegenbeispiel-geführten Abstraktionsverfeinerungs-Schema zum Lösen von Spielen mit unendlichem Zustandsraum unter unvollständiger Information, in denen die Komponente endlich viele Aktionen hat aber keine endliche Menge von möglichen Beobachtungen gegeben ist. Diese Situation ist weit verbreitet z.B. bei der Synthese von Mutex-Protokollen oder Robotersteuerungen. In diesem Kontext entsprechen die Beobachtungen Beobachtungsprädikaten, die durch logische Formeln repräsentiert sind, wobei deren Berechnung ein integraler Bestandteil des Syntheseverfahrens ist. Das resultierende Verfahren kann zum Lösen von Spielen benutzt werden, die mit keiner verfügbaren Technik gelöst werden können.

Letztlich werden Systeme untersucht, in denen die Komponente unendlich viele Beobachtungsprädikate hat und zwischen unendlich vielen Aktionen auswählen kann. Gezeitete Spiele unter unvollständiger Information sind eine grundlegende Klasse von Spielen, bei denen dies der Fall ist. Wir erweitern das Abstraktionsverfeinerungs-Schema, um die erste systematische Methode zur Synthese von Beobachtungsprädikaten für gezeitete Controller zu entwickeln. Es wird demonstriert, dass eine Verfeinerung der Beobachtungen, basierend auf Gegenbeispielen, ein höheres Potential aufzeigt als eine Brute-Force-Aufzählung der Beobachtungen, insbesondere für Systeme, bei denen eine feine Granularität der Beobachtungen notwendig ist.

Dedicated to my parents Liliya and Vasil.

Acknowledgements

First and foremost I wish to express my sincere gratitude to my supervisor Bernd Finkbeiner. I am thankful for his guidance, inspiration, unconditional support and patience. He taught me the value of high quality research, and I will always look up to him as a model for researcher and supervisor. He gave me the freedom to pursue my ideas and encouraged me to work on interesting problems and not to give up if they would turn out to be hard.

I am honored and grateful to have Rupak Majumdar and Ufuc Topcu on my thesis committee. I am deeply thankful to Rupak for giving me the excellent opportunity to continue my academic path as a postdoc in his group at MPI-SWS.

I am furthermore grateful for having had the opportunity to work with remarkable people such as Andreas Podelski, Helmut Seidl and Ufuk Topcu.

I want to thank the International Max Planck Research School for Computer Science (IMPRS-CS) and Microsoft Research Cambridge for funding my research with scholarships and the German Science Foundation (DFG) for funding my research as part of the AVACS project.

I want to thank all current and former members of the Reactive Systems group that I had the pleasure to interact with: Jérôme Creci, Klaus Dräger, Rüdiger Ehlers, Peter Faymonville, Michael Gerke, Felix Klein, Lars Kuhtz, Andrey Kupriyanov, Hans-Jörg Peter, Markus Rabe, Christa Schäfer, Sven Schewe, Leander Tentrup, Hazem Torfah, Martin Zimmermann.

A special thanks goes to the members of the Mathematical Logic and Applications group at Sofia University where I did my bachelor studies. Without them opening for me the door to the amazing world of logic and automata theory, I probably would have never come to be where I stand today. This

I mostly own to Ivan Soskov, who will continue to live in the memory of the students whose lives he has enlightened.

Furthermore I would like to thank the members of the Rigorous Software Engineering group at MPI-SWS in Kaiserslautern for welcoming me with an inspiring research environment after my doctoral studies.

For all the fun I had throughout the years that I spent in Saarbrücken, I am incredibly thankful to the great friends I made there (a.k.a. m7++): Laura, Evangelia, Konstantin, André, Yassen, Stefan, Vitaly. A big thanks to another great friend goes to Ruzica for the fun trips, cultural events and other exciting adventures she organized a plenty.

Above all, my most special thanks go to my parents for their faith in me, for the love and encouragement that they have given me, and the valuable lessons that they have taught me.

Thank you all!

Contents

Chapter 1

Introduction

Complex computer systems play an important role in every part of everyday life. Notable examples include fly-by-wire and drive-by-wire technologies, plant controllers, medical devices, smart cards and authentication systems. It is apparent that the correctness of such systems is vital to human safety, privacy and well-being.

To increase the confidence in the correctness of a given system, formal verification techniques can be used to show that the system satisfies some specified properties. Synthesis methods, on the other hand, start with a formal specification of the system's behavior, or a partial implementation, and automatically construct an implementation guaranteed to fulfill the specification, or report that the given requirements cannot be realized, if no such implementation exists. Often, the goal is to synthesize an individual component within a system, such as controller or an individual process in a network of communicating processes, such that the overall system satisfies the specification.

The pioneering works on synthesis of finite-state *closed* [MW80, CE81] and *open* [PR89a] systems were published in the late 80's and the beginning of the 90's. An *open*, also called *reactive* system is a system whose execution is not supposed to terminate, and which during its execution interacts with an external environment. The synthesis of reactive systems from linear time temporal specifications (LTL formulas) is closely related to *Church's solvability problem* [Chu63] and is also solved using automata-theoretic techniques. The automata-theoretic approach to the synthesis of finite-state reactive systems extends also to branching time specifications and to the *incomplete information* setting [KV97]. In this setting one drops the assumption that the synthesized system can perfectly observe its environment. Therefore, the synthe-

sized implementation is required to only depend on information available to the system, e.g., on its input signals. The positive results from the works above do not transfer to the synthesis of distributed systems. Already early on Pnueli and Rosner established in [PR90] that unfortunately the distributed synthesis problem is undecidable.

Now, more than two decades later, synthesis is a very active and promising area of research. Below we give a, by far not complete, overview of the most notable results in the area. These include semi-decision procedures for undecidable synthesis problems, efficient algorithmic solutions and synthesis techniques for important application areas.

Schewe and Finkbeiner [SF07] proposed a semi-decision procedure for the undecidable distributed synthesis problem, which is based on bounding the size of the sought implementation and gradually increasing this bound until an implementation is found. The approach encodes the requirement that the implementation of each process of the distributed system may only use information which is accessible to this process. Furthermore, if an implementation is found, it is guaranteed to have minimal size.

Although decidable, the synthesis of monolithic systems from temporal logic specifications has been considered prohibitively expensive (LTL synthesis is 2EXPTIME-complete). In [PPS06] an expressive fragment of LTL, called GR(1) was identified, for which the synthesis problem can be solved in polynomial time. The specification language and algorithmic approach have found a large number of applications, for example in robotics and autonomous vehicle navigation [KGFP09, WTM10].

In practice, it is not always feasible to synthesize a large system in its entirety from a high-level logical specification. Sometimes, one is interested in repairing an incorrect system, resolving the remaining implementation decisions in a partial program, or synthesizing a certain intricate aspect of the system's behavior, for example synchronization. Automata-theoretic and game-based synthesis techniques were successfully applied to the repair of finite-state programs (possibly the result of the abstraction of complex programs) [JGB05, GBC06, JSGB12, vEJ13]. The game-theoretic approach has also been applied to the synthesis of converters for incompatible protocols [PAHSv02].

The mentioned works on synthesis of monolithic systems, repairs or individual components in a larger system, assume that the system/component can perfectly observe its environment. While dropping this assumption does not increase the complexity of

the synthesis problem when the synthesis problem is presented succinctly as a logical formula (for example LTL synthesis under incomplete information is 2EXPTIME-complete [KV97]), when the partial system and its environment are represented as finite-state machines considering incomplete information incurs an exponential blow-up [Rei84]. This is caused by the fact that the synthesis algorithm has to keep track of the set of global states that are possible given the current local state. To avoid the high complexity, some practical methods look for implementations that do not add extra state to the partial program and hence cannot track the execution history [VYY09]. In the same way methods applied to the synthesis of multi-process programs [CCH$^+$11, CRKB11] avoid the undecidable distributed synthesis problem.

Another application area is discrete controller synthesis, which is closely related to supervisory control of discrete event dynamic systems [RW89]. In supervisory control, partial observability is typically defined using a mask, which is a mapping from the (possibly infinite) state space to a (possibly infinite) space of observations [KGM93]. For general infinite-state systems, the control problem is undecidable [KG05].

The synthesis problem has been studied for some important classes of infinite-state systems used in formal analysis and verification. Timed automata are one such class, used to model systems with quantitative timing information. The game-based algorithm for discrete controller synthesis was extended to the real-time setting in [MPS95]. Later timed control with partial observability was shown too be undecidable [BDMP03].

Other classes of infinite-state systems for which the synthesis problem has been investigated are push-down systems [Cac02] and lossy channel systems [ABd08].

We identify the following two major challenges in reactive systems synthesis.

(I) **Partial observability.** The distributed synthesis problem is undecidable since individual processes in a system have incomplete information and can therefore have incomparable information. Even when the synthesis problem asks for the construction of a single monolithic component, incomplete information is the cause for increased complexity of the synthesis problem. Finally, as already mentioned, partial observability renders undecidable the synthesis problem for otherwise well behaved classes of infinite-state systems like timed automata. Nevertheless, in order to deliver realistic implementations, synthesis algorithms must

take into account the interface limitations of the synthesized system or component that determine what can be observed. For example, the information received by the controller is limited by the precision of its sensors, and a process in a concurrent system cannot access the local variables of the other processes.

(II) **Infinite-state systems.** It is not surprising that the undecidability of a given verification problem for a class of infinite-state systems transfers to the corresponding synthesis problem. Furthermore, in some cases even if the verification problem is decidable, the respective synthesis problem becomes undecidable. For example, the synthesis problem for Petri nets is undecidable even for safety properties with upward-closed sets of error states. However, the precision of infinite-state models is necessary for modeling data from possibly infinite domains, real-time or unbounded communication buffers.

Let us now continue with a simple motivating example illustrating the above points.

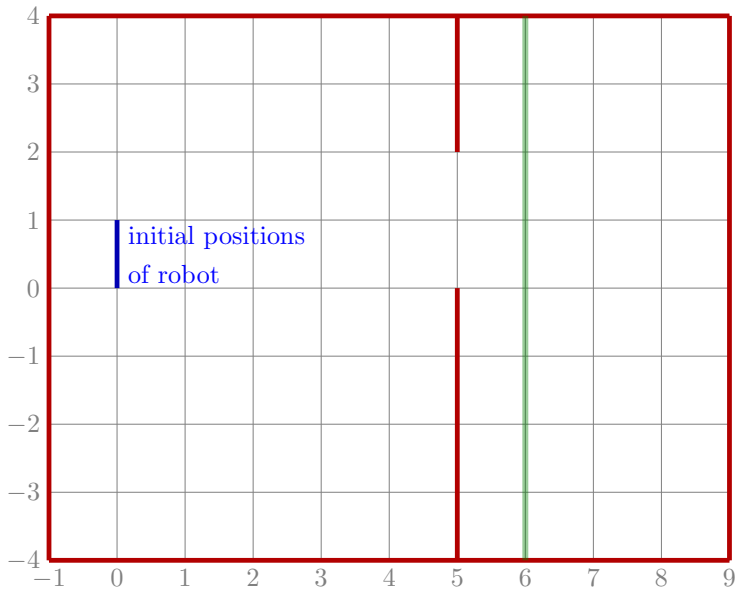


Figure 1.1: A toy example of a robot moving on a plane, navigating in two rooms.

Motivating example. Consider a robot moving on a plane, whose position is described by the coordinates $x \in \mathbb{R}$ and $y \in \mathbb{R}$. The set of initial states is described by a

formula φ_{Init}, and the property that the synthesized robot controller must satisfy is of the form $\Box\neg\varphi_{Err}$, meaning that the robot should avoid some set of error states.

In this setting there is a room that consists of two parts separated by a wall with a passage, as shown in Figure 1.1. The robot starts in a given position (or set of possible initial positions) in the left part of the room and should move to the right side of the room with 4 moves or less. From there on, the robot should move only in the right part of the room. The robot should not crash into the external or internal walls.

At each step the robot chooses to move in one of the four directions: north, south, east or west, indicated by the value of the controllable variable *move*, whose domain $\{N, S, E, W\}$ defines the possible choices of the controller. We assume w.l.o.g. that each action of the robot is enabled in each state. At each step, after the robot selects a direction, the environment executes the choice of the robot, namely the coordinates (x, y) are updated based on the robot's current position and in accordance with the value of the variable *move*. The position y of the robot cannot be observed by the controller (for example, due to sensors imprecision). In this example, there is initial uncertainty about the value of y, but moving south or north is by exactly 1 unit.

However, we will see that even this uncertainty is sufficient to result in the inability to control the system in a way that the error states are avoided. The reason is that the value of y cannot be observed by the controller, and although in a perfect-information game between the controller and its environment the controller has a strategy to win the game by avoiding the error states, there is no control strategy that does not depend on the (initial) value of y. If, on the other hand, we introduce an additional *observable variable* y^o, whose value is initially equal to the value of y, the controller is able to deduce information about the current value of y from the prefix that leads to the state.

The problem scenario described in the example above can be formalized as an infinite-state two-player game under incomplete information.

Two-player games of infinite duration are commonly used as a natural model of the ongoing an non-terminating interaction of a reactive system with its environment. The game is between $Player_\exists$ representing the system/component/controller and $Player_\forall$ that represents its environment. The synthesis problem then reduces to finding a *winning strategy* for $Player_\exists$ in this game (or determining that one does not exist). In

order for such a strategy to correspond to an actual implementation, it might be required to have some additional properties. If the controller cannot perfectly observe its environment, the requirement is that the strategy of $Player_{\exists}$ must be *consistent*, that is,depend only on what the controller can observe about the global system's execution. Thus, in this case the game is played by $Player_{\exists}$ *under incomplete information*.

1.1 Related Work

The classical solution for finite-state reachability games under *incomplete information* is due to Reif [Rei84]. He describes a *knowledge-based subset construction*, similar to automata determinization, that transforms games under incomplete information to perfect-information games. A knowledge-based subset construction for zero-sum games with observable ω-regular winning conditions based on this idea was defined in [CDHR06]. A powerset construction for not necessarily observable ω-regular objectives and also in the asynchronous setting was developed in [Puc10]. Algorithms that carry out this construction explicitly are in general not effective for infinite-state games.

Symbolic fixed-point algorithms based on antichains that avoid Reif's determinization procedure were proposed in [DWDR06, CDHR06]. These methods are effective for special classes of infinite-state games like discrete games on rectangular automata with a fixed finite set of observations [DWDR06]. In such cases they are applied to a given finite region algebra for the input infinite-state game.

The typical approach to treat infinite state spaces in software verification is *abstraction*, where a complex, possibly infinite-state system is mapped to a simpler finite-state one. Such an abstraction is required to be sound, meaning that if the abstract system is determined to be correct, the original system is guaranteed to be correct as well. For a detailed overview of state of-the-art abstraction techniques and in particular *abstraction for perfect-information games* ([HMMR00]), we refer the reader to Section 2.3. In addition to the transition relations of the two players in a game under incomplete information, an abstraction procedure for such games must definitely ensure that the observation equivalence is abstracted in a sound manner as well.

While for infinite-state systems that belong to specific classes one can consider an a priori fixed abstraction that is sufficiently precise to prove many properties of interest (e.g., the region graph of a timed automaton), this is not the case in general.

The key to effective construction of sufficiently precise abstraction is *abstraction refinement*. The most successful and widely used methodology for this is *counterexample-guided abstraction refinement (CEGAR)* [CGJ+00]. Chapter 5 provides pointers to the prominent works on this topic, as well as a comprehensive discussion of related literature on *abstraction refinement for perfect-information games* ([HJM03]). For games with complete information, one builds abstractions that overapproximate the power of the environment (i.e., the hostile player) and underapproximate the power of the controller (or the system's component to be synthesized). Thus, if the controller wins the abstract game it is guaranteed to also win the original one. Otherwise, an abstract counterexample, generated by the solving algorithm, is analyzed to determine whether the controller indeed cannot win the original game, or this counterexample results from the coarseness of the abstraction. In the latter case, the abstraction is refined to ensure the necessary precision that rules out this counterexample from the refined abstraction. For games under incomplete information, the situation is more complicated, because the strategic capabilities of a player depend not only on the available actions, but also on the knowledge about the current state of the game. If the abstract game provides less information to the system than the original one, then the controller might not be able to win the abstract game, because there it is unable to distinguish a certain pair of states and therefore is forced to apply the *same* action in the two states, where in the original game *different* moves could be selected. Therefore, the desired refinement procedure is one that is capable of distinguishing such states to allow for more precise observations which the controller can make in the refined abstraction.

Timed automata are an important class of infinite-state systems that enjoys a number of nice decidability and algorithmic properties, which readily transfer to the timed games setting under the perfect information hypothesis. However, computing a sufficient set of observations (if such exists) in a timed game under incomplete information is a challenging problem, since *timed control under partial observability* is in general undecidable ([BDMP03]). Therefore all known timed synthesis algorithms (discussed in detail in Chapter 6) had assumed an a priori given, fixed, finite set of observations.

For games under incomplete information defined by another well-known class of infinite-state systems, a fixed finite set of observations is defined by the given game, due to the nature of the class of system models. This is the class of *Lossy Channel Systems (LCSs)*, where it is natural to assume that a process has access to one symbol

(at the head) of a channel, which is the symbol it may read from that channel. Thus, since the number of channels is finite, and the message alphabet is also assumed to be finite, the set of possible observations a process can make is guaranteed to be finite. Works on games for LCSs, however, had not considered the incomplete information setting, as it can be seen in the overview of relevant literature given in Chapter 3.

1.2 Contribution

This thesis investigates the problem of automatically synthesizing correct by construction infinite-state systems under the partial observability hypothesis. We study the relation between the phenomenon of partial observability and the possibly infinite size of the sets of possible observations and possible actions of the existential player in the synthesis game. We use the characteristics of the synthesis problem that this relation provides to derive synthesis techniques for the respective types of infinite-state systems.

We begin with the simplest case, when the existential player in an infinite-state synthesis game has an a priori fixed *finite set of observations* and has a *finite set of possible actions*. In many cases the interface of a component naturally defines a finite set of observations. This holds, for example, for LCSs, where components are finite-state and communicate via a finite set of lossy unbounded FIFO channels whose message alphabets are finite. We first study the synthesis problem for this class of systems and define lossy channel games under incomplete information. We show, that games with safety and reachability winning conditions are decidable and finite-state observation-based strategies for the player who has incomplete information can be effectively computed. We describe symbolic game solving and strategy synthesis algorithms based on symbolic representations of upward and downward-closed sets of states. Then, we generalize these decidability results by identifying conditions that define classes of game structures for which incomplete information games with safety and reachability conditions respectively are decidable. We show that the new classes of game structures are not subsumed by classes for which these problems were previously known to be decidable.

In the second part of the thesis we *drop the assumption that a finite set of observations is provided* as input to the synthesis procedure. Thus, the task of computing a set of observations that suffice for realizability (existence of an implementation) becomes the main challenge. Here we consider systems whose sets of transitions are represented

symbolically by formulas in a logical theory, for example linear real arithmetic. In this framework, observations correspond to *observation predicates*, which are boolean expressions that can be used in conditionals in the resulting implementation (thus, we also call such observation predicates *decision predicates*). To address this challenge, we propose the first CEGAR scheme for solving infinite-state two-player safety games under incomplete information. The key idea is to start with some initial finite set of observation predicates and use an iterative procedure to extend it. By making the set of observation predicates a subset of the set of predicates used to construct the abstraction, we incorporate its refinement as part of the procedure for refining the abstraction. The abstraction phase integrates predicate abstraction and the knowledge-based subset construction. The result is a finite-state perfect-information game that soundly approximates the informedness of the component player. We provide a characterization of the concretizability of an abstract counterexample, which is a winning strategy for the environment player, by a first-order strategy-tree formula. In the refinement phase, our approach relies on extracting predicates from Craig interpolants computed for an unsatisfiable strategy-tree formula. In order to refine the informedness of the component player in the abstract game with observable refinement predicates, we develop a novel constraint-based interpolation technique which provides interpolants that meet arbitrary variable partitioning requirements. We establish a progress property of the refinement procedure and identify sufficient conditions that the games and the predicate generation methods need to satisfy to ensure the termination of the CEGAR loop. The resulting method for solving infinite-state two-player safety games under incomplete information is applicable to games that are out of the scope of other techniques available at the time of writing of this thesis.

In the third part of the thesis we turn to a class of systems in which, in addition to the possibly infinite set of observation predicates, the controller has *infinitely many possible actions* at each point. More specifically, we extend the CEGAR-based method to the synthesis of real-time controllers, where the controller has the power to decide how much time should elapse before taking a discrete transition. The main difficulty in this extension lies in the fact that the quantifier alternation resulting from the alternating choices of the two players in the corresponding game can no longer be handled by a straightforward replacement of the quantifiers over the variables whose domains are finite. We address this problem by developing a two-stage abstraction-refinement

procedure. To this end, we interpret the possible timing decisions of the controller as a type of observation predicates over clock variables that describe points in time where the controller does a discrete transition. We term these predicates *action points*, and thus the set of observation predicates is now partitioned into action points and decision predicates. The new abstraction-refinement layer is concerned with the action points and relies on extracting predicates from models of quantified first-order formulas. This is the first systematic method for the synthesis of observation predicates for timed control. Automatically refining the set of candidate observations based on counterexamples demonstrates better potential than brute-force enumeration of observation sets, in particular for systems where fine granularity of the observations is necessary.

1.3 Publications

The contents of this thesis is partially based on the following publications.

- Rayna Dimitrova and Bernd Finkbeiner. **Abstraction Refinement for Games with Incomplete Information**. Proceedings of the *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2008)*.
- Rayna Dimitrova and Bernd Finkbeiner. **Counterexample-guided Synthesis of Observation Predicates**. Proceedings of *10th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2012)*.
- Rayna Dimitrova and Bernd Finkbeiner. **Lossy Channel Games under Incomplete Information**. Proceedings of *1st International Workshop on Strategic Reasoning (SR 2013)*.

1.4 Organization of the Thesis

- We begin the next Chapter 2 by introducing the necessary preliminaries. We continue with the description of the computational model used throughout this thesis for specification and analysis of open systems, namely, two-player zero-sum games under incomplete information. Then we introduce symbolic game structures used to finitely and effectively represent infinite game structures.

We proceed with a discussion of the different ways in which a process of a distributed system observes the execution of its environment. We then give the definition of important notions such as strategy automata and counterexample trees for games with safety winning conditions, and recall the knowledge-based subset construction that transforms a game under incomplete information into a game with perfect information. We define the game solving problem and recall the scheme for solving finite-state safety games with perfect information.

Finally we define abstractions of games under incomplete information and the notion of soundness for such abstraction that takes observability into account.

- The first part of the thesis consists of Chapters 3 and 4, where we consider infinite-state games under incomplete information where the set of observations is a priori fixed, for example by the underlying system model. In Chapter 3 we study games defined by Lossy Channel Systems(LCSs) establishing decidability of incomplete-information games with safety and reachability winning conditions.

- In Chapter 4 we generalize the decidability results from the previous chapter by identifying lasses of game structures for which incomplete information games with safety and reachability conditions respectively are decidable.

- Chapter 5 contains the second major contribution of the thesis. It begins by presenting a sound predicate abstraction procedure for game structures under incomplete information. We describe a procedure for checking if a counterexample in the corresponding abstract safety game has a concretizaton or results from the abstraction being too coarse. In the latter case, the abstraction is refined in a way depending on the cause for the counterexample. We provide an interpolation-based refinement procedure that is used to refine the abstract observation equivalence relation in the case when its approximation is the reason for a nonconcretizable counterexample. We then give the CEGAR loop for safety games under incomplete information and reason about its progress and termination properties. We conclude the chapter with experimental results.

- Chapters 6 and 7 constitute the third part of the thesis, which is devoted to the question of automatically discovering observation predicates for timed control.

Chapter 6 provides the necessary definitions and introduces the timed controller synthesis problem with partial observability.

- Chapter 7 presents the key contribution of this part, namely a systematic method for the automatic synthesis of observation predicates for timed control, which extends the CEGAR approach described in the second part of the thesis.

Chapter 2

Infinite-State Games under Incomplete Information

2.1 Game Model and Representation

In this section we first introduce some basic notions and notation. Then, we describe the computational model, *two-player zero-sum games under incomplete information* used throughout this thesis for specification and analysis of open systems. Finally we introduce *symbolic game structures* as means to finitely represent games over infinite game structures as they arise for example in synthesis of controllers for infinite-state plants, mutex protocols, real-time systems and communication protocols.

2.1.1 Preliminaries

Notation. We denote with $\mathbb{N}$ and $\mathbb{N}_{>0}$ the sets of natural numbers and positive natural numbers, and $\mathbb{B} = \{0, 1\}$ is the set of boolean values. The sets $\mathbb{Q}$, $\mathbb{Q}_{\geq 0}$ and $\mathbb{Q}_{>0}$ consist of the rational numbers, the non-negative rational numbers and the positive rational numbers respectively. $\mathbb{R}$, $\mathbb{R}_{\geq 0}$ and $\mathbb{R}_{>0}$ are the respective sets of real numbers.

For a set A we denote with A^*, A^+, A^ω the sets of finite, finite and nonempty and infinite sequences of elements of A. For a finite or infinite sequence π of elements from a set A we denote with $|\pi|$ its length, which is the number of elements of π when π is finite and is $|\pi| = \infty$ when π is infinite. For $n \in \mathbb{N}$ with $n < |\pi|$, we denote with $\pi[n]$ and with $\pi[0, n]$ the $n+1$-th element of π and the prefix of length $n+1$, respectively. For $\pi \in A^+$, we denote with $\mathsf{last}(\pi)$ the last element of π.

For a variable x we denote with $\mathsf{Dom}(x)$ the domain of x. For a set X of variables, we define $\mathit{Vals}(X) = X \to \mathsf{Dom}(X)$ to be the set all total functions that map each variable $x \in X$ to a value in its domain, i.e., the set of valuations of the variables in X. For each variable x, we assume a variable x' with $\mathsf{Dom}(x') = \mathsf{Dom}(x)$ and for each $i \in \mathbb{N}$ a variable x^i with $\mathsf{Dom}(x^i) = \mathsf{Dom}(x)$. Given a set X of variables, we let $X' = \{x' \mid x \in X\}$ and $X^i = \{x^i \mid x \in X\}$ for $i \in \mathbb{N}$. Since the primed and the indexed versions of a variable have the same domain, we often use elements of $\mathit{Vals}(X)$, $\mathit{Vals}(X')$ and $\mathit{Vals}(X^i)$ for $i \in \mathbb{N}$ interchangeably, with the obvious interpretation.

Logical formulas. A *theory* Th is a set of models over a given signature Ξ. We consider only theories in which all symbols in Ξ are interpreted. Furthermore, in the first part of the thesis we are concerned only with quantifier-free formulas over Ξ.

We denote with $\mathcal{LF}_\Xi$ the set of quantifier-free first order formulas over the signature Ξ and with $\mathcal{LF}_\Xi[X]$ those of them whose variables are elements of the set X of variables. When Ξ is fixed and clear form the context, we write $\mathcal{LF}$ and $\mathcal{LF}[X]$ respectively.

A formula φ over Ξ is *satisfiable w.r.t. a theory* Th if there exists a model of Th in which φ is true. The formula φ *implies a formula* ψ *w.r.t.* Th if in every model of Th in which φ is true, ψ is also true.

For a formula φ, we denote with $\mathit{Vars}(\varphi)$ the set of free variables of φ. For sets X and X' of variables with $X \cap X' = \emptyset$, writing $\varphi[X]$ means that $\mathit{Vars}(\varphi) \subseteq X$ and $\varphi[X, X']$ means that $\mathit{Vars}(\varphi) \subseteq (X \dot\cup X')$. For a formula $\varphi[X]$ and $s \in \mathit{Vals}(X)$ we write $s \models \varphi$ iff φ is true for the valuation s of its variables. We define $[\![\varphi]\!] = \{s \in \mathit{Vals}(X) \mid s \models \varphi\}$. Similarly, for a formula $\varphi[X, X']$, $s \in \mathit{Vals}(X)$ and $s' \in \mathit{Vals}(X')$ we write $(s, s') \models \varphi$ iff φ is true for the valuation $s \dot\cup s'$ of its variables. In this case we define $[\![\varphi]\!] = \{(s, s') \in \mathit{Vals}(X) \times \mathit{Vals}(X') \mid (s, s') \models \varphi\}$.

For a formula φ, variables $x_1, \ldots, x_n$ and terms $e_1, \ldots, e_n$, $\varphi[e_1/x_1, \ldots, e_n/x_n]$ denotes the formula obtained from simultaneously substituting each free occurrence of x_i in φ by e_i for all $1 \leq i \leq n$. For a formula φ and set of variables X we denote with $\varphi[X'/X]$ the formula obtained by replacing each free occurrence of x by x' for all $x \in X$ and if $i \in \mathbb{N}$ we denote with $\varphi[X^i/X]$ the formula obtained by replacing each free occurrence of x by x^i for all $x \in X$.

Linear arithmetic. The theory of linear rational (real) arithmetic is defined by the signature $\Xi_{LRA} = \{0, 1, -, +, \leq\}$, where 0 and 1 have arity zero, $-$ has arity one and $+$ and $\leq$ have arity two, and a set axioms. The signature can be extended by further symbols while preserving expressiveness. We assume that the symbols $<$, $\geq$, $>$, $=$, $\neq$, $-$ with arity two, $c.$ with arity one and c with arity zero, for $c \in \mathbb{Q}$ ($c \in \mathbb{R}$), are also part of the signature Ξ_{LRA}. A linear inequality is a formula of the form $a_1x_1 + \ldots + a_nx_n \lhd a$, where $n \in \mathbb{N}$, $x_1, \ldots, x_n$ are variables, $a \in \mathbb{Q}$, $a_1, \ldots, a_n \in \mathbb{Q}$ and $\lhd \in \{<, \leq\}$.

Given a set X of variables such that $\mathsf{Dom}(x) = \mathbb{R}_{\geq 0}$ for each $x \in X$, $\mathcal{B}[X]$ is the set of formulas, where $x \in X$, $c \in \mathbb{Q}$, that are generated by the following grammar:

$$\varphi := \mathit{true} \mid x < c \mid x \leq c \mid x > c \mid x \geq c \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi.$$

$\mathcal{C}[X]$ is the subset of $\mathcal{B}(X)$ that is defined by the grammar:

$$\varphi := \mathit{true} \mid \mathit{false} \mid x < c \mid x \leq c \mid x > c \mid x \geq c \mid \varphi \wedge \varphi.$$

Finally, $\mathcal{D}[X]$ is the set of formulas, where $x, x_1, x_2 \in X$, $c \in \mathbb{Q}$ and $\sim \in \{<, \leq, >, \geq\}$, that are generated by the following grammar:

$$\varphi := \mathit{true} \mid x \sim c \mid x_1 - x_2 \sim c \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi.$$

2.1.2 Two-Player Games

The computational model that we consider is that of *two-player zero-sum games under incomplete information*. The games are played for an infinite duration which makes them a suitable model for open reactive systems, which never terminate. The two players in such a game correspond to a system (or a component of a system) and its environment. $\mathit{Player}_\exists$ stands for the system (component) and resolves the *friendly nondeterminism*, playing the game under incomplete information. $\mathit{Player}_\exists$, who represents the environment, on the other hand, has complete (perfect) information. This asymmetric situation is quite common, for example in controller synthesis, where the *controller* observes the plant via imprecise sensors and thus has incomplete information about the global state. The *plant*, on the other hand, has perfect information, since the controller must guarantee the property against all possible behaviors of its environment, and this is not influenced by the observation capabilities of the plant.

We first define game structures with perfect information, in which both players can completely observe the state of the game, and later extend these with an *observation equivalence relation* that defines the observations that $Player_\exists$ is able to make.

Definition 2.1.1. A *game structure with perfect information* $G = (S_\exists, S_\forall, I, \Sigma_\exists, T_\exists, T_\forall)$ consists of a set $S_\exists$ of $Player_\exists$ states, a set $S_\forall$ of $Player_\forall$ states, a set $I \subseteq S$ of initial states, where $S = S_\exists \dot\cup S_\forall$, a set $\Sigma_\exists$ of $Player_\exists$ actions and transition relations $T_\exists \subseteq S_\exists \times \Sigma_\exists \times S_\forall$ and $T_\forall \subseteq S_\forall \times S$ for $Player_\exists$ and $Player_\forall$ respectively.

We define $T = \{(s, s') \in S \times S \mid (s, s') \in T_\forall \text{ or } (s, \sigma, s') \in T_\exists \text{ for some } \sigma \in \Sigma_\exists\}$.

The games that we consider are *turn-based*, that is, the game proceeds in discrete steps, where at each step one of the players makes a move. The game starts in some nondeterministically chosen initial state, and the moves made by the two players build up a play which is a finite or infinite sequence of states. If the current state is in $S_\exists$, then it is $Player_\exists$'s turn to make a move and if the current state is in $S_\forall$, then it is $Player_\forall$'s turn. Note that the target states of a $Player_\exists$ transitions are always $Player_\forall$ states, while the target states of $Player_\forall$ transitions can belong to either player. This allows us to model the interaction from the point of view of a controller (system component), where (a possibly unbounded number of) multiple steps can happen between the controller's transitions, without summarizing sequences of environment transitions. A move of $Player_\forall$ at a state $s \in S_\forall$ consists of choosing a successor of s w.r.t. the transition relation $T_\forall$, from where the game will continue. Moves of player $Player_\exists$ in a state $s \in S_\exists$, on the other hand, consist of choosing an action $\sigma \in \Sigma_\exists$, for which a *$\sigma$-successor*, that is a state s' with $(s, \sigma, s') \in T_\exists$, exists. The next state of the play is then *nondeterministically* chosen among the σ-successors of s.

A game structure $G = (S_\exists, S_\forall, I, \Sigma_\exists, T_\exists, T_\forall)$ is *$\Sigma_\exists$-deterministic* iff for every $s \in S_\exists$, $\sigma \in \Sigma_\exists$ and $s'_1, s'_2 \in S$, if $(s, \sigma, s'_1) \in T_\exists$ and $(s, \sigma, s'_2) \in T_\exists$, then $s'_1 = s'_2$. A game structure is *deterministic* if it is *$\Sigma_\exists$-deterministic* and the set I is a singleton.

Definition 2.1.2. A *game structure with incomplete information* $G = (S_\exists, S_\forall, I, =_o, \Sigma_\exists, T_\exists, T_\forall)$ additionally specifies an *observation equivalence* $=_o \subseteq S \times S$ on the set of states S. The components $S_\exists$, $S_\forall$, I, $\Sigma_\exists$, $T_\exists$ and $T_\forall$ are as in game structures with perfect information. The relation $=_o$ is required to satisfy the following conditions:

(i) For $s_1, s_2 \in S$, if $s_1 =_o s_2$, then either $s_1 \in S_\exists$ and $s_2 \in S_\exists$ or $s_1 \in S_\forall$ and $s_2 \in S_\forall$. That is, the relation $=_o$ respects the partitioning of S into $S_\exists$ and $S_\forall$.

(ii) For $s \in S_\exists$, $\sigma_1, \sigma_2 \in \Sigma_\exists$ and $s'_1, s'_2 \in S_\forall$, if $(s, \sigma_1, s'_1) \in T_\exists$, $(s, \sigma_2, s'_2) \in T_\exists$ and $\sigma_1 \neq \sigma_2$, and $s'_1 \neq s'_2$ then $s'_1 \neq_\mathsf{o} s'_2$, i.e., $Player_\exists$ distinguishes different successors.

(iii) For $s_1, s_2 \in S_\exists$ and $\sigma \in \Sigma_\exists$, if $s_1 =_\mathsf{o} s_2$ and there exists $s'_1 \in S_\forall$ such that $(s_1, \sigma, s'_1) \in T_\exists$, then there exists a $s'_2 \in S_\forall$ such that $(s_2, \sigma, s'_2) \in T_\exists$. That is, the set of available actions is the same for all observation-equivalent $Player_\exists$ states.

Conditions (i)-(iii) specify our requirements on the observation abilities of $Player_\exists$.

Let Obs be the set of equivalence classes of $=_\mathsf{o}$ in G. For a state $s \in S$ we denote with $obs(s)$ the equivalence class to which s belongs. In a game under incomplete information $Player_\exists$ does not observe the current state s of the game, but only $obs(s)$.

When $=_\mathsf{o}$ is the quality relation, the game structure is essentially with perfect information. When it is irrelevant or clear from the context, we write simply game structure. Often, we use a tuple $(S_\exists, S_\forall, I, =_\mathsf{o}, \Sigma_\exists, T_\exists, T_\forall)$ to describe a game structure with perfect information, which we explicitly mention, meaning that $=_\mathsf{o}$ is equality.

Definition 2.1.3. Given a game structure $G = (S_\exists, S_\forall, I, =_\mathsf{o}, \Sigma_\exists, T_\exists, T_\forall)$, we define the following *successor and predecessor functions* on sets of states.

- $\mathsf{Post}_\exists : 2^S \times \Sigma_\exists \to 2^S$, where

$$\mathsf{Post}_\exists(R, \sigma) = \{s' \in S \mid \exists s \in S.\ s \in R \text{ and } (s, \sigma, s') \in T_\exists\},$$

- $\mathsf{Post}_\forall : 2^S \to 2^S$, where

$$\mathsf{Post}_\forall(R) = \{s' \in S \mid \exists s \in S.\ s \in R \text{ and } (s, s') \in T_\forall\},$$

- $\mathsf{Post} : 2^S \to 2^S$, where $\mathsf{Post}(R) = \big(\bigcup_{\sigma \in \Sigma_\exists} \mathsf{Post}_\exists(R, \sigma)\big) \cup \mathsf{Post}_\forall(R)$,

- $\mathsf{Pre}_\exists : 2^S \times \Sigma_\exists \to 2^S$, where

$$\mathsf{Pre}_\exists(R, \sigma) = \{s \in S_\exists \mid \exists s' \in S.\ s' \in R \text{ and } (s, \sigma, s') \in T_\exists\},$$

- $\mathsf{Pre}_\forall : 2^S \to 2^S$, where

$$\mathsf{Pre}_\forall(R) = \{s \in S_\forall \mid \exists s' \in S.\ s' \in R \text{ and } (s, s') \in T_\forall\}.$$

The function $\mathsf{Enabled} : \Sigma_\exists \to S_\exists$ is defined as $\mathsf{Enabled}(\sigma) = \mathsf{Pre}_\exists(S, \sigma)$.

Paths, plays and prefixes Let $G = (S_\exists, S_\forall, I, =_\mathsf{o}, \Sigma_\exists, T_\exists, T_\forall)$ be a game structure with perfect or incomplete information. A *path* in G is a finite or infinite sequence $\pi \in S^* \cup S^\omega$ of states such that $(\pi[i], \pi[i+1]) \in T$ for all $0 \leq i < |\pi| - 1$. A state $s \in S$ for which there is no state $s' \in S$ with $(s, s') \in T$ is called a *dead-end*. We call a path π in G *maximal* iff π is infinite or it is finite and $\mathsf{last}(\pi)$ is a dead-end. A *play* in G is a maximal path π in G such that $\pi[0] \in I$. A *prefix* in G is a finite path π in G such that $\pi[0] \in I$. We denote with $\mathsf{Prefs}(G)$ the set of prefixes in G and let $\mathsf{Prefs}_\exists(G) = \{\pi \in \mathsf{Prefs}(G) \mid \mathsf{last}(\pi) \in S_\exists\}$ and $\mathsf{Prefs}_\forall(G) = \{\pi \in \mathsf{Prefs}(G) \mid \mathsf{last}(\pi) \in S_\forall\}$.

For a path π we denote with $\pi|_\exists$ the projection of π on $S_\exists$ and define $|\pi|_\exists = |\pi|_\exists|$.

Winning conditions For a game structure $G = (S_\exists, S_\forall, I, =_\mathsf{o}, \Sigma_\exists, T_\exists, T_\forall)$, a *winning condition* is a set $\Omega \subseteq S^\omega \cup S^*$ that defines the set of sequences of states that are *winning for Player*$_\exists$. We consider only *zero-sum* games, in which a play π is winning for *Player*$_\forall$ iff it is not winning for player *Player*$_\exists$. i.e., the set of sequences of states won by *Player*$_\forall$ is $(S^\omega \cup S^*) \setminus \Omega$. In this thesis we focus mainly on games with *safety* winning conditions for *Player*$_\exists$ defined by a set *Err* of error states that *Player*$_\exists$ has to avoid. In the first part we will also look at games with *reachability* winning conditions for *Player*$_\exists$ defined by a set *Goal* of goal states that *Player*$_\exists$ has to reach.

Definition 2.1.4. A game structure G with perfect information (incomplete information) and a set of states *Err* in G define a *safety game with perfect information* (*safety game with incomplete information*) $\mathsf{Safety}(G, Err) = (G, \Omega_{Err})$, where the set of sequences of states winning for *Player*$_\exists$ is $\Omega_{Err} = \{\pi \in S^\omega \mid \forall i \geq 0.\ \pi[i] \notin Err\}$.

A play π in G is *winning for Player*$_\exists$ in $\mathsf{Safety}(G, Err)$ iff $\pi \in \Omega_{Err}$.

Definition 2.1.5. A game structure G with perfect information (incomplete information) and a set of states *Goal* in G define a *reachability game with perfect information* (*reachability game with incomplete information*) $\mathsf{Reach}(G, Goal) = (G, \Omega_{Goal})$, where the set of state sequences winning for *Player*$_\exists$ is $\Omega_{Goal} = \{\pi \in S^\omega \cup S^* \mid \exists 0 \leq i < |\pi|.\ \pi[i] \in Goal\}$. A play π in G is *winning for Player*$_\exists$ in $\mathsf{Reach}(G, Err)$ iff $\pi \in \Omega_{Goal}$.

The classical way to handle finite plays is to treat them separately from the winning condition Ω or to assume that the game structure does not contain dead-ends. In definitions 2.1.1 and 2.1.2 we do not disallow dead-end states. For the purpose of considering incomplete-information games in which *Player*$_\exists$ has a safety winning condition, we can assume that the game structure does not contain dead-end states that

belong to $Player_\forall$. Note, however that we do not, in general, make such an assumption for $Player_\exists$ dead-end states. The reason is that even if a game structure G is such that $S_\exists$ does not contain dead-ends, an abstraction of G that underapproximates the possible choices of $Player_\exists$ may introduce dead-end sates for $Player_\exists$, unless certain level of precision is required by the abstraction function.

Strategies. Let $G = (S_\exists, S_\forall, I, =_\mathsf{o}, \Sigma_\exists, T_\exists, T_\forall)$ be a game structure.

For a state $s \in S_\exists$ we define $\mathsf{Enabled}(s, G) = \{\sigma \in \Sigma_\exists \mid \exists s' \in S.\ (s, \sigma, s') \in T_\exists\}$ and for a set $R \subseteq S$ of $Player_\exists$ states we let $\mathsf{Enabled}(R, G) = \bigcap_{s \in R} \mathsf{Enabled}(s, G)$. Note that condition (iii) from Definition 2.1.2 implies that $\mathsf{Enabled}(s_1, G) = \mathsf{Enabled}(s_2, G)$ for every $s_s, s_2 \in S_\exists$ with $s_1 =_\mathsf{o} s_2$. Thus, for each set $R \subseteq o$ for some $o \in Obs$ it holds that $\mathsf{Enabled}(R, G) = \mathsf{Enabled}(s, G)$ for each $s \in R$.

A partial function $f_\exists : \mathsf{Prefs}_\exists(G) \to \Sigma_\exists$ is *non-blocking* if it is defined for every prefix $\pi \in \mathsf{Prefs}_\exists(G)$ for which $\mathsf{last}(\pi)$ is not a dead-end. A partial function $f_\forall : \mathsf{Prefs}_\forall(G) \to S$ is non-blocking if it is defined for every $\pi \in \mathsf{Prefs}_\forall(G)$ for which $\mathsf{last}(\pi)$ is not a dead-end.

Definition 2.1.6. A *strategy for* $Player_\exists$ is a partial function $f_\exists : \mathsf{Prefs}_\exists(G) \to \Sigma_\exists$ that is non-blocking and $f_\exists(\pi) \in \mathsf{Enabled}(\mathsf{last}(\pi), G)$ for every $\pi \in \mathsf{Prefs}_\exists(G)$.

Definition 2.1.7. A *strategy for* $Player_\forall$ is a partial function $f_\forall : \mathsf{Prefs}_\forall(G) \to S$ that non-blocking and $(\mathsf{last}(\pi), f_\forall(\pi)) \in T_\forall$ for every $\pi \in \mathsf{Prefs}_\exists(G)$.

A path π in the game structure G *complies with a strategy* $f_\exists$ for $Player_\exists$ iff for every $0 \leq i < |\pi| - 1$ for which $\pi[i] \in S_\exists$ it hods that $(\pi[i], f_\exists(\pi[0, i]), \pi[i + 1]) \in T_\exists$. A path π in the game structure G *complies with a strategy* $f_\forall$ for $Player_\forall$ iff for every $0 \leq i < |\pi| - 1$ for which $\pi[i] \in S_\forall$ it hods that $\pi[i + 1] = f_\forall(\pi[0, i])$.

For a strategy $f_\exists$ for $Player_\exists$ we denote with $\mathsf{Prefs}(f_\exists)$ the set of prefixes in $\mathsf{Prefs}(G)$ that comply with $f_\exists$ and for a strategy $f_\forall$ for $Player_\forall$ we denote with $\mathsf{Prefs}(f_\forall)$ the set of prefixes in $\mathsf{Prefs}(G)$ that comply with $f_\forall$.

The outcome of a strategy $f_\exists$ for $Player_\exists$ is a set of plays $\mathsf{Outcome}(f_\exists)$ such that $\pi \in \mathsf{Outcome}(f_\exists)$ iff π complies with $f_\exists$. The outcome of a strategy $f_\forall$ for $Player_\forall$ is a set of plays $\mathsf{Outcome}(f_\forall)$ such that $\pi \in \mathsf{Outcome}(f_\forall)$ iff π complies with $f_\forall$. The outcome of strategies $f_\exists$ and $f_\forall$ for $Player_\exists$ and $Player_\forall$, respectively, is a set of plays $\mathsf{Outcome}(f_\exists, f_\forall)$ such that $\pi \in \mathsf{Outcome}(f_\exists, f_\forall)$ iff π complies with $f_\exists$ and $f_\forall$.

Definition 2.1.8. A strategy $f_\exists$ for $Player_\exists$ in the game $\mathsf{Safety}(G, Err)$ is *winning for* $Player_\exists$ iff every play $\pi \in \mathsf{Outcome}(f_\exists)$ π is winning for $Player_\exists$, i.e., $\pi \in \Omega_{Err}$. A strategy $f_\forall$ for $Player_\forall$ in $\mathsf{Safety}(G, Err)$ is *winning for* $Player_\forall$ iff every play $\pi \in \mathsf{Outcome}(f_\forall)$ is winning for $Player_\forall$, that is, $\pi \notin \Omega_{Err}$.

2.1.3 Symbolic Representation

Since the game structures we study in this thesis are potentially infinite graphs, they are described symbolically, using logical formulas from a fixed theory. In this subsection we define symbolic game structures that serve as such a finite representation.

The communication between $Player_\exists$ and $Player_\forall$ is modeled by a finite set V of *variables*, which is partitioned into three disjoint sets: $V_\exists$, $V_\forall$ and $\{t\}$. The idea behind this partitioning is the following. $Player_\forall$ updates (and can observe) the variables in $V_\forall \dot\cup \{t\}$ and $Player_\exists$ updates (and can observe) the variables in $V_\exists \dot\cup \{t\}$. The subset $V_\forall^o \subseteq V_\forall$ consists of the input variables for $Player_\exists$, i.e., these are variables whose values $Player_\exists$ can read but not update. The remaining variables in $V_\forall$, i.e., the variables in $V_\forall \setminus V_\forall^o$ are private variables for $Player_\forall$ and cannot be observed by $Player_\exists$. The variables in $V_\exists$ are the output variables of $Player_\exists$ (which $Player_\forall$ can only read). The value of the variable t determines whose turn it is to make a move.

For a set of variables $X \subseteq V$ we denote with $Obs(X) = X \cap (V_\exists \dot\cup V_\forall^o \dot\cup \{t\})$ the set of variables in X whose values $Player_\exists$ can observe. The function Obs extends to sets of variables that contain variables from V' or indexed versions of the variables V.

Definition 2.1.9. A *symbolic game structure* $\mathcal{G} = (V_\exists, V_\forall, V_\forall^o, t, \varphi_{Init}, \mathcal{T}_\exists, \mathcal{T}_\forall)$ consists of disjoint sets of variables $V_\exists$ and $V_\forall$, set $V_\forall^o \subseteq V_\forall$ of observable variables, a variable $t \notin (V_\exists \dot\cup V_\forall)$, a formula φ_{Init} over V, and formulas $\mathcal{T}_\exists$ and $\mathcal{T}_\forall$ over $V \cup V'$.

A symbolic game structure is *well-defined* iff it satisfies the following conditions:

(1) For every $s \in Vals(V)$ and $s' \in Vals(V')$ the following hold:

- if $(s, s') \in [\![\mathcal{T}_\exists]\!]$, then $s(t) = \exists$, $s'(t) = \forall$ and $s(V_\forall) = s'(V_\forall)$ and
- if $(s, s') \in [\![\mathcal{T}_\forall]\!]$, then $s(t) = \forall$ and $s(V_\exists) = s'(V_\exists)$.

(2) For every $s_1, s_2 \in Vals(V)$ with $obs(s_1) = obs(s_2)$ and $s_1' \in Vals(V')$, if $(s_1, s_1') \in [\![\mathcal{T}_\exists]\!]$ there exists a $s_2' \in Vals(V')$ such that $(s_2, s_2') \in [\![\mathcal{T}_\exists]\!]$ and $obs(s_1') = obs(s_2')$.

In this thesis we consider only well-defined symbolic game structures and from now on, whenever we say that a $\mathcal{G}$ is a symbolic game structure we assume it is well-defined.

A well-defined symbolic game structure $\mathcal{G} = (V_\exists, V_\forall, V_\forall^o, t, \varphi_{Init}, \mathcal{T}_\exists, \mathcal{T}_\forall)$ defines a game structure with incomplete information $G = (S_\exists, S_\forall, I, \Sigma_\exists, T_\exists, T_\forall, =_o)$ as follows:

- $S_\exists = \{s \in \mathit{Vals}(V) \mid s(t) = \exists\}$,
- $S_\forall = \{s \in \mathit{Vals}(V) \mid s(t) = \forall\}$,
- $I = \{s \in \mathit{Vals}(V) \mid s \models \varphi_{Init}\}$,
- $\Sigma_\exists = \mathit{Vals}(V_\exists)$,
- $T_\exists = \{(s, \sigma, s') \in \mathit{Vals}(V) \times \mathit{Vals}(V_\exists) \times \mathit{Vals}(V') \mid (s, s') \in [\![\mathcal{T}_\exists]\!] \text{ and } s'(V_\exists) = \sigma\}$,
- $T_\forall = \{(s, s') \in \mathit{Vals}(V) \times \mathit{Vals}(V') \mid (s, s') \in [\![\mathcal{T}_\forall]\!]\}$,
- $s_1 =_o s_2$ iff $s_1(\mathit{Obs}(V)) = s_2(\mathit{Obs}(V))$.

Since $\mathcal{G}$ is well-defined, G satisfies the requirements of Definition 2.1.2. Note that if $V_\forall^o = V_\forall$, then G is a game structure with perfect information.

Remark. We could, in general, omit the requirement that $\mathit{Player}_\forall$ cannot modify the portion of the state that belongs to $\mathit{Player}_\exists$, i.e., the variables $V_\exists$, from the definition of well-defined symbolic game structure. That is, we could leave out the condition stating that $(s, s') \in [\![\mathcal{T}_\forall]\!]$ implies $s(V_\exists) = s'(V_\exists)$. In this case the above definition of the corresponding explicit game structure remains the same and the requirements of Definition 2.1.2 are still met. We would furthermore like to point out that the results established in this part of the thesis still hold with this more general definition.

The explicit game structure G defined by $\mathcal{G}$ has the following property.

Property 1. *If $s_1, s_2 \in S_\exists$ are such that $s_1 =_o s_2$ and $s_1', s_2' \in S$ are such that $(s_1, \sigma, s_1') \in T_\exists$ and $(s_2, \sigma, s_2') \in T_\exists$ for some $\sigma \in \Sigma_\exists$, then it holds that $s_1' =_o s_2'$.*

If $s \in S_\exists$ and $\sigma \in \Sigma_\exists$, then there exists at most one $s' \in S$ such that $(s, \sigma, s) \in T_\exists$.

If $s_1, s_2 \in S_\exists$ are such that $s_1 =_o s_2$ and $s_1', s_2' \in S$ are such that $(s_1, \sigma_1, s_1') \in T_\exists$ and $(s_2, \sigma_2, s_2') \in T_\exists$ for some $\sigma_1, \sigma_2 \in \Sigma_\exists$, then $\sigma_1 \neq \sigma_2$ implies that $s_1' \neq_o s_2'$.

Given a symbolic game structure $\mathcal{G}$ and a formula φ_{Err}, the *symbolic safety game* $\mathsf{Safety}(\mathcal{G}, \varphi_{Err})$ is a finite representation of the safety game $\mathsf{Safety}(G, [\![\varphi_{Err}]\!])$.

Example 2.1.1. Let us consider the motivating example from the introduction. The interaction between the robot controller and its environment is modeled as the symbolic game structure $\mathcal{G} = (V_\exists, V_\forall, V_\forall^o, t, \varphi_{Init}, \mathcal{T}_\exists, \mathcal{T}_\forall)$ defined as follows.

The output variables of are $V_\exists = \{move\}$ and the input (observable) variables are $V_\forall^o = \{x\}$. The unobservable by the controller variables are $V_\forall \setminus V_\forall^o = \{y, steps, err\}$.

The set of initial states is described symbolically by the formula

$$\varphi_{Init} \;:=\; t = \exists \;\wedge\; move = N \;\wedge\; x = 0 \;\wedge\; y \geq 0 \;\wedge\; y \leq 1 \;\wedge\; steps = 0 \;\wedge\; \neg err.$$

The transition relation of $Player_\forall$ is described by the formula

$$\begin{aligned}\mathcal{T}_\forall \;:=\;& t = \forall \wedge t' = \exists \wedge move' = move \wedge (\varphi_N \vee \varphi_S \vee \varphi_E \vee \varphi_W) \wedge \\ & steps' = steps + 1 \wedge \big((\varphi_{bad} \wedge err') \vee (\neg\varphi_{bad} \wedge err' = err)\big), \text{ where}\end{aligned}$$

$$\varphi_N := move = N \wedge y' = y + 1 \wedge x' = x,$$

$$\varphi_S := move = S \wedge y' = y - 1 \wedge x' = x,$$

$$\varphi_E := move = E \wedge x' = x + 2 \wedge y' = y,$$

$$\varphi_W := move = W \wedge x' = x - 2 \wedge y' = y,$$

$$\varphi_{bad} := \varphi_{hit-wall} \vee \varphi_{go-back}, \text{ where}$$

$$\varphi_{hit-wall} := (x < 5 \wedge x' \geq 5 \vee x > 5 \wedge x' \leq 5) \wedge (y \leq 0 \vee y \geq 2),$$

$$\varphi_{go-back} := x > 5 \wedge x' \leq 5.$$

The transition relation of $Player_\exists$ is described by the formula

$$\begin{aligned}\mathcal{T}_\exists \;:=\;& t = \exists \wedge t' = \forall \wedge (move' = N \vee move' = S \vee move' = E \vee move' = W) \wedge \\ & x' = x \wedge y' = y \wedge steps' = steps \wedge err' = err.\end{aligned}$$

The winning condition for $Player_\exists$ is defined by the formula φ_{Err} describing error states $\varphi_{Err} := err = true \vee (steps > 3 \wedge x < 6) \vee x \geq 9 \vee x \leq -1 \vee y \geq 4 \vee y \leq -4$. □

Definition 2.1.10. A symbolic game structure $\mathcal{G} = (V_\exists, V_\forall, V_\forall^o, t, \varphi_{Init}, \mathcal{T}_\exists, \mathcal{T}_\forall)$ is *effective* if the following functions can be defined and can be effectively computed.

- $\mathsf{Post}_\exists : \mathcal{LF}_\Xi[V] \times \Sigma_\exists \to \mathcal{LF}_\Xi[V]$, where $[\![\mathsf{Post}_\exists(\varphi, \sigma)]\!] = \mathsf{Post}_\exists([\![\varphi]\!], \sigma)$,
- $\mathsf{Post}_\forall : \mathcal{LF}_\Xi[V] \to \mathcal{LF}_\Xi[V]$, where $[\![\mathsf{Post}_\forall(\varphi)]\!] = \mathsf{Post}_\forall([\![\varphi]\!])$,
- $\mathsf{Pre}_\exists : \mathcal{LF}_\Xi[V] \times \Sigma_\exists \to \mathcal{LF}_\Xi[V]$, where $[\![\mathsf{Pre}_\exists(\varphi, \sigma)]\!] = \mathsf{Pre}_\exists([\![\varphi]\!], \sigma)$,

- $\mathsf{Pre}_\forall : \mathcal{LF}_\Xi[V] \to \mathcal{LF}_\Xi[V]$, where $[\![\mathsf{Pre}_\forall(\varphi)]\!] = \mathsf{Pre}_\forall([\![\varphi]\!])$.

If $\mathcal{G} = (V_\exists, V_\forall, V_\forall^o, t, \varphi_{Init}, \mathcal{T}_\exists, \mathcal{T}_\forall)$ is an effective symbolic game structure and $\Sigma_\exists = \mathit{Vals}(V_\exists)$, we can effectively compute the function $\mathsf{Enabled} : \Sigma_\exists \to \mathcal{LF}_\Xi[V]$ such that $[\![\mathsf{Enabled}(\sigma)]\!] = \{s \in \mathit{Vals}(V) \mid \exists s' \in \mathit{Vals}(V).\ (s, s') \models \mathcal{T}_\exists \wedge \forall x \in V_\exists.\ \sigma(x) = s'(x)\}$.

Note that if there exists a quantifier elimination procedure for the underlying logical theory Th, then the symbolic game structure is effective.

2.1.4 Discussion of the Game Model

The literature on two-player games on graphs contains a variety of definitions of game structures. In the perhaps most common setting, the game is defined by a directed graph, usually without labels on the edges, in which the vertices are partitioned among the players. In this model, at each step of the game the respective player chooses a successor node from which the game is to continue. Another definition adopted in a number of works, e.g. [DR11], employs a directed graph whose edges are labeled with the elements of a finite set of actions. The vertices are not partitioned according to the players, but at each step the first player selects an action and the second player selects one of the successors of the current node w.r.t. this action. Here, we chose to consider another variation, where the edges originating from $\mathit{Player}_\exists$ nodes in the game graph are labeled with actions from a set $\Sigma_\exists$ (the possible choices of $\mathit{Player}_\exists$), while the ones originating from $\mathit{Player}_\forall$ nodes are not labeled and $\mathit{Player}_\forall$ is responsible for choosing the actual successor of the node. We decided to define explicitly the choices of $\mathit{Player}_\exists$ as a set of actions $\Sigma_\exists$, as this allows us to explicitly refer to these choices and to easily distinguish the cases when the set of choices of $\mathit{Player}_\exists$ is finite or infinite.

Note that, according to the game rules we gave, the successor of a $\mathit{Player}_\exists$ state once $\mathit{Player}_\exists$ has selected an action from $\Sigma_\exists$ is chosen nondeterministically. For generality we have allowed $\mathit{Player}_\exists$ states to have multiple successors with a particular action. However, the games we will consider in this thesis will be $\Sigma_\exists$-deterministic. That, is the game structures will be defined in such a way that an action of $\mathit{Player}_\exists$ that is enabled in a given state will determine a unique successor of that state. For example, game structures defined by symbolic game structures are always $\Sigma_\exists$-deterministic.

Definition 2.1.2 states the properties that the observation equivalence is required to satisfy. Conditions (i) and (ii) are common and are fulfilled by typical game models.

Condition (iii), namely that $Player_\exists$ knows the actions available to him, is sometimes omitted in definitions from the literature. However, we will see that it is rather natural. For game structures defined by symbolic game structures this requires that the guards of $Player_\exists$ actions refer only to observable variables, which is usually the case.

2.2 Games under Incomplete Information

2.2.1 Observation-based Strategies

Observations and Synchronization The communication and synchronization between the processes of a distributed system or a plant and its controller influence observability. In a synchronous setting a process reads an input at each step, thus observing each change of its environment. In the asynchronous setting of [PR89b, SF06] a process performs an input/output operation independently of its environment and may thus be unaware of some changes of its inputs. In [Puc10], on the other hand, asynchronism is defined via observability, and a player is aware of exactly those transitions that have an effect that is observable by him. This is also the observation model used for timed control under partial observability in [CDL$^+$07]. There the controller is oblivious to the number of stuttering observations arising from sequences of timed transitions and unobservable changes of the internal state of the plant.

Let $G = (S_\exists, S_\forall, I, =_o, \Sigma_\exists, T_\exists, T_\forall)$ be a game structure with incomplete information. Recall that the function $obs : S \to Obs$ maps a state s to the corresponding equivalence class observed by $Player_\exists$. The sequence of observations that $Player_\exists$ makes during a play of the game depends on the type of interaction between the two players. In general, it is given by a function $obs_{pref} : \mathsf{Prefs}(G) \to Obs^*$, which maps a prefix π to the sequence of observations made by $Player_\exists$ so far. In the special case when $Player_\exists$ makes an observation at each step of the game, the sequence of observations is given by the *synchronous observation function* obs_s defined below. If, on the other hand $Player_\exists$ observes only the sequence of his own states, the observation of a prefix is defined by the *asynchronous observation function* obs_a below. Finally, the case when $Player_\exists$ can observe only changes of the state-based observations is modeled by the *stuttering-free observation function* obs_f whose definition is given below.

- $obs_s : \mathsf{Prefs}(G) \to Obs^*$, is defined for $\pi = s_0 s_1 \dots s_n$ as follows

$$obs_s(\pi) = obs(s_0) obs(s_1) \dots obs(s_n);$$

- $obs_a : \mathsf{Prefs}(G) \to Obs^*$ is defined for $\pi = s_0 s_1 \dots s_n$ as follows

$$obs_a(\pi) = \begin{cases} obs_a(\pi) = obs(s_n) & \text{if } n = 0 \text{ and } s_n \in S_\exists, \\ obs_a(\pi) = \epsilon & \text{if } n = 0 \text{ and } s_n \in S_\forall, \\ obs_a(\pi) = obs_a(s_0 \dots s_{n-1}) \cdot obs(s_n) & \text{if } n > 0 \text{ and } s_n \in S_\exists, \\ obs_a(\pi) = obs_a(s_0 \dots s_{n-1}) & \text{if } n > 0 \text{ and } s_n \in S_\forall; \end{cases}$$

- $obs_f : \mathsf{Prefs}(G) \to Obs^*$ is defined for $\pi = s_0 s_1 \dots s_n$ as follows

$$obs_f(\pi) = \begin{cases} obs_f(\pi) = obs(s_n) & \text{if } n = 0, \\ obs_f(\pi) = obs_f(s_0 \dots s_{n-1}) \cdot obs(s_n) & \text{if } n > 0 \text{ and } s_n \neq_o s_{n-1}, \\ obs_f(\pi) = obs_f(s_0 \dots s_{n-1}) & \text{if } n > 0 \text{ and } s_n =_o s_{n-1}. \end{cases}$$

Definition 2.2.1. Let $G = (S_\exists, S_\forall, I, =_o, \Sigma_\exists, T_\exists, T_\forall)$ be a game structure with incomplete information. Given a function $obs_{pref} : \mathsf{Prefs}(G) \to Obs^*$, a strategy $f_\exists$ for $Player_\exists$ in G is *obs_{pref}-consistent* iff for all $Player_\exists$ prefixes $\pi_1, \pi_2 \in \mathsf{Prefs}_\exists(f_\exists)$ for which $obs_{pref}(\pi_1) = obs_{pref}(\pi_2)$, it holds that $f_\exists(\pi_1) = f_\exists(\pi_2)$.

Let us define the equivalence relation $=_s$ on $\mathsf{Prefs}(G)$ as follows: $\pi_1 =_s \pi_2$ iff $obs_s(\pi_1) = obs_s(\pi_2)$. The relation $=_s$ is extended to an equivalence relation on plays such that $\pi_1 =_s \pi_2$ iff $|\pi_1| = |\pi_2|$ and $obs_s(\pi_1[i]) = obs_s(\pi_2[i])$ for each $0 \leq i < |\pi_1|$.

A winning condition Ω is called *observable* iff for each pair of plays π_1 and π_2 with $\pi_1 =_s \pi_2$ it holds that $\pi_1 \in \Omega$ iff $\pi_2 \in \Omega$. A set $R \subseteq S$ is called *observable* iff for every $s \in R$ and every $s' \in S$ with $obs(s') = obs(s)$ it holds that $s' \in R$. A safety game under incomplete information $\mathsf{Safety}(G, Err)$ for which the set of states Err is observable has an observable winning condition. Note that in a symbolic game structure with incomplete information $\mathcal{G} = (V_\exists, V_\forall, V_\forall^o, t, \varphi_{Init}, \mathcal{T}_\exists, \mathcal{T}_\forall)$, a formula $\varphi[Obs(V)]$ that contains only observable variables denotes a set of states that is observable. A safety game whose winning condition is not observable can be easily transformed into a safety game with observable winning condition such that $Player_\exists$ has an obs_s-consistent wining strategy in the transformed game iff he has an obs_s-consistent winning strategy in the original game. For finite-state games the transformation is linear.

Strategy automata As a representation of obs_{pref}-consistent strategies we consider (possibly infinite) Mealy automata that read (input) observations and produce (output) actions (or $\perp$, meaning no output is produced at the current step) at each step, while also updating the internal (memory) state. The sequence of observations supplied to the strategy automaton is determined by the observation function obs_{pref}.

Definition 2.2.2. A *strategy automaton* for some set of observations O and some set of actions $\Sigma_\exists$ is a (Mealy) automaton $\mathcal{M} = (Q, q^0, O \times \Sigma_\exists^\perp, \rho)$ with alphabet $O \times \Sigma_\exists^\perp$, where $\Sigma_\exists^\perp = \Sigma_\exists \dot\cup \{\perp\}$, and transition relation $\rho \subseteq Q \times (O \times \Sigma_\exists^\perp) \times Q$ such that:

(i) for each $q \in Q$ and $o \in O$ there exist $\sigma \in \Sigma_\exists^\perp$ and $q' \in Q$ with $(q, (o, \sigma), q') \in \rho$ (that is, ρ is input enabled for the observations O),

(ii) if $(q, (o, \sigma_1), q_1') \in \rho$ and $(q, (o, \sigma_2), q_2') \in \rho$, then $\sigma_1 = \sigma_2$ and $q_1' = q_2'$ (meaning that ρ deterministic in the observations O).

Let $G = (S_\exists, S_\forall, I, =_\mathsf{o}, \Sigma_\exists, T_\exists, T_\forall)$ be a game structure with incomplete information. A strategy automaton $\mathcal{M} = (Q, q^0, Obs \times \Sigma_\exists^\perp, \rho)$ is called *non-blocking* if for every $(q, (o, \sigma), q') \in \rho$ such that $o \subseteq S_\exists$ and $\mathsf{Enabled}(o, G) \neq \emptyset$, it holds that $\sigma \neq \perp$. The strategy automaton $\mathcal{M}$ is called $\Sigma_\exists$-*correct* if for every $(q, (o, \sigma), q') \in \rho$ it holds that if $o \subseteq S_\forall$, then $\sigma = \perp$ and if $o \subseteq S_\exists$ and $\sigma \neq \perp$, then $o \subseteq \mathsf{Enabled}(\sigma)$, that is, the proposed action is enabled in the current state of the game.

Given an observation function $obs_{pref} : \mathsf{Prefs}(G) \to Obs^*$, a non-blocking and $\Sigma_\exists$-correct strategy automaton $\mathcal{M} = (Q, q^0, Obs \times \Sigma_\exists^\perp, \rho)$ defines an obs_{pref}-consistent strategy $f_\exists$ for $Player_\exists$. Let $\pi \in \mathsf{Prefs}_\exists(G)$ and $obs_{pref}(\pi) = o_0 \dots o_n$. According to the properties of the automaton $\mathcal{M}$ there exists a unique sequence $\sigma_0 \dots \sigma_{n-1}$ such that there is a run (also unique) of $\mathcal{M}$ on the word $(0_0, \sigma_0) \dots (o_{n-1}, \sigma_{n-1})$. Let q be the last state of this run. According to conditions (i) and (ii) there is a unique $\sigma \in \Sigma_\exists^\perp$ such that there is a $q' \in Q$ with $(q, (o_n, \sigma), q') \in \rho$. If $\sigma \neq \perp$ we define $f_\exists(\pi) = \sigma$, otherwise we leave $f_\exists$ undefined. Since the strategy automaton $\mathcal{M}$ is non-blocking, the function $f_\exists$ is non-blocking as well, and since $\mathcal{M}$ is $\Sigma_\exists$-correct, $f_\exists$ is a strategy for $Player_\exists$.

A strategy $f_\exists$ is called *memoryless* if for all $\pi_1, \pi_2 \in \mathsf{Prefs}(G)$ with $\mathsf{last}(\pi_1) = \mathsf{last}(\pi_2)$ it holds that $f_\exists(\pi_1) = f_\exists(\pi_2)$. An obs_{pref}-consistent memoryless strategy (for some observation function obs_{pref}) can be defined by a strategy automaton with one state. Memoryless strategies for $Player_\forall$ are defined analogously. A strategy $f_\exists$ is *finite-state* if it can be defined by a strategy automaton with finite set of states Q.

For infinite-state games that are finitely represented by symbolic game structures we consider also *semi-symbolic strategy automata*. Finite semi-symbolic strategy automata can be used to finitely represent $Player_\exists$ strategies in infinite-state games. Unlike in (explicit) strategy automata, here the transition edges are labeled with pairs consisting of a formula over the variables $Obs(V)$ (called *guard*) and an action from $\Sigma_\exists$.

Definition 2.2.3. A *semi-symbolic strategy automaton* for a symbolic game structure with incomplete information $\mathcal{G} = (V_\exists, V_\forall, V_\forall^o, t, \varphi_{Init}, \mathcal{T}_\exists, \mathcal{T}_\forall)$ is an automaton $\mathcal{M} = (Q, q^0, \mathcal{LF}_\Xi[Obs(V)] \times \Sigma_\exists^\perp, \rho)$ with alphabet $\mathcal{LF}_\Xi[Obs(V)] \times \Sigma_\exists^\perp$, where $\Sigma_\exists^\perp = \Sigma_\exists \dot\cup \{\perp\}$, and transition relation $\rho \subseteq Q \times (\mathcal{LF}_\Xi[Obs(V)] \times \Sigma_\exists^\perp) \times Q$ such that:

(i) for each $q \in Q$ and $o \in Obs$ there exist $(\varphi, \sigma) \in \mathcal{LF}_\Xi[Obs(V)] \times \Sigma_\exists^\perp$ and $q' \in Q$ such that $o \subseteq [\![\varphi]\!]$ and $(q, (\varphi, \sigma), q') \in \rho$ (i.e., ρ is defined for each $o \in Obs$),

(ii) if $(q, (\varphi_1, \sigma_1), q_1') \in \rho$, $(q, (\varphi_2, \sigma_2), q_2') \in \rho$, and $\varphi_1 \wedge \varphi_2$ is satisfiable, then $\sigma_1 = \sigma_2$ and $q_1' = q_2'$ (implying that ρ deterministic in the observations Obs).

The notions non-blocking and $\Sigma_\exists$-correct are defined similarly as for explicit strategy automata. The symbolic strategy automaton $\mathcal{M}$ is *finite* when Q and ρ are finite.

2.2.2 Determinacy and Counterexamples

A two-player zero-sum game is called *determined* if the wining sets for the two players, that is, the set of states starting from which a player has a winning strategy to win the game, partitions the set of states. The seminal result by Martin [Mar75] establishes that every deterministic game with perfect information whose winning set is a Borel set is determined. As a simple case of Borel games, safety games with perfect information over deterministic game structures are also determined. Furthermore they are *memoryless determined*, that is, even if we require the winning strategies to be memoryless.

Thus, in a safety game with perfect information $\mathsf{Safety}(G, Err)$ with deterministic game structure G, either $Player_\exists$ has a (memoryless) winning strategy or $Player_\forall$ has a (memoryless) winning strategy. This does not hold for nondeterministic games.

It is well-known that games under incomplete information are not determined, even for deterministic game structures and safety winning conditions.

Let (G, Ω) be a game under incomplete information such that $Player_\exists$ does not have an obs_s-consistent winning strategy in (G, Ω). A *counterexample* in (G, Ω) is a

graph with nodes labeled with states of the game and edges labeled with actions that is a witness for the fact that $Player_\exists$ has no obs_s-consistent winning strategy.

For perfect information safety games it suffices to consider counterexamples represented as trees in which all paths are finite.

Definition 2.2.4. A *counterexample tree* for a safety game with perfect information $\mathsf{Safety}(G, Err)$ with game structure $G = (S_\exists, S_\forall, I, \Sigma_\exists, T_\exists, T_\forall)$ is a labeled tree $C_t = (N, E, L_s, L_a)$ with nodes N, edges $E \subseteq N \times N$, function $L_s : N \to S$ that labels each node with a state and a function $L_a : E \to \Sigma_\exists \cup \{\epsilon\}$ that labels edges with $Player_\exists$-actions or the symbol ϵ, where he following conditions are satisfied:

(i) $L_a(n, n') \neq \epsilon$ iff $L_s(n) \in S_\exists$,

(ii) C_t has a single root node n_0 and $L_s(n_0) \in I$,

(iii) for every $e = (n, n') \in E$ with $L_s(n) \in S_\exists$, $L_s(n') \in \mathsf{Post}_\exists(L_s(n), L_a(e))$,

(iv) for every $(n, n') \in E$ with $L_s(n) \in S_\forall$, $L_s(n') \in \mathsf{Post}_\forall(L_s(n))$,

(v) for each $n \in N$ with $L_s(n) \in S_\exists \setminus Err$ and every $\sigma \in \mathsf{Enabled}(L_s(n), G)$ there exists exactly one $n' \in N$ such that $(n, n') \in E$ and $L_a(n, n') = \sigma$,

(vi) for each $n \in N$ with $L_s(n) \in S_\forall$ there is at most one $n' \in N$ with $(n, n') \in E'$,

(vii) all paths in C_t are finite and for every leaf n, $L_s(n) \in Err$ or $L_s(n)$ is a dead-end.

Proposition 2.2.1. *Let* $\mathsf{Safety}(G, Err)$ *be a perfect-information safety game with game structure* $G = (S_\exists, S_\forall, I, \Sigma_\exists, T_\exists, T_\forall)$. *Then, exactly one of the following holds:*

(1) there exists a memoryless winning strategy for $Player_\exists$.

(2) there exists a counterexample tree.

Proof. We define a deterministic game structure $G' = (S_\exists, S'_\forall, \{i'\}, \Sigma_\exists, T'_\exists, T'_\forall)$ such that (i) $Player_\exists$ has a winning strategy in $\mathsf{Safety}(G', Err)$ iff he has a winning strategy in $\mathsf{Safety}(G, Err)$ and (ii) $Player_\forall$ has a winning strategy in $\mathsf{Safety}(G', Err)$ iff there exists a counterexample tree in $\mathsf{Safety}(G, Err)$. Then, the claim follows from these properties and the fact that deterministic perfect information games with safety winning conditions are determined. The game structure G' is defined as follows:

- $S'_\forall = S_\forall \dot\cup \{(s, \sigma) \mid s \in S_\exists, \sigma \in \Sigma_\exists\} \dot\cup \{i'\}$,
- $T'_\exists = \{(s, \sigma, (s, \sigma)) \mid \sigma \in \mathsf{Enabled}(s, G)\}$,

- $T'_\forall = T_\forall \dot\cup \{((s,\sigma),s') \mid (s,\sigma,s') \in T_\exists\} \dot\cup \{(i',i) \mid i \in I\}$.

Clearly the game structure G' is deterministic, and it is easy to see that each memoryless winning strategy for $Player_\exists$ in one of the games is a memoryless winning strategy in the other game. The outcome of a winning strategy for $Player_\forall$ in $\mathsf{Safety}(G', Err)$ contains a counterexample tree in $\mathsf{Safety}(G, Err)$ and the properties of a counterexample trees imply that it defines a memoryless winning strategy for $Player_\forall$ in $\mathsf{Safety}(G', Err)$. □

For a counterexample tree $C_t = (N, E, L_s, L_a)$ and a node $n \in N$ we denote with $path(n) = n_0 n_1 \dots n_i$ the path in the tree where n_0 is the root and $n_i = n$. For $n \in N$ we define $pref(n) = L_s(n_0)L_a(n_0,n_1)L_s(n_1)\dots L_a(n_{i-1},n_i)L_s(n_i)$ where $path(n) = n_0 n_1 \dots n_i$. For $n \in N$ and $\sigma \in \Sigma_\exists$ we denote with $Children(n) = \{n' \mid (n,n') \in E\}$ the set of children of n in C_t, with $Children(n,\sigma) = \{n' \mid (n,n') \in E \text{ and } L_a(n,n') = \sigma\}$ the set of n's children for edges labeled with σ, and with $Parent(n)$ the parent of n.

For safety games under incomplete information we consider counterexamples represented as trees whose nodes are labeled with sets of states representing the knowledge of $Player_\exists$ about the current state of the game, and in which all paths are finite. Now, the leaf nodes are labeled with state sets that contain an error state or contain a state that is a dead-end. Note that by condition (iii) in the definition of game structures under incomplete information, if a set of $Player_\exists$-states that are equivalent w.r.t. $=_o$ contains a dead-end state, then all states in this set are dead-ends.

Definition 2.2.5. A *knowledge-based counterexample tree* for a safety game under incomplete information $\mathsf{Safety}(G, Err)$ with game structure $G = (S_\exists, S_\forall, I, =_o, \Sigma_\exists, T_\exists, T_\forall)$ is a labeled tree $C_k = (N, E, K_s, L_a)$ with nodes N, edges $E \subseteq N \times N$, function $K_s : N \to (2^S \setminus \emptyset)$ that labels each node with a set of states and a function $L_a : E \to \Sigma_\exists \cup \{\epsilon\}$ that labels edges with $Player_\exists$-actions or the symbol ϵ, where:

(i) for every $n \in N$ there exists $o \in Obs$ such that $K_s(n) \subseteq o$,

(ii) $L_a(n,n') \neq \epsilon$ iff $K_s(n) \subseteq S_\exists$,

(iii) for the root node $n_0 \in N$ it holds that $K_s(n_0) \subseteq I \cap o$ for some $o \in Obs$,

(iv) for every $e = (n,n') \in E$ with $K_s(n) \subseteq S_\exists$, $K_s(n') \subseteq \mathsf{Post}_\exists(K_s(n), L_a(e))$,

(v) for every $(n,n') \in E$ with $K_s(n) \subseteq S_\forall$, $K_s(n') \subseteq \mathsf{Post}_\forall(K_s(n))$,

(vi) for each node n with $K_s(n) \subseteq S_\exists \setminus Err$ and each $\sigma \in \mathsf{Enabled}(K_s(n), G)$ there exists exactly one edge $e = (n,n') \in E$ such that $L_a(e) = \sigma$,

(vii) for each node n with $K_s(n) \subseteq S_\forall$ there exists at most one $n' \in N$ such that (n, n'),

(viii) all paths in the tree C_k are finite, and if n is a leaf node, then it holds that $K_s(n) \cap Err \neq \emptyset$ or $K_s(n)$ contains a dead-end state.

Proposition 2.2.2. *In a safety game under incomplete information* $\mathsf{Safety}(G, Err)$ *with game structure* $G = (S_\exists, S_\forall, I, =_\mathsf{o}, \Sigma_\exists, T_\exists, T_\forall)$*, if there exists a knowledge-based counterexample tree, then* $Player_\exists$ *does not have an* obs_s*-consistent winning strategy.*

Proof. Let $C_k = (N, E, K_s, L_a)$ be a knowledge-based counterexample tree in the game $\mathsf{Safety}(G, Err)$. Assume that $f_\exists$ is an obs_s-consistent winning strategy for $Player_\exists$ in $\mathsf{Safety}(G, Err)$. By the definition of C_k and the fact that $f_\exists$ is obs_s-consistent, there exists a path π in C_k such that (1) for every prefix $\pi' = s_0, s_1, \ldots, s_n$ for which $s_i \in L_s(\pi[i])$ for each $0 \leq i \leq n$, it holds that $\pi' \in \mathsf{Prefs}(f_\exists)$, and (2) $\mathsf{last}(\pi)$ is a leaf. Since (2) holds, there exists a state $s \in L_s(\pi[i])$ such that $s \in Err$ or s is a dead-end. The definition of C_k implies that there exists a prefix $\pi' = s_0, s_1, \ldots, s_n$ such that for $s_i \in L_s(\pi[i])$ for each $0 \leq i \leq n$ and $s_n = s$. According to (1), $\pi \in \mathsf{Prefs}(f_\exists)$, which contradicts the assumption that $f_\exists$ is a winning strategy for $Player_\exists$. □

In a safety game $\mathsf{Safety}(G, Err)$ with a game structure $G = (S_\exists, S_\forall, I, =_\mathsf{o}, \Sigma_\exists, T_\exists, T_\forall)$, if G is finite, then knowledge-based counterexample trees in $\mathsf{Safety}(G, Err)$ are always finite. For infinite game structures, if the set $\Sigma_\exists$ of $Player_\exists$ actions is finite, it holds that if there is a knowledge-based counterexample tree in $\mathsf{Safety}(G, Err)$, there exists also a finite one. If, on the other hand $\Sigma_\exists$ is infinite, it is possible that all possible knowledge-based counterexample trees in $\mathsf{Safety}(G, Err)$ are infinite.

2.2.3 Knowledge-Based Subset Construction

An incomplete-information game can be translated to a perfect-information game in which $Player_\exists$ has a winning strategy if and only if $Player_\exists$ has an obs_s-consistent winning strategy in the original game with incomplete information.

The original powerset construction for games with partial information was proposed by Reif [Rei84] for games in which both players have reachability winning conditions and in which every infinite play is a draw. A knowledge-based subset construction for zero-sum games with observable ω-regular winning conditions based on this idea was defined in [CDHR06]. While in [Rei84] a player cannot detect private moves of

the opponent, the setting of [CDHR06] is that of synchronous observability. A powerset construction for not necessarily observable ω-regular objectives and also in the asynchronous setting was developed in [Puc10]. While in principle the translation to perfect information games can be defined also for games over infinite graphs (in the case of [Rei84] and [CDHR06] for graphs with finite branching degree), the construction of the perfect information game is in general not effective for infinite game structures.

For completeness of the exposition and in order to make the connection to the constructions presented further in this thesis, we define here the knowledge based subset construction for the case of synchronous observation functions and games with infinite game structures and safety winning conditions that are not necessarily observable.

The *synchronous knowledge-based subset construction* of a game structure with incomplete information $G = (S_\exists, S_\forall, I, =_o, \Sigma_\exists, T_\exists, T_\forall)$ is a game structure with perfect information G^k, where $G^k = (S^k_\exists, S^k_\forall, I^k, \Sigma^k_\exists, T^k_\exists, T^k_\forall)$ is defined as follows:

- $S^k_\exists = \{s_k \in 2^{S_\exists} \setminus \{\emptyset\} \mid \exists o \in Obs.\ s_k \subseteq o\}$,
- $S^k_\forall = \{s_k \in 2^{S_\forall} \setminus \{\emptyset\} \mid \exists o \in Obs.\ s_k \subseteq o\}$,
- $S^k = S^k_\exists \cup S^k_\forall$,
- $I^k = \{s_k \in S^k \mid \exists o \in Obs.\ s_k = I \cap o\}$,
- $\Sigma^k_\exists = \Sigma_\exists$,
- $T^k_\exists = \{(s_k, \sigma, s'_k) \in S^k_\exists \times \Sigma_\exists \times S^k_\forall \mid \exists o \in Obs.\ s'_k = \mathsf{Post}_\exists(s_k, \sigma) \cap o\}$,
- $T^k_\forall = \{(s_k, s'_k) \in S^k_\forall \times S^k \mid \exists o \in Obs.\ s'_k = \mathsf{Post}_\forall(s_k) \cap o\}$.

Condition (iii) from Definition 2.1.2 guarantees that under the assumption that the game structure G does not contain dead-end states, the game structure G^k obtained from G via the above construction does not contain dead-ends either.

The following result was established in each of the works [Rei84, CDHR06, Puc10] for the respective game model, types of winning conditions and construction. Here we give in the following theorem the proof for game model defined in Section 2.1.

Theorem 2.2.1. *Let* $\mathsf{Safety}(G, Err)$ *be a safety game under incomplete information and let* $Err^k = \{s_k \in S^k \mid s_k \cap Err \neq \emptyset\}$. *Player*$_\exists$ *has a winning strategy in* $\mathsf{Safety}(G^k, Err^k)$ *iff he has an obs*$_s$*-consistent winning strategy in* $\mathsf{Safety}(G, Err)$.

Proof. Let $f_\exists^k$ be a winning strategy for $Player_\exists$ in $\mathsf{Safety}(G^k, Err^k)$. We define the function $f_\exists$ as follows. For a prefix $\pi \in \mathsf{Prefs}_\exists(G)$, if there exists a prefix $\pi^k \in \mathsf{Prefs}(f_\exists^k)$ such that $|\pi| = |\pi^k|$ and $\pi[i] \in \pi[k]$ for all $0 < i < |\pi|$, then we define $f_\exists(\pi) = f_\exists(\pi^k)$ (note that there can be at most one such π^k). Otherwise, $f_\exists(\pi)$ is undefined. Since $f_\exists^k$ is a strategy for $Player_\exists$ and is winning, by condition (iii) from Definition 2.1.2 we have that the function $f_\exists$ is a strategy for $Player_\exists$. Since the strategy $f_\exists^k$ is winning for $Player_\exists$ and for each $\pi \in \mathsf{Prefs}(f_\exists)$ there exists a corresponding $\pi^k \in \mathsf{Prefs}(f_\exists^k)$, it is easy to see that $f_\exists$ is winning for $Player_\exists$ in $\mathsf{Safety}(G, Err)$. From the definition of $f_\exists$ it is clear that it is also obs_s-consistent. Now let $f_\exists$ be an obs_s-consistent winning strategy for $Player_\exists$ in $\mathsf{Safety}(G, Err)$. It is easy to see that for each prefix $\pi^k \in \mathsf{Prefs}(G^k)$ there exists a prefix $\pi \in \mathsf{Prefs}(G)$ such that $|\pi| = |\pi^k|$ and $\pi[i] = \pi^k[i]$ for each $0 < i \leq |\pi|$. For each $\pi^k \in \mathsf{Prefs}_\exists(G^k)$ we define $f_\exists^k(\pi^k) = f_\exists(\pi)$ for some such prefix π. Since $f_\exists$ is obs_s-consistent, the function $f_\exists$ is well defined. From condition (iii) from Definition 2.1.2 it follows that $f_\exists^k(\pi^k) \in \mathsf{Enabled}(\mathsf{last}(\pi^k))$. As $f_\exists$ is non-blocking, $f_\exists^k$ is non-blocking as well. It is easy to show that for each $\pi^k \in \mathsf{Prefs}(f_\exists^k)$ and each $s \in \mathsf{last}(\pi^k)$ there exists $\pi \in \mathsf{Prefs}(f_\exists)$ such that $s = \mathsf{last}(\pi)$. Thus, since $f_\exists$ is winning for $Player_\exists$ in $\mathsf{Safety}(G, Err)$, it holds that $f_\exists$ is winning for $Player_\exists$ in $f_\exists(G^k, Err^k)$. □

From the definitions of the game $\mathsf{Safety}(G^k, Err^k)$ and knowledge-based counterexample tree in $\mathsf{Safety}(G, Err)$ it is clear that a counterexample tree in $\mathsf{Safety}(G^k, Err^k)$ directly corresponds to a knowledge-based counterexample tree in $\mathsf{Safety}(G, Err)$. We thus establish the following proposition, which together with Proposition 2.2.1 and Theorem 2.2.1 yields Theorem 2.2.2 below.

Proposition 2.2.3. *Let* $\mathsf{Safety}(G, Err)$ *be a safety game under incomplete information and let* $Err^k = \{s_k \in S^k \mid s_k \cap Err \neq \emptyset\}$. *There exists a counterexample tree in the perfect-information game* $\mathsf{Safety}(G^k, Err^k)$ *iff there exists a knowledge-based counterexample tree in the game under incomplete information* $\mathsf{Safety}(G, Err)$.

Theorem 2.2.2. *Let* $\mathsf{Safety}(G, Err)$ *be a safety game under incomplete information and* obs_s *be the synchronous observation function. Exactly one of the following holds:*

(1) there exists an obs_s*-consistent winning strategy for* $Player_\exists$.

(2) there exists a knowledge-based counterexample tree.

Unlike in the perfect information case, winning strategies for $Player_\exists$ in a game under incomplete information may in general need memory.

2.2.4 The Game Solving and Strategy Synthesis Problems

Given a game (under incomplete information) (G, Ω) and an observation function $obs_{pref} : \mathsf{Prefs}(G) \to Obs(G)^*$ the *game solving problem* is the decision problem that asks to determine whether there exist an obs_{pref}-consistent winning strategy for $Player_\exists$ in (G, Ω). The *strategy synthesis* problem is to construct such a strategy if one exists.

The game solving and strategy synthesis problems *for a class of games* $\mathcal{C}$ *and a class of* $Player_\exists$ *strategies* Λ restrict the input games to the ones that belong to the class $\mathcal{C}$ and the strategies of $Player_\exists$ to the ones that belong to the class Λ.

In this thesis we consider classes of games that can be finitely represented. In the first two parts of the thesis we restrict our attention to safety winning conditions.

Solving Finite-State Safety Games with Perfect Information. For the class of finite-state safety games with perfect information the game solving and strategy synthesis problems can be solved by a simple fixpoint computation that yields the set of states from which $Player_\exists$ has a winning strategy. More precisely, given a perfect information safety game $\mathsf{Safety}(G, Err)$ with a finite-state game structure $G = (S_\exists, S_\forall, I, \Sigma_\exists, T_\exists, T_\forall)$, the algorithm computes the set of states $W_\exists$ such that $s \in W_\exists$ iff $Player_\exists$ has a winning strategy in the game $\mathsf{Safety}(G_s, Err)$, where $G_s = (S_\exists, S_\forall, \{s\}, \Sigma_\exists, T_\exists, T_\forall)$. $Player_\exists$ has a winning strategy in $\mathsf{Safety}(G, Err)$ iff $I \subseteq W_\exists$. We let $W_\exists = S \setminus F^n$, where $n \geq 0$ is such that $F^{n+1} \subseteq F^n$, where:

$$\begin{array}{lcl} F^0 & = & Err \cup D, \text{where } D \text{ is the set of dead-ends in } G, \\ F^{i+1} & = & \mathsf{Pre}_\forall(F^i) \cup \bigcap_{\sigma \in \Sigma_\exists} \Big(\big(S \setminus \mathsf{Enabled}(\sigma)\big) \cup \mathsf{Pre}_\exists(F^i, \sigma) \Big). \end{array}$$

The strategy synthesis problem is then also easily solved. The memoryless strategy that maps each $s \in W_\exists \cap S_\exists$ to a $\sigma \in \Sigma_\exists$ such that each $s' \in S_\exists$ with $(s, \sigma, s') \in T_\exists$ is in $W_\exists$ (such σ exists by the construction of $W_\exists$) is a winning strategy for $Player_\exists$.

For finite-state safety games under incomplete information the knowledge-based subset construction from Section 2.2.3 is effective and can be combined with the fixpoint algorithm to solve the game and effectively construct a finite-state strategy. Clearly, the same can be done for finite-state incomplete-information games with more general winning conditions. To avoid the determinization-like powerset construction of the straightforward algorithm for solving games under incomplete information outlined above, [DWDR06, CDHR06] propose a symbolic algorithm based on antichains as a

data structure to represent downward-closed sets of sets of states. Their fixpoint computation is directed by the objective and avoids the explicit powerset construction.

2.2.5 Discussion of the Knowledge-Based Subset Construction

We defined knowledge-based counterexample trees and presented the knowledge-based subset construction with respect to the synchronous observation function obs_s. Since the object of study of this thesis are infinite-state games under incomplete information, this construction will only be used as a correctness argument and in these cases we will be reasoning about obs_s-consistent strategies for $Player_\exists$ (in a possibly modified game structure, corresponding to strategies in the original game structure, satisfying stronger consistency requirements). The knowledge-based subset construction of Reif was extended to asynchronous games in [Puc10] using a construction similar to the one given in [CDL+07] for timed games with fixed observations. While in principle we could use a similar idea for the predicate abstraction technique that we present, our reasoning about the relation between abstract and concrete games relies on the fact that a step of the abstract game corresponds to a step of the concrete game, which would no longer be the case for a construction that treats asynchronism directly.

In [Rei84] and [CDHR06] the game graphs have finite branching degree. Here we make no such assumption and neither do we require that the winning condition is observable, as done in [CDHR06]. Such assumptions are not necessary for the correctness of the knowledge-based subset construction given here with respect to games in which the partially informed player has a safety objective.

2.3 Game Abstractions

Abstraction [CGL94, CC00, DGG97] is the key to effective verification of systems with infinite or very large state spaces. The abstract-interpretation theory for temporal specifications for transition systems was first extended to synthesis and control – more precisely to (multi-player) concurrent game structures and alternating μ-calculus properties – by Henzinger et al. in their work [HMMR00]. An abstraction is sound w.r.t. a given property if whenever the property is established for the abstract system it is guaranteed to hold for the original concrete system as well. In the case of game properties we can formulate soundness as: if there exists a strategy for the component

(controller) player in the abstract game, then there must exist one in the concrete game. This can be achieved by restricting the possible moves of the component (controller) and allowing more behaviors of its environment. This approach has been followed in a number of works [HMMR00, HJM03]. An alternative abstraction methodology for two-player games follows the paradigm of three-valued abstraction used for the verification branching time properties of transition systems [HJS01, SG04]. A three-valued abstraction framework for two-player turn-based games was proposed in [dAGJ04]. Unlike the aforementioned works on game abstraction that allow for proving the existence of a strategy of only one of the players, the three-valued abstraction preserves all alternating μ-calculus formulas and treats the two players symmetrically.

The first abstraction technique for game structures with incomplete information was developed in [DF08]. In the case of incomplete information, restricting the power of the existential player involves overapproximating the observation equivalence – or, equivalently, restricting the observation power of the existential player. While in the perfect information case [HMMR00, HJM03] the relation between the abstract and the concrete game structures is *alternating simulation* [AHKV98], in [DF08] the alternating simulation relates the abstract game and corresponding knowledge-based game.

An abstraction of a game structure under incomplete information is a game structure that satisfies a set of conditions relating the abstract game structure and the corresponding pair of concretization functions. The abstract game structure can be a game structure with incomplete information or one with perfect information. The formalization of the soundness of the abstraction includes the concrete and abstract observation functions, which in the case of game structures with perfect information is the identity. The set of conditions we require an abstraction to satisfy do not define the abstract game structure uniquely, and also do not imply soundness of the abstraction.

Definition 2.3.1. An *abstraction* of a game structure with incomplete information $G = (S_\exists, S_\forall, I, =_\mathsf{o}, \Sigma_\exists, T_\exists, T_\forall)$ is a game structure $G^\# = (S^\#_\exists, S^\#_\forall, I^\#, =^\#_\mathsf{o}, \Sigma^\#_\exists, T^\#_\exists, T^\#_\forall)$ together with the concretization functions $\gamma : S^\# \to 2^S$ and $\gamma_\exists : \Sigma^\# \to 2^{\Sigma_\exists}$ such that the following conditions are satisfied:

(i) the sets $\{\gamma(s^\#) \mid s^\# \in S^\#\}$ cover the set S,

(ii) the sets $\{\gamma_\exists(\sigma^\#) \mid \sigma^\# \in \Sigma^\#\}$ cover the set $\Sigma_\exists$,

(iii) $\gamma(s^\#) \subseteq S_\exists$ for each $s^\# \in S^\#_\exists$ and $\gamma(s^\#) \subseteq S_\forall$ for each $s^\# \in S^\#_\forall$,

(iv) the sets $\{\gamma(s_0^\#) \mid s_0^\# \in I^\#\}$ cover the set I,

(v) if $(s_1^\#, \sigma^\#, s_2^\#) \in T_\exists^\#$, then for every $s_1 \in \gamma(s_1^\#)$ there exist $\sigma \in \gamma_\exists(\sigma^\#)$ and $s_2 \in \gamma(s_2^\#)$ such that $(s_1, \sigma, s_2) \in T_\exists$,

(vi) if $(s_1^\#, s_2^\#) \in T_\forall^\#$, then there are $s_1 \in \gamma(s_1^\#)$ and $s_2 \in \gamma(s_2^\#)$ such that $(s_1, s_2) \in T_\forall$,

(vii) if $s_1 =_\mathsf{o} s_2$ then, for some $s_1^\#, s_2^\# \in S_\exists^\#$, $s_1 \in \gamma(s_1^\#)$, $s_2 \in \gamma(s_2^\#)$ and $s_1^\# =_\mathsf{o}^\# s_2^\#$.

When the concrete game structure G is not clear from the context, we write $\gamma(s^\#, G)$ and $\gamma_\exists(\sigma^\#, G)$ for $s^\# \in S^\#$ and $\sigma^\# \in \Sigma_\exists^\#$, respectively.

Condition (iii) requires that the abstraction respects the partitioning of the states into $S_\exists$ and $S_\forall$. Conditions (v) and (vi) ensure that transitions in the abstract game structure correspond to transitions in the concrete game structure. Finally, condition (vii) guarantees that the abstract observation equivalence relation is coarser than the concrete one. Note that conditions (v) and (vi) do not imply that the transitions of the abstract game structure are such that is simulates the concrete one.

To illustrate the definition above, we define the abstract game structure obtained via *predicate abstraction* from a symbolic game structure with *perfect information*, i.e. in the special case when $=_\mathsf{o}$ is the equality relation. The resulting game structure and its concretization functions will satisfy the conditions of Definition 2.3.1.

Let $\mathcal{G} = (V_\exists, V_\forall, V_\forall^o, t, \varphi_{Init}, \mathcal{T}_\exists, \mathcal{T}_\forall)$ be a symbolic game structure with perfect information, that is, $V_\forall^o = V_\forall$. Let $\mathcal{P}$ be a finite set of *predicates*, which are atomic formulas over the set X of variables in $\mathcal{G}$. The set $\mathit{Vals}(\mathcal{P}) = \mathbb{B}^\mathcal{P}$ consists of all truth valuations of the elements of $\mathcal{P}$. For each $a \in \mathit{Vals}(\mathcal{P})$ and $\varphi \in \mathcal{P}$ we write $a \models \varphi$ iff $a(\varphi) = \mathit{true}$. Additionally, we associate with each $a \in \mathit{Vals}(\mathcal{P})$ the formula $[a] = \big(\bigwedge_{\varphi \in \mathcal{P}, a \models \varphi} \varphi\big) \wedge \big(\bigwedge_{\varphi \in \mathcal{P}, a \not\models \varphi} \neg\varphi\big)$, describing a set of states in G, and let $[\![a]\!] = [\![[a]]\!]$.

We define the concretization function $\gamma : S^\# \to 2^S$ as $\gamma(s^\#) = [\![s^\#]\!]$ for each $s^\# \in S^\#$, where $S^\# = \mathit{Vals}(\mathcal{P})$ is the set of abstract states. By abuse of notation, γ can be extended to the set $2^{S^\#}$ such that for $A \subseteq S^\#$ we have $\gamma(A) = \bigcup_{a \in A} \gamma(a)$. Clearly, taking the abstraction function $\alpha : 2^S \to 2^{S^\#}$ defined such that $\alpha(C) = \bigcap\{A \in 2^{S^\#} \mid C \subseteq \gamma(A)\}$ gives us the standard Galois connection $(2^S, \subseteq) \underset{\gamma}{\overset{\alpha}{\rightleftarrows}} (2^{S^\#}, \subseteq)$ (i.e., the functions α and γ satisfy the properties $\alpha(\gamma(A)) = A$ and $C \subseteq \gamma(\alpha(C))$).

The game structure $\mathsf{AbstractPerfect}(\mathcal{G}, \mathcal{P}) = (S_\exists^\#, S_\forall^\#, I^\#, =_\mathsf{o}^\#, \Sigma_\exists^\#, T_\exists^\#, T_\forall^\#)$, which is the *abstraction of the perfect-information symbolic game structureG w.r.t. a set* $\mathcal{P}$ *such that* $(t = \forall) \in \mathcal{P}$ is a finite-state game structure defined as follows.

- $S^{\#}_{\exists} = \{s^{\#} \in \mathit{Vals}(\mathcal{P}) \mid s^{\#} \models t = \exists\}$,
- $S^{\#}_{\forall} = \{s^{\#} \in \mathit{Vals}(\mathcal{P}) \mid s^{\#} \models t = \forall\}$,
- $S^{\#} = S^{\#}_{\exists} \dot{\cup} S^{\#}_{\forall}$,
- $I^{\#} = \{s^{\#} \in S^{\#} \mid [\![s^{\#}]\!] \cap [\![\varphi_{\mathit{Init}}]\!] \neq \emptyset\}$,
- $=^{\#}_{\circ}$ is the equality relation,
- $\Sigma^{\#}_{\exists} = S^{\#}_{\forall}$,
- $(s^{\#}_1, s^{\#}_2) \in T^{\#}_{\exists}$ iff $\forall s_1 \in [\![s^{\#}_1]\!].\exists s_2 \in [\![s^{\#}_2]\!].\ (s_1, s_2) \models \mathcal{T}_{\exists}$,
- $(s^{\#}_1, s^{\#}_2) \in T^{\#}_{\forall}$ iff $\exists s_1 \in [\![s^{\#}_1]\!].\exists s_2 \in [\![s^{\#}_2]\!].\ (s_1, s_2) \models \mathcal{T}_{\forall}$.

The concretization function $\gamma_{\exists} : \Sigma^{\#}_{\exists} \to 2^{\Sigma_{\exists}}$ maps an abstract action $\sigma^{\#} \in \Sigma^{\#}_{\exists}$ to the set of concrete actions $\gamma_{\exists}(\sigma^{\#}) = \{\sigma \in \Sigma \mid \exists s \in \gamma(\sigma^{\#}).\ \sigma(V_{\exists}) = s(V_{\exists})\}$.

Example 2.3.1. We now consider a variation $\mathcal{G} = (V_{\exists}, V_{\forall}, V^{o}_{\forall}, t, \varphi_{\mathit{Init}}, \mathcal{T}_{\exists}, \mathcal{T}_{\forall})$ of the symbolic game structure from Example 2.1.1, in which $V^{o}_{\forall} = V_{\forall} = \{x, y, \mathit{steps}, \mathit{err}\}$. Let the set $\mathcal{P}$ of predicates consist of the atomic formulas occurring in φ_{Err}:

$$\begin{aligned}\mathcal{P}_{\mathit{Err}} \quad = \quad & \{(t = \forall), (x < 6), (x \geq 9), (x \leq -1),\\ & (\mathit{steps} > 3), (\mathit{err} = \mathit{true}), (y \geq 4), (y \leq -4)\}.\end{aligned}$$

The set of initial states is $I^{\#} = \{s_0\}$, where $s^{\#}_0 = (0, 1, 0, 0, 0, 0, 0, 0)$ is a valuation of the predicates in $\mathcal{P}_{\mathit{Err}}$ in the same order as listed above. The only successor in $T^{\#}_{\exists}$ of state $s^{\#}_0$, belonging to $\mathit{Player}_{\exists}$, is $s^{\#}_1 = (1, 1, 0, 0, 0, 0, 0, 0)$. We have $(s^{\#}_0, \sigma^{\#}, s^{\#}_1) \in T^{\#}_{\exists}$, where $\sigma^{\#} = s^{\#}_1$, and $\gamma_{\exists}(\sigma) = \{N, S, E, W\}$. The set of successors of state $s^{\#}_1$ in $T^{\#}_{\forall}$ is

$$\big\{(0, p_1, 0, p_2, 0, p_3, p_4, p_5) \mid p_1, p_2, p_3, p_4, p_5 \in \{0, 1\} \wedge (p_1 \vee \neg p_2) \wedge (\neg p_4 \vee \neg p_5)\big\}.$$

The abstraction of $T_{\forall}$ is a classical existential overapproximation of the concrete transition relation for $\mathit{Player}_{\forall}$. The abstraction of $T_{\exists}$, on the other hand, restricts the possible choices of $\mathit{Player}_{\exists}$. In the example above the initial state of the abstract game has a single successor state that groups all concrete successors of all initial states in the concrete game. If we take as the set of abstraction predicates the set $\mathcal{P}_0 = \mathcal{P}_{\mathit{Err}} \cup \{(\mathit{move} = N), (\mathit{move} = E), (\mathit{move} = S)\}$, then the initial state will have four successors, corresponding to the four possible values of the variable *move* in the concrete

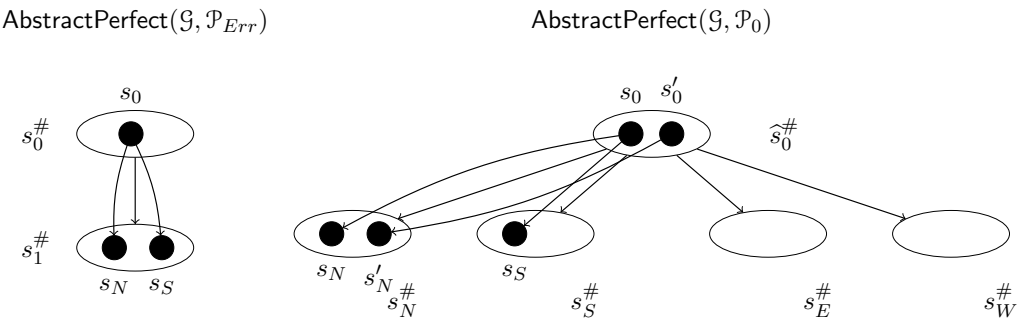


Figure 2.1: Successors of the initial states in the abstract game structures w.r.t. sets of predicates $\mathcal{P}_{Err}$ and $\mathcal{P}_0$. The state s_0 is a concrete state in $\gamma(s_0^\#)$ and s_N and s_S are successors of s_0 such that $s_N(move) = N$ and $s_S(move) = S$. Due to the imprecision of the abstraction, both s_N and s_S are in $\gamma(s_1^\#)$. The abstract game AbstractPerfect$(\mathcal{G}, \mathcal{P}_{Err})$ is more precise and s_N and s_S are in the concretization of different abstract states. The state s_0' is another concrete state, and both s_0 and s_0' are in $\gamma(\widehat{s}_0^\#)$. Thus, in the abstract game $Player_\exists$ must choose the same value for the variable *move* in both s_0 and s_0'.

game. The power of $Player_\exists$ will still be limited in the corresponding abstract game, since he is required to make *the same* choice in a *all* the possible concrete initial states.

Figure 2.1 depicts the abstract initial states and their successors in the game structures AbstractPerfect$(\mathcal{G}, \mathcal{P}_{Err})$ and AbstractPerfect$(\mathcal{G}, \mathcal{P}_0)$ respectively.

□

We now formulate precisely what it means for an abstraction of a given game structure with incomplete information to be sound (with respect to safety winning conditions). The soundness definition is parametrized by the observation functions for $Player_\exists$ in the concrete and in the abstract abstract games.

Definition 2.3.2. Let $G = (S_\exists, S_\forall, I, =_o, \Sigma_\exists, T_\exists, T_\forall)$ be a game structure with incomplete information and $G^\# = (S_\exists^\#, S_\forall^\#, I^\#, =_o^\#, \Sigma_\exists^\#, T_\exists^\#, T_\forall^\#)$ be an abstraction of G and let $Err \subseteq S$ and $Err^\# \subseteq S^\#$ be sets of concrete and abstract error states. Let obs_{pref} be an observation function for G and $obs_{pref}^\#$ be an observation function for $G^\#$. We say that (Safety$(G^\#, Err^\#), obs_{pref}^\#$) is a *sound abstraction* of (Safety$(G, Err), obs_{pref}$) iff when $Player_\exists$ has an $obs_{pref}^\#$-consistent winning strategy Safety$(G^\#, Err^\#)$ then $Player_\exists$ has an obs_{pref}-consistent winning strategy in Safety(G, Err).

Suppose that Safety$(\mathcal{G}, \varphi_{Err})$ is a perfect-information safety game with symbolic game structure $\mathcal{G} = (V_\exists, V_\forall, V_\forall^o, t, \varphi_{Init}, \mathcal{T}_\exists, \mathcal{T}_\forall)$ and $\mathcal{P}$ is a finite set of predicates that

contains the predicate $t = \forall$. Let $G^{\#} = \mathsf{AbstractPerfect}(\mathcal{G}, \mathcal{P}) = (S^{\#}_{\exists}, S^{\#}_{\forall}, I^{\#}, =^{\#}_{\mathsf{o}}, \Sigma^{\#}_{\exists}, T^{\#}_{\exists}, T^{\#}_{\forall})$. If $Err^{\#} = \{s^{\#} \in S^{\#} \mid [\![s^{\#}]\!] \cap [\![\varphi_{Err}]\!] \neq \emptyset\}$, then $(\mathsf{Safety}(G^{\#}, Err^{\#}), obs^{\#}_{s})$ is a sound abstraction of $(\mathsf{Safety}(G, Err), obs_{s})$. Formally, this claim follows easily as a corollary of the soundness property (proven as Theorem 5.1.1) of the abstraction of games under incomplete information described in Section 5.1.

We say that the abstraction $(\mathsf{Safety}(G^{\#}, Err^{\#}), obs^{\#}_{pref})$ of $(\mathsf{Safety}(G, Err), obs_{pref})$ *preserves the existence of finite-state winning* strategies for $Player_{\exists}$ if $Player_{\exists}$ has a finite-state obs_{pref}-consistent winning strategy in $\mathsf{Safety}(G, Err)$ iff $Player_{\exists}$ has a finite-state $obs^{\#}_{pref}$-consistent winning strategy in $\mathsf{Safety}(G^{\#}, Err^{\#})$.

Part I
Infinite-State Games with Fixed Observations

Chapter 3

Lossy Channel Games under Incomplete Information

In this part of the thesis we consider infinite-state games under incomplete information where the set of observations is a priori fixed, for example by the underlying system model. We study game structures for which the infinite set of states is ordered by a quasi ordering which is a *Better Quasi Ordering (BQO)* [Mil85, Nas65]. One particular such class of game structures is the class of game structures defined by *Lossy Channel Systems(LCSs)* [AJ93]. These are finite-control systems communicating via unbounded lossy FIFO channels. They are a common model for communication protocols such as link protocols, a canonical example of which is the Alternating Bit Protocol.

Motivating example. The Alternating Bit Protocol is a protocol for transferring messages in one direction between two processes, Sender and Receiver respectively, by retransmitting lost messages. Figure 3.1 shows the protocol modeled as a LCS. Each message sent from the Sender to Receiver consists of the actual message contents to be transmitted (ignored here) and a 1-bit sequence counter. The messages are sent on the unbounded FIFO channel K and acknowledged on the unbounded FIFO channel L. As we do not model the data part, the message alphabet of each channel is finite, in this case $\{0, 1\}$. However, the set of possible channel contents is infinite. Since the communication channels are *lossy*, messages may be lost nondeterministically.

Each sent message is resent by Sender until it receives an acknowledgement with the same sequence number. Upon this, Sender starts transmitting the next message

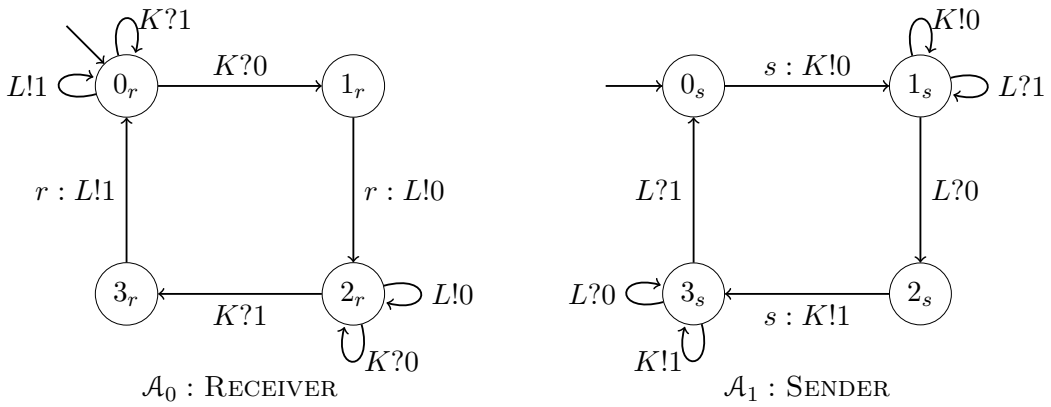


Figure 3.1: Alternating Bit Protocol modelled as a Lossy Channel System

with a flipped bit. Receiver initially sends 1s and waits until a message with sequence number 0 arrives. Upon this it starts sending acknowledgement 0 until a message with sequence number 1 is received, when the acknowledgement bit is flipped again.

The Alternating Bit Protocol works correctly even when messages and acknowledgments are lost nondeterministically. This means, that the sequence of data messages passed from Receiver to its client is a prefix of the sequence of data messages generated by Sender's client. That is, no data is lost, duplicated or reordered.

Once a communication protocol has been completely specified, such as the Alternating Bit Protocol shown here, one can apply a number of verification and analysis techniques to check that it satisfies the desired properties. The alternative would be to automatically synthesize the protocol from a given formal specification. However, the synthesis problems for decentralized protocols is in general undecidable, as individual processes might have access to incomparable information [PR90, Tri04]. We therefore look at a restriction of the general synthesis problem, where all except one processes are completely specified. This is not an uncommon situation. For example, it can be the case that the above protocol is only partially specified in that only the sender process is given and we wish to synthesize a receiver process such that the resulting protocol fulfills a given property. Another notable example is protocol converter synthesis [PRSV98, PAHSv02].

When synthesizing LCSs, the interface of the process to be synthesized determines in a natural way a finite set of observations that the generated implementation of this

process can use. This is the case if we make the realistic assumption that a process can observe only the element at the head of each channel it has (read) access to. In the above example, we can suppose that RECEIVER, i.e., the process we want to synthesize in this case, can only observe the element at the head of channel K. Despite the fact that a finite set of observations is predefined by the input, the synthesis problem under partial observability is not trivial. To see this, note that although the set of observations is finite, the set of possible knowledge sets, i.e., sets of states the system could be in according to the knowledge of the synthesized process, is still infinite.

Related Work. In the past two decades, the decidability of verification problems for LCSs has been well studied [AJ93, AJ96] and a large number of works have been devoted to developing automatic analysis techniques [ABJ98, AAB99, GRV04, GRV05] for this class of systems. In the control and synthesis setting, where games are the natural computational model, this class of systems has not yet been so well investigated. In [ABd08], Abdulla et al. establish decidability of two-player safety and reachability games where one (or both) player has downward-closed behavior (e.g., can lose messages), which subsumes games with lossy channels where one player (i.e., the environment) can lose messages. They, however, assume that the game is played under perfect information, which assumption disregards the fact that a process has no access to the local states of other processes or that it has only limited information about the contents of the channels. Recently, stochastic games on LCSs have been investigated [AHdA$^+$08, BS13, ACMS13], again under the perfect-information hypothesis.

The decidability results and algorithmic solutions for the analysis of LCSs typically rely on the monotonicity of the transition relation of a LCS w.r.t. the subword relation, which is a well-quasi ordering (WQO). It is well known that upward and downward-closed sets of words used in the analysis of lossy channel systems can be effectively represented by finite sets of minimal elements and simple regular expressions [ACABJ04], respectively. Algorithms for solving games under incomplete information usually manipulate explicitly [Rei84] or symbolically represented sets of sets of states [CDHR06]. Thus, unsurprisingly, we employ the fact that the subword relation is in fact a BQO, a stronger notion than WQO that is preserved by the powerset operation [Mar01].

Contribution. This chapter is devoted to *lossy channel games under incomplete information* defined by Dimitrova and Finkbeiner in [DF13] and describes the results presented in [DF13]. These games are defined by what we call *partially specified LCSs* defined in Section 3.2. We extend the subword relation to a relation on states of the game structure that is a BQO and provide symbolic representations of upward and downward-closed sets of states of the game. Based on these representations we provide algorithms for solving lossy channel games under incomplete information with safety and reachability winning conditions and for computing finite-memory winning strategies for $Player_{\exists}$ (that corresponds to the synthesized process). Finally, we show that unfortunately the undecidability results established in [ABd08] for perfect information lossy channel games with weak parity winning condition in which only one player can lose messages transfer to the incomplete-information setting considered here.

In Chapter 4 we proceed to generalize the results from Chapter 3 by identifying conditions that define classes of game structures for which incomplete information games with safety and reachability conditions respectively are decidable. To the best of the author's knowledge, the only previously identified generic class of infinite-state incomplete-information games for which the game solving problem for safety and reachability is known to be decidable is the class of *games with finite R-stable quotient*. We demonstrate that there exist game structures that do not have a finite R-stable quotient but directly translate into game structures that fall into the classes we identified.

3.1 Preliminaries

Lossy channel systems. Lossy channel systems are asynchronous distributed systems composed of finitely many finite-state processes communicating through a finite set of unbounded FIFO channels that can nondeterministically lose messages.

Definition 3.1.1. A *lossy channel system (LCS)* is a tuple $\mathcal{L} = (\{\mathcal{A}_i\}_{i=0}^n, C, M, \{\Sigma_i\}_{i=0}^n)$, where for each *process identifier* $p \in \{0, \ldots, n\}$, $\mathcal{A}_p$ is a finite automaton describing the behavior of process p, C is a finite set of *channels*, M is a finite set of *messages* and $\Sigma = \dot{\bigcup}_{i=0}^n \Sigma_i$ is the union of the disjoint finite sets of *actions* for the processes. The automaton $\mathcal{A}_p = (Q_p, q_p^0, \delta_p)$ for a process p consists of a finite set Q_p of *control locations*, an *initial location* q_p^0 and a finite set δ of *transitions* of the form (q, a, Gr, Op, q'), where $q, q' \in Q_p$, $a \in \Sigma_p$, $Gr : C \to \{true, (= \epsilon), \in (m \cdot M^*) \mid m \in M\}$

and $Op : C \to \{!m, ?m, nop \mid m \in M\}$. Intuitively, the function Gr maps each channel to a guard, which can be an emptiness test, a test of the letter at the head of the channel or *true*. The function Op gives the update operation for the respective channel, which is either a write, a read or *nop*, which leaves the channel unchanged.

Remark. Unlike in the classical definition of LCSs, the transitions in the definition above are allowed to carry guards that test a channel for emptiness or probe its first letter. An extension of the basic model with regular guards was introduced in [BBS06].

A *configuration* $(q_0, \ldots, q_n, w) \in Q \times W$ of $\mathcal{L}$, where $Q = Q_0 \times \ldots \times Q_n$ and $W = \{w \mid w : C \to M^*\}$ is the set of possible channel valuations, is a tuple of the locations of the processes and a function $w : C \to M^*$ that maps each channel to its contents. The *initial configuration* of $\mathcal{L}$ is $(q_0^0, \ldots, q_n^0, \epsilon)$, where $\epsilon(c) = \epsilon$ for each $c \in C$.

The *strong labeled transition relation* $\to \subseteq (Q \times W) \times \Sigma \times (Q \times W)$ of $\mathcal{L}$ consists of all tuples $((q_0, \ldots, q_n, w), \sigma, (q_0', \ldots, q_n', w'))$ (denoted $(q_0, \ldots, q_1, w) \xrightarrow{\sigma} (q_0', \ldots, q_n', w')$) such that if $\sigma \in \Sigma_p$, then $q_{p'}' = q_{p'}$ for each $p' \neq p$, and there is $(q_p, \sigma, Gr, Op, q_p') \in \delta_p$ such that for each $c \in C$ all of the following conditions hold:

(i) if $Gr(c) = (\in m \cdot M^*)$ then $w(c) \in m \cdot M^*$,

(ii) if $Gr(c) = (= \epsilon)$ then $w(c) = \epsilon$,

(iii) if $Op(c) = !m$, then $w'(c) = w(c) \cdot m$,

(iv) if $Op(c) = ?m$, then $m \cdot w'(c) = w(c)$,

(v) if $Op(c) = nop$, then $w'(c) = w(c)$.

Let $\preceq$ denote the (not necessarily contiguous) subword relation on M^*. We extend $\preceq$ to W as follows: $w_1 \preceq w_2$ for $w_1, w_2 \in W$ iff $w_1(c) \preceq w_2(c)$ for every $c \in C$.

The *weak labeled transition relation* $\Rightarrow \subseteq (Q \times W) \times \Sigma \times (Q \times W)$ for $\mathcal{L}$ is defined as follows: $(q, w) \stackrel{\sigma}{\Rightarrow} (q', w')$ iff there exist w_1 and w_2 such that $w_1 \preceq w$ and $w' \preceq w_2$ and $(q, w_1) \xrightarrow{\sigma} (q', w_2)$, i.e., the channels can lose messages before and after the transition.

Well-quasi orderings. A *well-quasi ordering* (WQO) $(X, \preceq)$ consists of a set X and a reflexive and transitive relation $\preceq$ on X such that for every infinite sequence $x_0, x_1, \ldots$ of elements of X there exist indices $0 \leq i < j$ such that $x_i \preceq x_j$. A WQO $(X, \preceq)$ is *decidable*, if for each x_1 and x_2 in X it can be effectively checked that $x_1 \preceq x_2$.

For a finite alphabet M and subword ordering $\preceq$ on M^*, $(M^*, \preceq)$ is a WQO. If $\preceq$ is the ordering on the set W defined above for a finite set of channels C, $(W, \preceq)$ is a WQO as well. Furthermore, both of these WQOs are decidable.

Let $(X, \preceq)$ be a WQO. A set $U \subseteq X$ is *upward-closed* w.r.t. $\preceq$ iff for every $x \in U$ and every $x' \in X$ with $x \preceq x'$ it holds that $x' \in U$. A set $D \subseteq X$ is *downward-closed* w.r.t. $\preceq$ iff for every $x \in D$ and every $x' \in X$ with $x' \preceq x$ it holds that $x' \in D$. When $(X, \preceq)$ is clear from the context we just refer to upward and downward-closed sets. We denote with $\mathcal{U}(X)$ and $\mathcal{D}(X)$ the sets of upward-closed and downward-closed subsets of X respectively. Upward-closed sets of WQOs have the following important property.

Property 2. *Any increasing sequence $U_0 \subseteq U_1 \ldots$ of elements of $\mathcal{U}(X)$ must eventually stabilize, i.e., there exists an index $k \geq 0$ such that $(\bigcup_{i \geq 0} U_i) = U_k = U_{k+1} = \ldots$.*

It is well known that each upward-closed set can be uniquely and finitely represented by its set of minimal elements. This is the case, since, if U is an upward-closed set and $\mathsf{Min}(U)$ is the set of $\preceq$-minimal elements of U, then $\mathsf{Min}(U)$ is guaranteed to be finite by the definition of WQO. For a set $Y \subseteq X$, we define $Y\uparrow = \{x' \in X \mid \exists x \in Y.\ x \preceq x'\}$. Thus, $U = \mathsf{Min}(U)\uparrow$. In the particular case of the subword ordering $(M^*, \preceq)$, upward closed sets can be also represented by regular expressions (or finite automata), since they are regular languages over M. This representation readily extends to upward-closed sets in $(W, \preceq)$ for a LCS, by using *indexed regular expressions*.

Downward-closed sets are, in the general case, more difficult to represent effectively. Downward-closed languages over a finite alphabet M, however, are regular and thus can be represented by regular expressions. Furthermore, they can be represented by a subclass of regular expressions, called *simple regular expressions* (SREs) introduced in [ACABJ04]. For an alphabet M, the SREs are defined by the following grammar:

$$\begin{array}{rcl} \mathit{atom} & ::= & (m + \epsilon) \mid (m_1 + \ldots + m_n)^* \\ \mathit{product} & ::= & \epsilon \mid \mathit{atom} \cdot \mathit{product} \\ \mathit{SRE} & ::= & \emptyset \mid \mathit{product}[+\mathit{SRE}]. \end{array}$$

For a finite set of channels C and finite message alphabet M, $\mathsf{SRE}(C, M)$ is the set of indexed SREs, which are mappings from C to the set of SREs over M. A downward-closed subset of W can be represented as a finite union of indexed SREs.

Clearly, membership and inclusion for indexed SREs, as well as for finitely represented upward-closed sets in $(W, \preceq)$ can be effectively checked.

Let $(X, \preceq)$ be a quasi-ordering and the relation $\sqsubseteq$ on $\mathcal{P}(X)$ (the powerset of X) be defined such that $X_1 \sqsubseteq X_2$ iff for every $x_1 \in X_1$ there exists a $x_2 \in X_2$ such that $x_1 \preceq x_2$. If $(X, \preceq)$ is a WQO, then $(\mathcal{P}_{\mathsf{fin}}(X), \sqsubseteq)$ is a WQO, where $\mathcal{P}_{\mathsf{fin}}(X)$ is the set of finite subsets of X. This property does not extend to infinite subsets for the general case of any WQO. However, we will now see that most natural WQOs enjoy this property.

Better-quasi orderings. The theory of *better-quasi orderings* (BQOs) is a refinement of the theory of WQOs. Although the notion of BQO is stronger than WQO, the WQOs typically used in verification are also BQOs. We now recall the definition of BQO and state the properties that are of interest in the context of this thesis.

The sets $\mathbb{N}^{<*}$ and $\mathbb{N}^{<\omega}$ consist of the finite, respectively infinite, strictly increasing sequences of natural numbers. For $\nu_1 \in \mathbb{N}^{<*}$ and $\nu_2 \in \mathbb{N}^{<*} \cup \mathbb{N}^{<\omega}$, $\nu_1 < \nu_2$ denotes that ν_1 is a proper prefix of ν_2. For $\nu \in \mathbb{N}^{<*}$, $set(\nu)$ is the set of numbers in ν, and for a non empty $\nu \in \mathbb{N}^{<*}$, $tail(\nu)$ is the sequence obtained from ν by removing its first element.

A *barrier* is an infinite subset β of $\mathbb{N}^{<*}$ that satisfies the following conditions:

- there do not exist $\nu_1, \nu_2 \in \beta$ such that $set(\nu_1) \subsetneq set(\nu_2)$ and
- for every $\nu \in \mathbb{N}^{<\omega}$ there exists $\nu' \in \beta$ such that $\nu' < \nu$.

A quasi-ordering $(X, \preceq)$ is a BQO if for each function $f : \beta \to X$ where β is a barrier, there exist $\nu_1, \nu_2 \in \beta$ such that $tail(\nu_1) < \nu_2$ and $f(\nu_1) \preceq f(\nu_2)$.

By taking the set $\mathbb{N}$ as a barrier it is easily shown that each BQO is a WQO [Abd10].

BQOs enjoy the following property, used in Section 3.3 of this thesis.

Property 3. *If $(X, \preceq)$ is a BQO then, $(\mathcal{U}(X), \supseteq)$ and $(\mathcal{D}(X), \subseteq)$ are also BQOs.*

As for WQOs, if $(X, \preceq)$ is a BQO, then $(\mathcal{P}_{\mathsf{fin}}(X), \sqsubseteq)$ is also a BQO.

Among others, the subword ordering $(M^*, \preceq)$ and the ordering $(W, \preceq)$ on the set of channel valuations in a LCS are also BQOs.

3.2 Lossy Channel Games under Incomplete Information

Partially specified LCSs. In a *partially specified* lossy channel system, we distinguish between two types of nondeterminism: the "hostile" nondeterminism due to the model, and the "friendly" one resulting from unresolved implementation decisions that

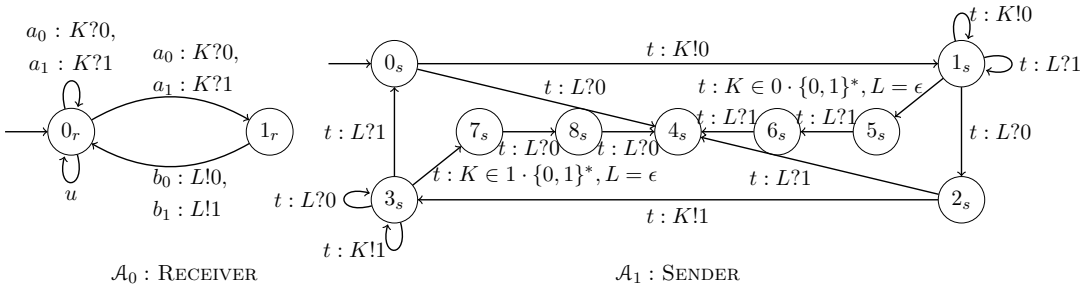


Figure 3.2: A communication protocol with a partially specified Receiver process. For process Receiver we have $\Sigma_0 = \{a_0, a_1, b_0, b_1, u\}$ and $\Sigma_\exists = \{b_0, b_1\}$. The property that the implementation must satisfy is that location 4 in Sender is not reachable, i.e., the receiver does not acknowledge messages that have not been sent, and once all messages and acknowledgements from previous phases have been consumed, the receiver can only send one delayed acknowledgement. Note that by using an extra channel and an extra location in process Receiver we can ensure that the error location is in process Receiver.

can be resolved in a favorable way. We consider the case when these decisions are within a single process, and thus, w.l.o.g. assume that the system consist of only two processes: the process under consideration and the parallel composition of the remaining ones.

Definition 3.2.1. A *partially specified lossy channel system* is tuple $(\mathcal{L}, \Sigma_{\mathsf{c}}, C_{obs})$, where $\mathcal{L} = (\mathcal{A}_0, \mathcal{A}_1, C, M, \Sigma_0, \Sigma_1)$ is a LCS, $\Sigma_{\mathsf{c}} \subseteq \Sigma_0$ is a subset of the actions of the *partially specified process* $\mathcal{A}_0$ and $C_{obs} \subseteq C$ is a set of *observable channels* that includes the set of all channels occurring in guards or read operations in $\mathcal{A}_0$.

Example 3.2.1. Figure 3.2 depicts a partially specified communication protocol consisting of two processes, Sender and Receiver, exchanging messages over the unreliable channels K and L. Process Sender sends messages to Receiver over channel K and Receiver acknowledges the receipt of a message using channel L. The two processes are represented as nondeterministic finite-state automata. Process Sender essentially runs the Alternating Bit Protocol. Process Receiver, however, is only partially specified: its alphabet of actions $\Sigma_0 = \{a_0, a_1, b_0, b_1, u\}$ is partitioned according to the unresolved decisions in the process specification: The subset $\Sigma_\exists = \{b_0, b_1\}$ of controllable actions specifies the unresolved implementation decisions concerning the write operations, i.e., what acknowledgement bit to send on channel L at location 1_r.

The the protocol must satisfy the conjunction of the following requirements:

1. Receiver does not acknowledge messages that have not been sent. That is, when the current location of Sender is 2_s, the language of channel L is 0^* and

for current location 0_s, the language of channel L is 1^*.

2. Once all messages and acknowledgements trailing from previous phases have been consumed (or lost), RECEIVER sends at most one delayed acknowledgement.

The above property is encoded as the unreachability of location 4_s in process SENDER. The transitions from locations 0_s and 2_s to 4_s are enabled when the first part of the property is violated. When the second pat of the property is violated, the sequences of transitions 1_s, 5_s, 6_s to 4_s and 3_s, 7_s, 8_s to 4_s respectively, can be taken. Note that by adding an extra *error location* in RECEIVER and either an extra symbol to the alphabet of channel K or an extra channel, we can encode the property in a way that the location that must be avoided is in the partially specified process $\mathcal{A}_0$. □

Let $(\mathcal{L}, \Sigma_{\mathsf{c}}, C_{obs})$ be a partially specified LCS with $\mathcal{L} = (\mathcal{A}_0, \mathcal{A}_1, C, M, \Sigma_0, \Sigma_1)$.

We define $H_{obs} = C_{obs} \to (M \cup \{\epsilon\})$. The function $obs : (Q \times W) \to (Q_0 \times H_{obs})$ maps each configuration (q_0, q_1, w) to the tuple $obs((q_0, q_1, w)) = (q_0, h)$, where for each $c \in C_{obs}$, if $w(c) = \epsilon$, then $h(c) = \epsilon$ and if $w(c) = m \cdot w'$ for some $m \in M$ and $w' \in M^*$, then $h(c) = m$. That is, for non-empty $c \in C_{obs}$, $h(c)$ is the letter at the head of $w(c)$.

Let $\mathsf{Enabled}((q_0, q_1, w)) = \{\sigma \in \Sigma_0 \mid \exists (q_0', q_1', w').\ (q_0, q_1, w) \xrightarrow{\sigma} (q_0', q_1', w')\}$. Note that for configurations (q_0, q_1, w) and (q_0', q_1', w') with $obs((q_0, q_1, w)) = obs((q_0', q_1', w'))$ it holds that $\mathsf{Enabled}((q_0, q_1, w)) = \mathsf{Enabled}((q_0', q_1', w'))$, and, abusing notation, we denote this set of actions with $\mathsf{Enabled}((q_0, h))$, where $(q_0, h) = obs(q_0, q_1, w)$.

Let us denote $\Sigma_c^\flat = \Sigma_{\mathsf{c}} \dot\cup \{\flat\}$. We define the functions $Act_\exists : Q_0 \times H_{obs} \to 2^{\Sigma_c^\flat}$ and $Act_\forall : Q_0 \times H_{obs} \times \Sigma_c^\flat \to 2^{\Sigma_0}$ as follows. For $o \in Q_0 \times H_{obs}$ we let $Act_\exists(o)$ be the smallest subset of $\Sigma_c^\flat$ such that $(\mathsf{Enabled}(o) \cap \Sigma_\exists) \subseteq Act_\exists(o)$ and if $\mathsf{Enabled}(o) \cap \Sigma_{\mathsf{c}} = \emptyset$ or $\mathsf{Enabled}(o) \cap (\Sigma_0 \setminus \Sigma_{\mathsf{c}}) \neq \emptyset$ then $\flat \in Act_\exists(o)$. For $o \in Q_0 \times H_{obs}$ and $\sigma_\exists \in \Sigma_c^\flat$ we define $Act_\forall(o, \sigma_\exists) = (\{\sigma_\exists\} \cap \Sigma_{\mathsf{c}}) \cup (\mathsf{Enabled}(o) \setminus \Sigma_{\mathsf{c}})$. The function $Act_\exists$ maps an observation to the set consisting of the enabled controllable actions and the special element $\flat$, and the function $Act_\forall$, given an observation and a controllable action or $\flat$, gives the set of actions that are the provided controllable action, or uncontrollable actions from Σ_0.

A *controller* for the partially specified LCS $(\mathcal{L}, \Sigma_{\mathsf{c}}, C_{obs})$ is a finite automaton $\mathcal{A}_c = (Q_c, q_c^0, (Q_0 \times H_{obs}) \times (\Sigma_c^\flat \times \Sigma_0), \rho)$ with alphabet $(Q_0 \times H_{obs}) \times (\Sigma_c^\flat \times \Sigma_0)$, whose transition relation $\rho \subseteq (Q_c \times ((Q_0 \times H_{obs}) \times (\Sigma_c^\flat \times \Sigma_0)) \times Q_c)$ has the following properties:

(i) for each $q \in Q_c$, $o \in Q_0 \times H_{obs}$, $\sigma_\exists \in \Sigma_c^\flat$, $\sigma \in \Sigma_0$, and $q_1', q_2' \in Q_c$, it holds that if $(q, (o, (\sigma_\exists, \sigma)), q_1') \in \rho$ and $(q, (o, (\sigma_\exists, \sigma)), q_2') \in \rho$, then $q_1' = q_2'$ (ρ is deterministic),

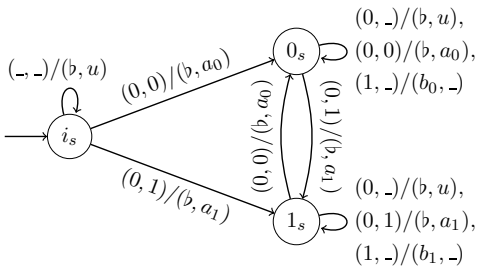


Figure 3.3: A controller for the partially specified LCS shown in Fig. 3.2.

(ii) for each $q \in Q_c$ and $o \in Q_0 \times H_{obs}$ there exist $\sigma_\exists \in Act_\exists(o)$, $\sigma \in \Sigma_0$ and $q' \in Q_c$ such that $(q,(o,(\sigma_\exists,\sigma)),q') \in \rho$ (there is a transition for each observation),

(iii) if $(q,(o,(\sigma_\exists,\sigma_1)),q'_1) \in \rho$ and $\sigma_2 \in Act_\forall(o,\sigma_\exists)$, then $(q,(o,(\sigma_\exists,\sigma_2)),q'_2) \in \rho$ for some $q'_2 \in Q_c$ (ρ does not restrict the actions $Act_\forall(o,\sigma_\exists)$),

(iv) if $(q,(o,(\sigma_{\exists 1},\sigma_1)),q'_1) \in \rho$ and $(q,(o,(\sigma_{\exists 2},\sigma_2)),q'_2) \in \rho$, then $\sigma_{\exists 1} = \sigma_{\exists 2}$ (the action from $\Sigma_c^\flat$ depends only on the current state and observation).

Figure 3.3 shows a controller for the partially specified LCS from Example 3.2.1.

Let $\mathcal{A}_c = (Q_c, q_c^0, (Q_0 \times H_{obs}) \times (\Sigma_c^\flat \times \Sigma_0), \rho)$ be a controller for the partially specified LCS $(\mathcal{L}, \Sigma_\mathtt{c}, C_{obs})$. The *product* of $\mathcal{L}$ and $\mathcal{A}_c$, denoted $\mathcal{C} \mid\mid \mathcal{L}$ is the LCS $\mathcal{L}^c = (\mathcal{A}_0^c, \mathcal{A}_1, C, M, \Sigma_0, \Sigma_1)$ where the automaton $\mathcal{A}_0^c = (Q_0^c, q_{0,c}^0, \delta^c)$ consists of:

- the set of locations $Q_0^c = Q_0 \times Q_c$,
- the initial location $q_{0,c}^0 = (q_0^0, q_c^0)$,
- the transition relation δ^c is the smallest subset of $(Q_0 \times Q_c) \times \Sigma_0 \times \mathcal{GR} \times \mathcal{OP} \times (Q_0 \times Q_c)$, where $\mathcal{GR} = (C_{obs} \to \{true, (= \epsilon), \in (m \cdot M^*) \mid m \in M\})$ and $\mathcal{OP} = (C \to \{!m, ?m, nop \mid m \in M\})$, such that if $(q_0, \sigma, Gr, Op, q'_0) \in \delta_0$, $(q_c, (q_0, h)/(\sigma_\exists, \sigma), q'_c) \in \rho$ and for every $c \in C_{obs}$ one of the following holds:
  - $h(c) = \epsilon,\ Gr(c) \in \{(= \epsilon), true\},\ Op(c) \in \{!m, nop \mid m \in M\}$,
  - $h(c) = m,\ Gr(c) \in \{(\in m \cdot M^*), true\},\ Op(c) \in \{?m, !m', nop \mid m' \in M\}$,

  then $(q_0, \sigma, Gr^c, Op, q'_0) \in \delta_0^c$, where for every $c \in C_{obs}$ we define

$$Gr^c(c) = \begin{cases} (= \epsilon) & \text{if } h(c) = \epsilon, \\ (\in m \cdot M^*) & \text{if } h(c) = m. \end{cases}$$

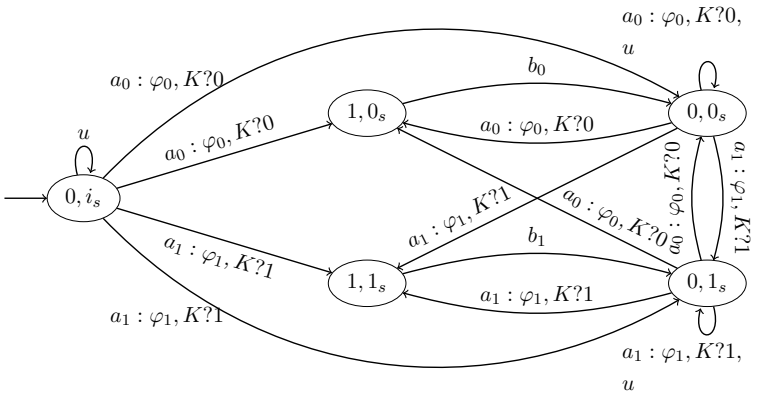


Figure 3.4: The nondeterministic finite-state automaton for the RECEIVER process obtained by the composition of the partially specified process in Fig. 3.2 and the controller automaton in Fig. 3.3. The result has been simplified by merging transitions. The guards φ_0 and φ_1 stand for $K \in 0 \cdot \{0,1\}^*$ and $K \in 1 \cdot \{0,1\}^*$ respectively. Note that in this case we could have omitted the guards as they are implied by those of the read operations.

Figure 3.4 depicts the RECEIVER process in the LCS obtained by the composition of the LCS from Example 3.2.1 and the controller in Fig. 3.3.

Given a partially specified LCS $(\mathcal{L}, \Sigma_{\mathsf{c}}, C_{obs})$ with $\mathcal{L} = (\mathcal{A}_0, \mathcal{A}_1, C, M, \Sigma_0, \Sigma_1)$ and a set $Q_{Err} \subseteq Q_0$ of error locations in $\mathcal{A}_0$, the *safety synthesis problem for LCSs* asks to construct a controller $\mathcal{C}$ for $(\mathcal{L}, \Sigma_{\mathsf{c}}, C_{obs})$ such that the set of locations Q_{Err} is not reachable in $\mathcal{C} \parallel \mathcal{L}$. Similarly, for a set $Q_{Goal} \subseteq Q_0$ of goal locations in $\mathcal{A}_0$, the *reachability synthesis problem for LCSs* asks to construct a controller $\mathcal{C}$ for $(\mathcal{L}, \Sigma_{\mathsf{c}}, C_{obs})$ such that the set of locations Q_{Goal} is reachable on every path in $\mathcal{C} \parallel \mathcal{L}$.

LC-Game structures with incomplete information. Let $(\mathcal{L}, \Sigma_\exists, C_{obs})$ be a partially specified LCS, where $\mathcal{L} = (\mathcal{A}_0, \mathcal{A}_1, C, M, \Sigma_0, \Sigma_1)$. We define the corresponding game structure with incomplete information $G(\mathcal{L}, \Sigma_\exists, C_{obs})$ as the tuple $G(\mathcal{L}, \Sigma_{\mathsf{c}}, C_{obs}) = (S_\exists, S_\forall, I, =_{\mathsf{o}}, \Sigma_\exists, T_\exists, T_\forall)$ that consists of the following components:

- $S_\exists = \{\exists\} \times \{0,1\} \times Q \times W \times \Sigma_c^\flat \times (\Sigma_0 \cup \{\flat\})$,
- $S_\forall = \{\forall\} \times \{0,1\} \times Q \times W \times \Sigma_c^\flat \times (\Sigma_0 \cup \{\flat\})$,
- $I = \{(\exists, 0, q_0^0, q_1^0, \epsilon, \flat, \flat), (\forall, 1, q_0^0, q_1^0, \epsilon, \flat, \flat)\}$,
- $=_{\mathsf{o}}$, which is such that $(t, p, q_0, q_1, w, \sigma_\exists, \sigma) =_{\mathsf{o}} (t', p', q_0', q_1', w', \sigma_\exists{}', \sigma')$ iff

  - $p = p' = 0$, $t = t'$, $obs((q_0, q_1, w)) = obs((q_0', q_1', w'))$, $\sigma_\exists = \sigma_\exists'$, $\sigma = \sigma'$; or
  - $p = p' = 1$, $t = t'$, $q_0 = q_0'$, $\sigma_\exists = \sigma_\exists'$ and $\sigma = \sigma'$.

- $\Sigma_\exists = \Sigma_c^\flat$,

- $T_\exists = \{((\exists, 0, q, w, \sigma_\exists, \sigma), (\forall, 0, q, w, \sigma_\exists', \sigma)) \mid \sigma_\exists' \in Act_\exists(obs((q, w)))\}$,

- $$\begin{aligned} T_\forall \;=\; & \{((\forall, 0, q, w, \sigma_\exists, \sigma), (\exists, 0, q', w', \sigma_\exists, \sigma')) \mid \sigma' \in Act_\forall(obs((q, w)), \sigma^\exists), (q, w) \overset{\sigma'}{\Rightarrow} (q', w')\} \\ \cup\; & \{((\forall, 0, q, w, \sigma_\exists, \sigma), (\forall, 1, q', w', \sigma_\exists, \sigma')) \mid \sigma' \in Act_\forall(obs((q, w)), \sigma^\exists), (q, w) \overset{\sigma'}{\Rightarrow} (q', w')\} \\ \cup\; & \{((\forall, 1, q, w, \sigma_\exists, \sigma), (\exists, 0, q', w', \sigma_\exists, \sigma)) \mid \exists \sigma' \in \Sigma_1.(q, w) \overset{\sigma'}{\Rightarrow} (q', w')\} \\ \cup\; & \{((\forall, 1, q, w, \sigma_\exists, \sigma), (\forall, 1, q', w', \sigma_\exists, \sigma)) \mid \exists \sigma' \in \Sigma_1.(q, w) \overset{\sigma'}{\Rightarrow} (q', w')\}. \end{aligned}$$

The first two components t and p of a state $(t, p, q_0, q_1, w, \sigma_\exists, \sigma)$ identify the player to whom the state belongs and the process to be executed, respectively. The tuple (q_0, q_1, w) encodes the current configuration of $\mathcal{L}$. Finally, the last two components $\sigma_\exists$ and σ encode the last action chosen by $Player_\exists$ and the action of the last transition of process $\mathcal{A}_0$ that was executed. The game starts at one of two initial states depending on which process is initially scheduled, and the initial state in which process 0 is scheduled belongs to $Player_\exists$ and the initial state in which process 1 is scheduled belongs to $Player_\forall$. The transition relation $T_\exists$ allows $Player_\exists$ to choose in each state from $S_\exists$ an action in $\Sigma_c^\flat$, that is either an action from Σ_c of a transition enabled in the current state, or can be the special element $\flat$ in case no transition with action in Σ_c is enabled or if there exists an enabled transition with action from $\Sigma_0 \setminus \Sigma_\mathsf{c}$. The transition relation $T_\forall$ of $Player_\forall$ encodes the possible transitions of $\mathcal{L}$ respecting the currently scheduled process and the current choice of $Player_\exists$ in case this is process 0. When process 0 is scheduled, any transition of $\mathcal{A}_0$ is with an action either in $\Sigma_0 \setminus \Sigma_\mathsf{c}$ or with the action in Σ_c that was chosen by $Player_\exists$. Note that if in the successor state process 0 is scheduled, then the turn is given again to $Player_\exists$ to make a choice.

Remark. The second component of states in S is used to model the interleaving semantics and is updated by $Player_\forall$. For simplicity, in the definition of the game structure above we do not make any assumptions about the choice of which process to be executed. One natural assumption one might make is that the selected process must have at least one transition enabled in the current state. This and other restrictions on the scheduling decisions of $Player_\forall$ can be easily imposed in the above model.

The observation equivalence $=_{\mathsf{o}}$ allows $Player_{\exists}$ to distinguish states in which process 0 is scheduled according to the current process 0 location, the letters at the heads of the observable channels and the actions. States in which process 1 is scheduled, on the other hand, cannot be distinguished based on the symbols of the observable channels.

Remark. Here, like in the classical definition of LCSs we allow for message losses both before and after an actual transition. This differs from the definition of lossy channel games in [ABd08], where messages can be lost only before a transition, which implies that player cannot immediately lose the message he just wrote. We considered the choice we made appropriate in our case, since all message losses are controlled by $Player_{\forall}$ representing the environment. Note that we allow a message at the head of an observable channel to be lost after it has been observed but not consumed by $Player_{\exists}$.

LC-Games under incomplete information. Given a partially specified lossy channel system $(\mathcal{L}, \Sigma_{\mathsf{c}}, C_{obs})$ with $\mathcal{L} = (\mathcal{A}_0, \mathcal{A}_1, C, M, \Sigma_0, \Sigma_1)$ and a set $Q_{Err} \subseteq Q_0$ of error locations in $\mathcal{A}_0$, we define the set Err of error states in the game structure $G(\mathcal{L}, \Sigma_{\mathsf{c}}, C_{obs}) = (S_{\exists}, S_{\forall}, I, =_{\mathsf{o}}, \Sigma_{\exists}, T_{\exists}, T_{\forall})$ as $Err = \{(t, p, q_0, q_1, w, \sigma_{\exists}, \sigma) \mid q_0 \in Q_{Err} \wedge p = 0\}$. The corresponding safety game under incomplete information is $\mathsf{Safety}(G, Err)$. Similarly, given a set $Q_{Goal} \subseteq Q_0$ of goal locations in $\mathcal{A}_0$ we define the reachability game under incomplete information $\mathsf{Reach}(G, Goal)$, where the set of goal states is $Goal = \{(t, p, q_0, q_1, w, \sigma_{\exists}, \sigma) \mid q_0 \in Q_{Goal} \wedge p = 0\}$.

Note that each $Player_{\exists}$-state in the game structure $G(\mathcal{L}, \Sigma_{\mathsf{c}}, C_{obs})$ has a successor. For a safety game, we can easily instrument $\mathcal{L}$ and the set of error locations Q_{Err} in a way that every $Player_{\forall}$ state has a successor and still ensure that plays reaching a state in G that corresponds to a deadlock in $\mathcal{L}$ are not winning for $Player_{\exists}$. Thus, we can assume w.l.o.g. that when we consider lossy channel games with safety winning conditions, all plays in the game structure are infinite.

Note that by the definition of the observation equivalence $=_{\mathsf{o}}$ in $G(\mathcal{L}, \Sigma_{\mathsf{c}}, C_{obs})$, the sets Err and $Goal$ are observable by construction. Thus, the games $\mathsf{Safety}(G, Err)$ and $\mathsf{Reach}(G, Goal)$ have winning conditions that are observable w.r.t. the observation functions obs_s, obs_a and obs_f. The algorithms presented in Section 3.3, however, rely only on observability w.r.t. the observation function obs_s.

Proposition 3.2.1. *Let $(\mathcal{L}, \Sigma_{\mathsf{c}}, C_{obs})$ be a partially specified lossy channel system with $\mathcal{L} = (\mathcal{A}_0, \mathcal{A}_1, C, M, \Sigma_0, \Sigma_1)$ and $Q_F \subseteq Q_0$ be a set of locations in $\mathcal{A}_0$. Let $G(\mathcal{L}, \Sigma_{\mathsf{c}}, C_{obs})$*

be the game structure with incomplete information defined by $(\mathcal{L}, \Sigma_{\mathsf{c}}, C_{obs})$ *and* $F = \{(t, p, q_0, q_1, w, \sigma_\exists, \sigma) \mid q_0 \in Q_F \wedge p = 0\}$. *Then, the following properties hold.*

1. *There exists a finite controller* $\mathcal{C}$ *for* $(\mathcal{L}, \Sigma_{\mathsf{c}}, C_{obs})$ *such that the set of locations* Q_F *is not reachable in* $\mathcal{C} \,||\, \mathcal{L}$ *if and only if* $Player_\exists$ *has a finite-state* obs_a*-consistent winning strategy in the incomplete-information game* $\mathsf{Safety}(G, F)$.

2. *There exists a finite controller* $\mathcal{C}$ *for* $(\mathcal{L}, \Sigma_{\mathsf{c}}, C_{obs})$ *such that the set of locations* Q_F *is reachable on every path in* $\mathcal{C} \,||\, \mathcal{L}$ *if and only if* $Player_\exists$ *has a finite-state* obs_a*-consistent winning strategy in the incomplete-information game* $\mathsf{Reach}(G, F)$.

3.3 Algorithms for Safety and Reachability Games

3.3.1 Monotonicity Properties of the Transition Relations

Let $(\mathcal{L}, \Sigma_{\mathsf{c}}, C_{obs})$ be a partially specified LCS with $\mathcal{L} = (\mathcal{A}_0, \mathcal{A}_1, C, M, \Sigma_0, \Sigma_1)$ and $G(\mathcal{L}, \Sigma_{\mathsf{c}}, C_{obs}) = (S_\exists, S_\forall, I, =_{\mathsf{o}}, \Sigma_\exists, T_\exists, T_\forall)$ be the corresponding game structure with incomplete information. As before, we denote with $S = S_\exists \dot\cup S_\forall$ the set of states of G and with Obs the set of equivalence classes of S w.r.t. the equivalence relation $=_{\mathsf{o}}$.

We define the relation $\preceq$ on $S = S_\exists \dot\cup S_\forall$ as follows: for $s = (t, p, q_0, q_1, w, \sigma_\exists, \sigma)$ and $s' = (t', p', q_0', q_1', w', \sigma_\exists', \sigma')$, we have $s \preceq s'$ iff the following conditions hold:

- $t = t'$, $p = p'$, $q_0 = q_0'$, $q_1 = q_1'$, $\sigma_\exists = \sigma_\exists'$, $\sigma = \sigma'$,
- $obs((q_0, q_1, w)) = obs((q_0', q_1', w'))$ and $w \preceq w'$.

Since all the sets $\{\exists, \forall\}$, $\{0, 1\}$, Q_0, Q_1, Σ_0 are finite and the quasi-ordering $(W, \preceq)$ is a BQO, it holds that $(S, \preceq)$ is a BQO as well.

The definition of $(S, \preceq)$ directly implies that if $s \preceq s'$ then $s =_{\mathsf{o}} s'$.

By definition, the transition relation $\Rightarrow$ of the LCS $\mathcal{L}$ enjoys the monotonicity property of being *downward-closed*: If for configurations (q_0, q_1, w) and (q_0', q_1', w') and action σ, $(q_0, q_1, w) \xrightarrow{\sigma} (q_0', q_1', w')$, then for every w'' with $w \preceq w''$ it holds that $(q_0, q_1, w'') \xrightarrow{\sigma} (q_0', q_1', w')$. Furthermore, if for configurations (q_0, q_1, w) and (q_0', q_1', w') and action σ, $(q_0, q_1, w) \xrightarrow{\sigma} (q_0', q_1', w')$, then for every w'' with $w'' \preceq w'$ it holds that $(q_0, q_1, w) \xrightarrow{\sigma} (q_0', q_1', w'')$. We will now see how these properties extend to G.

The definition of $\preceq$ on S implies that the following properties hold for the functions $Act_\exists$ and $Act_\forall$ defined in Section 3.2.

- For states $s = (\exists, p, q_0, q_1, w, \sigma_\exists, \sigma)$ and $s' = (\exists, p', q_0', q_1', w', \sigma_\exists', \sigma')$ such that $s \preceq s'$, it holds that $Act_\exists(obs(q_0, q_1, w)) = Act_\exists(obs(q_0', q_1', w'))$.
- For states $s = (\forall, p, q_0, q_1, w, \sigma_\exists, \sigma)$ and $s' = (\forall, p', q_0', q_1', w', \sigma_\exists', \sigma')$ such that $s \preceq s'$, it holds that $Act_\forall(obs(q_0, q_1, w), \sigma_\exists) = Act_\forall(obs(q_0', q_1', w'), \sigma_\exists')$.

As a consequence we can conclude that the transition relation for $Player_\exists$ is monotonic and that the transition relation for $Player_\forall$ is downward-closed.

Property 4. *If $(s, \sigma, s') \in T_\exists$ and $s \preceq s''$ for some state $s'' \in S_\exists$, then there exists a state $s''' \in S_\forall$ such that $(s'', \sigma, s''') \in T_\exists$ and $s' \preceq s'''$.*

Property 5. *If $(s, s') \in T_\forall$ and $s \preceq s''$ for some state $s'' \in S_\forall$, then $(s'', s') \in T_\forall$. Thus, if $S' \subseteq S$ is a set of states, the set $\mathsf{Pre}_\forall(S')$ is an upward-closed set of states.*

By the second property of the transition relation $\Rightarrow$ of $\mathcal{L}$ we have the following.

Property 6. *If $(s, s') \in T_\forall$ and $s'' \preceq s'$ for some state $s'' \in S$, then $(s, s'') \in T_\forall$. Thus, if $S' \subseteq States$ then $\mathsf{Post}_\forall(S')$ is a downward closed set. Furthermore, note that by the definition of LCSs, if S' is finite, then so is $\mathsf{Post}_\forall(S')$.*

By the definition of the relation $\preceq$ it immediately follows that each observable set of states is both upward-closed and downward-closed. Hence, if the wining condition of a lossy channel game is defined by an observable set of error or goal states, then clearly this set of states is both upward-closed and downward-closed.

The above properties tell us, that upward and downward-closed sets of states will be useful for solving lossy channel games under incomplete information.

Let $U \subseteq S$ be an upward-closed set of states in G and let $o \in Obs$. By the definition of $\preceq$ we have that the set $U \cap o$ is also upward-closed. Similarly, for a downward-closed set $D \subseteq S$, the set $D \cap o$ is downward-closed. Let us define the families of state sets

$$\begin{aligned} \mathcal{U}_{obs}(G) &= \{U \subseteq S \mid U \neq \emptyset,\ U = U{\uparrow} \text{ and } \exists o \in Obs. U \subseteq o\} \text{ and} \\ \mathcal{D}_{obs}(G) &= \{D \subseteq S \mid D \neq \emptyset,\ D = D{\downarrow} \text{ and } \exists o \in Obs. D \subseteq o\}. \end{aligned}$$

3.3.2 Effective Representation of Upward and Downward-Closed Sets

We show how each element of $\mathcal{U}_{obs}(G)$ can be finitely represented as a tuple that has a part describing an equivalence class in Obs and a part defining an upward-closed set of states. Similarly we can finitely represent each set in $\mathcal{D}_{obs}(G)$ as a tuple that has a part describing an equivalence class and a part defining a downward-closed set of states.

3. LOSSY CHANNEL GAMES UNDER INCOMPLETE INFORMATION

Let $\Phi_{\mathcal{U}_{obs}} \subseteq \{\exists, \forall\} \times \{0,1\} \times Q_0 \times H_{obs} \times \Sigma_\exists \times (\Sigma_0 \cup \{\flat\}) \times \mathcal{P}_{\mathsf{fin}}(Q_1 \times W)$ be such that $(t, p, q_0, h, \sigma_\exists, \sigma, R) \in \Phi_{\mathcal{U}_{obs}}$ iff the following two conditions are satisfied:

(i) if $p = 1$, then $h = \epsilon$ and

(ii) if $p = 0$, then for each $c \in C_{obs}$, $h(c) = \epsilon$ implies $w(c) = \epsilon$ for each $(q_1, w) \in R$.

The idea is that $(t, p, q_0, h, \sigma_\exists, \sigma)$ defines an equivalence class and R describes an upward closed set of states. Formally, the semantics of $\Phi_{\mathcal{U}_{obs}}$ is defined as follows.

$$[\![(t, p, q_0, h, \sigma_\exists, \sigma, R)]\!] = \{(t, p, q_0, q_1, w, \sigma_\exists, \sigma) \in S \quad | \quad \exists w', w'' \in W.(q_1, w') \in R \wedge (\forall c \in C_{obs}.h(c) = \epsilon \Rightarrow w''(c) = \epsilon) \; w' \preceq w'' \wedge w = h \odot w''\},$$

where for $c \in C_{obs}$, $(h \odot w'')(c) = h(c) \cdot w''(c)$ and for $c \in C \setminus C_{obs}$, $(h \odot w'')(c) = w''(c)$.

The operation $\odot : H_{obs} \times W \times W$ above prepends the first letters of the nonempty observable channels to the contents of the respective channel in the valuation.

In a similar manner, we can represent each set in $\mathcal{D}_{obs}(G)$ using elements of the set $\Phi_{\mathcal{D}_{obs}} \subseteq \{\exists, \forall\} \times \{0,1\} \times Q_0 \times H_{obs} \times \Sigma_\exists \times (\Sigma_0 \cup \{\flat\}) \times \mathcal{P}_{\mathsf{fin}}(Q_1 \times \mathsf{SRE}(C, M))$ defined such that $(t, p, q_0, h, \sigma_\exists, \sigma, R) \in \Phi_{\mathcal{D}_{obs}}$ iff the following two conditions are satisfied:

(i) if $p = 1$, then $h = \epsilon$ and

(ii) if $p = 0$, then for each $c \in C_{obs}$, $h(c) = \epsilon$ implies $r(c) = \epsilon$ for each $(q_1, r) \in R$.

The semantics of $\Phi_{\mathcal{D}_{obs}}$ is such that:

$$[\![(t, p, q_0, h, \sigma_\exists, \sigma, R)]\!] = \{(t, p, q_0, q_1, w, \sigma_\exists, \sigma) \in S \quad | \quad \exists r \in \mathsf{SRE}(C, M).\ \exists w' \in [\![r]\!].\ (q_1, r) \in R \wedge w = h \odot w'\}.$$

By the definition of $\preceq$ on S, each observable set of states is both upward-closed and downward-closed. Therefore, each such set can be represented in any of the above two ways. In particular, this holds for the sets of error and goal states in safety and reachability lossy channel games under incomplete information.

Since $(S, \preceq)$ is a BQO, by Property 3 it holds that $(\mathcal{U}_{obs}(G), \supseteq)$ and $(\mathcal{D}_{obs}(G), \subseteq)$ are BQOs. Using the above representations, the respective inclusion relations can be effectively decided, and hence, the BQOs $(\mathcal{U}_{obs}(G), \supseteq)$ and $(\mathcal{D}_{obs}(G), \subseteq)$ are decidable.

3.3.3 Effective Successor and Predecessor Operations

We define the function $\mathsf{Pre}^{obs}_{\forall} : \mathcal{U}_{obs}(G) \to \mathcal{P}_{\mathsf{fin}}(\mathcal{U}_{obs}(G))$ that maps a set $U \in \mathcal{U}_{obs}(G)$ to a finite set of upward-closed sets that partitions the respective set of $\forall$-predecessors of U according to the observations $Player_{\exists}$ makes. Similarly, we define the successor function $\mathsf{Post}^{obs}_{\forall} : \mathcal{D}_{obs}(G) \to \mathcal{P}_{\mathsf{fin}}(\mathcal{D}_{obs}(G))$. Since the transition relation of G has finite branching, if $D \in \mathcal{D}_{obs}(G)$ is finite then each $D' \in \mathsf{Post}^{obs}_{\forall}(D)$ is finite too. Formally,

$$\begin{array}{lcl} \mathsf{Pre}^{obs}_{\forall}(U) & = & \{U' \in \mathcal{U}_{obs}(G) \mid \exists o \in Obs.\ U' = \mathsf{Pre}_{\forall}(U) \cap o\}, \\ \mathsf{Post}^{obs}_{\forall}(D) & = & \{D' \in \mathcal{D}_{obs}(G) \mid \exists o \in Obs.\ D' = \mathsf{Post}_{\forall}(D) \cap o\}. \end{array}$$

We define $\mathsf{Pre}^{obs}_{\exists} : \mathcal{U}_{obs}(G) \times \Sigma_{\exists} \to \mathcal{P}_{\mathsf{fin}}(\mathcal{U}_{obs}(G))$ and $\mathsf{Post}^{obs}_{\exists} : \mathcal{D}_{obs}(G) \times \Sigma_{\exists} \to \mathcal{D}_{obs}(G)$:

$$\begin{array}{lcl} \mathsf{Pre}^{obs}_{\exists}(U, \sigma_{\exists}) & = & \{U' \in \mathcal{U}_{obs}(G) \mid \exists o \in Obs.\ U' = \mathsf{Pre}_{\exists}(U) \cap o\}, \\ \mathsf{Post}^{obs}_{\exists}(D, \sigma_{\exists}) & = & \mathsf{Post}_{\exists}(D, \sigma_{\exists}) \text{ if } D \subseteq S_{\exists}. \end{array}$$

Based on the representation for $\mathcal{U}_{obs}(G)$ and $\mathcal{D}_{obs}(G)$ we described in Section 3.3.2, we define in the following two subsections functions for symbolic computation of $\mathsf{Pre}^{obs}_{\forall}(U)$, $\mathsf{Pre}^{obs}_{\exists}(U, \sigma_{\exists})$, $\mathsf{Post}^{obs}_{\forall}(D)$ and $\mathsf{Post}^{obs}_{\exists}(D, \sigma_{\exists})$ for $U \in \mathcal{U}_{obs}(G)$ and $D \in \mathcal{D}_{obs}(G)$.

Predecessor Operations

For the function that maps an element of $\Phi_{\mathcal{U}_{obs}}$ to a finite subset of $\Phi_{\mathcal{U}_{obs}}$ we use the same symbol as for the corresponding function $\mathsf{Pre}^{obs}_{\forall} : \mathcal{U}_{obs}(G) \to \mathcal{P}_{\mathsf{fin}}(\mathcal{U}_{obs}(G))$. We define $\mathsf{Pre}^{obs}_{\forall} : \Phi_{\mathcal{U}_{obs}} \to \mathcal{P}_{\mathsf{fin}}(\Phi_{\mathcal{U}_{obs}})$ as follows: for $(t', p', q'_0, h', \sigma'_{\exists}, \sigma', R') \in \Phi_{\mathcal{U}_{obs}}$ we let

$$\begin{array}{lcl} \mathsf{Pre}^{obs}_{\forall}((t', p', q'_0, h', \sigma'_{\exists}, \sigma', R')) & = & \{\ (\forall, 0, q_0, h, \sigma'_{\exists}, \sigma, R) \in \Phi_{\mathcal{U}_{obs}} \mid (q_1, w) \in R \text{ iff} \\ & & \quad \sigma' \in Act_{\forall}((q_0, h), \sigma'_{\exists}), \\ & & \quad \exists Gr. \exists Op. \exists w'. (q_0, \sigma', Gr, Op, q'_0) \in \delta_0, \\ & & \quad (q_1, w') \in R',\ (h, w) \in \mathsf{Pre}_{\mathcal{L}}((h', w'), Gr, Op, 0)\} \\ & \cup & \{\ (\forall, 1, q'_0, h, \sigma'_{\exists}, \sigma', R) \in \Phi_{\mathcal{U}_{obs}} \mid (q_1, w) \in R \text{ iff} \\ & & \quad \exists \sigma \in \Sigma_1. \exists Gr. \exists Op. \exists q'_1. \exists w'. (q_1, \sigma, Gr, Op, q'_1) \in \delta_1, \\ & & \quad (q'_1, w') \in R',\ (h, w) \in \mathsf{Pre}_{\mathcal{L}}((h', w'), Gr, Op, 1)\}. \end{array}$$

We define the function $\mathsf{Pre}_{\mathcal{L}} : (H_{obs} \times M^*) \times \mathcal{GR} \times \mathcal{OP} \times \{0, 1\} \to \mathcal{P}_{\mathsf{fin}}(H_{obs} \times M^*)$ such that $(h, w) \in \mathsf{Pre}_{\mathcal{L}}((h', w'), Gr, Op, p)$ iff

- for all $c \in C_{obs}$, $(h(c), w(c)) \in \mathsf{Extract}(p, c, (Gr(c), Op(c)) \otimes^{-1} (h'(c) \cdot w'(c)))$ and
- for all $c \in (C \setminus C_{obs})$, $h'(c) = \epsilon$ and $w'(c) \in (Gr(c), Op(c)) \otimes^{-1} w'(c)$,

where the definitions of the operator $\otimes^{-1}$ and the function $\mathsf{Extract} : \{0,1\} \times C_{obs} \times \mathcal{P}_{\mathsf{fin}}(M^*) \to \mathcal{P}_{\mathsf{fin}}((M \cup \{\epsilon\}) \times M^*)$ are given below.

For $w \in M^*$ and guard g the set $\mathsf{Sat}(w, g) \in \mathcal{P}_{\mathsf{fin}}(M^*)$ is defined as follows:

$$\mathsf{Sat}(w,g) = \begin{cases} \{w\} & \text{if } g = \mathit{true}, \\ \{w\} & \text{if } g = (\in m \cdot M^*) \text{ and } w = m \cdot w', \\ \{m \cdot w\} & \text{if } g = (\in m \cdot M^*) \text{ and } w \notin m \cdot M^*, \\ \{w\} & \text{if } g = (= \epsilon) \text{ and } w = \epsilon, \\ \emptyset & \text{if } g = (= \epsilon) \text{ and } w \neq \epsilon. \end{cases}$$

For $w' \in M^*$, guard g and op we define the set $(g, op) \otimes^{-1} w'$ as follows:

$$(g, op) \otimes^{-1} w' = \begin{cases} \mathsf{Sat}(w'', g) & \text{if } op = (!m) \text{ and } w' = w'' \cdot m, \\ \mathsf{Sat}(w', g) & \text{if } op = (!m) \text{ and } \mathsf{last}(w') \neq m, \\ \mathsf{Sat}(m \cdot w', g) & \text{if } op = (?m), \\ \mathsf{Sat}(w', g) & \text{if } op = (nop). \end{cases}$$

Finally, the function $\mathsf{Extract} : \{0,1\} \times C_{obs} \times \mathcal{P}_{\mathsf{fin}}(M^*) \to \mathcal{P}_{\mathsf{fin}}((M \cup \{\epsilon\}) \times M^*)$ is defined as $\mathsf{Extract}(p, c, A) = \bigcup_{w \in A} \mathsf{Extract}'(p, c, w)$, where:

$$\mathsf{Extract}'(p,c,w) = \begin{cases} \{(\epsilon, w)\} & \text{if } p = 1, \\ \{(\epsilon, \epsilon)\} \cup \{(m, \epsilon) \mid m \in M\} & \text{if } p = 0 \text{ and } w = \epsilon, \\ \{(m, w')\} \cup \{(m_1, m \cdot w') \mid m_1 \in M\} & \text{if } p = 0 \text{ and } w = m \cdot w'. \end{cases}$$

Similarly as for the respective function for $\mathit{Player}_\forall$, we define a function that maps an element of $\Phi_{\mathcal{U}_{obs}}$ to a finite subset of $\Phi_{\mathcal{U}_{obs}}$, corresponding to the function $\mathsf{Pre}^{obs}_\exists : \mathcal{U}_{obs}(G) \times \Sigma_\exists \to \mathcal{P}_{\mathsf{fin}}(\mathcal{U}_{obs}(G))$. We define $\mathsf{Pre}^{obs}_\exists : \Phi_{\mathcal{U}_{obs}} \times \Sigma_\exists \to \mathcal{P}_{\mathsf{fin}}(\Phi_{\mathcal{U}_{obs}})$ as follows:

$$\mathsf{Pre}^{obs}_\exists((t', p', q'_0, h', \sigma'_\exists, \sigma', R'), \sigma_\exists) = \begin{cases} \{(\exists, 0, q'_0, h', \sigma''_\exists, \sigma', R') \mid \sigma'' \in \Sigma_\exists\} & \text{if } t' = \forall, \\ & \quad p' = 0, \\ & \quad \sigma' = \sigma_\exists, \\ \emptyset & \text{otherwise.} \end{cases}$$

Successor Operations

For the function that maps an element of $\Phi_{\mathcal{D}_{obs}}$ to a finite subset of $\Phi_{\mathcal{D}_{obs}}$ we use the same symbol as for the corresponding function $\mathsf{Post}^{obs}_\forall : \mathcal{D}_{obs}(G) \to \mathcal{P}_{\mathsf{fin}}(\mathcal{D}_{obs}(G))$. We

define $\mathsf{Post}^{obs}_{\forall} : \Phi_{\mathcal{D}_{obs}} \to \mathcal{P}_{\mathsf{fin}}(\Phi_{\mathcal{D}_{obs}})$ as follows: for $(t, p, q_0, h, \sigma_\exists, \sigma, R) \in \Phi_{\mathcal{D}_{obs}}$ we let

$$\begin{array}{rcl}
\mathsf{Post}^{obs}_{\forall}((t,0,q_0,h,\sigma_\exists,\sigma,R)) & = & \{\ (t',p',q_0',h',\sigma_\exists,\sigma',R') \mid (q_1',e') \in R' \text{ iff} \\
& & \quad \sigma \in Act_\forall((q_0,h),\sigma_\exists), (t' = \exists \Leftrightarrow p' = 0), \\
& & \quad \exists Op.\exists Gr \exists e.(q_0,\sigma', Gr, Op, q_0') \in \delta_0, \\
& & \quad (q_1',e) \in R,\ (h',e') \in \mathsf{Post}_{\mathcal{L}}((h,e), Gr, Op, p')\}, \\
\mathsf{Post}^{obs}_{\forall}((t,1,q_0,h,\sigma_\exists,\sigma,R)) & = & \{\ (t',p',q_0,h',\sigma_\exists,\sigma,R') \mid (q_1',e') \in R' \text{ iff} \\
& & \quad (t' = \exists \Leftrightarrow p' = 0), \\
& & \quad \exists \sigma' \exists Op.\exists Gr \exists q_1.\exists e.(q_1,\sigma', Gr, Op, q_1') \in \delta_1, \\
& & \quad (q_1,e) \in R,\ (h',e') \in \mathsf{Post}_{\mathcal{L}}((h,e), Gr, Op, p')\}.
\end{array}$$

We define the function $\mathsf{Post}_{\mathcal{L}} : (H_{obs} \times \mathsf{SRE}(M,C)) \times \mathcal{GR} \times \mathcal{OP} \times \{0,1\} \to \mathcal{P}_{\mathsf{fin}}(H_{obs} \times \mathsf{SRE}(M,C))$ such that for an indexed SRE $e = e_1 + \ldots + e_n$, $\mathsf{Post}((h,e), Gr, Op, p') = \bigcup_{i=1}^{n} \mathsf{Post}((h,e_i), Gr, Op, p')$ and for a product e, $(h',e') \in \mathsf{Post}_{\mathcal{L}}((h,e), Gr, Op, p')$ iff

- for all $c \in C_{obs}$, $(h'(c), e'(c)) \in \mathsf{Extract}(p', c, ((h(c) + \epsilon) \cdot e(c)) \otimes (Gr(c), Op(c)))$,
- for all $c \in (C \setminus C_{obs})$, $h'(c) = \epsilon$ and $e'(c) = e(c) \otimes (Gr(c), Op(c))$,

where the definitions of the operator $\otimes$ and the function $\mathsf{Extract} : \{0,1\} \times C_{obs} \times \mathsf{SRE}(M,C) \to \mathcal{P}_{\mathsf{fin}}((M \cup \{\epsilon\}) \times \mathsf{SRE}(M,C))$ are given below.

For $e \in \mathsf{SRE}(M)$ and guard g we define the SRE $\mathsf{Sat}(e,g)$ as follows:

- $\mathsf{Sat}(e, true) = e$,
- $\mathsf{Sat}(e, (\in m \cdot M^*)) = \emptyset$ if $e = \epsilon$, and otherwise if $e = e' \cdot e''$, then
$$\mathsf{Sat}(e, (\in m \cdot M^*)) = \begin{cases} e & \text{if } e' = (m' + \epsilon) \text{ and } m' = m, \\ e & \text{if } e' = (m_1 + \ldots + m_n)^* \text{ and } \\ & \quad m \in \{m_1, \ldots, m_n\}, \\ \mathsf{Sat}(e'', (\in m \cdot M^*)) & \text{otherwise,} \end{cases}$$
- $\mathsf{Sat}(e, (= \epsilon)) = \epsilon$.

For $e \in \mathsf{SRE}(M)$, $g \in Gr$ and $op \in Op$ we define the SRE $e \otimes (g, op)$ as follows:

- $e \otimes (g, !m) = \mathsf{Sat}(e,g) \cdot (m + \epsilon)$,
- $e \otimes (g, nop) = \mathsf{Sat}(e,g)$,
- $e \otimes (g, ?m) = \emptyset$ if $\mathsf{Sat}(e,g) = \epsilon$ or $\mathsf{Sat}(e,g) = \emptyset$, and if $\mathsf{Sat}(e,g) = e' \cdot e''$, then
$$e \otimes (g, ?m) = \begin{cases} e'' & \text{if } e' = (m' + \epsilon) \text{ and } m = m', \\ e & \text{if } e' = (m_1 + \ldots + m_n)^* \text{ and } m \in \{m_1, \ldots, m_n\}, \\ \emptyset & \text{otherwise.} \end{cases}$$

Finally, for $\mathsf{Extract} : \{0,1\} \times C_{obs} \times \mathsf{SRE}(M) \to \mathcal{P}_{\mathsf{fin}}((M \cup \{\epsilon\}) \times \mathsf{SRE}(M))$:

$$\mathsf{Extract}(p', c, e) = \begin{cases} \{(\epsilon, e)\} & \text{if } p' = 1, \\ \{(\epsilon, \epsilon)\} & \text{if } p' = 0 \text{ and } e = \epsilon, \\ \{(m_1, e_1)\} \cup \mathsf{Extract}(0, c, e_1) & \text{if } p' = 0 \text{ and } e = (m_1 + \epsilon) \cdot e_1, \\ \{(m_i, e') \mid i \in [1, n]\} \cup \mathsf{Extract}(0, c, e_1) & \text{if } p' = 0 \text{ and } e = (m_1 + \ldots + m_n)^* \cdot e_1. \end{cases}$$

We define a partial function $\mathsf{Post}^{obs} : \Phi_{\mathcal{D}_{obs}} \times \Sigma_\exists \to \Phi_{\mathcal{D}_{obs}}$ that corresponds to the function $\mathsf{Post}^{obs}_\exists : \mathcal{D}_{obs}(G) \times \Sigma_\exists \to \mathcal{D}_{obs}(G)$ as follows:

$$\mathsf{Post}^{obs}_\exists((t, p, q_0, h, \sigma_\exists, \sigma, R), \sigma'_\exists) = \begin{cases} (\forall, 0, q_0, h, \sigma'_\exists, \sigma, R) & \text{if } t = \exists, p = 0, \\ \text{undefined} & \text{otherwise} \end{cases}$$

3.3.4 Solving Safety Lossy Channel Games

In this subsection we describe a decision procedure for safety lossy channel games under incomplete information, which is based on a backward fixed point computation. The proposed algorithm is in the spirit of the *set-saturation methods* for verification of well-structured systems. The term set-saturation method was used in [FS01] to describe verification procedures that rely on the fact that each infinite increasing sequence of upward-closed sets of states eventually stabilizes.

Reducing obs_a-consistency to obs_s-consistency Each step in the fixed point computation presented below corresponds to a step in the game. Thus, this construction is correct w.r.t. $Player_\exists$ strategies that are obs_s-consistent. However, when $Player_\exists$ plays according to a obs_a-consistent strategy he only observes the steps at which the current state belongs to $Player_\exists$ (corresponding to states at which process 0 is scheduled). To circumvent this problem, our algorithm performs the fixed point computation on a game structure with incomplete information $\widetilde{G}$ obtained from $G(\mathcal{L}, \Sigma_\exists, C_{obs})$ by adding a *skip transition* for process 1. In $\widetilde{G}(\mathcal{L}, \Sigma_\exists, C_{obs}) = (S_\exists, S_\forall, I, =_{\mathsf{o}}, \Sigma_\exists, T_\exists, \widetilde{T_\forall})$, all components except for the transitions for $Player_\forall$ are as in $G(\mathcal{L}, \Sigma_\exists, C_{obs})$, and

$$\begin{array}{rcll} \widetilde{T_\forall} = T_\forall & \cup & \{((\forall, 1, q, w, \sigma_\exists, \sigma), (\exists, 0, q, w', \sigma_\exists, \sigma)) & \mid \quad w' \preceq w\} \\ & \cup & \{((\forall, 1, q, w, \sigma_\exists, \sigma), (\forall, 1, q, w', \sigma_\exists, \sigma)) & \mid \quad w' \preceq w\}. \end{array}$$

The game structure $\widetilde{G}$ can be defined by a modified partially specified LCS $\widetilde{\mathcal{L}} = (\mathcal{A}_0, \widetilde{\mathcal{A}_1}, C, M, \Sigma_0, \widetilde{\Sigma_1})$ as explained in Section 3.2. $\widetilde{\mathcal{L}}$ is such that $\widetilde{\mathcal{A}_1} = (Q_1, q_1^0, \widetilde{\delta_1})$, where $\widetilde{\delta_1} = \delta_1 \dot{\cup} \{(q, idle, true, nop, q) \mid q \in Q_1\}$, and $\widetilde{\Sigma_1} = \Sigma_1 \dot{\cup} \{idle\}$.

The game structure $\widetilde{G}(\mathcal{L}, \Sigma_\exists, C_{obs})$ has the following property.

Property 7. *In* $\mathsf{Safety}(G(\mathcal{L}, \Sigma_\exists, C_{obs}), Err)$ *Player*$_\exists$ *has an* obs_a*-consistent winning strategy iff he has an* obs_s*-consistent winning strategy in* $\mathsf{Safety}(\widetilde{G}(\mathcal{L}, \Sigma_\exists, C_{obs}), Err)$.

Proof. To see that for each obs_a-consistent winning strategy $f_\exists$ for *Player*$_\exists$ in the game $\mathsf{Safety}(G, Err)$, where $G = G(\mathcal{L}, \Sigma_\exists, C_{obs})$, we can define a obs_s-consistent winning strategy $\widetilde{f_\exists}$ for *Player*$_\exists$ in $\mathsf{Safety}(\widetilde{G}, Err)$, where $\widetilde{G} = \widetilde{G}(\mathcal{L}, \Sigma_\exists, C_{obs})$, consider the function $delete_{skip} : \mathsf{Prefs}_\exists(\widetilde{G}) \to \mathsf{Prefs}(G)$ that maps a *Player*$_\exists$-prefix in $\widetilde{G}$ to the prefix in G obtained from it by deleting all *skip* steps. Then, we define $\widetilde{f_\exists}(\widetilde{\pi}) = f_\exists(delete_{skip}(\widetilde{\pi}))$. Since for every $\widetilde{\pi}_1, \widetilde{\pi}_2 \in \mathsf{Prefs}_\exists(\widetilde{G})$, if $obs_s(\widetilde{\pi}_1) = obs_s(\widetilde{\pi}_2)$ then also $obs_a(delete_{skip}(\widetilde{\pi}_1)) = obs_a(delete_{skip}(\widetilde{\pi}_2))$, $\widetilde{f_\exists}$ is obs_s-consistent. Since for each $\widetilde{\pi}$ in $\widetilde{G}$, $\widetilde{\pi}$ and $delete_{skip}(\widetilde{\pi})$ visit exactly the same observations in *Obs*, $\widetilde{f_\exists}$ is winning.

The other direction follows from Proposition 3.3.3 below where we show that we can construct a finite-state obs_a-consistent winning strategy in the game $\mathsf{Safety}(G, Err)$ whenever *Player*$_\exists$ has a obs_s-consistent winning strategy in $\mathsf{Safety}(\widetilde{G}, Err)$. □

Note that $\mathcal{U}_{obs}(\widetilde{G}) = \mathcal{U}_{obs}(G)$, and in the following we use them interchangeably. The input to the game solving algorithm described in the next paragraph is the safety LC-game $\mathsf{Safety}(\widetilde{G}(\mathcal{L}, \Sigma_\exists, C_{obs}), Err)$, where the game structure $\widetilde{G}(\mathcal{L}, \Sigma_\exists, C_{obs})$ is represented by the partially specified LCS $\widetilde{\mathcal{L}}$ defined above.

Algorithm Let us denote with $\mathcal{L}(G)$ the set

$$\mathcal{L}(G) = \{L \in \mathcal{P}_{\mathsf{fin}}(\mathcal{U}_{obs}(G)) \mid L \neq \emptyset \text{ and } \exists o \in Obs. \forall U \in L.\ U \subseteq o\}.$$

Note that by definition for each $L \in \mathcal{L}(G)$ there exists a unique $o \in Obs$ such that $o \cap U \neq \emptyset$ for some $U \in L$. We denote this observation with $obs(L)$.

The algorithm for solving safety LC-games under incomplete information that we present computes a set $B \subseteq \mathcal{L}(G)$. Intuitively, each L in the computed set B has the following property: if $K \subseteq S$ is the set of states that the game can be currently in according to *Player*$_\exists$'s knowledge, and $K \cap U \neq \emptyset$ for every $U \in L$, then *Player*$_\exists$ cannot win the safety game when his initial (current) knowledge is K. Considering the

set I of initial states of the game, if for some $L \in B$ it holds that $I \cap U \neq \emptyset$ for all $U \in L$, then $Player_\exists$ has no obs_s-consistent winning strategy in $\mathsf{Safety}(\widetilde{G}, Err)$.

We define a quasi-ordering $\sqsubseteq$ on $\mathcal{L}(G)$ such that for $L, L' \in \mathcal{L}(G)$, we have

$$L \sqsubseteq L' \text{ iff for every } U \in L \text{ there exists a } U' \in L' \text{ such that } U \supseteq U'.$$

Since $(\mathcal{U}_{obs}(G), \supseteq)$ is a BQO, by the properties of BQOs, $\sqsubseteq$ is also a BQO.

From the definition of $\sqsubseteq$ it is clear that if $L \sqsubseteq L'$ and for some K it holds that $K \cap U' \neq \emptyset$ for each $U' \in L'$, then it also holds that $K \cap U \neq \emptyset$ for each $U \in L$. Thus, as we will see, it suffices to consider in the fixed point iteration the set B consisting only of minimal elements. That is, the set B is a finite representation of the possibly infinite set $B\uparrow$. Our algorithm computes a least fixed point by iteratively applying the function $\mathsf{Update}(A) = \mathsf{Min}(A \cup \mathsf{UPre}(A))$, where for a set A, $\mathsf{Min}(A)$ is the set of minimal elements of A w.r.t. $\sqsubseteq$ and the function UPre is defined as follows.

Algorithm: SolveLC-GamesSafety

Input: safety LC-game $\mathsf{Safety}(\widetilde{G}(\mathcal{L}, \Sigma_\exists, C_{obs}), Err)$ with
$G = (S_\exists, S_\forall, I, =_\mathsf{o}, \Sigma_\exists, T_\exists, T_\forall)$ and observable set Err

Output: $(winner, strategy)$ or $(winner, counterexample\ tree)$

$B := \emptyset$; $B' := \{\{Err \cap o\} \mid o \in Obs\}$; `/* initialize the set of L-sets */`

while $B' \not\sqsubseteq B$ **do**
  $B := B'$;
  if $\exists L \in B. \forall U \in L.\ I \cap U \neq \emptyset$ **then**
    $C_k := \text{ConstructTree}(\widetilde{G}, B)$;
    return $(\forall, C_k)$; `/* return counterexample tree */`
  $B' := \mathsf{Min}(B \cup \mathsf{UPre}(B))$; `/* update the set B */`

$f_\exists := \text{ConstructStrategy}(\widetilde{G}, B)$;
return $(\exists, f_\exists)$; `/* return strategy for` $Player_\exists$ `*/`

Algorithm 1: Solving safety LC-games under incomplete information.

Definition 3.3.1. We define the function $\mathsf{UPre} : \mathcal{P}_{\mathsf{fin}}(\mathcal{L}(G)) \to \mathcal{P}_{\mathsf{fin}}(\mathcal{L}(G))$ such that for a finite subset B of $\mathcal{L}(G)$, $\mathsf{UPre}(B)$ is the smallest (with respect to set inclusion) finite subset of $\mathcal{L}(G)$ that contains each set $L \subseteq \bigcup_{L' \in B} \bigcup_{U' \in L'} \left((\bigcup_{\sigma \in \Sigma_0} \mathsf{Pre}_\exists^{obs}(U, \sigma)) \cup \mathsf{Pre}_\forall^{obs}(U) \right)$ that satisfies one of the following two conditions:

- $L \in \mathcal{P}(\mathcal{P}(S_\exists))$ and for every action $\sigma_\exists \in Act_\exists(obs(L))$ there exists a set $L' \in B$ such that for every $U' \in L'$ it holds that $\mathsf{Pre}_\exists^{obs}(U', \sigma) \cap L \neq \emptyset$;
- $L \in \mathcal{P}(\mathcal{P}(S_\forall))$ and there is $L' \in B$ such that for each $U' \in L'$, $\mathsf{Pre}_\forall^{obs}(U') \cap L \neq \emptyset$.

The procedure SolveLC-GamesSafety for solving safety LC-games under incomplete information is given as Algorithm 1. It updates a set B of elements of $\mathcal{L}(G)$ by applying the function Update, and terminates when one of two possible conditions is satisfied. If a set L such that $I \cap U \neq \emptyset$ for each $U \in L$ is added to B, SolveLC-GamesSafety returns a finite tree constructed by the procedure ConstructTree. If the termination condition of the loop is fulfilled without adding such a set L to the set B, then the procedure returns a finite strategy for $Player_\exists$ constructed by the procedure ConstructStrategy.

The procedures ConstructTree and ConstructStrategy are described as part of the proofs of Proposition 3.3.2 and Proposition 3.3.3 respectively.

Each step of algorithm SolveLC-GamesSafety can be effectively computed. Using the representation from Section 3.3.2, the elements of $\mathcal{L}(G)$ are finite subsets of $\Phi_{\mathcal{U}_{obs}}$. Thus, equality between elements of $\mathcal{L}(G)$ is decidable, and $(\mathcal{L}(G), \sqsubseteq)$ is a decidable BQO. Since the sets B and B' are finite, the termination condition of the loop can be effectively checked, and the functions UPre and Min are computable. For the computation of UPre we rely of the functions $\mathsf{Pre}_\forall^{obs} : \Phi_{\mathcal{U}_{obs}} \to \mathcal{P}_{\mathsf{fin}}(\Phi_{\mathcal{U}_{obs}})$ and $\mathsf{Pre}_\exists^{obs} : \Phi_{\mathcal{U}_{obs}} \times \Sigma_\exists \to \mathcal{P}_{\mathsf{fin}}(\Phi_{\mathcal{U}_{obs}})$, which we described in Section 3.3.3.

Correctness The following three propositions establish the total correctness of Algorithm SolveLC-GamesSafety.

Proposition 3.3.1. *Algorithm* SolveLC-GamesSafety *terminates.*

Proof. Let $B_0 \subseteq B_1 \subseteq B_2 \ldots$ be the sequence of subsets of $\mathcal{L}(G)$ computed by Algorithm SolveLC-GamesSafety. Suppose that the algorithm does not terminate. This means that $B_0 \subsetneq B_1 \subsetneq B_2 \ldots$ and there exists a sequence $L_0, L_1, L_2, \ldots$ of distinct elements of $\mathcal{L}(G)$ such that for each $i \geq 0$, $L_{i+1} \in B_{i+1}$ and $L_{i+1} \notin B_i$. Since $\sqsubseteq$ is a BQO (and hence a WQO), there exist indices $0 \leq i < j$ such that $L_i \sqsubseteq L_j$. Since $L_i \in B_j$, this contradicts to the fact that B_j consists of $\sqsubseteq$-minimal elements only. □

Proposition 3.3.2. *If algorithm* SolveLC-GamesSafety *terminates and returns a tree C_k, then C_k is a knowledge-based counterexample tree in $\mathsf{Safety}(\widetilde{G}, Err)$.*

Proof. Let $L_0 \in B$ be an element of $\mathcal{L}(G)$ such that $I \cap L_0 = \emptyset$ for every $U \in L_0$. Procedure ConstructTree constructs a tree $C_k = (N, E, K_s, L_a)$ with nodes N, edges $E \subseteq N \times N$, function $K_s : N \to (2^S \setminus \emptyset)$ that labels each node with a set of states and a function $L_a : E \to \Sigma_\exists \cup \{\epsilon\}$ that labels edges with $Player_\exists$-actions. More precisely, the nodes of C_k are labeled with elements of $\mathcal{D}_{obs}(G)$.

Furthermore, the construction ensures, relying on the definition of the function UPre, that for each $n \in N$ there exists $L \in B$ such that $K_s(n) \cap U \neq \emptyset$ for each $U \in L$.

The tree C_k is constructed in a top-down manner as follows.

Let us denote for each $L \in B$ with $rank(L)$ the least $i \geq 0$ such that $L \in B_i$.

By the definition of I and $\mathcal{L}(G)$ it follows that there exists a single state $s_0 \in I$ such that $s_0 \in U$ for each $U \in L_0$. The root n_0 of C_k is labeled with $\{s_0\} \in \mathcal{D}_{obs}(G)$. At each step the procedure picks an unexplored leaf node n and processes n as follows:

- If $K_s(n) \cap Err \neq \emptyset$, n is closed (not expanded further) and remains a leaf in C_k.
- If $K_s(n) \cap Err = \emptyset$ and $K_s(n) \subseteq S_\exists$, then for each $\sigma_\exists \in Act_\exists(o)$, where $o \in Obs$ is the unique observation such that $K_s(n) \cap o \neq \emptyset$, we add to N a node n' and let $K_s(n') = D'$ where $D' = \mathsf{Post}_\exists^{obs}(K_s(n), \sigma_\exists)$ and $L_a((n, n')) = \sigma$.
- If $K_s(n) \cap Err = \emptyset$ and $K_s(n) \subseteq S_\forall$, then there exists a $D' \in \mathsf{Post}_\forall^{obs}(K_s(n))$ such that there exists an $L' \in B$ for which $D' \cap U' \neq \emptyset$ for each $U' \in L'$. We pick one such D' and L' with minimal rank and add to N a node n' with $K_s(n') = D'$.

For each edge $(n, n') \in E$ with $K_s(n) \subseteq S_\forall$ we let $L_a((n, n')) = \epsilon$.

The above construction is guaranteed to terminate, since the transition relation of $\widetilde{G}$ has finite branching and along each edge (n, n') of the tree the rank of the corresponding elements of B strictly decreases. For n with $K_s(n) \subseteq S_\exists$ this follows by the definition of rank and the construction of B, and for $K_s(n) \subseteq S_\forall$ by the choice of n'.

For each leaf n it holds that $K_s(n) \subseteq Err$, since $L = \{Err \cap o\}$ for some $o \in Obs$, for each L with $rank(L) = 0$. All other conditions of Definition 2.2.5 also follow easily from the construction of C_k, and thus C_k is a knowledge-based counterexample tree. □

Proposition 3.3.3. *If algorithm* SolveLC-GamesSafety *terminates returning a strategy $f_\exists$, then $f_\exists$ is an obs_a-consistent wining strategy for $Player_\exists$ in $\mathsf{Safety}(G, Err)$.*

Proof. Here we consider the case when the while-loop of SolveLC-GamesSafety terminates and for every $L \in B$ there exists a $U \in L$ such that $I \cap U = \emptyset$. In this case the procedure ConstructStrategy constructs a finite strategy represented as a strategy automaton $\mathcal{M} = (Q_\mathcal{M}, q^0_\mathcal{M}, Obs \times \Sigma^\perp_\exists, \rho)$, as described below.

A function $[\![]\!] : Q_{\mathcal{M}} \to \mathcal{D}_{obs}(G)$ maps each $q_{\mathcal{M}} \in Q_{\mathcal{M}}$ to a downward-closed set $[\![q_{\mathcal{M}}]\!] \subseteq S$ that is such that for each $L \in B$ there exists $U \in L$ such that $[\![q_{\mathcal{M}}]\!] \cap U = \emptyset$.

The set $Q_{\mathcal{M}}$ is defined as follows. First, for each $o \in Obs$ we define the set $\mathcal{V}_o \subseteq \mathcal{P}(S)$:

$$\mathcal{V}_o = \{U \in \mathcal{U}_{obs}(G) \mid \exists L \in B.\ U \in L \wedge obs(L) = o\}.$$

Now, if $q_{\mathcal{M}}^{sink}$ is a fresh state, we let

$$\begin{array}{rcl} Q_{\mathcal{M}} & = & \{q_{\mathcal{M}}^{sink}\} \\ & \dot\cup & \{q_{\mathcal{M}} \in \mathcal{P}_{\mathsf{fin}}(\mathcal{V}_o) \mid o \in Obs \wedge \forall L \in B. obs(L) = o \Rightarrow \exists U \in L.\ U \in q_{\mathcal{M}}\}. \end{array}$$

We define the function $[\![]\!] : Q_{\mathcal{M}} \to \mathcal{D}_{obs}(G)$ where for $o \in Obs$, $q_{\mathcal{M}} \in Q_{\mathcal{M}} \cap \mathcal{P}_{\mathsf{fin}}(\mathcal{V}_o)$, $[\![q_{\mathcal{M}}]\!] = o \setminus \left(\bigcup_{U \in q_{\mathcal{M}}} U\right)$. For $q_{\mathcal{M}}^{sink}$ we let $[\![q_{\mathcal{M}}^{sink}]\!] = \emptyset$.

Since each U in q is an upward-closed set and o is downward-closed, $[\![q_{\mathcal{M}}]\!] \in \mathcal{D}_{obs}(G)$.

We claim that the following properties hold for the set $Q_{\mathcal{M}}$, the function $[\![]\!]$ and the functions $\mathsf{Post}_\exists$ and $\mathsf{Post}_\forall$ in the game structure $\widetilde{G}$.

Property 8. *If $q_{\mathcal{M}} \in Q_{\mathcal{M}}$ and $q_{\mathcal{M}} \in \mathcal{P}_{\mathsf{fin}}(\mathcal{V}_o \cap \mathcal{P}(S_\exists))$ for some $o \in Obs$, then **there exists** $\sigma \in Act_\exists(o)$ such that for each $L \in B$ there is $U \in L$ with $\mathsf{Post}_\exists([\![q_{\mathcal{M}}]\!], \sigma) \cap U = \emptyset$.*

Property 9. *If $q_{\mathcal{M}} \in Q_{\mathcal{M}}$ and $q_{\mathcal{M}} \in \mathcal{P}_{\mathsf{fin}}(\mathcal{V}_o \cap \mathcal{P}(S_\forall))$ for some $o \in Obs$, then **for every** $o' \in Obs$ it holds that for each $L \in B$ there is $U \in L$ such that $(\mathsf{Post}_\forall([\![q_{\mathcal{M}}]\!]) \cap o') \cap U = \emptyset$.*

To verify Property 8 we assume that its statement does not hold. Let $q_{\mathcal{M}} \in Q_{\mathcal{M}}$ and $q_{\mathcal{M}} \in \mathcal{P}_{\mathsf{fin}}(\mathcal{V}_o \cap \mathcal{P}(S_\exists))$ for some $o \in Obs$. By our assumption, for each $\sigma \in Act_\exists(o)$, there exists $L'_\sigma \in B$ such that for all $U' \in L'_\sigma$ it holds that $\mathsf{Post}_\exists([\![q_{\mathcal{M}}]\!], \sigma) \cap U' \neq \emptyset$.

Let $L = \{\mathsf{Pre}_\exists(U', \sigma) \cap o \mid \sigma \in Act_\exists(o), U' \in L'_\sigma\}$. Then, for each $U \in L$ it holds that $[\![q_{\mathcal{M}}]\!] \cap U \neq \emptyset$. By the construction of B, there exists $L'' \in B$ such that $L'' \sqsubseteq L$. According to the definition of $\sqsubseteq$, for each $U'' \in L''$ it holds that $[\![q_{\mathcal{M}}]\!] \cap U'' \neq \emptyset$. Since $obs(L'') = o$, by the construction of $Q_{\mathcal{M}}$, there exists a $U'' \in L'' \cap q_{\mathcal{M}}$. Thus, $[\![q_{\mathcal{M}}]\!] \cap U'' = \emptyset$, which is a contradiction to the fact about L'' established above.

Property 9 follows using similar reasoning about the construction of B.

If for a set of states $V \subseteq S$ we have that $V \subseteq [\![q_1]\!]$ and $V \subseteq [\![q_2]\!]$, for some $q_1, q_2 \in Q_{\mathcal{M}}$, then $V \subseteq [\![q]\!]$, where $q = q_1 \cup q_2 \in Q_{\mathcal{M}}$. Thus, if for a set $V \subseteq S$ there exists a $q \in Q_{\mathcal{M}}$ such that $V \subseteq [\![q]\!]$, then there exists a $q' \in Q_{\mathcal{M}}$ such that $V \subseteq [\![q']\!]$ and which is the largest w.r.t. set inclusion element of $Q_{\mathcal{M}}$ with this property, that is, for every $q'' \in Q_{\mathcal{M}}$ with $V \subseteq [\![q'']\!]$ it holds that $q'' \subseteq q'$.

We will use the following relations to define the transition relation ρ of $\mathcal{M}$:

$$\rho_\exists \subseteq (Q_{\mathcal{M}} \cap \mathcal{P}_{\mathsf{fin}}(\mathcal{P}(S_\exists))) \times \Sigma_\exists \times (Q_{\mathcal{M}} \cap \mathcal{P}_{\mathsf{fin}}(\mathcal{P}(S_\forall))), \text{ and}$$

$$\rho_\forall \subseteq (Q_\mathcal{M} \cap \mathcal{P}_{\mathsf{fin}}(\mathcal{P}(S_\forall))) \times Obs \times (Q_\mathcal{M} \cap \mathcal{P}_{\mathsf{fin}}(\mathcal{P}(S_\exists))).$$

For $q \in Q_\mathcal{M} \cap \mathcal{P}_{\mathsf{fin}}(\mathcal{P}(S_\exists))$, $q' \in Q_\mathcal{M} \cap \mathcal{P}_{\mathsf{fin}}(\mathcal{P}(S_\forall))$ and $\sigma \in \Sigma_\exists$ we let

$$(q, \sigma, q') \in \rho_\exists \text{ iff } \mathsf{Post}_\exists([\![q]\!], \sigma) \subseteq [\![q']\!].$$

For $q \in Q_\mathcal{M} \cap \mathcal{P}_{\mathsf{fin}}(\mathcal{P}(S_\forall))$, $q' \in Q_\mathcal{M} \cap \mathcal{P}_{\mathsf{fin}}(\mathcal{P}(S_\exists))$ and $o' \in Obs$ we let

$$\begin{aligned}(q, o', q') \in \rho_\forall \quad \text{iff} \quad & \exists o_0 \in Obs \exists q_0 \in Q_\mathcal{M} \ldots \exists o_n \in Obs \exists q_n \in Q_\mathcal{M}. \\ & (\forall 0 \leq i < j < n.\ q_i \neq q_j) \wedge \\ & q_0 = q \wedge q_n = q' \wedge o_n = o' \wedge (\forall 0 \leq i \leq n. [\![q_i]\!] \subseteq o_i) \wedge \\ & (\forall 0 \leq i < n. \mathsf{Post}_\forall([\![q_i]\!]) \cap o_{i+1} \neq \emptyset \wedge (\mathsf{Post}_\forall([\![q_i]\!]) \cap o_{i+1}) \subseteq [\![q_{i+1}]\!]) \wedge \\ & (\forall 0 \leq i < n. \forall \widetilde{q} \in Q_\mathcal{M}.\ (\mathsf{Post}_\forall([\![q_i]\!]) \cap o_{i+1}) \subseteq [\![\widetilde{q}]\!] \Longrightarrow \widetilde{q} \subseteq q_{i+1}).\end{aligned}$$

For each set $V \subseteq S_\forall$ of states and observation $o \in Obs$ that are such that $V \subseteq o$ and $\mathsf{Post}_\forall(V) \cap o \neq \emptyset$, it holds that $V \subseteq \mathsf{Post}_\forall(V) \cap o$ and therefore, if $V \cap U \neq \emptyset$ for some then also $\mathsf{Post}_\forall(V) \cap U \neq \emptyset$. Thus, if for some $V \subseteq S_\forall$ and $o \in Obs$ we have that $V \subseteq o$, $\mathsf{Post}(V) \cap o \neq \emptyset$, and $q \in Q_\mathcal{M}$ is the largest w.r.t. set inclusion element of $Q_\mathcal{M}$ such that $V \subseteq [\![q]\!]$ and $q' \in Q_\mathcal{M}$ is the largest w.r.t. set inclusion element of $Q_\mathcal{M}$ such that $\mathsf{Post}_\forall(q) \cap o \subseteq [\![q']\!]$, then it holds that $q' \subseteq q$ (and hence $[\![q']\!] \supseteq [\![q]\!]$).

Taking into account the above observation, the definition of $\rho_\forall$ entails that if $(q, o', q'_1) \in \rho_\forall$ and $(q, o', q'_2) \in \rho_\forall$, then $q'_1 \subseteq q'_2$ or $q'_2 \subseteq q'_1$ and thus, $q_1 \cap q_2 \in Q_\mathcal{M}$.

The following properties of the relations $\rho_\exists$ and $\rho_\forall$ are a direct consequence of their definitions and Property 8 and Property 9, respectively:

- If $q \in Q_\mathcal{M}$ and $q \in \mathcal{P}_{\mathsf{fin}}(\mathcal{V}_o \cap \mathcal{P}(S_\exists))$ for some $o \in Obs$, then there exist an action $\sigma \in Act_\exists(o)$ and a state $q' \in Q_\mathcal{M}$ such that $(q, \sigma, q') \in \rho_\exists$.
- If $q \in Q_\mathcal{M}$ and $q \in \mathcal{P}_{\mathsf{fin}}(\mathcal{V}_o \cap \mathcal{P}(S_\forall))$ for some $o \in Obs$, then for every $o' \in Obs$ such that $o' \subseteq S_\exists$ and there exist states $s_0, \ldots, s_n \in S$ such that $s_0 \in [\![q]\!]$, $s_n \in o'$ and $(s_i, s_{i+1}) \in \widetilde{T_\forall}$ for each $i < n$, there exists $q' \in Q_\mathcal{M}$ such that $(q, o', q') \in \rho_\forall$. Moreover, $s_n \in [\![\bigcap_{(q,o',q') \in \rho_\forall} q']\!]$, where $\bigcap_{(q,o',q') \in \rho_\forall} q'$ is guaranteed to be an element of $Q_\mathcal{M}$ by the observation we made above.

Since for every $L \in B$ there exists $U \in L$ such that $I \cap U = \emptyset$, there must exist a $q^0_\mathcal{M} \in Q_\mathcal{M}$ such that $\{(\forall, 1, q^0_0, q^0_1, \epsilon, \flat, \flat)\} \subseteq [\![q^0_\mathcal{M}]\!]$. We fix one such $q^0_\mathcal{M}$ as initial state.

We define the transition relation $\rho_\mathcal{M}$ of $\mathcal{M}$ as follows. Let $q \in Q_\mathcal{M}$ and $o' \in Obs$.

- If $q \in Q_\mathcal{M} \cap \mathcal{P}_{\mathsf{fin}}(\mathcal{P}(S_\forall))$, $o' \subseteq S_\exists$ and there exists $q' \in Q_\mathcal{M}$ such that $(q, o', q') \in \rho_\forall$ we let $q' = \bigcup_{(q,o',\widetilde{q}) \in \rho_\forall} \widetilde{q}$, fix an action $\sigma \in Act_\exists(o')$ and a state $q'' \in Q_\mathcal{M}$ such that $(q', \sigma, q'') \in \rho_\exists$ (such q'' exists by the above property), and let $(q, (o', \sigma), q'') \in \rho_\mathcal{M}$.

- If $q \in Q_{\mathcal{M}} \cap \mathcal{P}_{\mathsf{fin}}(\mathcal{P}(S_\forall))$ and $o' \subseteq S_\forall$, we let $(q, (o', \bot), q^{sink}_{\mathcal{M}}) \in \rho_{\mathcal{M}}$.
- If $q \in Q_{\mathcal{M}} \cap \mathcal{P}_{\mathsf{fin}}(\mathcal{P}(S_\forall))$ and $o' \subseteq S_\exists$ and there is no $q' \in Q_{\mathcal{M}}$ with $(q, o', q') \in \rho_\forall$, we fix an action $\sigma \in Act_\exists(o')$ and let $(q, (o', \sigma), q^{sink}_{\mathcal{M}}) \in \rho_{\mathcal{M}}$.
- If $q \in Q_{\mathcal{M}} \cap \mathcal{P}_{\mathsf{fin}}(\mathcal{P}(S_\exists))$ and $o' \subseteq S_\forall$, we let $(q, (o', \bot), q^{sink}_{\mathcal{M}}) \in \rho_{\mathcal{M}}$.
- If $q \in Q_{\mathcal{M}} \cap \mathcal{P}_{\mathsf{fin}}(\mathcal{P}(S_\exists))$ and $o' \subseteq S_\exists$, we fix an action $\sigma \in Act_\exists(o')$ and a state $q' \in Q_{\mathcal{M}}$ such that $(q, \sigma, q') \in \rho_\exists$, and let $(q, (o', \sigma), q') \in \rho_{\mathcal{M}}$.
- If $q = q^{sink}_{\mathcal{M}}$ and $o' \subseteq S_\forall$, we let $(q, (o', \bot), q^{sink}_{\mathcal{M}}) \in \rho_{\mathcal{M}}$.
- If $q = q^{sink}_{\mathcal{M}}$ and $o' \subseteq S_\exists$, we fix $\sigma \in Act_\exists(o')$ and let $(q, (o', \sigma), q^{sink}_{\mathcal{M}}) \in \rho_{\mathcal{M}}$.

It is easy to verify that $\mathcal{M}$ fulfills the conditions of Definition 2.2.2. Furthermore, the definitions of $Act_\exists$ and $\rho_{\mathcal{M}}$ ensure that $\mathcal{M}$ is non-blocking and $\Sigma_\exists$-correct.

Thus, $\mathcal{M}$ defines an obs_a consistent strategy $f_\exists$ for $Player_\exists$ in G, as described in Section 2.2. To see that $f_\exists$ is winning for $Player_\exists$ assume the opposite and consider a play $\pi \in \mathsf{Outcome}(f_\exists)$ such that for some $0 \leq i < |\pi|$ it holds that $\pi[i] \in Err$. By induction on i it is easy to see that there exists a $q_{\mathcal{M}} \in Q_{\mathcal{M}}$ such that $\pi[i] \in [\![q_{\mathcal{M}}]\!]$. Thus, for some $o \in Obs$ we obtain $[\![q_{\mathcal{M}}]\!] \cap (Err \cap o) \neq \emptyset$. Since $L = \{Err \cap o\} \in B$ this contradicts the definition of $Q_{\mathcal{M}}$, and hence $f_\exists$ is winning for $Player_\exists$.

Notice that the strategy automaton $\mathcal{M}$ is finite and can be effectively constructed by the procedure ConstructStrategy. The set of states $Q_{\mathcal{M}}$, the alphabet $Obs \times \Sigma_\exists^\bot$ and the transition relation ρ are finite. The conditions in the definitions of $\rho_\exists$ and $\rho_\forall$ can be checked effectively, since $\mathsf{Pre}_\exists$ and $\mathsf{Pre}_\forall$ are computable for elements $\mathcal{U}_{obs}(G)$ and, if $q = \{U_1, \ldots, U_n\}$, $[\![q]\!] \subseteq o$, $q' = \{U'_1, \ldots, U'_m\}$, $[\![q']\!] \subseteq o'$, then:

$$
\begin{array}{lll}
\mathsf{Post}_\exists([\![q]\!], \sigma) \subseteq [\![q']\!] & \iff & \\
\mathsf{Post}_\exists([\![q]\!], \sigma) \subseteq o' \cap \overline{(\bigcup_{i=1}^m U'_i)} & \iff & \\
\mathsf{Post}_\exists([\![q]\!], \sigma) \subseteq o' \text{ and for all } i = 1, \ldots, m, \mathsf{Post}_\exists([\![q]\!], \sigma) \cap U'_i = \emptyset & \iff & \text{(def. } T_\exists\text{)} \\
\mathsf{Post}_\exists(o, \sigma) \cap o' \neq \emptyset \text{ and for all } i = 1, \ldots, m, \mathsf{Post}_\exists([\![q]\!], \sigma) \cap U'_i = \emptyset & \iff & \\
o \cap \mathsf{Pre}_\exists(o', \sigma) \neq \emptyset \text{ and for all } i = 1, \ldots, m, [\![q]\!] \cap \mathsf{Pre}_\exists(U'_i, \sigma) = \emptyset & \iff & \\
o \cap \mathsf{Pre}_\exists(o', \sigma) \neq \emptyset \text{ and for all } i = 1, \ldots, m, o \cap \mathsf{Pre}_\exists(U'_i, \sigma) \subseteq \bigcup_{j=1}^n U_j; &&
\end{array}
$$

$$
\mathsf{Post}_\forall([\![q]\!]) \cap o' \neq \emptyset \iff [\![q]\!] \cap \mathsf{Pre}_\forall(o') \neq \emptyset \iff o \cap \mathsf{Pre}_\forall(o') \not\subseteq \bigcup_{j=1}^n U_j;
$$

$$
\begin{array}{ll}
\mathsf{Post}_\forall([\![q]\!]) \cap o' \subseteq [\![q']\!] & \Leftrightarrow \\
\mathsf{Post}_\forall([\![q]\!]) \cap o' \subseteq o' \cap \overline{(\bigcup_{i=1}^{m} U'_i)} & \Leftrightarrow \\
\text{for all } i = 1, \ldots, m, \mathsf{Post}_\forall([\![q]\!]) \cap o' \cap U'_i = \emptyset & \Leftrightarrow \\
\text{for all } i = 1, \ldots, m, [\![q]\!] \cap \mathsf{Pre}_\forall(o' \cap U'_i) = \emptyset & \Leftrightarrow \\
\text{for all } i = 1, \ldots, m, o \cap \mathsf{Pre}_\forall(o' \cap U'_i) \subseteq \bigcup_{j=1}^{n} U_j. &
\end{array}
$$

Thus, the relation ρ can be computed, since $Q_{\mathcal{M}}$ is finite and the relations $\rho_\exists$ and $\rho_\forall$ are computed by checking conditions that are in the form given above. □

3.3.5 Solving Reachability Lossy Channel Games

The algorithm for lossy channel games under incomplete information with reachability winning conditions constructs a finite set of *AND-OR* trees whose nodes are labeled with elements of $\mathcal{D}_{obs}(G)$ (more precisely, with finite elements of $\mathcal{D}_{obs}(G)$). The procedure for reachability games falls into the class of what is called *tree-saturation methods* in [FS01]. It can be seen as an extension of the idea of the approach to downward-closed games with reachability winning conditions described in [ABd08].

Function $\mathsf{Post}^*_\forall$ The sets of states that label the nodes of the constructed trees represent the knowledge of $Player_\exists$ about the current state of the game after a given prefix. Since $Player_\exists$ can only observe the heads the observable channels and messages can be lost by $Player_\forall$, the knowledge of $Player_\exists$ at each point of the play is indeed a downward-closed set, element of $\mathcal{D}_{obs}(G)$. Furthermore, since the transition relations of G have finite branching, our construction ensures that the node labels are finite sets.

To update the knowledge of $Player_\exists$, we define the function $\mathsf{Post}^*_\forall : \mathcal{D}^{\mathsf{fin}}_{obs}(G) \to \mathcal{P}_{\mathsf{fin}}(\mathcal{D}^{\mathsf{fin}}_{obs}(G))$, where $\mathcal{D}^{\mathsf{fin}}_{obs}(G) \subseteq \mathcal{D}_{obs}(G)$ is the subset of $\mathcal{D}_{obs}(G)$ consisting of the finite sets. The function $\mathsf{Post}^*_\forall$ extends the restriction of $\mathsf{Post}^{obs}_\forall$ on $\mathcal{D}^{\mathsf{fin}}_{obs}(G)$. For $D \in \mathcal{D}^{\mathsf{fin}}_{obs}(G)$, the set $\mathsf{Post}^*_\forall(D) \in \mathcal{P}_{\mathsf{fin}}(\mathcal{D}^{\mathsf{fin}}_{obs}(D))$ is a finite set of elements of $\mathcal{D}^{\mathsf{fin}}_{obs}(S)$, each of which is a set of states that $Player_\exists$ knows, according to his current observation, the game may be in after a sequence of transitions of $Player_\forall$ corresponding to a sequence of transitions of process 1 possibly preceded by a transition of process 0. Formally, for $D \in \mathcal{D}^{\mathsf{fin}}_{obs}(G)$ we have $D' \in \mathsf{Post}^*_\forall(D)$ iff there exists a sequence $D_0, D_1, \ldots, D_n \in \mathcal{D}^{\mathsf{fin}}_{obs}(G)$ such that the following conditions hold:

- $D_0 = D$ and for every $1 \leq i \leq n$, $D_{i-1} \subseteq S_\forall$ and $D_i \in \mathsf{Post}^{obs}_\forall(D_{i-1})$,

- for every $0 \leq i < j < n$ it holds that $D_i \not\subseteq \mathit{Goal}$ and $D_i \not\subseteq D_j$,
- D' satisfies one of the following conditions:
  - (i) $D' = D_n$, $D' \subseteq \mathit{Goal}$, or
  - (ii) there exists a $0 \leq i < n$ such that $D_i \subseteq D_n$ and $D' = D_n$, or
  - (iii) $D' = \{(\exists, 0, q_0, q_1, w, \sigma_\exists, \sigma) \mid (\forall, 1, q_0, q_1, w, \sigma_\exists, \sigma) \in \bigcup_{i=0}^{n} D_i\}$.

Proposition 3.3.4. *For each $D \in \mathcal{D}^{\mathsf{fin}}_{obs}(G)$ the set $\mathsf{Post}^*_\forall(D)$ is finite.*

Proof. The elements of the set $\mathsf{Post}^*_\forall(D)$ correspond to paths in a tree with root labeled with D in which the children of a node labeled with D_i correspond to (are labeled with) the elements of $\mathsf{Post}^*(D_i)$. Since the transition relations of $G(\mathcal{L}, \Sigma_\exists, C_{obs})$ have finite branching and the set *Obs* is finite, the degree of each node in this tree is finite.

Each element of $\mathsf{Post}^*_\forall(D)$ corresponds to a path in this tree: in cases (i) and (ii) this is a label of a node in the tree and in case (iii) this set is computed based on the labels of a finite path in the tree. Thus, if we assume that the set $\mathsf{Post}^*(D)$ is infinite, it means that the tree must be infinite. By König's lemma we have that there exists an infinite path in the tree such that for each node on this path there exist infinitely many distinct elements of $\mathsf{Post}^*(D)$ corresponding to nodes/paths in this node's subtree.

Since $(\mathcal{D}^{\mathsf{fin}}_{obs}(G), \subseteq)$ is a BQO (and hence a WQO), there must be nodes n' and n'' on this path such that for the respective sets D' and D'' it holds that $D' \subseteq D''$. By the definition of $\mathsf{Post}^*_\forall(D)$ no new elements of $\mathsf{Post}^*_\forall(D)$ can appear in the subtree of n'' which is a contradiction to the choice of the infinite path. □

Proposition 3.3.5. *For each $D \in \mathcal{D}^{\mathsf{fin}}_{obs}(G)$, if there exists an infinite path π such that $\pi[0] \in D$ and $\pi[i] \in S_\forall$ and $\pi[i] \notin \mathit{Goal}$ for each $i \geq 0$, then there exists $D' \in \mathsf{Post}^*_\forall(D)$ such that condition (ii) from the definition of $\mathsf{Post}^*_\forall$ is satisfied.*

Proof. If such a path π exists, there exists an infinite sequence $D_0, D_1, D_2, \ldots$ of elements of $\mathcal{D}^{\mathsf{fin}}_{obs}(G)$ such that $D_0 = D$ and for every $1 \leq i$, $D_{i-1} \subseteq S_\forall$ and $D_i \in \mathsf{Post}^{obs}_\forall(D_{i-1})$. Since $(\mathcal{D}^{\mathsf{fin}}_{obs}(G), \subseteq)$ is a WQO, there exist indices $0 \leq i < j$ such that $D_i \subseteq D_j$. Now, taking the finite sequence $D_0, \ldots, D_j$, for the set $D' = D_j$ we have that $D' \in \mathsf{Post}^*_\forall(D)$ where the condition of Proposition (ii) is satisfied. □

Algorithm The procedure for solving reachability games, given as Algorithm 2, performs a forward exploration of the finite downward-closed sets of states representing the knowledge of $\mathit{Player}_\exists$. It constructs a set of AND-OR trees rooted at the different

possible knowledge sets for $Player_\exists$ at location q_0^0. The labeled graph consisting of the constructed trees is $\mathcal{F} = (N, E, L_D, L_a)$ with nodes N, edges $E \subseteq N \times N$, function $L_D : N \to \mathcal{D}_{obs}^{\mathsf{fin}}(G)$ that labels each node with an element of $\mathcal{D}_{obs}^{\mathsf{fin}}(G)$ and a function $L_a : E \to \Sigma_\exists \cup \{\epsilon\}$ that labels edges with $Player_\exists$-actions.

The procedure's input is a LC-game $\mathsf{Reach}(G(\mathcal{L}, \Sigma_\exists, C_{obs}), Goal)$ described symbolically by a partially specified LCS and a symbolically represented set of goal states.

The set of trees $\mathcal{F}$ is constructed starting from the root nodes labeled with the elements of the set $\{(\exists, 0, q_0^0, q_1^0, \epsilon, \flat, \flat)\} \cup \mathsf{Post}_\forall^*(\{(\forall, 1, q_0^0, q_1^0, \epsilon, \flat, \flat)\} \setminus Goal)$, representing the possible initial knowledge sets of $Player_\exists$. Children of a node in the tree are added for the elements of the respective set of successor sets computed by $\mathsf{Post}_\exists^{obs}$ and $\mathsf{Post}_\forall^*$. Nodes whose label is a subset of the goal states are declared successful for $Player_\exists$ and their successors are not explored. Nodes for which there exists an ancestor labeled with a subset of the node's label are declared unsuccessful for $Player_\exists$ and also not explored further. Similarly, nodes labeled with a set that includes as a subset a set on the path used by $\mathsf{Post}_\forall^*$ are also unsuccessful for $Player_\exists$ and not explored further.

Once the set $\mathcal{F}$ of trees is constructed, each node n is evaluated by assigning to it a boolean value $win(n)$. The evaluation proceeds bottom-up from the leaves, which have already been evaluated. The values of non-leaf nodes are computed based on those of their children, by interpreting the choices of $Player_\exists$ disjunctively and the choices of $Player_\forall$ conjunctively. If some root node is evaluated with *false*, then SolveLC-GamesReachability returns $\forall$ indicating that $Player_\exists$ does not have an obs_a-consistent winning strategy. Otherwise, SolveLC-GamesReachability returns a strategy for $Player_\exists$ constructed by the procedure ConstructStrategy.

Each step of algorithm SolveLC-GamesReachability can be effectively computed. Using the representation from Section 3.3.2, the labels of nodes in $\mathcal{F}$ are represented by elements of $\Phi_{\mathcal{D}_{obs}}$. Thus, checking inclusion between node labels is decidable. The computation of the node labels relies on the functions $\mathsf{Post}_\exists^{obs} : \Phi_{\mathcal{D}_{obs}} \times \Sigma_\exists \to \Phi_{\mathcal{D}_{obs}}$ and $\mathsf{Post}_\forall^* : \Phi_{\mathcal{D}_{obs}} \to \mathcal{P}_{\mathsf{fin}}(\Phi_{\mathcal{D}_{obs}})$. The latter function is based on the function $\mathsf{Post}_\forall^{obs} : \Phi_{\mathcal{D}_{obs}} \to \mathcal{P}_{\mathsf{fin}}(\Phi_{\mathcal{D}_{obs}})$ and by Proposition 3.3.4 can be effectively computed by exploring a finite tree, since $\mathsf{Post}_\forall^{obs}$ can be computed and finite union of elements of $\Phi_{\mathcal{D}_{obs}}$ with the same observation is an element of $\Phi_{\mathcal{D}_{obs}}$ and can be computed.

Algorithm: SolveLC-GamesReachability

Input: reachability LC-game $\mathsf{Reach}(G(\mathcal{L}, \Sigma_\exists, C_{obs}), Goal)$ with
$G = (S_\exists, S_\forall, I, =_o, \Sigma_\exists, T_\exists, T_\forall)$ and observable set $Goal$

Output: $(winner, strategy)$ or $winner$

$ToExplore := \{n_0\}$; **add** n_0 to N **with** $L_D(n_0) := \{(\exists, 0, q_0^0, q_1^0, \epsilon, \flat, \flat)\}$;
$\mathcal{V} := \mathsf{Post}^*_\forall(\{(\forall, 1, q_0^0, q_1^0, \epsilon, \flat, \flat)\} \setminus Goal)$;
foreach $o \in Obs$ *such that there is* $D' \in \mathcal{V}$ *with* $D' \subseteq o$ **do**
  $D' := \bigcup_{D \in \mathcal{V}, D \subseteq o} D$; **add new** n' to N **with** $L_D(n') = D'$;
  if $D' \subseteq S_\forall$ *and* $D' \not\subseteq Goal$ **then** $win(n') := false$; `/* close` n' `*/`
  else $ToExplore := ToExplore \cup \{n'\}$
while $ToExplore \neq \emptyset$ **do**
  pick and remove n **from** $ToExplore$;
  if $L_D(n) \subseteq Goal$ **then** $win(n) := true$; `/* close` n `*/`
  else if *exists an ancestor* n' *of* n *such that* $L_D(n') \subseteq L_D(n)$ **then**
    $win(n) := false$; `/* close` n `*/`
  else if $L_D(n) \subseteq S_\exists$ **then**
    foreach $\sigma_\exists \in Act_\exists(o)$, *where* $o \in Obs$ *with* $L_D(n) \subseteq o$ **do**
      add new n' to N **with** $L_D(n') = \mathsf{Post}^{obs}_\exists(L_D(n), \sigma_\exists)$;
      $L_a((n, n')) = \sigma_\exists$;
      $ToExplore := ToExplore \cup \{n'\}$
  else if $L_D(n) \subseteq S_\forall$ **then**
    $\mathcal{V} := \mathsf{Post}^*_\forall(L_D(n))$;
    foreach $o \in Obs$ *such that there is* $D' \in \mathcal{V}$ *with* $D' \subseteq o$ **do**
      $D' := \bigcup_{D \in \mathcal{V}, D \subseteq o} D$;
      add new n' to N **with** $L_D(n') = D'$, $L_a((n, n')) = \epsilon$;
      if $D' \subseteq S_\forall$ *and* $D' \not\subseteq Goal$ **then** $win(n') := false$; `/* close` n' `*/`
      else $ToExplore := ToExplore \cup \{n'\}$
$ToExplore := Leaves(\mathcal{F})$;
while $ToExplore \neq \emptyset$ **do**
  pick and remove n **from** $ToExplore$ **such that**
  $n' := Parent(n)$ *and each* $n'' \in Children(n')$ *is evaluated*;
  if $L_D(n') \subseteq S_\exists$ **then** $win(n') = \bigvee_{n'' \in Children(n')} win(n'')$;
  else $win(n') = \bigwedge_{n'' \in Children(n')} win(n'')$;
  if n' *is a root and* $win(n') = false$ **then return** $\forall$;
  else $ToExplore := ToExplore \cup \{n'\}$
$f_\exists :=$ ConstructStrategy$(G, \mathcal{F})$; **return** $(\exists, f_\exists)$;

Algorithm 2: Solving reachability LC-games under incomplete information.

Correctness The following three propositions establish the total correctness of Algorithm SolveLC-GamesReachability.

Proposition 3.3.6. *Algorithm* SolveLC-GamesReachability *terminates.*

Proof. Let us suppose that the construction of the graph $\mathcal{F}$ does not terminate. This means that the graph $\mathcal{F}$ is infinite. The transition relations of $G(\mathcal{L}, \Sigma_\exists, C_{obs})$ as well as the set *Obs* are finite, and as shown in Proposition 3.3.4, for each $D \in \mathcal{D}_{obs}(G)$ the set $\mathsf{Post}^*_\forall(D)$ is finite. Therefore, the graph $\mathcal{F}$ constructed by the algorithm is the union of a finite set of disjoint trees in which each node has finite degree. Thus, by König's Lemma there exists an infinite simple path $n_0, n_1, n_2, \ldots$ in $\mathcal{F}$. Consider the infinite sequence $L_D(n_0), L_D(n_1), L_D(n_2), \ldots$ of labels of the nodes on this path. Since $(\mathcal{D}^{\mathsf{fin}}_{obs}(G), \subseteq)$ is a BQO (and hence also a WQO), there must exist indices $0 \leq i < j$ such that $L_D(n_i) \subseteq L_D(n_j)$. Thus, by construction n_j must be a leaf in $\mathcal{F}$, which contradicts to the fact that $n_0, n_1, n_2, \ldots$ is an infinite path.

Thus, the graph $\mathcal{F}$ is finite and the first two loops of SolveLC-GamesReachability terminate. The loop that evaluates the nodes in $\mathcal{F}$ clearly also terminates, since each node in $\mathcal{F}$ is added to the set *ToExplore* at most once. □

Let us denote with $\widetilde{\mathcal{F}} = (N, \widetilde{E}, L_D, \widetilde{L_a})$ the graph obtained from $\mathcal{F}$ by replacing each edge (n, n') where n' is a leaf node for which there is an ancestor n'' such that $L_D(n'') \subseteq L_D(n')$ with an edge (n, n'') labeled by $\widetilde{L_a}((n, n'')) = L_a((n, n'))$ and adding a self-loop labeled with ϵ for each leaf n with $L_D(n) \subseteq S_\forall$ and $L_D(n) \not\subseteq \mathit{Goal}$. Note that in both cases the back-jump edges originate from nodes n with $L_D(n) \subseteq S_\forall$.

Paths in the trees $\mathcal{F}$ and in the graph $\widetilde{\mathcal{F}}$ correspond to sets of paths in the game structure $G(\mathcal{L}, \Sigma_\exists, C_{obs})$. The function $\mathit{States}_G : N \to S$ maps a node n to the set of states of G defined as follows $\mathit{States}_G(n) = L_D(n) \cup \{(\forall, 0, q_0, q_1, w, \sigma_\exists, \sigma) \mid (\exists, 1, q_0, q_1, w, \sigma_\exists, \sigma) \in L_D(n)\}$. Note that for a node n with $L_D(n) \subseteq S_\exists$ the set $\mathit{States}_G(n)$ contains also the counterparts from $S_\forall$ of the states in $L_D(n)$, taking into account the construction of the node labels in $\mathcal{F}$. The function Paths_G extends the function States_G to paths in $\mathcal{F}$ and $\widetilde{\mathcal{F}}$ as follows. Given a path ν, the set $\mathit{Paths}_G(\nu)$ of paths in G contains a path π iff there exists a function $f : \mathbb{N} \to \mathbb{N}$ such that:

- $f(i) \leq f(i+1)$ for each $i \in \mathbb{N}$, and the co-domain of f is $[0, |\nu|)$,
- $\pi[i] \in \mathit{States}(\nu[f(i)])$ for each $0 \leq i < |\pi|$, and
- if $\pi[i] \in S_\exists$ then $f(i+1) > f(i)$ for each $0 \leq i < |\pi| - 1$.

Since for each newly introduced edge (n, n'') replacing an edge (n, n') it holds that $L_D(n'') \subseteq L_D(n')$, we have by the definition of $\mathcal{F}$ that for every finite path $\nu = n_0, n_1, \ldots, n_k$ in $\widetilde{\mathcal{F}}$ and every $s \in L_D(n_k)$, there exists $\pi \in \mathit{Paths}_G(\nu)$ such that $\mathsf{last}(\pi) = s$. Thus, for each (possibly infinite) path ν in $\widetilde{\mathcal{F}}$ there exists $\pi \in \mathit{Paths}_G(\nu)$.

Proposition 3.3.7. *If algorithm* SolveLC-GamesReachability *terminates returning* $\forall$*, then* $\mathit{Player}_\exists$ *does not have an* obs_a*-consistent winning strategy in* $\mathsf{Reach}(G, \mathit{Goal})$.

Proof. If algorithm SolveLC-GamesReachability returns $\forall$, then there is a root node n_0 in $\mathcal{F}$ for which $\mathit{win}(n_0) = \mathit{false}$. We will show that for any obs_a-consistent strategy $f_\exists$ for $\mathit{Player}_\exists$ we can use the tree rooted an n_0 to show that there exists an infinite play $\pi \in \mathsf{Outcome}(f_\exists)$ that never visits a state in Goal.

Let us fix for each node n in $\widetilde{\mathcal{F}}$ with $L_D(n) \subseteq S_\forall$ and $\mathit{win}(n) = \mathit{false}$ a node $n' \in \mathit{Children}(n)$ by fixing a partial function $\rho_\forall : N \to N$ that maps each such node to a respective successor. By the definition of win and the fact that we added back-jump edges in $\widetilde{\mathcal{F}}$ it follows that for each such node the function is defined.

By the construction of the trees $\mathcal{F}$ and the definition of the function Paths_G, we have that for each two prefixes $\pi_1 \in \mathit{Paths}_G(\nu)$ and $\pi_2 \in \mathit{Paths}_G(\nu)$, where $\nu = n_0, n_1, \ldots, n_k$ is a path in $\widetilde{\mathcal{F}}$, it holds that $\mathit{obs}_a(\pi_1) = \mathit{obs}_a(\pi_2)$. Therefore, since the strategy $f_\exists$ is obs_a-consistent, we can define a function $\rho_\exists : N^+ \to N$, where for a finite path $\nu = n_0 \ldots n_k$ in $\widetilde{\mathcal{F}}$, if there is a $\pi \in \mathit{Paths}_G(\nu)$ we define $\rho_\exists(\pi) = n'$, where n' is the single child of n with $L_a((n, n')) = f_\exists(\pi)$ (such n' exists according to the definition of strategies and the construction of $\mathcal{F}$), and otherwise we leave $\rho_\exists(\pi)$ undefined.

Since, as explained above, for each path ν in $\widetilde{\mathcal{F}}$ there exists $\pi \in \mathit{Paths}_G(\nu)$, the function $\rho_\exists$ and $\rho_\forall$ starting from a root n_0 with $\mathit{win}(n_0) = \mathit{false}$ construct an infinite path $n_0, n_1, \ldots$ such that $L_D(n_i) \cap \mathit{Goal} = \emptyset$ for each $i \geq 0$. There exists $\pi \in \mathit{Paths}_G(n_0, n_1, \ldots)$, and furthermore $\pi \in \mathsf{Outcome}(f_\exists)$ by the definition of $\rho_\exists$. □

Proposition 3.3.8. *If* SolveLC-GamesReachability *terminates returning a strategy* $f_\exists$*, then* $f_\exists$ *is an* obs_a*-consistent wining strategy for* $\mathit{Player}_\exists$ *in* $\mathsf{Reach}(G, \mathit{Goal})$.

Proof. Here we consider the case when algorithm SolveLC-GamesReachability terminates and the root of each tree in $\mathcal{F}$ is evaluated to *true*. In this case the procedure ConstructStrategy constructs a finite strategy represented as a strategy automaton $\mathcal{M} = (Q_\mathcal{M}, q^0_\mathcal{M}, \mathit{Obs} \times \Sigma^\perp_\exists, \rho)$, as described below.

The set of states of the strategy automaton is defined as

$$Q_\mathcal{M} = \{q^0_\mathcal{M}, q^{\mathit{sink}}_\mathcal{M}\} \dot\cup \{D \mid \exists n \in N.\ D = L_D(n) \wedge \mathit{win}(n) = \mathit{true}\}.$$

By the definition of the function *win*, we have that for each $n \in N$ such that $L_D(n) \subseteq S_\exists$, $win(n) = true$ and $L_D(n) \not\subseteq Goal$, there exists $\sigma \in \Sigma_\exists$ such that for the node n' with $L_a((n,n')) = \sigma$ it holds that $win(n') = true$. Furthermore, for every $n'' \in N$ with $L_D(n'') = L_D(n)$ it holds that $win(n'') = true$ and there exists $n''' \in Children(n'')$ such that $L_a((n'',n''')) = \sigma$ and $win(n''') = win$. Thus, we can construct a function $\rho_\exists : \{q \in Q_\mathcal{M} \mid q \subseteq S_\exists \wedge q \not\subseteq Goal\} \to \Sigma_\exists$ such that

$$\rho_\exists(q) = \sigma \Rightarrow (\forall n \in N.\, L_D(n) = q \Rightarrow \exists n' \in Children(n)\; L_a((n,n')) = \sigma \wedge win(n') = true).$$

Note that for $q \in \{q \in Q_\mathcal{M} \mid q \subseteq S_\exists \wedge q \not\subseteq Goal\}$ it holds that $\mathsf{Post}^{obs}_\exists(q, \rho_\exists(q)) \in Q_\mathcal{M}$.

We define the function $\rho_\forall : \{q \in Q_\mathcal{M} \mid q \subseteq S_\forall\} \times Obs \to Q_\mathcal{M}$:

$$\rho_\forall(q,o) = \begin{cases} \bigcup_{D \in \mathsf{Post}_\forall^*(q), D \subseteq o} D & \text{if } \exists D \in \mathsf{Post}^*_\forall(q).\; D \subseteq o \\ q^{sink}_\mathcal{M} & \text{otherwise.} \end{cases}$$

Let $fix : Obs \cap \mathcal{P}(S_\exists) \to \Sigma_\exists$ be a function that fixes for each observation $o \subseteq S_\exists$ some action $\sigma \in Act_\exists(o)$. Now, using the functions $\rho_\exists$ and $\rho_\forall$ we define the transition relation $\rho_\mathcal{M}$ of $\mathcal{M}$, which is the least relation such that for $q \in Q_\mathcal{M}$ and $o \in Obs$:

- If $q = q^0_\mathcal{M}$, $o \subseteq S_\exists$ and there exists a root node n with $L_D(n) \subseteq o$, then $(q,(o,\sigma),q') \in \rho_\mathcal{M}$, where $\sigma = \rho_\exists(L_D(n)))$ and $q' = \mathsf{Post}^{obs}_\exists(L_D(n),\sigma)$.
- If $q = q^0_\mathcal{M}$, $o \subseteq S_\exists$ and no root n has $L_D(n) \subseteq o$, then $(q,(o,fix(o)),q^{sink}_\mathcal{M}) \in \rho_\mathcal{M}$.
- If $q \in Q_\mathcal{M} \cap \mathcal{P}_{\mathsf{fin}}(\mathcal{P}(S_\exists))$, $q \not\subseteq Goal$ and $o \subseteq S_\exists$, then $(q,(o,\sigma),q') \in \rho_\mathcal{M}$, where $\sigma = \rho_\exists(q)$ and $q' = \mathsf{Post}^{obs}_\exists(q,\sigma)$.
- If $q \in Q_\mathcal{M} \cap \mathcal{P}_{\mathsf{fin}}(\mathcal{P}(S_\forall))$, $q \not\subseteq Goal$, $o \subseteq S_\exists$ and $q' = \rho_\forall(q,o) \neq q^{sink}_\mathcal{M}$, then $(q,(o,\sigma),q'') \in \rho_\mathcal{M}$, where $\sigma = \rho_\exists(q')$ and $q'' = \mathsf{Post}^{obs}_\exists(q',\sigma)$.
- If $q \in Q_\mathcal{M} \cap \mathcal{P}_{\mathsf{fin}}(\mathcal{P}(S_\forall))$, $q \not\subseteq Goal$, $o \subseteq S_\exists$ and $q' = \rho_\forall(q,o) = q^{sink}_\mathcal{M}$, then $(q,(o,fix(o)),q^{sink}_\mathcal{M}) \in \rho_\mathcal{M}$.
- If $q \subseteq Goal$ or $q = q^{sink}_\mathcal{M}$ and $o \subseteq S_\exists$, then $(q,(o,fix(o)),q) \in \rho_\mathcal{M}$.
- If $o \subseteq S_\forall$, then $(q,(o,\bot),q^{sink}_\mathcal{M}) \in \rho_\mathcal{M}$.

It is easy to verify that $\mathcal{M}$ fulfills the conditions of Definition 2.2.2. Furthermore, the definitions of $Act_\exists$ and $\rho_\mathcal{M}$ ensure that $\mathcal{M}$ is non-blocking and $\Sigma_\exists$-correct.

Thus, $\mathcal{M}$ defines an *obs*-a consistent strategy $f_\exists$ for $Player_\exists$ in G, as described in Section 2.2. To see that $f_\exists$ is winning for $Player_\exists$ assume the opposite and consider an infinite play $\pi \in \mathsf{Outcome}(f_\exists)$ such that for all $i \geq 0$ it holds that $\pi[i] \notin Goal$.

By the definition of $f_\exists$ and $\mathcal{M}$, each play in $\mathsf{Outcome}(f_\exists)$ corresponds to a run of $\mathcal{M}$ that does not reach $q_{\mathcal{M}}^{sink}$. By the construction of $\mathcal{M}$ and $\mathcal{F}$ such a run corresponds to a path in the tree, that is such that if it reaches leaf node n then n must be such that $L_D(n) \subseteq \mathit{Goal}$. Since we assumed that the play π does not reach *Goal*, the corresponding path must be of the form $n_0, n_1, \ldots, n_k$, where $L_D(n_k) \subseteq S_\exists$ and $\mathit{win}(n_i) = \mathit{true}$ for each $0 \leq i \leq k$. Thus, there exists an index $i_0 \geq 0$ such that for each $i \geq i_0$ it holds that $\pi[i] \in \mathit{States}(n_k)$. Thus, by Proposition 3.3.5 and the construction of $\mathcal{F}$, either $k = 0$ and there exists a root node n' with $\mathit{win}(n') = \mathit{false}$ or there exists $n' \in \mathit{Children}(n_{k-1})$ such that $\mathit{win}(n') = \mathit{false}$. The first case contradicts the precondition of the claim we are proving and the second contradicts the fact that $\mathit{win}(n_{k-1}) = \mathit{true}$ and $L_D(n_{k-1}) \subseteq S_\forall$. Thus, the proof by contradiction is completed.

Notice that the strategy automaton $\mathcal{M}$ is finite and can be effectively constructed by the procedure ConstructStrategy. The set of states $Q_{\mathcal{M}}$, the alphabet $\mathit{Obs} \times \Sigma_\exists^\perp$ and the transition relation ρ are finite. The transition relation ρ can be computed, since each $q_{\mathcal{M}}$ is a downward-closed set, the set $Q_{\mathcal{M}}$ is finite and the relations $\rho_\exists$ and $\rho_\forall$ are defined in terms of the functions $\mathsf{Post}_\exists^{obs}$ and $\mathsf{Post}_\forall^*$. □

We denote with $\mathcal{C}_{LCS,Safety}$ and $\mathcal{C}_{LCS,Reach}$ the classes of games under incomplete information with observable safety and reachability winning conditions respectively, whose game structures are defined by partially specified LCSs. The following theorem summarizes the results from sections 3.3.4 and 3.3.5.

Theorem 3.3.1. *The game solving and strategy synthesis problems are decidable for*

- *the class of games $\mathcal{C}_{LCS,Safety}$ and the class of obs_a-consistent $Player_\exists$ strategies.*
- *the class of games $\mathcal{C}_{LCS,Reach}$ and the class of obs_a-consistent $Player_\exists$ strategies.*

3.4 Undecidability of Parity LC-Games under Incomplete Information

We now turn to more general ω-regular observable winning conditions for $Player_\exists$. For perfect information lossy channel games in which only one player can lose messages, undecidability results for *weak parity* winning conditions were established in [ABd08]. In this section we show that these results carry on to our setting of lossy channel games under incomplete information. Notice that the perfect information games of [ABd08] are not a special case of the incomplete information games for LCSs which we study in

this thesis. This is because in a lossy channel game under incomplete information, the information which $Player_\exists$ receives about the current state is from a finite set, while in the perfect information case there are infinitely many possible observations.

An *observable priority function* $pr : Obs \to \{0, 1, \ldots, n\}$ for $n \in \mathbb{N}$ in a game structure $G = (S_\exists, S_\forall, I, =_o, \Sigma_\exists, T_\exists, T_\forall)$ with incomplete information maps each observation to a non-negative integer priority. Let Obs be finite. For an infinite play $\pi = s_0, s_1, s_2 \ldots$ we define $pr(\pi) = \min\{pr(o) \mid o \in InfObs(\pi)\}$, where $InfObs(\pi) = \{o \in Obs \mid \forall i.\exists j.\ i < j \wedge s_j \in o\}$ is the set of observations from that occur infinitely often in π, and define $wpr(\pi) = \min\{pr(o) \mid o \in Obs \wedge \exists i.\ s_i \in o\}$. An observable *parity* winning condition for $Player_\exists$ in the game structure G is defined by an observable priority function pr as the set $Parity(G, pr) = \{\pi \in S^\omega \mid pr(\pi) \text{ is even}\}$ of infinite plays for which the minimal priority occurring infinitely often is even. An observable *weak parity* winning condition $Player_\exists$ in G is also defined by an observable priority function pr and is the set of infinite plays $WeakParity(G, pr) = \{\pi \in S^\omega \mid wpr(\pi) \text{ is even}\}$ for which the minimal priority occurring in the play is even.

We denote with $\mathcal{C}_{LCS,Parity}$ and $\mathcal{C}_{LCS,WeakParity}$ the classes of games under incomplete information with observable parity and weak parity winning conditions respectively, whose game structures are defined by partially specified LCSs.

Theorem 3.4.1. *The game solving problem is undecidable for the class of games $\mathcal{C}_{LCS,WeakParity}$ and the class of obs_a-consistent $Player_\exists$ strategies.*

Proof. In [ABd08] it was shown that in the perfect information setting the weak parity problem for B-LCS games, which are games played on a finite set of channels in which player A has a weak parity objective and only player B is allowed to lose messages, is undecidable. Their proof (given for A-LCS games but easily transferable into a proof for B-LCS games) is based on a reduction from the infinite computation problem for transition systems based on lossy channel systems, which is undecidable [AJ96].

This reduction can be adapted for our framework, with $Player_\exists$ in the role of player A and $Player_\forall$ in the role of player B. The fact that here $Player_\exists$ choses only transition labels and plays under incomplete information does not affect the proof for B-LCS games, since there player A just follows passively, while player B simulates the original system. The values of the priority function used in [ABd08] do not depend on the contents of the channels. Thus, we can define an observable priority function.

We now present the adapted version of the proof given in [ABd08].

The *infinite computation problem* asks if for a LCS $\mathcal{L} = (\{\mathcal{A}_i\}_{i=0}^n, C, M, \{\Sigma_i\}_{i=0}^n)$ and a location $q_{init} = (q_0^{init}, \ldots, q_n^{init})$ there exists a channel contents w such that there exists an infinite computation of $\mathcal{L}$ starting from (q_{init}, w), that is, whether there is an infinite sequence of configurations $\pi = (q^0, w^0), (q^1, w^1), (q^1, w^1), \ldots$ such that $q^0 = q_{init}$, $w^0 = w$ and for each $i \geq 0$ there is $\sigma_i \in \Sigma$ such that $(q^i, w^i) \xrightarrow{\sigma_i} (q^{i+1}, w^{i+1})$.

We show that the infinite computation problem for LCS can be reduced to the game solving problem for the class of games $\mathcal{C}_{LCS,WeakParity}$ and the class of obs_a-consistent $Player_\exists$ strategies. To this end, for a given LCS $\mathcal{L} = (\{\mathcal{A}_i\}_{i=0}^n, C, M, \{\Sigma_i\}_{i=0}^n)$ we define a partially specified LCS $(\widetilde{\mathcal{L}}, \Sigma_{\mathsf{c}}, C_{obs})$, where $\widetilde{\mathcal{L}} = (\{\widetilde{\mathcal{A}_i}\}_{i=0}^1, C, M, \{\widetilde{\Sigma_i}\}_{i=0}^1)$ and:

- $\widetilde{\Sigma_0} = \{\sigma_{c,m} \mid c \in C, m \in M\} \dot\cup \{\widetilde{\sigma_{init}}, \widetilde{\sigma_d}\} \dot\cup \Sigma$,
- $\widetilde{\Sigma_1} = \{\widetilde{\sigma_1}\}$,
- $\widetilde{\mathcal{A}_0} = (\widetilde{Q_0}, \widetilde{q_0^0}, \widetilde{\delta_0})$, $\widetilde{Q_0} = \{\widetilde{q_0}, \widetilde{q_d}\} \dot\cup (Q_0 \times \ldots \times Q_n)$ and $\widetilde{\delta_0}$ is the least set with:
  - $(\widetilde{q_0^0}, \widetilde{\sigma_{c,m}}, \mathit{true}, c!m, \widetilde{q_0^0}) \in \widetilde{\delta_0}$, for each $c \in C$ and $m \in M$,
  - $(\widetilde{q_0^0}, \widetilde{\sigma_{init}}, \mathit{true}, \mathit{nop}, q_{init}) \in \widetilde{\delta_0}$,
  - $(q, \sigma, \mathit{Gr}, \mathit{Op}, q') \in \widetilde{\delta_0}$ for each $p \in [0, n]$ and $(q_p, \sigma, \mathit{Gr}, \mathit{Op}, q'_p) \in \delta_p$, such that $q(p) = q_p$, $q'(p) = q'_p$ and $q(p') = q'(p')$ for all $p' \in [0, 1]$ with $p' \neq p$,
  - $(q, \widetilde{\sigma_d}, \mathit{true}, \mathit{nop}, q_d)$ for each $q \in Q_0 \times \ldots \times Q_n$,
- $\widetilde{\mathcal{A}_1} = (\widetilde{Q_1}, \widetilde{q_1^0}, \widetilde{\delta_1})$, where $\widetilde{Q_1} = \{\widetilde{q_1^0}\}$ and $\widetilde{\delta_1} = \{(\widetilde{q_1^0}, \widetilde{\sigma_1}, \mathit{true}, \mathit{nop}, \widetilde{q_1^0})\}$,
- $\Sigma_{\mathsf{c}} = \emptyset$, $C_{obs} = C$.

Let $G(\widetilde{\mathcal{L}}, \Sigma_{\mathsf{c}}, C_{obs}) = (S_\exists, S_\forall, I, =_{\mathsf{o}}, \Sigma_\exists, T_\exists, T_\forall)$. Now, let the observable priority function $pr : \mathit{Obs} \to \{0, 1, 2\}$ be defined as follows:

- $pr(o) = 2$ if $(t, 0, (\widetilde{q_0^0}, \widetilde{q_1^0}), w, \sigma_\exists, \sigma) \in o$ for some t, w, $\sigma_\exists$ and σ,
- $pr(o) = 1$ if $(t, 0, (\widetilde{q_0}, \widetilde{q_1^0}), w, \sigma_\exists, \sigma) \in o$ for some $\widetilde{q_0} \in Q$, and t, w, $\sigma_\exists$ and σ,
- $pr(o) = 0$ if $(t, 0, (\widetilde{q_d}, \widetilde{q_1^0}), w, \sigma_\exists, \sigma) \in o$ for some t, w, $\sigma_\exists$ and σ,
- $pr(o) = 0$ if $(t, 1, q, w, \sigma_\exists, \sigma) \in o$ for some t, q, w, $\sigma_\exists$ and σ.

Note that since Obs is finite, the function pr has finite representation.

We claim that in $(G, \mathit{WeakParity}(G, pr))$, $Player_\exists$ has an obs_a-consistent winning strategy iff the answer to the infinite computation problem for $\mathcal{L}$ and q_{init} is negative.

Suppose that $f_\exists$ is an obs_a-consistent winning strategy for $Player_\exists$ in the game $(G, \mathit{WeakParity}(G, pr))$ and assume that there exists a channel contents w and an

infinite computation $\nu = (q^0, w^0), (q^1, w^1), (q^2, w^2), \ldots$ with $q^0 = q_{\varphi_{Init}}$ and $w^0 = w$. By the construction of $(\widetilde{\mathcal{L}}, \Sigma_{\mathsf{c}}, C_{obs})$, there exists an infinite play $\pi \in \mathsf{Outcome}(f_\exists)$, which corresponds to w and ν, such that there exists an index $i > 0$ such that for all $0 \leq j < i$ we have $pr(o_j) = 2$ and for all $j \geq i$ we have $pr(o_j) = 1$, where $o_0 o_1 o_2 \ldots$ is the sequence of observations corresponding to π. Thus, $\pi \notin \mathit{WeakParity}(G, pr)$, which contradicts the fact that $f_\exists$ is a winning strategy for $\mathit{Player}_\exists$ in the game $(G, \mathit{WeakParity}(G, pr))$.

Now, assume that there does not exist $w \in W$ for which there is an infinite computation of $\mathcal{L}$ starting from (q_{init}, w). Let $f_\exists$ be the strategy for $\mathit{Player}_\exists$ in G that is such that $f_\exists(\pi) = \flat$ for each $\pi \in \mathsf{Prefs}_\exists(G)$. Clearly $f_\exists$ is obs_a-consistent and each play $\pi \in \mathsf{Outcome}(f_\exists)$ is infinite. We will show that $f_\exists$ is winning for $\mathit{Player}_\exists$ in $(G, \mathit{WeakParity}(G, pr))$. If we assume the contrary, then by the construction of the game and the definition of the priority function, there exists an infinite play $\pi \in \mathsf{Outcome}(f_\exists)$ such that there exists an index $i > 0$ such that for all $0 \leq j < i$ we have $pr(o_j) = 2$ and for all $i \leq j$ we have $pr(o_j) = 1$. Let w be the contents of the channels in state $\pi[i]$. Then, by the definition of G and pr we have that there exists an infinite computation of $\mathcal{L}$ starting from (q_{init}, w), which contradicts our assumption. □

Corollary 3.4.1. *The game solving problem is undecidable for the class of games: $\mathcal{C}_{LCS,Parity}$ and the class of obs_a-consistent $\mathit{Player}_\exists$ strategies.*

As noted in [ABd08], the above proof can be straightforwardly adapted to show the undecidability also for Büchi and co-Büchi winning conditions. Given a set $F \subseteq S$ of states in a game structure $G = (S_\exists, S_\forall, I, =_{\mathsf{o}}, \Sigma_\exists, T_\exists, T_\forall)$ with incomplete information, the Büchi winning condition for F is $\mathit{Buchi}(G, F) = \{\pi \in S^\omega \mid \forall i \geq 0. \exists j > i.\ \pi[j] \in F\}$. The co-Büchi winning condition for F is $\mathit{coBuchi}(G, F) = \{\pi \in S^\omega \mid \exists i \geq 0. \forall j \geq i.\ \pi[j] \notin F\}$. To obtain a reduction to a Büchi winning condition, encoded as an observable parity condition, we use the construction from the proof and define the set of states $F' = \{s \in S \mid \exists o. s \in o \wedge pr'(o) = 0\}$, where:

- $pr'(o) = 0$ if $(t, 0, (\widetilde{q_0^0}, \widetilde{q_1^0}), w, \sigma_\exists, \sigma) \in o$ for some t, w, $\sigma_\exists$ and σ,
- $pr'(o) = 1$ if $(t, 0, (\widetilde{q_0}, \widetilde{q_1^0}), w, \sigma_\exists, \sigma) \in o$ for some $\widetilde{q_0} \in Q$, and t, w, $\sigma_\exists$ and σ,
- $pr'(o) = 0$ if $(t, 0, (\widetilde{q_d}, \widetilde{q_1^0}), w, \sigma_\exists, \sigma) \in o$ for some t, w, $\sigma_\exists$ and σ,
- $pr'(o) = 0$ if $(t, 1, q, w, \sigma_\exists, \sigma) \in o$ for some t, q, w, $\sigma_\exists$ and σ.

Similarly, for a co-Büchi winning condition, $F'' = \{s \in S \mid \exists o. s \in o \wedge pr''(o) = 1\}$, where:

- $pr''(o) = 2$ if $(t, 0, (\widetilde{q_0^0}, \widetilde{q_1^0}), w, \sigma_\exists, \sigma) \in o$ for some t, w, $\sigma_\exists$ and σ,
- $pr''(o) = 1$ if $(t, 0, (\widetilde{q_0}, \widetilde{q_1^0}), w, \sigma_\exists, \sigma) \in o$ for some $\widetilde{q_0} \in Q$, and t, w, $\sigma_\exists$ and σ,
- $pr''(o) = 2$ if $(t, 0, (\widetilde{q_d}, \widetilde{q_1^0}), w, \sigma_\exists, \sigma) \in o$ for some t, w, $\sigma_\exists$ and σ,
- $pr''(o) = 2$ if $(t, 1, q, w, \sigma_\exists, \sigma) \in o$ for some t, q, w, $\sigma_\exists$ and σ.

Chapter 4

Games with Fixed Observations

In the previous chapter we presented algorithms for solving incomplete-information LC-games with safety and reachability winning conditions. Now we extract and formulate general sets of conditions on the game structures under which the algorithmic schemes resulting from generalizing these algorithms are applicable. In addition to the existence of a BQO on the set of states of the game that relates in a suitable way to the observation equivalence relation, there are, broadly speaking, two other types of conditions. One part of the conditions establish the existence of a symbolic representation that enables the effective computation of each step of the respective backward or forward algorithm. The second part of the conditions require in addition that the transition relations of the two players satisfy some monotonicity properties that ensure the algorithms' termination. We conclude this chapter with an example of a game structure that meets the requirements of the classes of games defined here, but does not fall into any of the previously known decidable classes of infinite-state incomplete-information games.

4.1 Monotonic and Downward-Closed BQO Games

In LC-games, the transition relation of $Player_\forall$ satisfies a strong monotonicity property which allowed us to define effective symbolic representations closed under predecessor of successor operations respectively, and which guaranteed the termination of the respective backward or forward algorithm. Generally, given a suitable effective symbolic representation, the requirement on the transition relation of $Player_\forall$ can be relaxed by adding a monotonicity condition regarding the transition relation of $Player_\exists$ and still

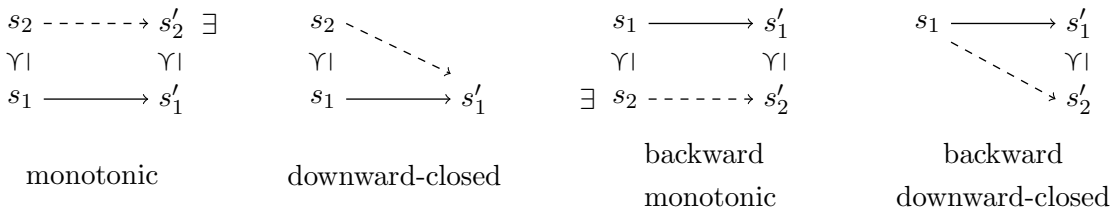


Figure 4.1: Monotonicity properties of (transition) relations.

ensure termination. In the following, we give the definitions of the different monotonicity conditions and state the properties of the game structures that fulfill them.

Definition 4.1.1. Let $(S, \preceq)$ be a quasi-ordering. A relation $R \subseteq S \times S$ is called

- *monotonic* if for each $s_1, s_2, s_1' \in S$ for which $(s_1, s_1') \in R$ and $s_1 \preceq s_2$, there exists $s_2' \in S$ such that $(s_2, s_2') \in R$ and $s_1' \preceq s_2'$.
- *downward-closed* if for each $s_1, s_2, s_1' \in S$ for which $(s_1, s_1') \in R$ and $s_1 \preceq s_2$, it holds that $(s_2, s_1') \in R$.
- *backward-monotonic* if for each $s_1, s_1', s_2' \in S$ for which $(s_1, s_1') \in R$ and $s_2' \preceq s_1'$, there exists $s_2 \in S$ such that $(s_2, s_2') \in R$ and $s_2 \preceq s_1$.
- *backward-downward-closed* if for each $s_1, s_1', s_2' \in S$ for which $(s_1, s_1') \in R$ and $s_2' \preceq s_1'$, it holds that $(s_1, s_2') \in R$.

Figure 4.1 depicts the properties of a binary relation $\rightarrow$ from Definition 4.1.1.

For a game structure $G = (S_\exists, S_\forall, I, =_\mathsf{o}, \Sigma_\exists, T_\exists, T_\forall)$ and an action $\sigma \in \Sigma_\exists$ we denote $T_{\exists_\sigma} = \{(s, s') \in S_\exists \times S_\forall \mid (s, \sigma, s') \in T_\exists\}$. For each observation $o \in Obs$, $\mathsf{Enabled}(o) = \{\sigma \in \Sigma_\exists \mid \exists s \in o. \exists s' \in S. (s, \sigma, s') \in T_\exists\}$ is the set of actions enables in o.

Definition 4.1.2. Let $G = (S_\exists, S_\forall, I, =_\mathsf{o}, \Sigma_\exists, T_\exists, T_\forall)$ be a game structure (with incomplete information) and $(S_\exists \dot\cup S_\forall, \preceq)$ be a quasi-ordering. $(G, \preceq)$ is called

- $\exists$-*monotonic*($\exists$-*downward-closed*) if $T_{\exists_\sigma}$ is monotonic(downward-closed) for all σ.
- $\forall$-*monotonic*($\forall$-*downward-closed*) if $T_\forall$ is monotonic(downward-closed).
- *monotonic* (*downward-closed*) if it is both $\exists$-monotonic and $\forall$-monotonic (both $\exists$-downward-closed and $\forall$-downward-closed).

The notions of $\exists$-*($\forall$)-backward-monotonic*,$\exists$-*($\forall$)-backward-downward-closed*, *backward-monotonic* and *backward-downward-closed* game structures are defined analogously, using the respective notions from Definition 4.1.1.

Each $*$-downward-closed game structure is $*$-monotonic. Depending on the monotonicity notions satisfied by the transition relations of a game structure, the respective Pre- and Post-operators enjoy the useful properties stated below. Recall that for a WQO or BQO $(S, \preceq)$ we denote with $\mathcal{U}(S)$ the set of upward-closed subsets of S and with $\mathcal{D}(S)$ the set of downward-closed subsets of S.

Proposition 4.1.1. *Let $G = (S_\exists, S_\forall, I, =_\mathsf{o}, \Sigma_\exists, T_\exists, T_\forall)$ be a game structure (with incomplete information) and $(S_\exists \dot\cup S_\forall, \preceq)$ be a quasi-ordering. If $(G, \preceq)$ is*

- $\forall$*-monotonic, then* $\mathsf{Pre}_\forall(U) \in \mathcal{U}(S)$*, for every* $U \in \mathcal{U}(S)$*.*
- $\forall$*-downward-closed, then* $\mathsf{Pre}_\forall(V) \in \mathcal{U}(S)$*, for every set* $V \subseteq S$*.*
- $\exists$*-monotonic, then* $\mathsf{Pre}_\exists(U, \sigma) \in \mathcal{U}(S)$*, for every* $U \in \mathcal{U}(S)$ *and every* $\sigma \in \Sigma_\exists$*.*
- $\exists$*-downward-closed, then* $\mathsf{Pre}_\exists(V, \sigma) \in \mathcal{U}(S)$*, for every* $V \subseteq S$ *and* $\sigma \in \Sigma_\exists$*.*
- $\forall$*-backward-monotonic, then* $\mathsf{Post}_\forall(D) \in \mathcal{D}(S)$ *for every* $D \in \mathcal{D}(S)$*.*
- $\forall$*-backward-downward-closed, then* $\mathsf{Post}_\forall(V) \in \mathcal{D}(S)$*, for every* $V \subseteq S$*.*
- $\exists$*-backward-monotonic, then* $\mathsf{Post}_\exists(D, \sigma) \in \mathcal{D}(S)$*, for* $D \in \mathcal{D}(S)$ *and* $\sigma \in \Sigma_\exists$*.*
- $\exists$*-backward-downward-closed, then* $\mathsf{Post}_\exists(V, \sigma) \in \mathcal{D}(S)$*, for* $V \subseteq S$ *and* $\sigma \in \Sigma_\exists$*.*

Definition 4.1.3. A *well-quasi ordered (WQO) game structure* is a tuple $(G, \preceq)$, where $G = (S_\exists, S_\forall, I, =_\mathsf{o}, \Sigma_\exists, T_\exists, T_\forall)$ is a game structure (with incomplete information) and $(S_\exists \dot\cup S_\forall, \preceq)$ is a WQO. An WQO game structure $(G, \preceq)$ is a called a *better-quasi ordered (BQO) game structure* if the WQO $(S_\exists \dot\cup S_\forall, \preceq)$ is a BQO.

By definition, each BQO game structure is a WQO game structure.

A *constraint* ϕ for a given game structure $G = (S_\exists, S_\forall, I, =_\mathsf{o}, \Sigma_\exists, T_\exists, T_\forall)$ denotes a possibly infinite set of states $[\![\phi]\!]$ in G. Constraints are used to represent (infinite) sets of states in symbolic algorithms. For example, in the algorithms described in Section 3.3.4 and Section 3.3.5 the sets of constraints used are the sets $\Phi_{\mathcal{U}_{obs}}$ and $\Phi_{\mathcal{D}_{obs}}$. In order to use a set of constraints in a symbolic algorithm, it has to satisfy the respective expressiveness and effectiveness conditions defined below.

Definition 4.1.4. A set of constraints Φ is Pre-*effective* for a BQO game structure $(G, \preceq)$ with $G = (S_\exists, S_\forall, I, =_o, \Sigma_\exists, T_\exists, T_\forall)$ if it fulfills the following conditions:

(i) for each $\phi, \phi' \in \Phi$ and $s \in S$ we can decide whether $[\![\phi]\!] \subseteq [\![\phi']\!]$, $[\![\phi]\!] = \emptyset$, $c \in [\![\phi]\!]$;

(ii) for each $U \in \mathcal{U}(S)$ there exists $\phi_U \in \Phi$ such that $[\![\phi_U]\!] = U$;

(iii) for each $\phi_1, \phi_2 \in \Phi$ and $o \in \mathit{Obs}$ for which it holds that $[\![\phi_1]\!] \subseteq o$ and $[\![\phi_2]\!] \subseteq o$, we can compute $\phi \in \Phi$ such that $[\![\phi]\!] = [\![\phi_1]\!] \cup [\![\phi_2]\!]$;

(iv) for each $\phi \in \Phi$ and $o \in \mathit{Obs}$ we can compute $\phi' \in \Phi$ such that $[\![\phi']\!] = [\![\phi]\!] \cap o$;

(v) for each $\phi \in \Phi$ and $\sigma \in \Sigma_\exists$ we can compute $\phi' \in \Phi$ such that $[\![\phi']\!] = \mathsf{Pre}_\exists([\![\phi]\!], \sigma)$;

(vi) for each $\phi \in \Phi$ we can compute $\phi' \in \Phi$ such that $[\![\phi']\!] = \mathsf{Pre}_\forall([\![\phi]\!])$.

Definition 4.1.5. A set of constraints Φ is Post-*effective* for a BQO game structure $(G, \preceq)$ with $G = (S_\exists, S_\forall, I, =_o, \Sigma_\exists, T_\exists, T_\forall)$ if it fulfills the following conditions:

(i) for each $\phi, \phi' \in \Phi$ and $s \in S$ we can decide whether $[\![\phi]\!] \subseteq [\![\phi']\!]$, $[\![\phi]\!] = \emptyset$, $c \in [\![\phi]\!]$;

(ii) for each $D \in \mathcal{D}(S)$ there exists $\phi_D \in \Phi$ such that $[\![\phi_D]\!] = D$;

(iii) for each $\phi_1, \phi_2 \in \Phi$ and $o \in \mathit{Obs}$ for which it holds that $[\![\phi_1]\!] \subseteq o$ and $[\![\phi_2]\!] \subseteq o$, we can compute $\phi \in \Phi$ such that $[\![\phi]\!] = [\![\phi_1]\!] \cup [\![\phi_2]\!]$;

(iv) for each $\phi \in \Phi$ and $o \in \mathit{Obs}$ we can compute $\phi' \in \Phi$ such that $[\![\phi']\!] = [\![\phi]\!] \cap o$;

(v) for each $\phi \in \Phi$ and $\sigma \in \Sigma_\exists$ we can compute $\phi' \in \Phi$ such that $[\![\phi']\!] = \mathsf{Post}_\exists([\![\phi]\!], \sigma)$;

(vi) for each $\phi \in \Phi$ we can compute $\phi' \in \Phi$ such that $[\![\phi']\!] = \mathsf{Post}_\forall([\![\phi]\!])$.

The following theorems generalize the results stated in 3.3.1. They establish generic conditions on the input games that ensure that the algorithms presented in Section 3.3 decide the game solving problem and solve the strategy synthesis problems for the resulting class of games, considering the class of obs_a-consistent $\mathit{Player}_\exists$ strategies.

Theorem 4.1.1. *Let $(G, \preceq)$ be a BQO game structure, where $G = (S_\exists, S_\forall, I, =_o, \Sigma_\exists, T_\exists, T_\forall)$, Φ be a* Pre*-effective set of constraints for G, and $\phi_{Err} \in \Phi$.*

If $(G, \preceq)$ and ϕ_{Err} enjoy the following properties:

- *the sets $\Sigma_\exists$, Obs and I are finite,*
- *for each $o \in \mathit{Obs}$ we can compute* $\mathsf{Enabled}(o)$*,*

- *for each* $o \in Obs$, *if* $\sigma \in \mathsf{Enabled}(o)$ *then there is* $o' \in Obs$ *with* $\mathsf{Post}_\exists(o, \sigma) \subseteq o'$,
- *for each* $s_1, s_2 \in S$, $s_1 \preceq s_2$ *implies* $s_1 =_\mathsf{o} s_2$,
- *the transition relations, the relation* $\preceq$ *and* ϕ_{Err} *satisfy one of the conditions:*

  (I) $(G, \preceq)$ *is monotonic and* $[\![\phi_{Err}]\!]$ *is upward-closed, or*

  (II) $(G, \preceq)$ *is* $\forall$*-downward-closed,*

then we can decide whether $Player_\exists$ *has an* obs_a*-consistent winning strategy in the game* $\mathsf{Safety}(G, [\![\phi_{Err}]\!])$ *or not, and if the answer is positive we can compute such a strategy.*

Proof. To solve a safety BQO game under incomplete information, given a Pre-effective set of constraints, we apply Algorithm 1 using the elements of Φ to represent elements of $\mathcal{U}_{obs}(G)$, instead of using $\Phi_{\mathcal{U}_{obs}}$ as we did in the case of LC-games.

Since Φ is closed under intersection with elements of Obs, we can clearly initialize the set B in Algorithm 1 with elements of Φ. Now the elements of $\mathcal{L}(G)$ are finite subsets of Φ. Since we can check inclusion between elements of Φ, we can also check equality between elements of $\mathcal{L}(G)$, and hence the loop condition. The set I is finite and we can check membership in elements of Φ, which entails that we can check whether $I \cap [\![\phi]\!] = \emptyset$ for $\phi \in \Phi$. As we can check inclusion between elements of Φ, $(\mathcal{L}(G), \sqsubseteq)$ is a decidable BQO, and hence the function Min is computable. Conditions (iv), (v) and (vi) from the definition of Pre-effective set of constraints imply that the function UPre is computable. Thus, the set B can be effectively updated.

If $Player_\exists$ has an obs_a-consistent winning strategy in $\mathsf{Safety}(G, [\![\phi_{Err}]\!])$, a finite such strategy can be effectively constructed by the algorithm working with the effective set of constraints Φ. As stated in the proof of Proposition 3.3.3 to construct the transition relation of the strategy automaton it suffices to be able to compute $\mathsf{Pre}_\exists$ and $\mathsf{Pre}_\forall$ for constraints, finite union of constraints with the same observation, and intersection with elements of Obs and to perform emptiness and inclusion checks for elements of Φ.

Note, however, that to be able to construct a knowledge-based counterexample tree in $\mathsf{Safety}(\widetilde{G}, [\![\phi_{Err}]\!])$ as in the proof of Proposition 3.3.2, we would need the set of constraints to be Post-effective and the transition relations $T_\exists$ and $T_\forall$ to have finite branching, in order to be able to compute the labels of the nodes of the tree. □

Theorem 4.1.2. *Let* $(G, \preceq)$ *be a BQO game structure, where* $G = (S_\exists, S_\forall, I, =_\mathsf{o}, \Sigma_\exists, T_\exists, T_\forall)$, Φ *be a* Post*-effective set of constraints for* G, *and* $\phi_{Goal} \in \Phi$.

If $(G, \preceq)$ *enjoys the following properties:*

- *the sets* $\Sigma_\exists$, Obs *and* I *are finite,*

- *the transition relations $T_\exists$ and $T_\forall$ have finite branching,*
- *for each $o \in Obs$ we can compute* $\mathsf{Enabled}(o)$,
- *for each $o \in Obs$, if $\sigma \in \mathsf{Enabled}(o)$ then there is $o' \in Obs$ with $\mathsf{Post}_\exists(o, \sigma) \subseteq o'$,*
- *for each $s_1, s_2 \in S$, $s_1 \preceq s_2$ implies $s_1 =_{\mathsf{o}} s_2$,*
- *the transition relations, the relation $\preceq$ and I satisfy one of the conditions:*

  (I) $(G, \preceq)$ is backward-monotonic and the set I is downward-closed, or

  (II) $(G, \preceq)$ is $\forall$-backward-downward-closed,

then we can decide whether $Player_\exists$ has an obs_a-consistent winning strategy in the game $\mathsf{Reach}(G, [\![\phi_{Goal}]\!])$ *or not, and if the answer is positive we can compute such a strategy.*

Proof. To solve a reachability BQO game under incomplete information, given a Post-effective set of constraints, we apply Algorithm 2 using the elements of Φ to represent elements of $\mathcal{D}^{\mathsf{fin}}_{obs}(G)$, instead of using $\Phi_{\mathcal{D}_{obs}}$ as we did in the case of LC-games.

The labels of the constructed AND-OR tree are elements of Φ. To compute the nodes' labels, the functions $\mathsf{Post}^{obs}_\exists$ and $\mathsf{Post}^*_\forall$ and finite union of sets in $\mathcal{D}^{\mathsf{fin}}_{obs}(G)$ with the same observation must be computable. Conditions (iv) and (v) from Definition 4.1.5 imply that $\mathsf{Post}^{obs}_\exists$ is computable, and using condition (iii) we know that finite union of constraints with the same observation is also computable. As we saw in Proposition 3.3.4, the function $\mathsf{Post}^*_\forall$ applied to an element of Φ can be computed by exploring a finite tree, because Obs and the branching of $T_\forall$ are finite and $(S, \preceq)$ is a BQO, and using the function $\mathsf{Post}^{obs}_\forall$, which is computable since Φ satisfies conditions (iv) and (vi) from Definition 4.1.5. We can decide inclusion between node labels and check inclusion in $[\![\phi]\!]$ since Φ is Post-effective. The AND-OR tree is finite, since $(S, \preceq)$ is a BQO.

If $Player_\exists$ has an obs_a-consistent winning strategy in $\mathsf{Reach}(G, [\![\phi_{Goal}]\!])$, a finite such strategy can be effectively constructed by the algorithm working with the effective set of constraints Φ. As seen in the proof of Proposition 3.3.8, to construct the transition relation of the strategy automaton it suffices to be able to compute $\mathsf{Post}^{obs}_\exists$, $\mathsf{Post}^*_\forall$ and finite union of constraints with the same observation. □

The preconditions of Theorem 4.1.1 and Theorem 4.1.2 guarantee the termination of the set-saturation and tree-saturation methods respectively. The first one guarantees that the fact about stabilization of infinite monotonic sequences of upward-closed sets can be applied, while the second one ensures the finiteness of the AND-OR tree. Notice

that if for each $s_1, s_2 \in S$ such that $s_1 \preceq s_2$ it holds that $s_1 =_o s_2$, then every observable set is both-upward and downward-closed. Hence, by requiring observability of a set of error or goal states we can ensure that they are upward as well as downward-closed.

4.2 R-stable Games

In this section we consider a class of infinite-state games of imperfect information defined in [DWDR06], called R-stable games, for which safety and reachability games were shown to be decidable [DWDR06]. We first describe the game model used in [DWDR06, De 06] and give a simple translation from their model into ours. In the graph games of [DWDR06, De 06] edges are labeled with actions and one player is responsible for choosing the action, while the other resolves the remaining nondeterminism by choosing a successor state from where the game proceeds.

Definition 4.2.1. ([De 06]) A *game of imperfect information* $G = (S, E, F, \Sigma, \Delta, O)$ is a tuple where S is a set of states, E is the set of initial states, F is the set of final states, Σ is a finite alphabet of actions, $\Delta \subseteq S \times \Sigma \times S$ is a set of labeled transitions, such that for each $s \in S$ and each $\sigma \in \Sigma$ there exists $s' \in S$ such that $(s, \sigma, s') \in \Delta$, and $O \subseteq 2^S$ is a set of observations. If for each $o_1, o_2 \in O$ it holds that $o_1 \cap o_2 \neq \emptyset$ implies that $o_1 = o_2$ the game G is called a game of *incomplete information*.

For a set $V \subseteq S$ and $\sigma \in \Sigma$, $\mathsf{Post}_\sigma(V) = \{s' \mid \exists s \in V.\ (s, \sigma, s') \in \Delta\}$.

Each game of imperfect information can be transformed into an equivalent game of incomplete information. For finite games this can be done in polynomial time.

Definition 4.2.2. ([De 06]) An *observation-based strategy* in an imperfect-information game $G = (S, E, F, \Sigma, \Delta, O)$ is a function $\lambda : O^+ \to \Sigma$. The notions of strategy outcome and safe strategies are defined in a standard way. The game solving problem for safety games of imperfect information asks to decide if there exists a safe observation-based strategy in the given game. For formal definitions we refer the reader to [De 06].

In [DWDR06] a fixpoint algorithm for solving finite-state safety and reachability games of imperfect (incomplete) information was proposed. The fixpoint of a controllable predecessor operator is computed on the lattice of antichains of state sets and yields the set of states from which the imperfectly (incompletely) informed player has an observation based strategy (w.r.t. a synchronous observation function). The approach

has been extended to general ω-regular objectives in [CDHR06]. For the infinite-state case [DWDR06] identifies a class of games for which the antichain-based approach can be used to solve safety and reachability games of imperfect information.

Let $\{r_1, r_2, \ldots, r_n\}$ be a finite partition of a possibly infinite state set S. A set $A \subseteq S$ is *R-definable* if $A = \bigcup_{r \in R'}$ for some $R' \subseteq R$.

Definition 4.2.3. ([DWDR06]) An imperfect-information game $G = (S, E, F, \Sigma, \Delta, O)$ is *R-stable* if the following conditions hold:

(i) for every $r \in R$ and every $\sigma \in \Sigma$, the set $\mathsf{Post}_\sigma(r)$ is R-definable,

(ii) every $o \in O$ is R-definable,

(iii) E and F are R-definable,

(iv) for all $r, r' \in R$ and for all $\sigma \in \Sigma$ it holds that:

$$\big(\exists s \in r.\mathsf{Post}_\sigma(\{s\}) \cap r' \neq \emptyset\big) \Longrightarrow \big(\forall s \in r.\mathsf{Post}_\sigma(\{s\}) \cap r' \neq \emptyset\big).$$

Theorem 4.2.1. *([DWDR06]) Safety and reachability games with imperfect information are decidable for the class of R-stable games.*

We can translate each game of incomplete information $G = (S, E, F, \Sigma, \Delta, O)$ with safety (respectively reachability) objective to a safety (respectively reachability) game $\mathsf{Safety}(\widetilde{G}, F)$ (respectively $\mathsf{Reach}(\widetilde{G}, F)$) where the game structure $\widetilde{G} = \mathit{translate}(G) = (S_\exists, S_\forall, I, =_\mathsf{o}, \Sigma_\exists, T_\exists, T_\forall)$ with incomplete information is such that:

- $S_\exists = S$, $S_\forall = S \times \Sigma$, $I = E$ and
- for $\widetilde{s}_1, \widetilde{s}_2 \in \widetilde{S}$, where $\widetilde{S} = S_\exists \dot\cup S_\forall$, we have $\widetilde{s}_1 =_\mathsf{o} \widetilde{s}_2$ iff $\widetilde{s}_1, \widetilde{s}_2 \in S_\exists$ and there exists an $o \in O$ such that $\widetilde{s}_1 \in o$ and $\widetilde{s}_2 \in o$, or $\widetilde{s}_1, \widetilde{s}_2 \in S_\forall$, $\widetilde{s}_1 = (s_1, \sigma)$ and $\widetilde{s}_2 = (s_2, \sigma)$ for some σ, and there exists an $o \in O$ such that $s_1 \in o$ and $s_2 \in o$,
- $\Sigma_\exists = \Sigma$, $T_\exists = \{(s, \sigma, (s, \sigma)) \mid s \in S, \sigma \in \Sigma\}$ and $T_\forall = \{((s, \sigma), s') \mid (s, \sigma, s') \in \Delta\}$.

The game structure $\widetilde{G} = \mathit{translate}(G)$ has the following properties:

- $\mathit{Player}_\exists$ has an obs_s-consistent winning strategy in the game $\mathsf{Safety}(\widetilde{G}, F)$ iff there exists a safe observation-based strategy for G.

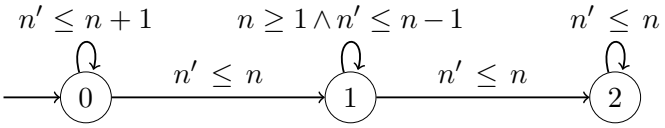


Figure 4.2: Game under imperfect information that is not R-stable for any finite R.

- *Player*$_\exists$ has an obs_s-consistent winning strategy in the game $\mathsf{Reach}(\widetilde{G}, F))$ iff there exists a reaching observation-based strategy for G.
- Every obs_s-consistent strategy for *Player*$_\exists$ in $\widetilde{G}$ is also obs_a-consistent.

Therefore obs_s and obs_a consistent *Player*$_\exists$ strategies coincide for games obtained via the above translation, and thus we can reduce the game solving and strategy synthesis problems for the games from Definition 4.2.1 to the game solving and strategy synthesis problems for the class of games obtained by the above translation and the class of obs_a-consistent (obs_s-consistent) *Player*$_\exists$ strategies.

In Section 4.1 we gave sufficient conditions for the decidability of the game solving and strategy synthesis problems for a class of safety and a class of reachability games under incomplete information. We will now show that the classes of game structures considered in Theorem 4.1.1 and Theorem 4.1.2 are not contained in the class of R-stable games studied in [DWDR06]. To this end, we will give an example of a game of incomplete information $G = (S, E, F, \Sigma, \Delta, O)$ that is not R-stable, but which, using the above transformation, translates into a game, whose game structure meets the requirements of Theorem 4.1.1 and Theorem 4.1.2.

Example 4.2.1. We define the set S of states of the game $G = (S, E, F, \Sigma, \Delta, O)$ as $S = \{(q, n) \mid q \in \{0, 1, 2\}, n \in \mathbb{N}\}$. The set of initial states is $E = \{(0, 0)\}$ and the set of final states is $F = \{(2, n) \mid n \in \mathbb{N}\}$. We let $\Sigma = \{\sigma\}$ and define the transition relation

$$\begin{aligned}\Delta &= \{((0,i),\sigma,(0,j)) \mid j \leq i+1\} \cup \{((1,i),\sigma,(1,j)) \mid i \geq 1 \wedge j \leq i-1\} \\ &\cup \{((q,i),\sigma,(q',j)) \mid (q=0 \wedge q'=1 \vee q=1 \wedge q'=2 \vee q=2 \wedge q'=2) \wedge j \leq i\}.\end{aligned}$$

The transition relation Δ is depicted in Figure 4.2.

Finally the finite set of observations is $O = \{o_{(0,0)}, o_{(0,1)}, o_{(1,0)}, o_{(1,1)}, o_{(2,0)}, o_{(2,1)}\}$, where for each $q \in \{0, 1, 2\}$, we have $o_{(q,0)} = \{(q, 0)\}$ and $o_{(q,1)} = \{(q, n) \mid n > 0\}$.

Let us assume that $R = \{r_1, r_2, \ldots, r_n\}$ is a finite partition of S and G is R-stable. We note $s_1 \sim_R s_2$ iff there exists an $r \in R$ such that $s_1 \in r$ and $s_2 \in r$.

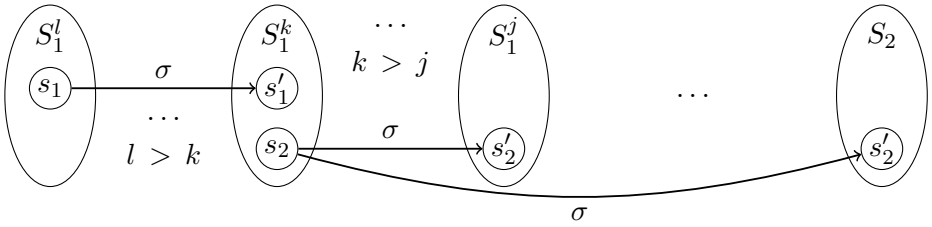


Figure 4.3: Illustration of the inductive step of the proof that the partition R of S' cannot be finite if it satisfies condition (iv) of the definition of R-stable.

Let $S_i = \{(q,n) \in S \mid q = i\}$ for $i = 0,1,2$ and furthermore let $S_i^l = \{(q,n) \in S_i \mid n = l\}$ for all natural numbers $l \in \mathbb{N}$. By condition (ii) of the definition of R-stable games, which requires that each $o \in O$ is R-definable, we have that for each $i,j \in \{0,1,2\}$ with $i \neq j$ it holds for each $s_1 \in S_i$ and each $s_2 \in S_j$ that $s_1 \not\sim_R s_2$.

We now show that for every $l \in \mathbb{N}$, every $0 \leq k < l$, every $s_1 \in S_1^l$ and every $s_2 \in S_1^k$ it holds that $s_1 \not\sim_R s_2$. We will thus arrive at contradiction to fact that R was chosen to be finite. The proof is by induction on l. The claim follows trivially for $l = 0$.

Now let $l > 0$ and $0 \leq k < l$. Note that by the definition of the transition relation Δ we have $\mathsf{Post}_\sigma(S_1^l) \cap S_1^k \neq \emptyset$ and that $\mathsf{Post}_\sigma(S_1^k) \cap S_1^j \neq \emptyset$ implies $j < k$. Suppose that for some $s_1 \in S_1^l$, $s_2 \in S_1^k$ we have $s_1 \sim_R s_2$. There exists $s_1' \in S_1^k$, as illustrated in Fig. 4.3, such that $s_1 \xrightarrow{\sigma} s_1'$. Condition (iv) of the definition of R-stable games implies that there exists a state s_2' such that $s_2 \xrightarrow{\sigma} s_2'$ and $s_1' \sim_R s_2'$. There are two possibilities according to Δ: either $s_2' \in S_1^j$ for some $j < k$ or $s_2' \in S_2$. In the first case by the induction hypothesis and in the second case by condition (ii) of Definition 4.2.3 it follows that $s_1' \not\sim_R s_2'$, which contradicts the choice of s_2'. Therefore, $s_1 \not\sim_R s_2$. □

Let $\widetilde{G} = \mathit{translate}(G) = (S_\exists, S_\forall, I, =_o, \Sigma_\exists, T_\exists, T_\forall)$ be the game structure with incomplete information obtained from G using the translation that we described above.

We will provide a BQO $\preceq$ and Pre-effective and Post-effective sets of constraints Φ_1 and Φ_2, and show that the requirements of Theorems 4.1.1 and 4.1.2 are met.

The set $\Sigma_\exists$ is finite, since Σ is finite. As the set O of observations in G is also finite, the set Obs is finite as well. Since $E = \{(0,0)\}$ is finite, so is the set I of initial states in $\widetilde{G}$. The finiteness of Σ implies that the transition relation $T_\exists$ for $\mathit{Player}_\exists$ has finite branching. By the definition of Δ in our example we have that for each $(((q,i),\sigma),(q',j)) \in T_\forall$ it holds that $0 \leq j \leq i+1$. Thus, $T_\forall$ has finite branching as well. As for each $o \in \mathit{Obs}$ we have $\mathsf{Enabled}(o) = \Sigma_\exists$, the function $\mathsf{Enabled}$ is computable.

If $\widetilde{s}_1, \widetilde{s}_2 \in S_\exists$ are such that $\widetilde{s}_1 =_\mathsf{o} \widetilde{s}_2$ and $\sigma \in \Sigma_\exists$, we have $(s_1, \sigma) =_\mathsf{o} (s_2, \sigma)$. Thus, for each $o \in Obs$ and $\sigma \in \Sigma_\exists$ there exists an $o' \in Obs$ such that $\mathsf{Post}_\exists(o, \sigma) \subseteq o'$.

Let us define the relation $\preceq$ on $\widetilde{S}$ such that $\widetilde{s}_1 \preceq \widetilde{s}_2$ iff one of the following holds:

- $\widetilde{s}_1 = (q_1, n_1)$, $\widetilde{s}_2 = (q_2, n_2)$, $q_1 = q_2$, and either it holds that $n_1 = 0$ and $n_2 = 0$, or it holds that $n_1 > 0$, $n_2 > 0$ and $n_1 \leq n_2$, or
- $\widetilde{s}_1 = ((q_1, n_1), \sigma_1)$, $\widetilde{s}_2 = ((q_2, n_2), \sigma_2)$, $q_1 = q_2$, $\sigma_1 = \sigma_2$, and either it holds that $n_1 = 0$ and $n_2 = 0$, or it holds that $n_1 > 0$, $n_2 > 0$ and $n_1 \leq n_2$

Clearly, this definition together with that of O implies that $s_1 \preceq s_2$ entails $s_1 =_\mathsf{o} s_2$.

Note that, in our example, if $(((q, i), \sigma), (q', j)) \in T_\forall$, then also $(((q, k), \sigma), (q', l)) \in T_\forall$ for each $k \geq i$ and $l \leq j$. Therefore, the transition relation $T_\forall$ is downward-closed, as well as backward-downward-closed, and hence also monotonic and backward monotonic. Since the transition relation $T_\exists$ does not modify the S-component of the state, each of the relations $T_{\exists_\sigma}$ for $\sigma \in \Sigma_\exists$ is monotonic and backward-monotonic.

Clearly, $(\widetilde{S}, \preceq)$ is a BQO. According to the definition of $\preceq$, the set $I = \{(0, 0)\}$ is both upward- and downward-closed, and so is the set $F = \{(2, n) \mid n \in \mathbb{N}\}$.

Let $\Phi_1 = \{\phi^1_s \mid s \in \widetilde{S}\}$, and let:

$$
\begin{array}{lcl}
[\![\phi^1_{(q,0)}]\!] & = & \{(q, 0)\} \text{ for each } q \in \{0, 1, 2\}, \\
[\![\phi^1_{(q,n)}]\!] & = & \{(q, n') \mid n' \geq n\} \text{ for each } n \geq 1 \text{ and } q \in \{0, 1, 2\}, \\
[\![\phi^1_{((q,0),\sigma)}]\!] & = & \{((q, 0), \sigma)\} \text{ for each } q \in \{0, 1, 2\}, \\
[\![\phi^1_{((q,n),\sigma)}]\!] & = & \{((q, n'), \sigma) \mid n' \geq n\} \text{ for each } n \geq 1 \text{ and } q \in \{0, 1, 2\}.
\end{array}
$$

Let $\Phi_2 = \{\phi^2_s \mid s \in \widetilde{S}\} \dot\cup \{\phi^2_{(q,\infty)}, (\phi^2_{((q,\infty),\sigma)} \mid q \in \{0, 1, 2\}\}$, and;

$$
\begin{array}{lcl}
[\![\phi^2_{(q,0)}]\!] & = & \{(q, 0)\} \text{ for each } q \in \{0, 1, 2\}, \\
[\![\phi^2_{(q,n)}]\!] & = & \{(q, n') \mid n' \leq n\} \text{ for each } n \geq 1 \text{ and } q \in \{0, 1, 2\}, \\
[\![\phi^2_{(q,\infty)}]\!] & = & \{(q, n') \mid n' \in \mathbb{N}\} \text{ for each } q \in \{0, 1, 2\}, \\
[\![\phi^2_{((q,0),\sigma)}]\!] & = & \{((q, 0), \sigma)\} \text{ for each } q \in \{0, 1, 2\}, \\
[\![\phi^2_{((q,n),\sigma)}]\!] & = & \{((q, n'), \sigma) \mid n' \leq n\} \text{ for each } n \geq 1 \text{ and } q \in \{0, 1, 2\}, \\
[\![\phi^2_{((q,\infty),\sigma)}]\!] & = & \{((q, n'), \sigma) \mid n' \in \mathbb{N}\} \text{ for each } q \in \{0, 1, 2\}.
\end{array}
$$

It is easy to see that the sets Φ_1 and Φ_2 are Pre-effective and Post-effective sets of constraints respectively, for the game structure $\widetilde{G}$.

Thus, together with $\preceq$ and the sets of constraints above, the games $\mathsf{Safety}(\widetilde{G}, F)$ and $\mathsf{Reach}(\widetilde{G}, F)$ satisfy the conditions identified in Section 4.1.

Part II
Counterexample-Guided Abstraction Refinement for Games under Incomplete Information

Chapter 5

Counterexample-Guided Abstraction Refinement for Games under Incomplete Information

The goal of abstraction is to map a complex (possibly infinite-state) system to a simpler, finite-state system of manageable size that preserves enough information about the original system to prove or disprove the property of interest. However, it is most often the case that the resulting approximation is too coarse, and solving the decision problem on the abstract instance does not yield a definite answer to the decision problem for the concrete system. One possibility in this case is to report an indefinite answer, or a potential false positive or false negative respectively. Alternatively, abstraction-based verification methods [CGJ+00, LBBO01, DD02, CCGS03, SG04] that employ the *abstraction refinement* paradigm automatically refine the current abstraction to obtain more precise approximation of the original system. Typically, the refinement procedures rely on identifying the reason for the indefinite answer or the false counterexample, in order to identify missing information that is necessary in the abstraction. Based on this information, the abstraction is then refined and the process is repeated in the so-called *abstraction-refinement* loop until a definite result for the concrete system is obtained.

In the past years several abstraction-refinement frameworks for games with perfect information have been developed. Ball and Kupferman [BK06] describe an abstraction-

refinement framework for concurrent multi-player games. Their abstraction is based on the three-valued semantics for the alternating μ-calculus. Upon an indefinite answer, the refinement procedure identifies a state in which the evaluation of the specification became indefinite and a subformula which has undefined value in this state, and splits the state into states in which this subformula has a precise value. An approach to solving two-player turn-based games via three-valued abstraction refinement was proposed in [dAR07]. The abstraction is analyzed using must and may controllable predecessor operators, computing sets of *never-win*, *must-win* and *may-win* states and the refinement reduces the number of may-win states to get closer to a definite answer. Henzinger et al. [HJM03] extends the paradigm of *counterexample-guided abstraction refinement (CEGAR)*, one of the most successful and widely used techniques for automatic abstraction refinement in verification, to games with perfect information.

To the best of our knowledge, the only line of work that applies abstraction techniques to synthesis of infinite-state systems under partial observability is [KGMM09, KGMM12]. There, the input infinite-state discrete event system is abstracted w.r.t. a *fixed, possibly infinite abstract domain* and widening is used to enforce convergence. A limitation of the procedure presented there is that it is a backward approach, suitable for computing safe memoryless controllers or controller strategies that record the last k observations along the execution for a fixed k. For general controllers with memory, the procedure does not give an answer to the decision problem but instead pre-computes allowed sets of actions for possible knowledge sets that an online controller can use to determine the correct actions at the states in this set, or the absence of a safe action.

One of the most popular and successful techniques for generating refinement predicates in verification [McM06, BDFW07] is *Craig interpolation*. There one infers from an unconcretizable abstract counterexample path a formula that intuitively corresponds to an explanation for the unconcretizability. This formula refers only to the variables associated with a particular position of the abstract counterexample path, and hence the predicates occurring in this formula can be used to refine the abstraction.

In this chapter we describe a CEGAR scheme for (infinite-state) games under incomplete information with safety winning conditions. The approach is based on *predicate abstraction* [DD02] and constructs a finite-state abstract game structure with perfect information directly from the given symbolic game structure with incomplete information. The abstract game structure simulates, in the sense of an alternating simulation

relation, the knowledge based game corresponding to the symbolic game structure. It is thus a sound abstraction of the original game. The predicates play a two-fold role in the abstraction of a game structure with incomplete information. As usual, they define the abstract state-space and determine the precision of the abstract transition relation. Here, however this includes the *approximation of the observation equivalence* that is part of the process of constructing a game structure with perfect information. To refine the latter approximation we developed an extension of the constraint-based interpolation technique from [RSS07], which extension provides Craig interpolants that meet a variable partitioning requirement provided as input.

5.1 Abstraction for Incomplete-Information Games

The abstraction method for game structures with incomplete information we present in this section is based on *predicate abstraction*. Predicate abstraction [DD02] is a technique successfully used in software verification [BMMR01, HJMS03, PR07, DKFW10]. Given a concrete system and a finite set of predicates, it constructs an abstract finite-state system that simulates the concrete one. In classical predicate abstraction for verification, states in the abstract system are boolean vectors interpreted as valuations of the abstraction predicates. Each abstract state thus represents a (possibly infinite) set of concrete states, and thus the construction can be seen as a powerset construction.

The abstract game construction that we give here integrates predicate abstraction with the knowledge-based subset construction described in Section 2.2.3.

5.1.1 Abstraction Predicates

The abstraction is parametrized by a finite set of predicates $\mathcal{P}$, which determines its precision, that is the precision of the abstract transition relations for the two players. The precision of the transition relation for player $Player_\exists$ depends on the observations that he can make, which is related to the type of the predicates in $\mathcal{P}$, and more specifically, to the observability of the variables occurring in these predicates.

Predicates The set $\mathcal{AP}$ consists of the atomic formulas in the underlying logical theory and from now on we refer to its elements as *predicates*. For a given set of variables X we define $\mathcal{AP}[X] = \{\varphi \in \mathcal{AP} \mid \mathit{Vars}(\varphi) \subseteq X\}$. For a set $\mathcal{P} \subseteq \mathcal{AP}$ of

predicates we denote with $\mathcal{B}(\mathcal{P})$ the closure of $\mathcal{P}$ under boolean operators. Given a formula φ, we denote with $\mathit{Preds}(\varphi)$ is the set of predicates that occur in φ.

Recall that for a set $\mathcal{P}$ of predicates, $\mathit{Vals}(\mathcal{P}) = \mathbb{B}^{\mathcal{P}}$ is the set of all truth valuations of the elements of $\mathcal{P}$. For each $a \in \mathit{Vals}(\mathcal{P})$ and $\varphi \in \mathcal{P}$ we write $a \models \varphi$ iff $a(\varphi) = \mathit{true}$. Given a finite set $\mathcal{P} \subseteq \mathcal{AP}[X]$ of predicates over a set of variables X, we associate with each $a \in \mathit{Vals}(\mathcal{P})$ a formula $[a] = \big(\bigwedge_{\varphi\in\mathcal{P}, a\models\varphi} \varphi\big) \wedge \big(\bigwedge_{\varphi\in\mathcal{P}, a\not\models\varphi} \neg\varphi\big)$ and let $[\![a]\!] = [\![[a]]\!]$.

For sets of predicates $\mathcal{P}$ and $\mathcal{Q}$ with $\mathcal{Q} \subseteq \mathcal{P}$ we define the equivalence relation $=_{\mathcal{Q}}$ on $\mathit{Vals}(\mathcal{P})$ as follows: for $a_1, a_2 \in \mathit{Vals}(\mathcal{P})$, $a_1 =_{\mathcal{Q}} a_2$ iff $a_1(\varphi) = a_2(\varphi)$ for each $\varphi \in Q$.

Observation Predicates Let $\mathcal{G} = (V_\exists, V_\forall, V_\forall^o, t, \varphi_{\mathit{Init}}, \mathcal{T}_\exists, \mathcal{T}_\forall)$ be a symbolic game structure with an underlying set of variables $V = V_\exists \,\dot\cup\, V_\forall \,\dot\cup\, \{t\}$. In this chapter we consider only symbolic game structures for which the set $\Sigma_\exists$ of actions in the respective explicit game structure is finite. That is, the domain of each variable $x \in V_\exists$ is finite.

In order to obtain a sound abstraction of $\mathit{Player}_\exists$'s informedness, predicates in which (both observable and) unobservable variables occur are considered unobservable. For a set $\mathcal{P} \subseteq \mathcal{AP}[V]$ of predicates, we define $\mathit{Obs}(\mathcal{P}) = \{\varphi \in \mathcal{P} \mid \mathit{Vars}(\varphi) \subseteq \mathit{Obs}(\mathit{Vars}(\varphi))\}$.

For a finite set $\mathcal{P} \subseteq \mathcal{AP}[V]$ we define

$$\mathit{EnabledObs}(\mathcal{P}) = \bigcup_{a\in \mathit{Vals}(\mathit{Obs}(\mathcal{P}))} \mathit{Preds}\big(\bigvee_{\sigma\in\Sigma_\exists} \mathsf{Pre}_\exists([a], \sigma)\big).$$

By the property (2) of $\mathcal{G}$, we can ensure that the set $\mathit{EnabledObs}(\mathcal{P})$ consists of observable predicates from $\mathcal{AP}[V]$ and its definition entails the following property.

Property 10. *If $s_1, s_2 \in \mathit{Vals}(V)$ are such that $s_1 \models \varphi$ iff $s_2 \models \varphi$ for every $\varphi \in \mathit{EnabledObs}(\mathcal{P})$, then for each $s_1' \in \mathit{Vals}(V)$ and $\sigma_1 \in \Sigma_\exists$ for which $(s_1, \sigma_1, s_1') \in [\![\mathcal{T}_\exists]\!]$ there exist $s_2' \in \mathit{Vals}(V)$ and $\sigma_2 \in \Sigma_\exists$ such that $(s_2, \sigma_2, s_2') \in [\![\mathcal{T}_\exists]\!]$ and such that the following holds: $s_1 \models \varphi$ iff $s_2 \models \varphi$ for every $\varphi \in \mathit{Obs}(\mathcal{P})$.*

Precision We say that a set of predicates $\mathcal{P} \subseteq \mathcal{AP}[V]$ is *precise with respect to a variable $x \in V$* for which the set $\mathsf{Dom}(x)$ is finite, iff $(x = c) \in \mathcal{P}$ for each $c \in \mathsf{Dom}(x)$. We say that a set of predicates $\mathcal{P} \subseteq \mathcal{AP}[V]$ is *precise with respect to the set $\Sigma_\exists = \mathit{Vals}(V_\exists)$ of $\mathit{Player}_\exists$ actions* iff $\mathcal{P}$ is precise with respect to each variable $x \in V_\exists$.

5.1.2 Abstract Game Structure with Perfect Information

Let $\mathcal{G} = (V_\exists, V_\forall, V_\forall^o, t, \varphi_{Init}, \mathcal{T}_\exists, \mathcal{T}_\forall)$ be a symbolic game structure and $V = V_\exists \dot\cup V_\forall \dot\cup \{t\}$. Let $\mathcal{P} \subseteq \mathcal{AP}[V]$ be a finite set of predicates that is precise w.r.t. the variable t.

Let $\mathcal{P}_\exists = \mathit{EnabledObs}(\mathcal{P})$ and let $\mathit{Obs}_\exists = \mathit{Vals}(\mathcal{P} \cup \mathcal{P}_\exists)/_{=_{\mathit{Obs}(\mathcal{P}\cup\mathcal{P}_\exists)}}$ be the set of equivalence classes of $\mathit{Vals}(\mathcal{P} \cup \mathcal{P}_\exists)$ w.r.t. $=_{\mathit{Obs}(\mathcal{P}\cup\mathcal{P}_\exists)}$. Let $\mathit{Obs}_\forall = \mathit{Vals}(\mathcal{P})/_{=_{\mathit{Obs}(\mathcal{P})}}$ be the set of equivalence classes of $\mathit{Vals}(\mathcal{P})$ w.r.t. $=_{\mathit{Obs}(\mathcal{P})}$.

The abstraction $\mathsf{Abstract}(\mathcal{G}, \mathcal{P})$ of $\mathcal{G}$ w.r.t. the finite set $\mathcal{P}$ of predicates is a game structure with perfect information that can be constructed directly from the symbolic game structure $\mathcal{G}$ using a decision procedure for the underlying logical theory. The set of states of the abstract game structure is a subset of $2^{\mathit{Vals}(\mathcal{P})} \cup 2^{\mathit{Vals}(\mathcal{P}\cup\mathcal{P}_\exists)}$ and the set of possible $\mathit{Player}_\exists$ actions is $\mathit{Vals}(\mathit{Obs}(\mathcal{P}))$, and thus the game structure is finite.

The game structure $\mathsf{Abstract}(\mathcal{G}, \mathcal{P}) = (S_\exists^\#, S_\forall^\#, I^\#, =_\mathsf{o}^\#, \Sigma_\exists^\#, T_\exists^\#, T_\forall^\#)$, which is the *abstraction of G w.r.t. $\mathcal{P}$* is a finite-state game structure defined as follows.

- $S_\exists^\# = \{s^\# \in 2^{\mathit{Vals}(\mathcal{P}\cup\mathcal{P}_\exists)} \setminus \emptyset \mid (\forall a \in s^\#.\ [\![a]\!] \neq \emptyset \wedge a \models t = \exists) \wedge (\exists o \in \mathit{Obs}_\exists.\ s^\# \subseteq o)\}$,
- $S_\forall^\# = \{s^\# \in 2^{\mathit{Vals}(\mathcal{P})} \setminus \emptyset \mid (\forall a \in s^\#.\ [\![a]\!] \neq \emptyset \wedge a \models t = \forall) \wedge (\exists o \in \mathit{Obs}_\forall.\ s^\# \subseteq o)\}$,
- $S^\# = S_\exists^\# \dot\cup S_\forall^\#$,
- $I^\# = \{s^\# \in S^\# \mid \exists o \in (\mathit{Obs}_\exists \dot\cup \mathit{Obs}_\forall).\ s^\# = I' \cap o\}$, where
  $I' = \{a \in \mathit{Vals}(\mathcal{P}) \cup \mathit{Vals}(\mathcal{P} \cup \mathcal{P}_\exists) \mid [\![a]\!] \cap [\![\varphi_{Init}]\!] \neq \emptyset\}$,
- $=_\mathsf{o}^\#$ is the equality relation,
- $\Sigma_\exists^\# = \mathit{Vals}(\mathit{Obs}(\mathcal{P}))$,
- $(s_1^\#, s_2^\#) \in T_\exists^\#$ iff the following conditions are satisfied

  (i)$_\exists$ $\forall a \in s_1^\#.\forall s \in [\![a]\!].\exists a' \in s_2^\#.\exists s' \in [\![a']\!].\ (s, s') \models \mathcal{T}_\exists$,

  (ii) $\forall a' \in s_2^\#.\exists a \in s_1^\#.\exists s \in [\![a]\!].\exists s' \in [\![a']\!].\ (s, s') \models \mathcal{T}_\exists$,

  (iii)

  $$\forall a_2 \in s_2^\#.\forall a_2' \in \mathit{Vals}(\mathcal{P}).\ (a_2(\mathit{Obs}(\mathcal{P})) = a_2'(\mathit{Obs}(\mathcal{P})) \wedge \exists a \in s_1^\#.\exists s \in [\![a]\!].\exists s' \in [\![a_2']\!].\ (s, s') \models \mathcal{T}_\exists) \Rightarrow a_2' \in s_2^\#,$$

- $(s_1^\#, s_2^\#) \in T_\forall^\#$ iff the following conditions are satisfied

(i)$_\forall$ $\exists a \in s_1^\#.\exists s \in [\![a]\!].\exists a' \in s_2^\#.\exists s' \in [\![a']\!].\ (s,s') \models \mathcal{T}_\forall$,

(ii) $\forall a' \in s_2^\#.\exists a \in s_1^\#.\exists s \in [\![a]\!].\exists s' \in [\![a']\!].\ (s,s') \models \mathcal{T}_\forall$,

(iii)

$$\forall a_2 \in s_2^\#.\forall a_2' \in \mathit{Vals}(\mathcal{P}).\ (a_2(\mathit{Obs}(\mathcal{P})) = a_2'(\mathit{Obs}(\mathcal{P})) \wedge \exists a \in s_1^\#.\exists s \in [\![a]\!].\exists s' \in [\![a_2']\!].\ (s,s') \models \mathcal{T}_\forall) \Rightarrow a_2' \in s_2^\#.$$

The abstract game structure is an explicit one. Since we allow predicates that relate variables from the sets $V_\exists$ and $V_\forall$, it is not possible in general to use the abstraction predicates and define the abstract game as a symbolic game. Note further that the construction defined above does not correspond to simply abstracting the game into a game under incomplete information using predicate abstraction, and then applying the knowledge-based subset construction w.r.t. an abstract observation equivalence. If we wanted to define the transitions for *predicate valuations* belonging to $\mathit{Player}_\exists$, then we would still need to take the abstracted observation equivalence into account, since the existence of a transition from a $\mathit{Player}_\exists$ valuation a to some a' might depend on the existence of transitions from a to other valuations observationally equivalent to a'.

With an abstract state $s^\# \in S^\#$ we associate the formula $[s^\#] = \bigvee_{a \in s^\#} [a]$ and define the concretization function $\gamma : S^\# \to 2^S$ such that $\gamma(s^\#) = [\![[s^\#]]\!]$.

The concretization function $\gamma_\exists : \Sigma^\# \to 2^{\Sigma_\exists}$ is defined as follows: $\gamma_\exists(\sigma^\#) = \{\sigma \in \Sigma_\exists \mid \exists s \in S.\ (\forall x \in V_\exists.\ s(x) = \sigma(x)) \wedge (\forall \varphi \in \mathit{Obs}(\mathcal{P}).\ s \models \varphi \Leftrightarrow \sigma^\# \models \varphi)\}$.

If $\mathcal{P}$ is precise w.r.t. $\Sigma_\exists$, for each $\sigma^\# \in \Sigma^\#$ the set $\gamma_\exists(\sigma^\#)$ contains at most one σ.

Let us discuss the definition of the abstract transition relations $T_\exists^\#$ and $T_\forall^\#$. Intuitively, condition (i)$_\exists$ guarantees that each choice that $\mathit{Player}_\exists$ can make in the game structure $G^\#$ can be consistently concretized *for all of the respective concrete states*. That is, in $G^\#$ $\mathit{Player}_\exists$ has potentially fewer choices and is therefore less powerful. Condition (i)$_\forall$, on the other hand requires that the choice can be concretized *for some concrete state*, which means that $\mathit{Player}_\forall$ is potentially more powerful in $G^\#$ than in $\mathcal{G}$. Note that the definitions of $T_\exists^\#$ and $T_\forall^\#$ above are equivalent respectively to:

$$(s_1^\#, s_2^\#) \in T_\exists^\# \iff \exists o \in \mathit{Obs}_\exists.\ s_2^\# = \{a' \in \mathit{Vals}(\mathcal{P}) \mid a' \in o \wedge \exists a \in s_1^\#.\exists s \in [\![a]\!].\exists s' \in [\![a']\!].\ (s,s') \models \mathcal{T}_\exists\} \tag{5.1}$$

$$(s_1^\#, s_2^\#) \in T_\forall^\# \iff \exists o \in Obs_\exists.$$
$$s_2^\# = \{a' \in Vals(\mathcal{P} \cup \mathcal{P}_\exists) \mid a' \in o \wedge \exists a \in s_1^\#.\exists s \in [\![a]\!].\exists s' \in [\![a']\!].\ (s, s') \models \mathcal{T}_\forall\} \tag{5.2}$$

However we chose to give the definition in terms of the respective conditions, since the fact that (5.1) implies condition (i)$_\exists$ holds under the specific condition that $\mathcal{P}_\exists \supseteq EnabledObs(\mathcal{P})$, and we want to explain the role of the predicates $EnabledObs(\mathcal{P})$ in the abstraction. More specifically, we want to shed light on the reason for the asymmetry in the definitions of $S_\exists^\#$ and $S_\forall^\#$, namely, why abstract states for $Player_\forall$ are sets of valuations of the predicates in $\mathcal{P}$, while abstract states for $Player_\exists$ are sets of valuations of the possibly larger set of predicates $\mathcal{P} \cup \mathcal{P}_\exists$, where $\mathcal{P}_\exists = EnabledObs(\mathcal{P})$.

The reason behind this definition is that we want to ensure the *monotonicity* of the abstraction, namely that the abstraction constructed w.r.t. a set of predicates $\mathcal{P} \supseteq \mathcal{Q}$ is *at least as precise* as the abstraction constructed w.r.t. $\mathcal{Q}$. Condition (i)$_\exists$ is similar to the definition of *must* transitions in three-valued abstractions [HJS01]. And while in terms of the valuations of abstract predicates the transitions of $Player_\exists$ resemble *must hyper-transitions* [SG04], in terms of the actual states of the abstract game the transitions of $Player_\exists$ are essentially must-transitions. Hence, we have to address the same problems concerning the monotonicity of abstraction refinement.

Example 5.1.1. This example illustrates the problem explained above. Consider a symbolic game structure $\mathcal{G}$ with $V_\exists = \{x_\exists\}$, $V_\forall = \{x_\forall^o\}$, where $V_\forall^o = \{x_\forall^o\}$.

Let $\mathcal{P} = \{t = \forall, t = \exists, x_\exists = 0, x_\exists = 1, x_\exists = 2, x_\forall^o > 0\}$ and $\mathcal{P}' = \mathcal{P} \cup \{x_\forall^o = x_\exists\}$.

Since all variables and hence all abstraction predicates are observable we can interpret each abstract state as a valuation of the abstraction predicates.

Figure 5.1 depicts on the top an abstract state in $\mathsf{Abstract}(\mathcal{G}, \mathcal{P})$ that belongs to $Player_\exists$ and has two successors. The corresponding state in $\mathsf{Abstract}(\mathcal{G}, \mathcal{P}')$, shown in the middle, has no successors. If however we consider the set of predicates $\mathcal{P}'' = \mathcal{P}' \cup \{x_\forall^o = 1, x_\forall^o = 2\}$, then the states in $\mathsf{Abstract}(\mathcal{G}, \mathcal{P}'')$ resulting from splitting this state, one of which is shown in the bottom of Figure 5.1, have their respective successors.

Must hyper-transitions proposed in [SG04] are used to obtain a monotonic abstraction-refinement framework for CTL, at the price of making the number of transitions exponential in the number of abstract states. Here we take a different approach by making use of two properties of the symbolic game structures defined in Chapter 2. Let $\mathcal{P} \subseteq \mathcal{P}'$

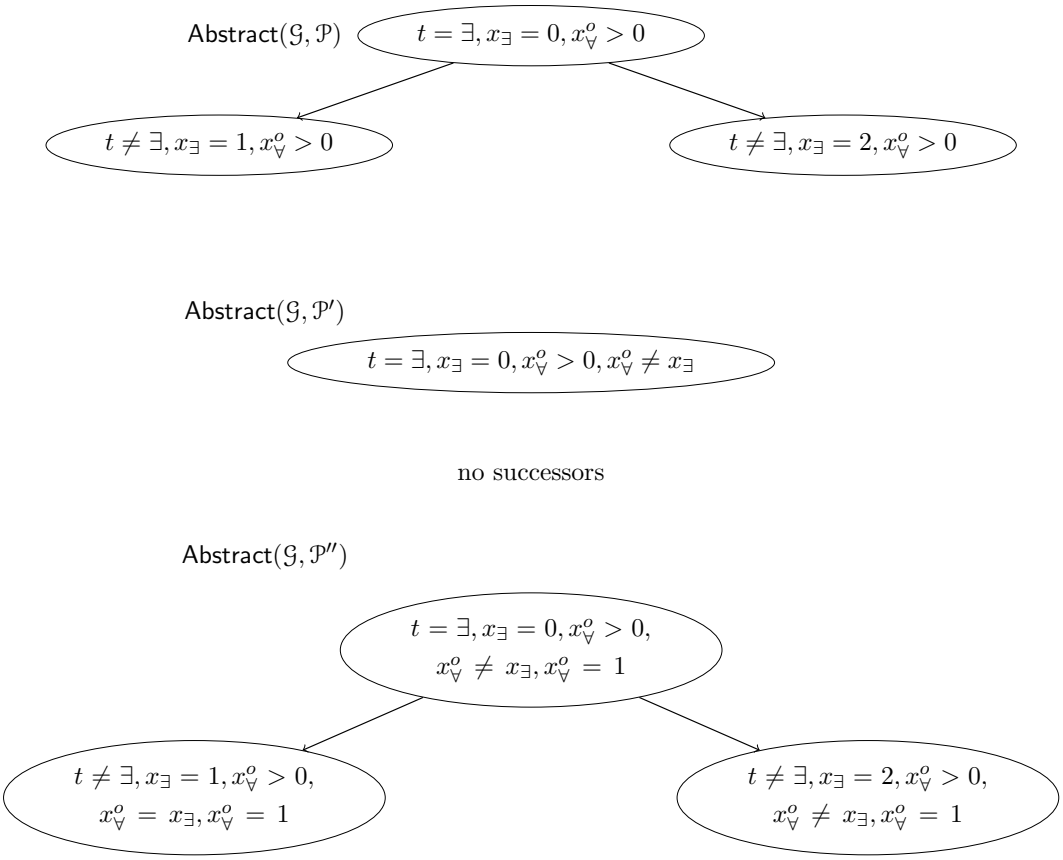


Figure 5.1: States in abstract game structures w.r.t. sets of predicates $\mathcal{P} \subset \mathcal{P}' \subset \mathcal{P}''$.

be two sets of predicates. Recall that the way in which the two players take turns is such that both the successors and the predecessors of a state in $S_\exists$ (respectively $S^\#_\exists$) are in $S_\forall$ (respectively $S^\#_\forall$). Thus, if the states in $S^\#_\exists$ are split with additional predicates in order to ensure that all concrete $Player_\exists$ transitions that were allowed by $\mathsf{Abstract}(\mathcal{G}, \mathcal{P})$ are also allowed by $\mathsf{Abstract}(\mathcal{G}, \mathcal{P}')$, the split does not have to be iterated and propagated. It suffices to split $Player_\exists$ states with predicates occurring in $\varphi = \bigvee_{\sigma \in \Sigma_\exists} \mathsf{Pre}_\exists([a], \sigma)$ for each $a \in Vals(Obs(\mathcal{P}'))$. Since the free variables of the formula $[a]$ are among $Obs(V)$, by the properties of $\mathcal{T}_\exists$ we have that $\varphi \equiv \bigvee_{\sigma \in \Sigma_\exists} [a]\eta_\sigma$, where the substitution η_σ is $\eta_\sigma = \{\sigma(x)/x' \mid x \in V_\exists \cup \{t\}\}$. Thus, the predicates occurring in φ are observable, and thus actually split the states that belong to $Player_\exists$.

Note that, furthermore, by the definition of $\mathsf{Abstract}(\mathcal{G}, \mathcal{P})$ using the set of predicates $\mathcal{P}_\exists$, if $\mathcal{G}$ does not contain dead-ends, neither does the resulting abstract game structure.

Example 5.1.2. Let us go back to Example 2.1.1 and consider the set of predicates consisting of the atomic formulas occurring in φ_{Err} and the predicates describing the possible actions of $Player_\exists$. Formally, let

$$\begin{aligned}\mathcal{P}_0 \;=\; & \{(t = \forall), (move = N), (move = E), (move = S),\\ & (x < 6), (x \geq 9), (x \leq -1), (steps > 3), (err = true), (y \geq 4), (y \leq -4)\}.\end{aligned}$$

Each of the predicates $(t = \forall), (move = 0), (move = N), (move = E), (move = S), (x < 6), (x \geq 9), (x \leq -1)$ is observable since only observable variables occur in it. The remaining predicates are not observable by $Player_\exists$.

Valuations are boolean vectors giving a value for each predicate. Two possible valuations are $a_0 = (0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0)$ and $a_1 = (0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0)$, where the values for the predicates in $\mathcal{P}_0$ are given in the same order in which they are listed above. In an abstract state, all valuations must agree on the values of the observable predicates. Since a_0 and a_1 differ only in the value of the unobservable predicate $(y \geq 4)$, $A = \{a_0, a_1\}$ is a possible abstract state, which belongs to $Player_\exists$. The set of concrete states corresponding to A is the set of states that satisfy the formula:

$$\begin{aligned}[A] \;=\; & (t \neq \forall) \wedge (move = N) \wedge (move \neq E) \wedge (move \neq S) \wedge\\ & (x < 6) \wedge (x < 9) \wedge (x > -1) \wedge (steps \leq 3) \wedge (err = false) \wedge (y > -4).\end{aligned}$$

□

We now establish that the definition above indeed yields an abstraction of the explicit game structure $G = (S_\exists, S_\forall, I, =_o, \Sigma_\exists, T_\exists, T_\forall)$ represented by $\mathcal{G}$.

Proposition 5.1.1. *Let* $\mathcal{G} = (V_\exists, V_\forall, V_\forall^o, t, \varphi_{Init}, \mathcal{T}_\exists, \mathcal{T}_\forall)$ *be a symbolic game structure with incomplete information with set of variables* $V = V_\exists \dot\cup V_\forall \dot\cup \{t\}$ *and let* $\mathcal{P} \subseteq \mathcal{AP}[V]$ *be a finite set of predicates that is precise w.r.t. the variable* t*. Then the game structure* $\mathsf{Abstract}(\mathcal{G}, \mathcal{P}) = (S_\exists^\#, S_\forall^\#, I^\#, =_\mathsf{o}^\#, \Sigma_\exists^\#, T_\exists^\#, T_\forall^\#)$ *is an abstraction of the explicit game structure* $G = (S_\exists, S_\forall, I, =_\mathsf{o}, \Sigma_\exists, T_\exists, T_\forall)$ *that is defined by* $\mathcal{G}$*.*

Proof. We show that $\mathsf{Abstract}(\mathcal{G}, \mathcal{P})$ together with the concretization functions defined above satisfies the conditions from Definition 2.3.1. Conditions (i), (ii) and (iv) hold since each of $Vals(\mathcal{P})$ and $Vals(\mathcal{P} \cup \mathcal{P}_\exists)$ partition $Vals(V)$ and $Vals(Obs(\mathcal{P}))$ partitions $Vals(Obs(V))$. Condition (iii) is implied by the definitions of $S_\exists^\#$ and $S_\forall^\#$. Condition (v) follows from the definition of $T_\exists^\#$, and condition (vi) follows from the definition of $T_\forall^\#$. Finally, condition (vii) is implied by the definitions of $Obs_\exists$ and $Obs_\forall$. □

5.1.3 Soundness of Predicate Abstraction

An abstraction of a game structure by Definition 2.3.1 is not necessarily sound. We now prove the soundness of the abstraction defined above. The key argument is that the abstract game structure simulates, in the sense of an alternating simulation relation, the knowledge-based game structure for the concrete one. This is implied by Properties 11 and 12 below, which follow directly from the definition of the abstract game and the properties of symbolic game structures.

Definition 5.1.1. For a path $\pi^\#$ in $\mathsf{Abstract}(\mathcal{G}, \mathcal{P})$, the *concretization* of $\pi^\#$ in G is

$$\gamma(\pi) = \{\pi \in S^* \mid \pi \text{ is a path in } G, |\pi| = |\pi^\#|, \forall i.\ 0 \leq i < |\pi| \Rightarrow \pi[i] \in \gamma(\pi^\#[i])\}.$$

Theorem 5.1.1. *Let* $\mathsf{Safety}(\mathcal{G}, \varphi_{Err})$ *be a safety game under incomplete information with symbolic game structure* $\mathcal{G} = (V_\exists, V_\forall, V_\forall^o, t, \varphi_{Init}, \mathcal{T}_\exists, \mathcal{T}_\forall)$ *and* $V = V_\exists \dot\cup V_\forall \dot\cup \{t\}$*. Let* $\mathcal{P} \subseteq \mathcal{AP}[V]$ *be a finite set of predicates that is precise w.r.t.* t*, and let* $G^\# = \mathsf{Abstract}(\mathcal{G}, \mathcal{P}) = (S_\exists^\#, S_\forall^\#, I^\#, =_\mathsf{o}^\#, \Sigma_\exists^\#, T_\exists^\#, T_\forall^\#)$*. If* $Err^\# = \{s^\# \in S^\# \mid [\![s^\#]\!] \cap [\![\varphi_{Err}]\!] \neq \emptyset\}$*, then* $(\mathsf{Safety}(G^\#, Err^\#), obs_s^\#)$ *is a sound abstraction of* $(\mathsf{Safety}(G, Err), obs_s)$*.*

Proof. By definition, the game structure $\mathsf{Abstract}(\mathcal{G}, \mathcal{P})$ has the following properties.

Property 11. *If* $s_1^\# \in S^\#$ *and* $s_1 \in S$ *are such that* $s_1 \in [\![s_1^\#]\!]$ *and* $(s_1, s_2) \in T_\forall$ *for some* $s_2 \in S$*, then there exists* $s_2^\# \in S^\#$ *such that* $(s_1^\#, s_2^\#) \in T_\forall^\#$ *and* $s_2 \in [\![s_2^\#]\!]$*. Furthermore, if* $\widetilde{s}_1 \in [\![s_1^\#]\!]$ *is such that* $\widetilde{s}_1 =_\mathsf{o} s_1$ *and* $\widetilde{s}_2 \in S$ *is such that* $(\widetilde{s}_1, \widetilde{s}_2) \in T_\forall$ *and* $\widetilde{s}_2 =_\mathsf{o} s_2$*, then it also holds that* $\widetilde{s}_2 \in [\![s_2^\#]\!]$*.*

Property 12. *If $s_1^\# \in S^\#$ and $s_2^\# \in S^\#$ are such that $(s_1^\#, \sigma^\#, s_2^\#) \in T_\exists^\#$ for some $\sigma^\# \in \Sigma^\#$, then for each $s_1 \in [\![s_1^\#]\!]$ there exist $\sigma \in \gamma_\exists(\sigma^\#)$ and $s_2 \in [\![s_2^\#]\!]$ such that $(s_1, \sigma, s_2) \in T_\exists$. Furthermore, for every $\widetilde{s}_1 \in [\![s_1]\!]$ with $\widetilde{s}_1 =_\mathsf{o} s_1$ it holds that there exists a $\widetilde{s}_2 \in [\![s_2]\!]$ such that $(\widetilde{s}_1, \sigma, \widetilde{s}_2) \in T_\exists$ and $\widetilde{s}_2 =_\mathsf{o} s_2$.*

The following property follows from the definition of $\mathsf{Abstract}(\mathcal{G}, \mathcal{P})$ and Property 10.

Property 13. *If $s_1^\# \in S^\#$ and $s_1 \in S$ are such that $s_1 \in [\![s_1^\#]\!]$ and $(s_1, \sigma, s_2) \in T_\exists$ for some $\sigma \in \Sigma_\exists$ and $s_2 \in S$, then there exists $s_2^\# \in S^\#$ and $\sigma^\# \in \Sigma^\#$ such that $(s_1^\#, \sigma^\#, s_2^\#) \in T_\exists^\#$, $s_2 \in [\![s_2^\#]\!]$ and $\sigma \in \gamma_\exists(\sigma^\#)$. Furthermore, if $\widetilde{s}_1 \in [\![s_1^\#]\!]$ is such that $\widetilde{s}_1 =_\mathsf{o} s_1$ and $\widetilde{s}_2 \in S$ and $\widetilde{\sigma} \in \Sigma_\exists$ are such that $(\widetilde{s}_1, \widetilde{\sigma}, \widetilde{s}_2) \in T_\exists$ and $\widetilde{s}_2 =_\mathsf{o} s_2$, then it also holds that $\widetilde{s}_2 \in [\![s_2^\#]\!]$.*

Let $f_\exists^\#$ be a winning strategy for $Player_\exists$ in the game $\mathsf{Safety}(G^\#, Err^\#)$ (since the game is under perfect information, the set of all $Player_\exists$ strategies and the set of the $obs_s^\#$-consistent ones coincide). We define a strategy $f_\exists$ for $Player_\exists$ in G as follows.

Let $[\mathsf{Prefs}(G)]_{=_s}$ be the set of equivalence classes of prefixes in G w.r.t. the equivalence relation $=_s$. It can easily be seen by induction on the length of the prefix and using Properties 11 and 13 that for each $\Pi \in [\mathsf{Prefs}(G)]_{=_s}$ there exists a unique prefix $\pi^\# \in \mathsf{Prefs}(G^\#)$ such that $\Pi \subseteq \gamma(\pi)$. Let $\Pi \in [\mathsf{Prefs}(G)]_{=_s}$ be such that $\Pi \subseteq \mathsf{Prefs}_\exists(G)$ and let $\pi^\#$ be the corresponding prefix in $G^\#$ such that $\Pi \subseteq \gamma(\pi^\#)$. If $\pi^\# \in \mathsf{Prefs}(f_\exists^\#)$, then there exist $\sigma^\#$ and $s^\#$ such that $(\mathsf{last}(\pi^\#), \sigma^\#, s^\#) \in T_\exists^\#$ and $\pi^\# \cdot s^\# \in \mathsf{Prefs}(f_\exists^\#)$. According to Property 12 there exists a $\sigma \in \gamma_\exists(\sigma)$ such that for every $\pi \in \Pi$, $\sigma \in \mathsf{Enabled}(\mathsf{last}(\pi))$ and for $s \in S$ with $(\mathsf{last}(\pi), \sigma, s) \in T_\exists$ it holds that $s \in [\![s^\#]\!]$. In this case we fix one such σ and define $f_\exists(\pi) = \sigma$ for each $\pi \in \Pi$. In the case when $\pi^\# \notin \mathsf{Prefs}(f_\exists^\#)$, we choose an arbitrary $\sigma \in \Sigma_\exists$ such that $\sigma \in \mathsf{Enabled}(\mathsf{last}(\pi))$ for each $\pi \in \Pi$ and define $f_\exists(\pi) = \sigma$, if such a σ exists, and otherwise $f_\exists(\pi)$ is undefined.

By definition, the function $f_\exists$ is a strategy for $Player_\exists$ and is obs_s-consistent.

Using the definition of $f_\exists$ and Properties 11, 13 and 12 we can see that for each play $\pi \in \mathsf{Outcome}(f_\exists)$, π is infinite and there exists a play $\pi^\# \in \mathsf{Outcome}(f_\exists^\#)$ such that $\pi \in \gamma(\pi^\#)$. Thus, the strategy $f_\exists$ is winning for $Player_\exists$ in $\mathsf{Safety}(\mathcal{G}, \varphi_{Err})$. □

5.1.4 From Abstract Strategies to Finite-State Concrete Strategies

Since $\mathsf{Safety}(G^\#, Err^\#)$ is a finite-state game with perfect information, if $Player_\exists$ has a winning strategy, then he has a memoryless winning strategy. Suppose that $f_\exists^\# : S_\exists^\# \to \Sigma^\#$ is a memoryless winning strategy for $Player_\exists$ in $\mathsf{Safety}(G^\#, Err^\#)$.

Let for each $s^\# \in S^\#$, $o(s)$ be the unique element of $Obs_\exists \cup Obs_\forall$ such that $s^\# \subseteq o$.

We define the function $\sigma^{\#}_{fix} : Obs_{\exists} \cup Obs_{\forall} \to \Sigma^{\#} \cup \{\bot\}$ as follows. For $o \in Obs_{\forall}$, $\sigma^{\#}_{fix}(o) = \bot$. For $o \in Obs_{\exists}$, $\sigma^{\#}_{fix}(o) = \bot$ if for every $\sigma \in \Sigma_{\exists}$ it holds that $[o] \wedge \mathsf{Enabled}(\sigma)$ is unsatisfiable and $\sigma^{\#}_{fix}(o) = \sigma^{\#}$ otherwise, for some arbitrarily fixed $\sigma^{\#} \in \Sigma^{\#}$ that is such that $[o] \Rightarrow \bigvee_{\sigma \in \gamma_{\exists}(\sigma^{\#})} \mathsf{Enabled}(\sigma)$ is valid.

We extend $f^{\#}_{\exists}$ to a total function $f^{\bot}_{\exists} : S^{\#} \to \Sigma^{\#} \cup \{\bot\}$ such that if $s^{\#} \in S^{\#}_{\exists}$ and $f^{\#}_{\exists}(s^{\#})$ is defined, then $f^{\bot}_{\exists}(s^{\#}) = f^{\#}_{\exists}(s^{\#})$, and otherwise $f^{\bot}_{\exists}(s^{\#}) = \sigma^{\#}_{fix}(o(s^{\#}))$.

We define a partial function $A : (Obs_{\exists} \cup Obs_{\forall}) \times (\Sigma^{\#} \cup \{\bot\}) \to 2^{(\mathcal{LF}[Obs(V)] \times \Sigma_{\exists})}$. For each $o \in Obs_{\exists} \cup Obs_{\forall}$, $A(o, \bot) = \{([o], \bot)\}$. For each $o \in Obs_{\exists}$ and $\sigma^{\#} \in \Sigma^{\#}$ that are such that $[o] \Rightarrow \bigvee_{\sigma \in \gamma_{\exists}(\sigma^{\#})} \mathsf{Enabled}(\sigma)$ is valid, we define $A(o, \sigma^{\#}) = \{([o] \wedge \varphi_0, \sigma_0), \ldots, ([o] \wedge \varphi_n, \sigma_n)\}$, where $\sigma_i \in \gamma_{\exists}(\sigma^{\#})$ for each $0 \leq i \leq n$, and the formulas $\varphi_0, \ldots, \varphi_n$ are such that: For each $0 \leq i \leq n$, the formula $\varphi_i \Rightarrow \mathsf{Enabled}(\sigma_i)$ is valid; the formula $[o] \Rightarrow \bigvee_{i=0}^{n} \varphi_i$ is valid; for each $0 \leq i < j \leq n$, $\varphi_i \wedge \varphi_j$ is unsatisfiable.

We use the function A to define the labels of the transitions in a semi-symbolic strategy automaton constructed from the abstract strategy $f^{\#}_{\exists}$. Let $O^{\#} = Obs_{\exists} \cup Obs_{\forall}$.

We define the semi-symbolic strategy automaton $\mathcal{M} = (Q^0_q, \mathcal{LF}[Obs(V)] \times \Sigma^{\bot}_{\exists}, \rho)$ that represents a winning strategy $f_{\exists}$ for $Player_{\exists}$ in G. The set of states is $Q = \{q^0, q_{\bot}\} \dot{\cup} S^{\#}$ where q^0 is the initial state and $q_{\bot}$ is a sink. The transition relation is

$$
\begin{array}{rcl}
\rho & = & \{(q^0, a, q) \mid q \in I^{\#}, a \in A(o(q), f^{\bot}_{\exists}(q))\} \\
 & \cup & \{(q, a, q') \mid q \in S^{\#}, q' \in S^{\#}, q' \in \mathsf{Post}(q), a \in A(o(q'), f^{\bot}_{\exists}(q'))\} \\
 & \cup & \{(q^0, a, q_{\bot}) \mid \exists o \in O^{\#}.\ a \in A(o, \sigma^{\#}_{fix}(o)) \wedge \forall q \in I^{\#}.\ o(q) \neq o\} \\
 & \cup & \{(q, a, q_{\bot}) \mid q \in S^{\#} \wedge \exists o \in O^{\#}.\ a \in A(o, \sigma^{\#}_{fix}(o)) \wedge \forall q' \in \mathsf{Post}(q).\ o(q') \neq o\} \\
 & \cup & \{(q_{\bot}, a, q_{\bot}) \mid \exists o \in O^{\#}.\ a \in A(o, \sigma^{\#}_{fix}(o))\}.
\end{array}
$$

The semi-symbolic strategy automaton $\mathcal{M}$ is finite. By definition, since $f^{\#}_{\exists}$ is a strategy for $Player_{\exists}$ in $G^{\#}$, $\mathcal{M}$ is non-blocking and $\Sigma_{\exists}$-correct. Thus, $\mathcal{M}$ is a finite representation of a strategy $f_{\exists}$ for $Player_{\exists}$ in $\mathcal{G}$. Since $f^{\#}_{\exists}$ is winning for $Player_{\exists}$ in $\mathsf{Safety}(G^{\#}, Err^{\#})$, it is easy to see that $f_{\exists}$ is winning for $Player_{\exists}$ in $\mathsf{Safety}(\mathcal{G}, \varphi_{Err})$.

5.2 Counterexample Tree Analysis

Since $\mathsf{Safety}(G^{\#}, Err^{\#})$ is a safety game with perfect information, by Proposition 2.2.1, if $Player_{\exists}$ does not have a winning strategy in $\mathsf{Safety}(G^{\#}, Err^{\#})$, there exists a counterexample tree $C^{\#}_t$ in $\mathsf{Safety}(G^{\#}, Err^{\#})$, which we call *abstract counterexample tree*.

An abstract counterexample may correspond to a counterexample in the the concrete game, i.e., be *concretizable*, or be an artifact of the abstraction. We give a characterization of concretizable counterexamples that allows us to effectively check if a counterexample is concretizable by using decision procedures for the logical theory.

5.2.1 Counterexample Concretization

In Section 2.2 we established that in safety games under incomplete information it suffices to consider knowledge-based counterexample trees as counterexamples. Thus, we define the concretization of an abstract counterexample tree $C_t^{\#}$ to be the (possibly empty) set of corresponding knowledge-based counterexample trees.

Definition 5.2.1. Given an abstract counterexample tree $C_t^{\#} = (N^{\#}, E^{\#}, {L_s}^{\#}, {L_a}^{\#})$ in $\mathsf{Safety}(G^{\#}, Err^{\#})$, the concretization $\gamma(C_t^{\#})$ of $C_t^{\#}$ is the smallest set that contains each knowledge-based counterexample tree $C_k = (N, E, K_s, L_a)$ in $\mathsf{Safety}(G, Err)$ that is such that: for each path $n_0 n_1 \dots n_m$ in C_k with n_0 being the root node, there exists a path $n_0^{\#} n_1^{\#} \dots n_m^{\#}$ in $C_t^{\#}$ where $n_0^{\#}$ is the root of $C_t^{\#}$, such that for each $0 \leq i \leq m$ it holds that $K_s(n_i) \subseteq \gamma({L_s}^{\#}(n_i^{\#}))$ and for each $0 \leq i \leq m-1$ it holds that either $L_a(n_i, n_{i+1}) = {L_a}^{\#}(n_i^{\#}, n_{i+1}^{\#}) = \epsilon$ or $L_a(n_i, n_{i+1}) \in \gamma_{\exists}({L_a}^{\#}(n_i^{\#}, n_{i+1}^{\#}))$.

An abstract counterexample tree $C_t^{\#}$ is *genuine* iff $\gamma(C_t^{\#}) \neq \emptyset$, and otherwise it is called *spurious*. We now show how to construct a *tree formula* that is satisfiable iff the abstract counterexample tree is genuine. The key idea is to symbolically simulate the perfect-information game over the equivalence classes of the prefixes of the concrete game structure G with incomplete information. The formula is constructed as a conjunction of a set of *trace formulas* for a finite set of traces defined by the abstract counterexample tree $C_t^{\#}$, where a trace is a finite sequence of actions of $Player_{\exists}$.

5.2.2 Trace formulas

The set of traces for an abstract counterexample tree consists of the finite sequences of $Player_{\exists}$ actions for which there should be a corresponding path to a leaf node in each counterexample tree in the concretization. Below we make this intuition precise.

Traces and counterexample paths. With each node n in $C_t^{\#}$, we associate a set $Traces(n)$ of traces, where a *trace* is a finite sequence $\tau \in \Sigma_{\exists}^*$ of actions of $Player_{\exists}$.

For each node n in $C_t^{\#}$, the set $Traces(n)$ is recursively defined as follows:

- If n is a leaf node with ${L_s}^{\#}(n) \in Err^{\#}$, then $\mathit{Traces}(n) = \{\epsilon\}$,
- If n is a leaf node with ${L_s}^{\#}(n) \in S^{\#}_{\exists} \setminus Err^{\#}$, then $\mathit{Traces}(n) = \Sigma_{\exists}$,
- If n is a leaf node with ${L_s}^{\#}(n) \in S^{\#}_{\forall} \setminus Err^{\#}$, then $\mathit{Traces}(n) = \{\epsilon\}$,
- If n is an internal node with ${L_s}^{\#}(n) \in S^{\#}_{\exists}$, then

$$\begin{array}{rcl} \mathit{Traces}(n) & = & \{\sigma \cdot \tau \mid \exists n' \in \mathit{Children}(n).\ \sigma \in \gamma_{\exists}({L_a}^{\#}(n,n')) \wedge \tau \in \mathit{Traces}(n')\} \cup \\ & & \{\sigma \in \Sigma_{\exists} \mid \forall n' \in \mathit{Children}(n).\ \sigma \notin \gamma_{\exists}({L_a}^{\#}(n,n'))\}, \end{array}$$

- If n is an internal node with ${L_s}^{\#}(n) \in S^{\#}_{\forall}$ and n' is the child of n in $C^{\#}_t$, then

$$\mathit{Traces}(n) = \mathit{Traces}(n').$$

Each trace τ induces a set of concrete counterexample paths in $\mathsf{Safety}(G, Err)$. A finite path $\pi = s_0 s_1 \dots s_m$ in the concrete game structure G is a *concrete counterexample path for the trace* $\tau = \sigma_1 \dots \sigma_k$ iff $s_{i_1} s_{i_2} \dots s_{i_l}$ with $l \leq k$ is the sequence consisting of all $\mathit{Player}_{\exists}$-states from π in the same order and the following conditions are satisfied:

- for every $1 \leq j \leq l$ with $i_j < m$, $(s_{i_j}, \sigma_j, s_{i_j+1}) \in T_{\exists}$, and
- $s_m \in Err$ or $s_m \in S_{\exists}$ and $\sigma_l \notin \mathsf{Enabled}(s_m)$.

Trace formulas for $C^{\#}_t$. A path $\rho = n_0 n_1 \dots n_m$ in $C^{\#}_t$ is an *abstract counterexample path for the trace* $\tau = \sigma_1 \dots \sigma_k$ iff $n_{i_1} n_{i_2} \dots n_{i_l}$ with $l \leq k$ is the sequence consisting of all $\mathit{Player}_{\exists}$-nodes in ρ in the same order and the following conditions are satisfied:

- for every $1 \leq j \leq l$ with $i_j < m$, $\sigma_l \in \gamma_{\exists}({L_a}^{\#}(n_{i_j}, n_{i_j+1}))$, and
- ${L_s}^{\#}(n_m) \in Err^{\#}$ or ${L_s}^{\#}(n_m) \in S^{\#}_{\exists}$ and $\sigma_l \notin \gamma_{\exists}({L_a}^{\#}(n_m, n'))$ for all $(n, n') \in E^{\#}$.

For a trace $\tau = \sigma_1 \dots \sigma_k$ and a path $\rho = n_0 n_1 \dots n_m$ in $C^{\#}_t$ that is an abstract error path for τ with l $\mathit{Player}_{\exists}$ nodes, we define the formula $\varphi_{\mathit{path}}(\tau, \rho)$, which characterizes the set of concrete error paths for τ corresponding to ρ, as follows:

$$\begin{array}{rcl} \varphi_{\mathit{path}}(\tau,\rho)[V^{n_0}, \dots, V^{n_m}] & = & \big(\bigwedge_{i=0}^{m} [{L_s}^{\#}(n_i)][V^{n_i}/V]\big) \wedge \\ & & \big(\bigwedge_{i<m, {L_s}^{\#}(n_i) \in S^{\#}_{\exists}} \mathcal{T}_{\exists}[V^{n_i}/V,\ V^{n_{i+1}}/V']\big) \wedge \\ & & \big(\bigwedge_{i<m, {L_s}^{\#}(n_i) \in S^{\#}_{\forall}} \mathcal{T}_{\forall}[V^{n_i}/V,\ V^{n_{i+1}}/V']\big) \wedge \\ & & \big(Err \vee (t = \exists \wedge \neg \mathsf{Enabled}(\sigma_l))\big)[V^{n_m}/V]. \end{array}$$

Let n_0 be the root node of $C_t^{\#}$ and let $\tau \in \mathit{Traces}(n_0)$. We denote with $\mathit{Paths}(\tau)$ the set of all abstract counterexample paths ρ for τ such that $\rho[0] = n_0$. The *trace formula* for τ is then defined as $\varphi_{\mathit{trace}}(\tau) = \varphi_{\mathit{Init}}[V^{n_0}/V] \wedge \bigvee_{\rho \in \mathit{Paths}(\tau)} \varphi_{\mathit{path}}(\tau, \rho)$.

By the definition of the formula $\varphi_{\mathit{trace}}(\tau)$, if $\varphi_{\mathit{trace}}(\tau)$ is satisfiable then there exists a prefix $\pi \in \mathsf{Prefs}(G)$ such that $\pi \models \varphi_{\mathit{trace}}(\tau)$ and $|\pi|_\exists \leq |\tau|$.

5.2.3 Tree formula

We define $\mathit{Traces}(C_t^{\#}) = \mathit{Traces}(n_0)$, where n_0 is the root of $C_t^{\#}$. If the abstract counterexample tree $C_t^{\#}$ is genuine, then for each $\tau \in \mathit{Traces}(C_t^{\#})$, the knowledge-based counterexample tree in G should provide a concrete counterexample path π_τ. Since the knowledge-based counterexample tree branches according to the choices of $\mathit{Player}_\exists$, the paths corresponding to two different traces τ_1 and τ_2 may differ after the position corresponding to the first position in which τ_1 and τ_2 differ. Furthermore, unlike for counterexample trees in the perfect information case, the concrete counterexample paths for τ_1 and τ_2 here may differ even before this position. However, in order to ensure that the paths form a knowledge-based counterexample tree, the states at the same level must be equivalent with respect to $=_{\mathsf{o}}$ up to the position corresponding to the first difference between τ_1 and τ_2. We encode this in the *tree formula* that characterizes the set of knowledge-based counterexample trees in the concretization of $C_t^{\#}$ by appropriately indexing the state variables in the formula. The tree formula for $C_t^{\#}$ is built from the trace formulas for the traces in $\mathit{Traces}(C_t^{\#})$ after applying a variable substitution to each of them, as we describe in the next paragraph.

Definition 5.2.2. Let $\pi_1, \pi_2 \in \mathit{Paths}(G)$, $\tau_1, \tau_2 \in \Sigma_\exists^*$, $\tau_1 \neq \tau_2$, and $|\pi_1|_\exists \leq |\tau_1|$ and $|\pi_2|_\exists \leq |\tau_2|$. We define $\mathsf{diff}(\pi_1, \pi_2, \tau_1, \tau_2) = \min\{m_1, m_2\}$, where

$$m = \begin{cases} \min\{j \mid \tau_1[j] \neq \tau_2[j]\} & \text{if } \exists j.\ j < \min\{|\tau_1|, |\tau_2|\} \text{ and } \tau_1[j] \neq \tau_2[j] \\ \min\{|\tau_1|, |\tau_2|\} & \text{otherwise} \end{cases}$$

$$m_i = \begin{cases} j_m + 1 & \text{if } \pi_i|_\exists = s_{j_0} \dots s_{j_l} \text{ and } l \geq m, \\ |\pi_i| & \text{if } |\pi_i|_\exists \leq m. \end{cases}$$

Intuitively, if π_1 and π_2 are counterexample paths for the traces $\tau_1 \in \mathit{Traces}(C_t^{\#})$ and $\tau_2 \in \mathit{Traces}(C_t^{\#})$ respectively, $\mathsf{diff}(\pi_1, \pi_2, \tau_1, \tau_2)$ is index of the first position starting from which the states on the two paths are allowed to be observably different.

Indexing of variables. For each trace $\tau \in \mathit{Traces}(C_t^\#)$ the trace formula $\varphi_{\mathit{trace}}(\tau)$ characterizes the set of concrete counterexample paths $s_0 s_1 \ldots_m$ with $s_0 \in I$ for which there exists a path $n_0 n_1 \ldots n_m$ in $C_t^\#$ such that for every $0 \leq i \leq m$, $s_i \in \gamma({L_s}^\#(n_i))$. By definition $\varphi_{\mathit{trace}}(\tau) = \varphi_{\mathit{Init}}[V^{n_0}/V] \wedge \bigvee_{\rho \in \mathit{Paths}(\tau)} \varphi_{\mathit{path}}(\tau, \rho)$ and the variables encoding a concrete state in $\gamma({L_s}^\#(n))$ for some $n \in N^\#$ are indexed with n. Let $n \in N^\#$ be a node on some $\rho \in \mathit{Paths}(\tau)$. We define $\tau'_n = \tau[1, l]$ where l is the number of nodes n' on $\mathit{path}(n)$ such that $n' \neq n$ and ${L_s}^\#(n') \in S_\exists^\#$. Thus, $\tau'_n \in \Sigma^\#$ is the sequence of $\mathit{Player}_\exists$ choices leading to n on the corresponding concrete counterexample path for τ.

For a trace τ we define the substitution η_τ, that distinguishes between the observable $\mathit{Obs}(V) = V_\forall^o \,\dot\cup\, V_\exists \,\dot\cup\, \{t\}$ and the unobservable variables $V \setminus \mathit{Obs}(V) = V_\forall \setminus V_\forall^o$:

$$\begin{aligned} \eta_\tau \;=\; & \{x^{n,\tau'_n}/x^n \mid x \in V_\forall^o \cup V_\exists \cup \{t\}, n \text{ is a node on some } \rho \in \mathit{Paths}(\tau)\} \cup \\ & \{x^{n,\tau}/x^n \mid x \in V_\forall \setminus V_\forall^o, n \text{ is a node on some } \rho \in \mathit{Paths}(\tau)\}. \end{aligned}$$

Tree formula for $C_t^\#$. Since a knowledge-based counterexample tree that concretizes $C_t^\#$ needs to provide a concrete counterexample path for every trace $\tau \in \mathit{Traces}(C_t^\#)$, the tree formula $\mathsf{TF}(C_t^\#)$ is defined to be the conjunction of the trace formulas for the traces in $\mathit{Traces}(C_t^\#)$, to each of which the respective variable substitution was applied:

$$\mathsf{TF}(C_t^\#) = \bigwedge_{\tau \in \mathit{Traces}(C_t^\#)} \big(\varphi_{\mathit{trace}}(\tau)\eta_\tau\big). \tag{5.3}$$

5.2.4 Concretizability Characterization

The theorem below states that the formula 5.3 characterizes the concretizability of $C_t^\#$.

Theorem 5.2.1. *Let* $\mathsf{Safety}(\mathcal{G}, \varphi_{Err})$ *be a safety game under incomplete information,* $\mathcal{P} \subseteq \mathcal{AP}[V]$ *be a finite set of predicates that is precise w.r.t. the turn variable* t *and let* $G^\# = \mathsf{Abstract}(\mathcal{G}, \mathcal{P})$ *and* $Err^\# = \{s^\# \in S^\# \mid [\![s^\#]\!] \cap [\![\varphi_{Err}]\!] \neq \emptyset\}$*. If* $C_t^\#$ *is an abstract counterexample tree in* $\mathsf{Safety}(G^\#, Err^\#)$*, then* $\gamma(C_t^\#) \neq \emptyset$ *iff* $\mathsf{TF}(C_t^\#)$ *is satisfiable.*

Proof. First, consider the case when the formula $\mathsf{TF}(C_t^\#)$ is satisfiable, and let M be a model of $\mathsf{TF}(C_t^\#)$. For a node $n \in N^\#$ and $\tau \in \mathit{Traces}(C_t^\#)$ we define $\tau_n = \tau[1, l]$ where l is the number of nodes n' on $\mathit{path}(n)$ such that $n' \neq n$ and ${L_s}^\#(n') \in S_\exists^\#$.

Let us define the valuation $s_{n,\tau} \in \mathit{Vals}(V)$ such that for $x \in V_\forall^o \cup V_\exists \cup \{t\}$, $s_{n,\tau}(x) = M(x^{n,\tau_n})$ and for $x \in V_\forall \setminus V_\forall^o$, $s_{n,\tau}(x) = M(x^{n,\tau})$. Note that by this,

$$\mathit{obs}(s_{n,\tau'}) = \mathit{obs}(s_{n,\tau''}) \text{ for every } \tau', \tau'' \in \mathit{Traces}(C_t^\#) \text{ for which } \tau'_n = \tau''_n. \tag{5.4}$$

We define the labeled tree $C_k = (N, E, K_s, L_a)$ with $N \subseteq N^{\#} \times \Sigma_{\exists}^*$ as the smallest graph that satisfies the following conditions:

- $(n_0^{\#}, \epsilon) \in N$ and $K_s((n_0^{\#}, \epsilon)) = I \cap obs(s_{n_0,\tau})$, where $n_0^{\#}$ is the root of $C_t^{\#}$ and $\tau \in \mathit{Traces}(C_t^{\#})$ (note that, by (5.4), the label of $(n_0^{\#}, \epsilon)$ is uniquely determined);

- if $(n, \tau) \in N$, $(n, n') \in E^{\#}$, $K_s((n, \tau)) \cap Err = \emptyset$, ${L_a}^{\#}(n, n') \in \Sigma_{\exists}^{\#}$ and $\sigma \in \gamma_{\exists}({L_a}^{\#}(n, n'))$, then let $S' = \mathsf{Post}_{\exists}(K_s((n, \tau)), \sigma) \cap obs(s_{n',\tau'})$ for some $\tau' = \tau \cdot \sigma \cdot \tau'' \in \mathit{Traces}(C_t^{\#})$, and if $S' \neq \emptyset$, then $(n', \tau \cdot \sigma) \in N$, $((n, \tau), (n', \tau \cdot \sigma)) \in E$ and $K_s((n', \tau \cdot \sigma)) = S'$ and $L_a(((n, \tau), (n', \tau \cdot \sigma))) = \sigma$;

- if $(n, \tau) \in N$, $K_s((n, \tau)) \cap Err = \emptyset$, $(n, n') \in E^{\#}$ and ${L_a}^{\#}(n, n') = \epsilon$, then let $S' = \mathsf{Post}_{\forall}(K_s(n, \tau)) \cap obs(s_{n',\tau'})$ for some $\tau' = \tau \cdot \tau'' \in \mathit{Traces}(C_t^{\#})$, and if $S' \neq \emptyset$, then $(n', \tau) \in N$, $((n, \tau), (n', \tau)) \in E$, $K_s((n', \tau)) = S'$, $L_a(((n, \tau), (n', \tau))) = \epsilon$.

Since $C_t^{\#}$ and $\Sigma_{\exists}$ are finite, C_k is finite as well. Conditions (i), (ii), (iv) and (v) from the definition of knowledge-based counterexample tree are implied by the definition of C_k. Conditions (iii) and (viii) hold by the fact that M is a model of $\mathsf{TF}(C_t^{\#})$ and that for each leaf n in $C_t^{\#}$ $\gamma(n)$ contains a state in Err or a dead end. Condition (vi) holds by (5.4) and the fact that if for states $s_1, s_2 \in S_{\exists}$ and $s_1', s_2' \in S$ and $\sigma \in \Sigma_{\exists}$ it holds that $obs(s_1) = obs(s_2)$ and $(s_1, \sigma, s_1') \in T_{\exists}$ and $(s_2, \sigma, s_2') \in T_{\exists}$, then $obs(s_1') = obs(s_2')$ for game structures defined symbolically (see Definition 2.1.9). Since $C_t^{\#}$ is a counterexample tree, C_k also satisfies condition (vii). Thus, C_k is a knowledge-based counterexample tree in the game $\mathsf{Safety}(G, Err)$.

Since M is a model of $\mathsf{TF}(C_t^{\#})$, the definition of C_k also implies that $C_k \in \gamma(C_t^{\#})$.

Now, for the other direction, suppose that $\gamma(C_t^{\#}) \neq \emptyset$ and let $C_k \in \gamma(C_t^{\#})$. Since C_k is a knowledge-based counterexample tree, for each trace $\tau \in \mathit{Traces}(C_t^{\#})$ there exists a concrete counterexample path $s_0 \dots s_m$ in G and a path $n_0 \dots n_m$ in C_k such that $s_i \in K_s(n_i)$ for each $0 \leq i \leq n$. By the properties of C_k it holds that we can use the set of concrete counterexample paths to define a model M for $\mathsf{TF}(C_t^{\#})$.

□

Example 5.2.1. Let us consider again Example 2.1.1 and the abstract game structure defined by the set of predicates $\mathcal{P}_0$ given in Example 5.1.2. Figure 5.2 shows an abstract counterexample tree in the abstract game $\mathsf{Safety}(\mathsf{Abstract}(\mathcal{G}), \varphi_{Err})$.

Each node i in the counterexample tree shown in Figure 5.2 is labeled by (an abstract state A_i and) a formula $[A_i]$ over $\mathcal{P}_0$ describing the set $\gamma(A_i)$ of concrete states in the game structure $\mathcal{G}$ for the corresponding abstract state A_i.

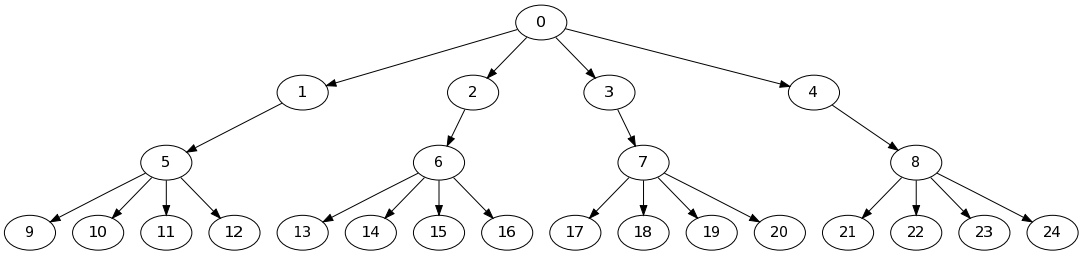


Figure 5.2: Counterexample tree in $\mathsf{Safety}(\mathsf{Abstract}(\mathcal{G}, \mathcal{P}), \varphi_{Err})$. The root is labeled with an abstract initial state, the leaves are labeled with abstract error states. Nodes $0, 5, 6, 7, 8$ belong to $Player_{\exists}$, the remaining nodes belong to $Player_{\forall}$.

$$
\begin{array}{lcllllllll}
[A_0] & = & (t \neq \forall) & \wedge & (move = N) & \wedge & (x < 6) & \wedge & (x < 9) & \wedge & (x > -1) \\
 & & & \wedge & (steps \leq 3) & \wedge & (err = false) & \wedge & (y < 4) & \wedge & (y > -4) \\
[A_1] & = & (t = \forall) & \wedge & (move = N) & \wedge & (x < 6) & \wedge & (x < 9) & \wedge & (x > -1) \\
 & & & \wedge & (steps \leq 3) & \wedge & (err = false) & \wedge & (y < 4) & \wedge & (y > -4) \\
\ldots & & \ldots & & & & & & & &
\end{array}
$$

For the root node 0 we have the set of traces $Traces(0) = \{N \cdot E, N \cdot S, N \cdot W, N \cdot N, E \cdot E, E \cdot S, E \cdot W, E \cdot N, S \cdot E, S \cdot S, S \cdot W, S \cdot N, W \cdot E, W \cdot S, W \cdot W, W \cdot N\}$.

For the trace $\tau = N \cdot N$ we have $Paths(\tau) = \{0 \cdot 1 \cdot 5 \cdot 9\}$. The corresponding formula $\varphi_{trace}(\tau)$ is not satisfiable and hence the abstract counterexample tree depicted in Figure 5.2 is not concretizable in $\mathsf{Safety}(\mathcal{G}, \varphi_{Err})$. The cause for this abstract counterexample is that the abstraction of the transition relation is not precise enough.

5.2.5 Sources of Spuriousness

The presented characterization of concretizable abstract counterexample trees allows us to easily identify, in the case of a spurious counterexample tree $C_t^{\#}$, the reason for the presence of the analyzed counterexample. In the CEGAR procedure that we describe in Section 5.4 we distinguish two possibilities. If each of the trace formulas in the conjunction $\mathsf{TF}(C_t^{\#})$ is satisfiable, this means that the the approximation of the observation equivalence defined by $Obs(\mathcal{P})$ is too coarse. If, on the other hand, for some trace τ the trace formula $\varphi_{trace}(\tau)$ is unsatisfiable, the cause of the abstract counterexample is the approximation of the transition relations on the level of $Vals(\mathcal{P})$.

In the next section we focus on the first of these two cases and develop a procedure for generating observable abstraction predicates.

5.3 Interpolation for Observation Refinement

In the case when each of the conjuncts in the unsatisfiable tree formula $\mathsf{TF}(C_t^{\#})$ for the abstract counterexample tree $C_t^{\#}$ is satisfiable, the goal of the refinement procedure is to compute a set of *observable* refinement predicates that allow to distinguish the sets of concrete counterexample paths for the traces in $\mathit{Traces}(C_t^{\#})$. Furthermore, the computed predicates should be state predicates. That is, they should refer only to the current values of the variables in individual states of a path and should not relate different states on the path. We call such predicates *localized*.

One of the techniques most widely and successfully used in verification [HJMM04] for generating refinement predicates is *Craig interpolation* [Cra57]. The main idea of this methodology is to extract the relevant information, in the form of a formula called *interpolant* from the proof of unsatisfiability of a formula characterizing the concretizability of a counterexample path. There, the path formula is split into a conjunction of two formulas – a formula describing a prefix of the path and a formula describing the corresponding suffix, and the two formulas share variables representing a single state. Therefore, the predicates occurring in the interpolant computed for the two formulas relate only current values of the variables of the analyzed system.

Here, since each pair of conjuncts of a tree formula $\mathsf{TF}(C_t^{\#})$ shares only observable variables, interpolants will contain only observable predicates. However, the conjuncts share variables corresponding to different states along a path, and thus an interpolant may contain predicates relating values of variables in different execution steps. Thus, the key challenge for the interpolant computation in our case is to ensure that the predicates in the resulting interpolant are localized.

The straight-forward application of existing interpolation methods (such as, for example,[JM06, RSS07]) can produce refinement predicates that are either guaranteed to be observable or guaranteed not to relate variables for different states, but not both.

5.3.1 Craig Interpolation

Definition 5.3.1. Let Th be a theory over a signature Ξ, all of whose symbols are interpreted in Th, and let φ and ψ be formulas over Ξ. Suppose that $\varphi \wedge \psi$ is unsatisfiable w.r.t. Th. We say that a formula θ is a (theory-specific) *interpolant for* (φ, ψ) iff φ implies θ w.r.t. Th, $\theta \wedge \psi$ is unsatisfiable w.r.t. Th and $\mathit{Vars}(\theta) \subseteq \mathit{Vars}(\varphi) \cap \mathit{Vars}(\psi)$.

A number of theories, such as for example the theories of linear rational or real arithmetic, admit quantifier free interpolation, that is, for each pair of quantifier-free formulas whose conjunction is unsatisfiable there exists a quantifier free interpolant.

For two formulas φ and ψ for which $\varphi \wedge \psi$ is unsatisfiable w.r.t. *Th*, if θ is an interpolant for (φ, ψ), then $\neg\theta$ is an interpolant for (ψ, φ) w.r.t. *Th*.

Definition 5.3.2. Let *Th* be a theory over a signature Ξ, and $\mathcal{AP}$ be the set of atomic formulas over the signature Ξ. We call a formula φ *localized w.r.t. a partitioning* $(X_0, \ldots, X_n)$ of the set $\mathit{Vars}(\varphi)$ of free variables of φ if for each atomic proposition $\psi \in \mathit{Preds}(\varphi)$ it holds that $\mathit{Vars}(\psi) \subseteq X_i$ for some $0 \leq i \leq n$.

Definition 5.3.3. Let *Th* be a theory over a signature Ξ and let φ and ψ be formulas over Ξ. Suppose that $\varphi \wedge \psi$ is unsatisfiable w.r.t. *Th*. Given a partitioning $(X_0, \ldots, X_n)$ of the set of variables $\mathit{Vars}(\varphi) \cup \mathit{Vars}(\psi)$, we say that θ is a *localized interpolant* for (φ, ψ) if θ is an interpolant for (φ, ψ) and θ is localized w.r.t. $(X_0, \ldots, X_n)$.

5.3.2 Observation Equivalence Refinement

We describe an algorithm RefineObservations, outlined as Algorithm 3, which computes a finite set of observable predicates given the set of trace formulas from $\mathsf{TF}(C_t^{\#})$. It uses as a black box a procedure LocalizedInterpolant, which, given two formulas and a partitioning of the variables occurring in them, returns an interpolant for the pair of formulas that is localized w.r.t. the given partition or returns $\bot$ if it was unable to compute such an interpolant. In Section 5.3.3 we provide an instantiation of this procedure for the case of linear rational arithmetic and reason about its properties.

Given the set of trace formulas $\mathit{Formulas}(C_t^{\#})$ for the traces in $C_t^{\#}$ whose conjunction is unsatisfiable, the function MinimalUnsatisfiableSubset returns a minimal set Φ of trace formulas whose conjunction is unsatisfiable. This set is minimal in the sense that for every $\Phi' \subsetneq \Phi$ the conjunction of its elements is satisfiable. Thus, the traces corresponding to the formulas in the set Φ are such that if $\mathit{Player}_\exists$ could distinguish between the corresponding sets of counterexample paths he could prevent the counterexample $C_t^{\#}$ by playing the respective sequences of actions. In order to enable $\mathit{Player}_\exists$ to play the actions occurring in these traces, we additionally refine the abstraction with predicates that guarantee that the refined abstraction is precise w.r.t. this set of actions. Namely, given a set of traces Υ, the set $\mathsf{Predicates}_\Sigma(\Upsilon)$ of predicates consists of all predicates $x = c$, where $x \in V_\exists$ and $\tau[i](x) = c$ for some $\tau \in \Upsilon$ and some $0 \leq i < |\tau|$.

The minimal unsatisfiable set of trace formulas Φ can be split into a formula φ_1 and the nonempty set $\Phi_2 = \Phi \setminus \{\varphi_1\}$ such that $\varphi_2 = \bigwedge_{\varphi \in \Phi_2} \varphi$ is satisfiable and $\varphi_1 \wedge \varphi_2$ is unsatisfiable. This means that every concrete counterexample path for the trace corresponding to φ_1 can be distinguished from some concrete counterexample path from each tuple of paths satisfying Φ_2, meaning that the paths differ before the first position in which the corresponding traces differ. Since the pair of satisfiable formulas (φ_1, φ_2) has unsatisfiable conjunction, we can compute an interpolant θ for (φ_1, φ_2). By the definition of the substitutions η_τ, θ refers only to indexed observable variables.

In order to ensure that each atomic formula in θ is a state predicate, i.e., it does not relate different predicates on a path, we pass to the interpolation procedure a partitioning $\mathsf{Partition}(\mathit{Vars}(\varphi_1) \cup \mathit{Vars}(\varphi_2), C_t^\#) = (\widetilde{V}_1, \ldots, \widetilde{V}_k)$ of the variables occurring in the trace formulas and require that in each predicate in θ only variables from a single $\widetilde{V}_i$ occur. The partitioning of the variables is according to their level in the tree, namely,

$$\widetilde{V}_i = \bigcup_{n \in N^\#, |path(n)|-1=i, \tau \in Traces(C_t^\#)} V^{n,\tau} \cap (\mathit{Vars}(\varphi_1) \cup \mathit{Vars}(\varphi_2)).$$

The procedure LocalizedInterpolant, applied to formulas φ_1 and φ_2 and a partitioning $(\widetilde{V}_1, \ldots, \widetilde{V}_k)$ of $\mathit{Vars}(\varphi_1) \cup \mathit{Vars}(\varphi_2)$, returns an interpolant localized w.r.t. $(\widetilde{V}_1, \ldots, \widetilde{V}_k)$, if it can find one. Note that even if the theory admits Craig interpolation, a localized interpolant for the given formulas and variable partitioning might not exist. In the latter case the procedure LocalizedInterpolant aborts returning $\bot$.

If LocalizedInterpolant returns an interpolant θ, the set $\mathsf{ExtractPredicates}(\theta)$ consists of the predicates from θ in which the indexed variables have been renamed to the original variables V, and which can be added to the set of refinement predicates.

Since the conjunction $\varphi_1 \wedge \varphi$ is satisfiable for each $\varphi \in \Phi$, it holds that $\theta \wedge \varphi$ is also satisfiable for each $\varphi \in \Phi$. Thus, if $|\Phi_2| > 1$, the predicates extracted from θ may be insufficient for $\mathit{Player}_\exists$ to prevent the counterexample tree $C_t^\#$. This is because they suffice to distinguish the paths satisfying φ_1 from the tuples of paths satisfying φ_2, i.e., the paths satisfying the trace formulas in Φ_2 that satisfy the equivalence condition, but not necessarily from the paths satisfying the formulas in Φ_2 that can be distinguished from each other. Therefore, the set Φ is updated to $\Phi' = \{\theta \wedge \varphi \mid \varphi \in \Phi_2\}$ and the process of computing an interpolant and a set of predicates is repeated.

Algorithm: RefineObservations

Input: set of trace formulas $\mathit{Formulas}(C_t^{\#}) = \{\varphi_{\mathit{trace}}(\tau)\eta_\tau \mid \tau \in \mathit{Traces}(C_t^{\#})\}$ for set of traces $\mathit{Traces}(C_t^{\#})$ in an abstract counterexample tree $C_t^{\#}$

Output: finite set $\mathcal{R}$ of refinement predicates

$\Phi :=$ MinimalUnsatisfiableSubset($\mathit{Formulas}(C_t^{\#})$);
$\mathcal{R} :=$ Predicates$_\Sigma$($\{\tau \in \mathit{Traces}(C_t^{\#}) \mid \varphi_{\mathit{trace}}(\tau)\eta_\tau \in \Phi\}$);

while $|\Phi| \geq 2$ **do**

  pick (φ_1, Φ_2) **such that:**
    $\Phi_2 \subseteq \Phi$, $\varphi_1 \in \Phi \setminus \Phi_2$,
    $\Phi_2 \neq \emptyset$, $\bigwedge_{\varphi\in\Phi_2} \varphi$ **is** SAT
    $\varphi_1 \wedge \left(\bigwedge_{\varphi\in\Phi_2} \varphi\right)$ **is** UNSAT;

  $\varphi_2 = \bigwedge_{\varphi\in\Phi_2} \varphi$;
  $\theta :=$ LocalizedInterpolant($\varphi_1, \varphi_2,$ Partition($\mathit{Vars}(\varphi_1) \cup \mathit{Vars}(\varphi_2), C_t^{\#}$));
  $\mathcal{R} := \mathcal{R} \cup$ ExtractPredicates(θ); `/* extract refinement predicates */`

  $\Phi := \{\theta \wedge \varphi \mid \varphi \in \Phi_2\}$; `/* update the set of conjuncts */`

return $\mathcal{R}$;

Algorithm 3: Refinement Predicates for the Observation Equivalence.

Example 5.3.1. To illustrate procedure RefineObservations, we consider a variant of Example 2.1.1, in which the set $V_\forall^o$ contains an additional variable y^o, that is initially equal to y and then is set to 0 by $\mathit{Player}_\forall$. Formally, we have the symbolic game structure $\mathcal{G}' = (V_\exists, V_\forall', V_\forall^{o\prime}, t, \varphi_{\mathit{Init}}', \mathcal{T}_\exists', \mathcal{T}_\forall')$, where $\mathcal{G} = (V_\exists, V_\forall, V_\forall^o, t, \varphi_{\mathit{Init}}, \mathcal{T}_\exists, \mathcal{T}_\forall)$ is the symbolic game structure from Example 2.1.1 and $V_\forall' = V_\forall \dot\cup \{y^o\}$, $V_\forall^{o\prime} = V_\forall^o \dot\cup \{y^o\}$, $\varphi_{\mathit{Init}}' = \varphi_{\mathit{Init}} \wedge y^o = y$, $\mathcal{T}_\exists' = \mathcal{T}_\exists \wedge y^{o\prime} = y$, $\mathcal{T}_\forall' = \mathcal{T}_\forall \wedge y^{o\prime} = 0$.

Let us look at the abstraction Abstract($\mathcal{G}', \mathcal{P}'$) w.r.t. the set of predicates

$$\begin{aligned}\mathcal{P}' \;=\; & \{(t=\forall), (\mathit{move}=N), (\mathit{move}=E), (\mathit{move}=S), (x<6), (x\geq 9),\\ & (x\leq -1), (x\geq 3), (x\leq 1), (x\geq 1), (x<0), (x<2), (x<4), (x\geq 5),\\ & (\mathit{steps}>3), (\mathit{steps}>2), (\mathit{steps}>0), (\mathit{steps}>1), (\mathit{err}=1),\\ & (y\geq 4), (y\leq -4), (y\geq 3), (y\leq -3), (y\geq 2), (y\leq -2), (y\leq -1)\}.\end{aligned}$$

In the abstract game Safety(Abstract($\mathcal{G}', \mathcal{P}'$), $\mathit{Err}^{\#}$), there exists a counterexample tree, part of which is shown in Figure 5.3. The path $\pi_1 = 0, 1, 5, 9, 25, 37, 85, 125$ is

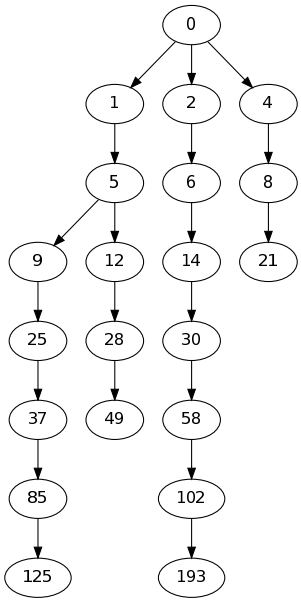


Figure 5.3: Paths in an abstract counterexample tree for $\mathsf{Safety}(\mathsf{Abstract}(\mathcal{G}', \mathcal{P}'), Err^{\#})$. Paths $\pi_1 = 0, 1, 5, 9, 25, 37, 85, 125$ and $\pi_2 = 0, 2, 6, 14, 30, 58, 102, 193$ correspond to traces $\tau_1 = N \cdot N \cdot N$ and $\tau_2 = E \cdot E \cdot E$ respectively. The conjunction of the corresponding trace formulas is unsatisfiable.

the only path in this tree corresponding to the trace $\tau_1 = N \cdot N \cdot N$ and the path $\pi_2 = 0, 2, 6, 14, 30, 58, 102, 193$ is the only path for the trace $\tau_2 = E \cdot E \cdot E$. The conjunction $\varphi_{trace}(\tau_1) \wedge \varphi_{trace}(\tau_2)$ is unsatisfiable, that is, the two formulas $\varphi_{trace}(\tau_1)$ and $\varphi_{trace}(\tau_2)$ form a minimal subset of the set of trace formulas for this counterexample tree, whose conjunction is unsatisfiable. The reason for which the counterexample tree is not concretizable is that for the initial states in which $y^o \leq 0$, the sequence τ_1, i.e., making 3 steps north does not lead to a bad state, while for initial states in which $y^o > 0$ the sequence τ_2, i.e., making 3 steps east does not lead to a bad state. Computing a Craig interpolant for these two trace formulas yields the predicate $(y_o \leq 0)$, which distinguishes the concrete counterexample paths for the traces in their first states.

In the example above, each pair of concrete counterexample paths for the two traces respectively could be distinguished by an observable predicate in their first state. In general, this is not always the case, as it can be seen in the next example. There, different tuples (in that case again pairs) of concrete counterexample paths, exhibit an observable difference at different levels of the spurious counterexample tree.

Example 5.3.2. Consider a simple model of a production system, that produces items differing in several characteristics: items of type 0 with weight in the interval $[1, 5]$ and items of type 1 with weight in the interval $[4, 10]$. The items come in two sizes: items of size 1 are those of type 0 with weight less than or equal to 2 and those of type 1 with weight less that or equal to 7. The remaining items, i.e., items of type 0 whose weight is greater than 2 and those of type 1 whose weight is greater than 7 have size 2. A robotic arm controller is required to sort the items correctly into the respective box according to their type. Via sensors the controller receives precise information about the size of the currently produced item, and inaccurate information about its weight.

The controller synthesis problem for this production system is modeled as a symbolic game $\mathsf{Safety}(\mathcal{G}, \varphi_{Err})$ with $\mathcal{G} = (V_\exists, V_\forall, V_\forall^o, t, \varphi_{Init}, \mathcal{T}_\exists, \mathcal{T}_\forall)$, where

- $V_\exists := \{box\},\ V_\forall := \{type, weight, size, weight^o, loc\},\ V_\forall^o := \{size, weight^o, loc\}$,
- $\varphi_{Init} := t = \forall \wedge box = 0 \wedge type = 0 \wedge weight = 0 \wedge size = 0 \wedge weight^o = 0 \wedge loc = 0$,
- $\mathcal{T}_\exists := t = \exists \wedge t' = \forall \wedge (box' = 0 \vee box' = 1) \wedge \mathsf{preserve}(type, weight, size, weight^o, loc)$,
- $\mathcal{T}_\forall := t = \forall \wedge (\varphi_0 \vee \varphi_1 \vee \varphi_2 \vee \varphi_3' \vee \varphi_3'' \vee \varphi_4) \wedge box' = box$,

$$
\begin{array}{rcl}
\varphi_0 & := & loc = 0 \wedge t' = \forall \wedge loc' = 1 \wedge \mathsf{preserve}(size, weight^o) \wedge \\
 & & (type' = 0 \wedge weight' \in [1,5] \vee type' = 1 \wedge weight' \in [4,10]), \\
\varphi_1 & := & loc = 1 \wedge t' = \forall \wedge loc' = 2 \wedge \mathsf{preserve}(type, weight, weight^o) \wedge \\
 & & \big(size' = 1 \wedge (type = 0 \wedge weight \leq 2 \vee type = 1 \wedge weight \leq 7) \vee \\
 & & size' = 2 \wedge (type = 0 \wedge weight > 2 \vee type = 1 \wedge weight > 7)\big), \\
\varphi_2 & := & loc = 2 \wedge t' = \exists \wedge loc' = 3 \wedge size' = 0 \wedge \mathsf{preserve}(type, weight) \wedge \\
 & & (weight^{o\prime} < 3 \leftrightarrow weight < 3) \wedge (weight^{o\prime} \in [3,6) \leftrightarrow weight \in [3,6)), \\
\varphi_3' & := & loc = 3 \wedge type = box \wedge loc' = 0 \wedge t' = \forall \wedge \\
 & & type' = 0 \wedge weight' = 0 \wedge size' = 0 \wedge weight^{o\prime} = 0, \\
\varphi_3'' & := & loc = 3 \wedge type \neq box \wedge loc' = 4 \wedge \mathsf{preserve}(t, type, weight, size, weight^o), \\
\varphi_4 & := & loc = 4 \wedge \mathsf{preserve}(t, loc, type, weight, size, weight^o),
\end{array}
$$

  where for a set of variables $X \subseteq V$, $\mathsf{preserve}(X) := \bigwedge_{x \in x}(x' = x)$,

- $\varphi_{Err} := t = \exists \wedge loc = 4$.

Note that, in order to correctly infer the type of the item at location 3, the controller has to observe and record the value of the observable variable *size* when the location is 2, and observe the value of the observable variable $weight^0$ when the location is 3.

Let for example $\mathcal{P} = \{(t = \forall), (box = 0), (loc = 0), (loc = 4), (loc = 1), (loc = 2), (loc \leq 1), (loc \geq 3)\}$ and consider the abstract game $\mathsf{Safety}(\mathsf{Abstract}(\mathcal{G}, \mathcal{P}), Err^\#)$.

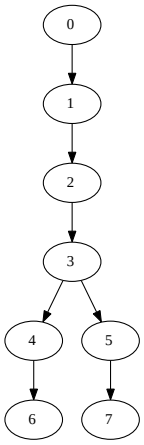


Figure 5.4: Counterexample tree $C_t^{\#}$ in the abstract game $\mathsf{Safety}(\mathsf{Abstract}(\mathcal{G}, \mathcal{P}), Err^{\#})$ from Example 5.3.2. The conjunction of the trace formulas for the tree $C_t^{\#}$ is unsatisfiable, and hence the abstract counterexample tree is not concretizable.

A counterexample tree $C_t^{\#}$ in $\mathsf{Safety}(\mathsf{Abstract}(\mathcal{G}, \mathcal{P}), Err^{\#})$ is shown in Figure 5.4. The set of traces for the root node is $\mathit{Traces}(0) = \{\sigma_0, \sigma_1\}$, where $\sigma_0(box) = 0$ and $\sigma_1(box) = 1$. Now, applying the procedure RefineObservations yields the predicates $size \geq 2$, $weight^o < 3$ and $weight^o < 6$, where predicate $size \geq 2$ is associated with node 2 in the tree $C_t^{\#}$ and the remaining two predicates are associated with node 3.

The procedure RefineObservations, shown as Algorithm 3, computes a set of refinement predicates, given the set of satisfiable trace formulas for a spurious abstract counterexample tree $C_t^{\#}$. The following lemma formalizes the property of the computed set of predicates that is important for showing the progress of the refinement step.

Lemma 5.3.1. *Let* $\mathsf{Safety}(\mathcal{G}, \varphi_{Err})$ *be a safety game under incomplete information,* $\mathcal{P} \subseteq \mathcal{AP}[V]$ *be a finite set of predicates that is precise w.r.t. the turn variable* t *and let* $G^{\#} = \mathsf{Abstract}(\mathcal{G}, \mathcal{P})$ *and* $Err^{\#} = \{s^{\#} \in S^{\#} \mid [\![s^{\#}]\!] \cap [\![\varphi_{Err}]\!] \neq \emptyset\}$.

Let $C_t^{\#}$ *be an abstract counterexample tree in* $\mathsf{Safety}(G^{\#}, Err^{\#})$ *such that* $\mathsf{TF}(C_t^{\#})$ *is unsatisfiable and* $\varphi_{trace}(\tau)$ *is satisfiable for each* $\tau \in \mathit{Traces}(C_t^{\#})$.

Consider a set $\Pi \subseteq \mathsf{Prefs}(G)$ *of prefixes such that for every* $\tau \in \mathit{Traces}(C_t^{\#})$, *there exists* $\pi_\tau \in \Pi$ *such that* $\pi_\tau \models \varphi_{trace}(\tau)$. *Suppose that* RefineObservations *applied to* $\mathit{Formulas}(C_t^{\#}) = \{\varphi_{trace}(\tau)\eta_\tau \mid \tau \in \mathit{Traces}(C_t^{\#})\}$ *completed successfully (every call to* LocalizedInterpolant *returned an interpolant) and returned a set* $\mathcal{R}$ *of predicates. Then there exist:*

- $\tau_1, \tau_2 \in \mathit{Traces}(C_t^{\#})$ *such that* $\tau_1 \neq \tau_2$ *and* $\mathsf{Predicates}_{\Sigma}(\{\tau_1, \tau_2\}) \subseteq \mathcal{R}$,

- *a predicate* $\psi \in \mathcal{R}$ *and*
- *a position* $j < \mathsf{diff}(\pi_{\tau_1}, \pi_{\tau_2}, \tau_1, \tau_2)$,

such that $\pi_{\tau_1}[j] \models \psi$ *and* $\pi_{\tau_2}[j] \models \neg\psi$.

Proof. Let k be the number of iterations of the loop in RefineObservations and $\theta_1 \dots \theta_k$ be the corresponding interpolants computed by LocalizedInterpolant. We will show that there exists an iteration i such that there exist $\tau_1, \tau_2 \in \mathit{Traces}(C_t^{\#})$ such that $\tau_1 \neq \tau_2$, $\pi_{\tau_1} \models \theta_i$ and $\pi_{\tau_2} \models \neg\theta_i$. Thus, there exist a predicate $\psi \in \mathit{Preds}(\theta_i)$ and a position $j < \mathsf{diff}(\pi_{\tau_1}, \pi_{\tau_2}, \tau_1, \tau_2)$ such that $\pi_{\tau_1}[j] \models \psi$ and $\pi_{\tau_2}[j] \models \neg\psi$ or $\pi_{\tau_1}[j] \models \neg\psi$ and $\pi_{\tau_2}[j] \models \psi$. Since $\mathcal{R} \supseteq \mathsf{ExtractPredicates}(\theta_i)$, the claim of the lemma holds.

We denote with Φ^i, φ_1^i, Φ_2^i and φ_2^i the values of Φ, φ_1, Φ_2 and φ_2 at iteration i.

It can easily be seen by induction on i that if for all iterations i' with $i' < i$ the above property does not hold for $\theta_{i'}$, then either the property holds for θ_i or for each $\varphi \in \Phi^i$ there exists a $\pi \in \Pi$ such that $\pi \models \varphi$. The statement clearly holds for $i = 1$ by he choice of Π and the definition of Φ^0. For $i + 1$ the statement follows from the inductive hypothesis and the fact that θ_i is an interpolant for $(\varphi_1^i, \varphi_2^i)$.

Thus, either there exists $i < k$ such that there exist $\tau_1, \tau_2 \in \mathit{Traces}(C_t^{\#})$ for which $\tau_1 \neq \tau_2$, $\pi_{\tau_1} \models \theta_i$ and $\pi_{\tau_2} \models \neg\theta_i$, or there exist $\tau_1, \tau_2 \in \mathit{Traces}(C_t^{\#})$ such that $\tau_1 \neq \tau_2$, $\pi_{\tau_1} \models \varphi_1^k$ and $\pi_{\tau_2} \models \varphi_2^k$ where $\Phi_2^k = \{\varphi_2^k\}$. In the latter case, since θ_k is an interpolant for $(\varphi_1^k, \varphi_2^k)$ it holds that $\pi_{\tau_1} \models \theta_k$ and $\pi_{\tau_2} \models \neg\theta_k$, which concludes the proof. □

5.3.3 Localized Interpolants for Linear Rational Arithmetic

For this section we fix the considered background theory *Th* to be the theory of linear rational arithmetic. Thus, the set $\mathcal{AP}$ of atomic predicates is the set of linear inequalities with rational coefficients. Each quantifier free linear arithmetic formula is a positive boolean combination of linear inequalities. In particular, when transformed into disjunctive normal form, it can be seen as a disjunction of systems of linear inequalities.

Constraint solving for interpolation for linear arithmetic A *mixed system*, denoted $Ax \leqslant a$, is a conjunction of strict and non-strict linear inequalities. The system consisting of the strict inequalities of $Ax \leqslant a$ is denoted by $A^{\mathsf{lt}}x < a^{\mathsf{lt}}$, and the one of the non-strict ones with $A^{\mathsf{le}}x < a^{\mathsf{le}}$. If m_A is the number of columns of the matrix A, we denote with $A|_k$ its k-th column for $1 \leq k$. A conic combination of the inequalities in $Ax \leqslant a$ is an inequality $\lambda Ax \leqslant \lambda a$ for a row vector $\lambda = (\lambda|_1, \dots, \lambda|_{m_A})$ with length

m_A and such that $\lambda|_k \geq 0$ for each $1 \leq k \leq m_A$. The subvectors of λ corresponding to the strict and non-strict inequalities in $Ax \leqslant a$ respectively are denoted with λ^{lt} and λ^{le}. If $x = (x_1, \ldots, x_{m_A})$, we define $\mathit{Vars}(Ax \leqslant a) = \{x_k \mid A|_k \neq 0\}$.

Rybalchenko and Sofronie-Stokkermans propose in [RSS07] an algorithm `LI` for computing linear interpolants for mixed systems of inequalities whose conjunction is not satisfiable. Their algorithm reduces interpolant generation to a linear programming problem and is based in Motzkin's transposition theorem below.

Theorem 5.3.2 ([Sch86])**.** *If $Ax < a$ is a system of strict inequalities and $Bx \leq b$ is a system of non-strict inequalities, then there exists a vector x_0 such that $Ax_0 < a$ and $Bx_0 \leq b$ iff for all vectors $\lambda \geq 0$ and $\mu \geq 0$ the following conditions hold true:*

- *if $\lambda A + \mu B = 0$ then $\lambda a + \mu b \geq 0$, and*
- *if $\lambda A + \mu B = 0$ and $\lambda \neq 0$, then $\lambda a + \mu b > 0$.*

The input of algorithm from [RSS07] consists of two mixed systems of inequalities $Ax \leqslant a$ and $Bx \leqslant b$ such that the conjunction $Ax \leqslant a \wedge Bx \leqslant b$ is not satisfiable. The output is an interpolant which is a linear inequality $ix \lhd \delta$ where $\lhd \in \{\leq, <\}$.

Theorem 5.3.3 ([RSS07])**.** *For mutually unsatisfiable mixed systems $Ax \leqslant a$ and $Bx \leqslant b$ of inequalities, there exists a linear inequality interpolant $ix \lhd \delta$ with $\lhd \in \{\leq, <\}$.*

As noted in [RSS07], an interpolant θ for disjunctions of mixed systems $\bigvee_k A_k \leqslant a_k$ and $\bigvee_l B_l x \leqslant b_l$ can be constructed by computing a linear interpolant $i_{kl}x \lhd \delta_{kl}$ for each pair $(A_j \leqslant a_j, B_k x \leqslant b_k)$ and then letting $\theta = \bigvee_j \bigwedge_k i_{jk}x \lhd \delta_{jk}$. Another alternative suggested there is to use their constraint-based algorithm to generate partial interpolants for conflict clauses in a DPLL procedure and then combine them using an extension of Pudlák's algorithm [Pud97, YM05].

Computing localized interpolants In order to compute localized interpolants for linear rational arithmetic we give an algorithm `LILA` (Linear Interpolation with Localized Atoms) that extends algorithm `LI` from [RSS07]. In addition to the systems of inequalities, algorithm `LILA` receives a partitioning of the variables which occur in the input systems. As a result, each atom in the generated interpolant, a conjunction of linear inequalities, is guaranteed to contain variables from exactly one partition.

Algorithm: LILA

Input: mixed systems $Ax \leqslant a$ and $Bx \leqslant b$ where $Ax \leqslant a \wedge Bx \leqslant b$ is UNSAT partitioning $(X_0, X_1, \ldots, X_n)$ of the variables in the vector x

Output: interpolant $\bigwedge_{j=0}^{n} i_j x \lhd_j \delta_j$ with $\lhd_j \in \{\leq, <\}$ and $Vars(i_j x \lhd_j \delta_j) \subseteq X_j$ or $\bot$ if such an interpolant could not be computed

$\chi_1 := \lambda \geq 0 \wedge \mu \geq 0 \wedge \lambda A + \mu B = 0;$
$\chi_2 := \lambda = \sum_{j=0}^{n} \lambda_j \wedge \bigwedge_{j=0}^{n} (\lambda_j \geq 0 \wedge i_j = \lambda_j A \wedge \delta_j = \lambda_j a \wedge \bigwedge_{k \in \overline{Ix(j)}} \lambda_j A_{|k} = 0);$

if exist $\lambda, \mu, \lambda_j, i_j, \delta_j$, for $0 \leq j \leq n$ **satisfying** $\chi_1 \wedge \chi_2 \wedge \lambda a + \mu b \leq -1$
then return $\bigwedge_{j=0}^{n} i_j x \leq \delta_j;$

elif exist $\lambda, \mu, \lambda_j, i_j, \delta_j$, for $0 \leq j \leq n$ **satisfying** $\chi_1 \wedge \chi_2 \wedge \lambda a + \mu b \leq 0 \wedge \lambda^{\mathsf{lt}} \neq 0$
then return $\bigwedge_{0 \leq j \leq n, \lambda_j^{\mathsf{lt}} \neq 0} i_j x < \delta_j \wedge \bigwedge_{0 \leq j \leq n, \lambda_j^{\mathsf{lt}} = 0} i_j x \leq \delta_j;$

elif exist $\lambda, \mu, \lambda_j, i_j, \delta_j$, for $0 \leq j \leq n$ **satisfying** $\chi_1 \wedge \chi_2 \wedge \lambda a + \mu b \leq 0 \wedge \mu^{\mathsf{lt}} \neq 0$
then return $\bigwedge_{j=0}^{n} i_j x \leq \delta_j;$

else return $\bot$

Algorithm 4: Computing localized interpolants for systems of inequalities.

The procedure LILA , shown as Algorithm 4, generates an interpolant for $(Ax \leqslant a, Bx \leqslant b)$ that is of the form $\bigwedge_{j=0}^{n} i_j x \lhd_j \delta_j$, where $n+1$ is the number of sets $(X^0, X^1, \ldots, X^n)$, and which is localized w.r.t. the given variable partitioning. Such an interpolant, however, might not exists. If this is the case, LILA returns the element $\bot$.

Each of the inequalities $i_j x \lhd_j \delta_j$ is a linear combination $\lambda_j Ax \leqslant \lambda_j a$ of inequalities in $Ax \leqslant a$. The coefficients are computed as a solution to a set of constraints, they are a satisfying assignment to the variables $\lambda_0, \lambda_1, \ldots, \lambda_n$. The subvectors $\lambda_j^{\mathsf{lt}}, \lambda_j^{\mathsf{le}}$ for $j = 0, 1, \ldots, n$ define linear combinations of strict and non-strict inequalities in $Ax \leqslant a$, respectively. The remaining variables in the constraints are $\lambda, \lambda^{\mathsf{lt}}, \lambda^{\mathsf{le}}, \mu, \mu^{\mathsf{lt}}, \mu^{\mathsf{le}}$.

The constraint $\chi_1 = \lambda \geq 0 \wedge \mu \geq 0 \wedge \lambda A + \mu B = 0$, together with the constraints on $\lambda, \mu, \lambda^{\mathsf{lt}}, \mu^{\mathsf{lt}}$ in each of the three cases, as in [RSS07], encode the possible cases according to Theorem 5.3.2. Constraint χ_2 encodes the requirement for localized atoms:

$$\chi_2 := \lambda = \sum_{j=0}^{n} \lambda_j \wedge \bigwedge_{j=0}^{n} \big(\lambda_j \geq 0 \wedge i_j = \lambda_j A \wedge \delta_j = \lambda_j a \wedge \bigwedge_{k \in \overline{Ix(j)}} \lambda_j A_{|k} = 0\big).$$

The set $Ix(j) = \{k \in \{1, \ldots, m_A\} \mid x_k \in V^j\}$ for a $0 \leq j \leq n$ is the set of column indices in the matrix A that correspond to variables in the set X_j. We denote with $\overline{Ix(j)} = \{1, \ldots, m_A\} \setminus Ix(j)$ the complement of $Ix(j)$. Thus, $\bigwedge_{k \in \overline{Ix(j)}} \lambda_j A_{|k} = 0$ requires that the coefficients in $i_j x \lhd_j \delta_j$ of all variables in x that are not in X_j are 0.

In general, a localized interpolant for a pair of systems of inequalities may not exist. Even if one does exist, it might not be of the following form, where $\lhd_j \in \{\leq, <\}$

$$\theta = \bigwedge_{j=0}^{n} i_j x \lhd_j \delta_j \qquad (\bigwedge \mathsf{LI})$$

that is considered by Algorithm 4. However, we establish in the following theorem that if an interpolant of the form $(\bigwedge \mathsf{LI})$ does exist, the procedure `LILA` is guaranteed to find one. Later we will see how we can use algorithm `LILA` to find also a localized interpolant that is a disjunction of linear inequalities, in case one exists.

Theorem 5.3.4. *Algorithm* `LILA` *is* **sound***: If, given mixed systems $Ax \leqslant a$ and $Bx \leqslant b$ and a variable partitioning $(X_0, X_1, \ldots, X_n)$, it returns a formula θ, then θ is an interpolant for $(Ax \leqslant a, Bx \leqslant b)$ that is localized w.r.t. $(X_0, X_1, \ldots, X_n)$.*
Algorithm `LILA` *is* **complete***: if there exists interpolant θ for $(Ax \leqslant a, Bx \leqslant b)$ that is localized w.r.t. $(X_0, X_1, \ldots, X_n)$ and satisfies $(\bigwedge \mathsf{LI})$, then the algorithm will find one.*

Proof. **Soundness.** Assume that the algorithm returns a formula θ. Thus, there exist row vectors λ, μ and λ_j, for $0 \leq j \leq n$, that satisfy χ_1 and χ_2. $\theta = \bigwedge_{j=0}^{n} i_j x \lhd_j \delta_j$, where for each $0 \leq j \leq n$, $i_j = \lambda_j A$, $\delta_j = \lambda_j a$ and $\lhd_j \in \{\leq, <\}$. Let $i_j = (i_j^0, \ldots, i_j^n)$ and $k \in \overline{Ix(j)}$. We have $i_j^k = \lambda_j A_{|k}$, and since λ_j satisfies constraint χ_2, $i_j^k = 0$. Thus, $Vars(i_j x \lhd_j \delta_j) \subseteq X_j$ for each $0 \leq j \leq n$, i.e., θ is localized w.r.t. $(X_0, X_1, \ldots, X_n)$.

It remains to show that θ is indeed an interpolant for $(Ax \leqslant a, Bx \leqslant b)$. The formula $Ax \leqslant a$ implies each of the conjuncts in θ, since each of them is a conic combination of the inequalities in $Ax \leqslant a$. Hence, $Ax \leqslant a$ implies θ. By [RSS07], $\lambda Ax \lhd \lambda a$, where $\lhd$ is $<$ iff θ was returned by the second case of `LILA` , is an interpolant for $(Ax \leqslant a, Bx \leqslant b)$, and hence its conjunction with $Bx \leqslant b$ is unsatisfiable. As $\lambda = \sum_{j=0}^{n} \lambda_j$, θ implies $\lambda Ax \lhd \lambda a$. Therefore, the conjunction of θ and $Bx \leqslant b$ is unsatisfiable as well. Since $\lambda Ax \lhd \lambda a$ is an interpolant $Vars(\lambda Ax \lhd \lambda a) \subseteq Vars(Ax \leqslant a) \cap Vars(Bx \leqslant b)$. As each variable x_k, where $x = (x_1, \ldots, x_{m_A})$ appears in exactly one X_j, it occurs in at most one of the conjuncts and hence, if $\lambda A|_k = 0$, then also $\lambda_j A|_k = 0$ for each $0 \leq j \leq n$. Therefore $Vars(\lambda_j Ax \lhd \lambda_j a) \subseteq Vars(Ax \leqslant a) \cap Vars(Bx \leqslant b)$ for each $0 \leq j \leq n$.

Completeness. Let θ be an interpolant for $(Ax \leqslant a, Bx \leqslant b)$ that is localized w.r.t. $(X_0, X_1, \ldots, X_n)$ and that is of the form $(\bigwedge \mathsf{LI})$.

Since $Ax \leqslant a$ implies θ, it implies also $i_j x \lhd_j \delta_j$ for each $0 \leq j \leq n$. Let $0 \leq j \leq n$ and consider $-i_j x \lhd'_j -\delta_j$, where $\lhd'_j$ is $<$ iff $\lhd_j$ is $\leq$. The conjunction of $Ax \leqslant a$ and $-i_j x \lhd'_j -\delta_j$ is unsatisfiable. Thus, by Theorem 5.3.2 there exist vectors $\lambda_j'^{\mathsf{lt}} \geq 0$ and $\lambda_j'^{\mathsf{le}} \geq 0$ and constant $\alpha_j \geq 0$ such that

- if $\lhd'_j$ is $<$, then $(\lambda_j'^{\mathsf{lt}}\alpha_j)\binom{A^{\mathsf{lt}}}{-i_j} + \lambda_j'^{\mathsf{le}}A^{\mathsf{le}} = 0$ and $(\lambda_j'^{\mathsf{lt}}\alpha_j)\binom{a^{\mathsf{lt}}}{-\delta_j} + \lambda_j'^{\mathsf{le}}a^{\mathsf{le}} \leq 0$ and
- if $\lhd'_j$ is $\leq$, then $\lambda_j'^{\mathsf{lt}}A^{\mathsf{lt}} + (\lambda_j'^{\mathsf{le}}\alpha_j)\binom{A^{\mathsf{le}}}{-i_j} = 0$ and $\lambda_j'^{\mathsf{lt}}a^{\mathsf{lt}} + (\lambda_j'^{\mathsf{le}}\alpha_j)\binom{a^{\mathsf{le}}}{-\delta_j} \leq 0$.

In both cases, since $Ax \leqslant a$ is satisfiable, it holds that $\alpha_j > 0$. We define λ'_j as the combination of λ'^{lt} and λ'^{le}. Then,

$$\lambda'_j A = \lambda_j'^{\mathsf{lt}}A^{\mathsf{lt}} + \lambda_j'^{\mathsf{le}}A^{\mathsf{le}} = \alpha_j i_j \text{ and } \lambda'_j a = \lambda_j'^{\mathsf{lt}}a^{\mathsf{lt}} + \lambda_j'^{\mathsf{le}}a^{\mathsf{le}} \leq \alpha_j \delta_j.$$

Now, consider θ as a mixed system $Ix \leqslant d$. Since the conjunction of θ and $Bx \leqslant b$ is unsatisfiable, by [RSS07] there exist vectors $\lambda'' = (\lambda''_0, \ldots, \lambda''_n) \geq 0$ and $\mu \geq 0$, such that $\lambda'' I + \mu B = 0$ and one of the following is satisfied: (1) $\lambda'' d + \mu b \leq -1$, or (2) $\lambda'' d + \mu b \leq 0$ and $\lambda''^{\mathsf{lt}} \neq 0$, or (3) $\lambda'' d + \mu b \leq 0$ and $\mu^{\mathsf{lt}} \neq 0$.

Let us define $\lambda_j = \frac{\lambda''_j}{\alpha_j}\lambda'_j$ for each $0 \leq j \leq n$ and $\lambda = \sum_{j=0}^{n} \lambda_j$.

Consider a $0 \leq j \leq n$ and $k \in \overline{Ix(j)}$. We have $\lambda_j A_{|k} = \frac{\lambda''_j}{\alpha_j}\lambda'_j A_{|k} = \frac{\lambda''_j}{\alpha_j}\alpha_j i_{j_k}$, where $i_j = (i_{j_1}, \ldots, i_{j_{m_A}})$. Since θ is localized w.r.t. $(X_0, X_1, \ldots, X_n)$, $\lambda_j A_{|k} = 0$. Therefore, for each $0 \leq j \leq n$ the condition $\bigwedge_{k \in \overline{Ix(j)}} \lambda_j A_{|k} = 0$ is satisfied.

We have $\lambda A = \big(\sum_{j=0}^{n} \lambda_j\big)A = \sum_{j=0}^{n}(\frac{\lambda''_j}{\alpha_j}\lambda'_j A) = \sum_{j=0}^{n}(\frac{\lambda''_j}{\alpha_j}\alpha_j i_j) = \sum_{j=0}^{n}(\lambda''_j i_j) = \lambda'' I$. Therefore, $\lambda A + \mu B = 0$. Similarly, $\lambda a \leq \lambda'' d$. Therefore, one of the following cases must hold. In case (1), $\lambda'' d + \mu b \leq -1$ and thus, $\lambda a + \mu b \leq -1$. In case (2), $\lambda'' d + \mu b \leq 0$ and $\lambda''^{\mathsf{lt}} \neq 0$. Thus, $\lambda a + \mu b \leq 0$. Since $\lambda''^{\mathsf{lt}} \neq 0$, for some j we have that θ_j is strict and $\lambda''_j \neq 0$. There are two possible cases. If (2.1) $\lambda_j'^{\mathsf{lt}} \neq 0$, then $\lambda_j^{\mathsf{lt}} \neq 0$ and hence $\lambda^{\mathsf{lt}} \neq 0$. Otherwise, we have (2.2) $\lambda_j'^{\mathsf{lt}} = 0$. As θ_j is strict, we have that $\lhd'_j$ is $\leq$. Therefore, Theorem 5.3.2, it holds that $\lambda_j'^{\mathsf{lt}}a^{\mathsf{lt}} + (\lambda_j'^{\mathsf{le}}\alpha_j)\binom{a^{\mathsf{le}}}{-\delta_j} < 0$. Hence, $\lambda'_j a < \alpha_j \delta_j$ and thus, $\lambda a + \mu b < \lambda'' d + \mu b$. This implies $\lambda a + \mu b < 0$. We can scale the vectors λ_k for $0 \leq k \leq n$ (and hence λ) and μ accordingly so that $\lambda a + \mu b \leq -1$. In case (3) $\lambda'' d + \mu b \leq 0$ and $\mu^{\mathsf{lt}} \neq 0$. Thus, also $\lambda a + \mu b \leq 0$. Thus, in all the possible cases, the guard of some of the if branches of algorithm `LILA` will be satisfied and an interpolant with the required properties will be returned. □

The procedure LocalizedInterpolant$_{LRA}$, given as Algorithm 5, implements LocalizedInterpolant for the theory of Linear Rational Arithmetic (LRA). The formulas are rewritten as disjunctions of mixed systems of inequalities and localized

interpolants θ_{kl} are computed for each pair of disjuncts. If for a pair $(A_k x \leqslant a_k, B_l x \leqslant b_l)$ of systems the procedure `LILA` returns $\bot$, it is called again with the arguments $A_k x \leqslant a_k$ and $B_l x \leqslant b_l$ swapped. Thus, if it succeeds to compute an interpolant θ'_{kl} for $(B_l x \leqslant b_l, A_k x \leqslant a_k)$, we let $\theta_{kl} = \neg\theta'_{kl}$, since $\neg\theta'_{kl}$ is an interpolant for $(A_k x \leqslant a_k, B_l x \leqslant b_l)$. Since $\theta'_{kl} = \bigwedge_j i'_j x \lhd'_j \delta'_j$, we have $\theta_{kl} = \bigvee_j i_j x \lhd_j \delta_j$, where $i_j = -i'_j$, $\delta_j = -\delta'_j$ and $\lhd_j = \leq$ iff $\lhd'_j = <$. Thus, each θ_{kl} is in this case of the from $(\bigvee \mathsf{LI})$.

Algorithm: LocalizedInterpolant$_{LRA}$

Input: *LRA* formulas φ and ψ such that $\varphi \wedge \psi$ is `UNSAT`
partitioning $(X_0, X_1, \ldots, X_n)$ of $Vars(\varphi) \cup Vars(\psi)$
Output: localized interpolant θ for (φ, ψ) or $\bot$

$\left(\bigvee_k A_k x \leqslant a_k\right) :=$ MixedSystems(φ);
$\left(\bigvee_l B_l x \leqslant b_l\right) :=$ MixedSystems(ψ);
foreach k,l **do**
  $\theta_{k,l} :=$ `LILA`$(A_k x \leqslant a_k, B_l x \leqslant b_l, (X_0, X_1, \ldots, X_n))$;
  if $\theta_{k,l} = \bot$ **then** $\theta_{k,l} := \neg$`LILA`$(B_l x \leqslant b_l, A_k x \leqslant a_k, (X_0, X_1, \ldots, X_n))$;
  if $\theta_{k,l} = \bot$ **then return** $\bot$;
return $\bigvee_k \bigwedge_l \theta_{kl}$;

Algorithm 5: Computing localized interpolants for pairs of formulas.

$$\theta = \bigvee_{j=0}^{n} i_j x \lhd_j \delta_j \qquad (\bigvee \mathsf{LI})$$

The following corollary of Theorem 5.3.4 establishes the soundness and relative completeness of algorithm LocalizedInterpolant$_{LRA}$.

Corollary 5.3.1. *Algorithm* LocalizedInterpolant$_{LRA}$ *is sound: If, given formulas φ and ψ and a variable partitioning $(X_0, X_1, \ldots, X_n)$, it returns a formula θ, then θ is an interpolant for (φ, ψ) that is localized w.r.t. $(X_0, X_1, \ldots, X_n)$.*

Furthermore, algorithm LocalizedInterpolant$_{LRA}$ *is guaranteed to find an interpolant for $\left(\bigvee_k A_k x \leqslant a_k, \bigvee_l B_l x \leqslant b_l\right)$ that is localized w.r.t. $(X_0, X_1, \ldots, X_n)$, if for each pair $(A_k x \leqslant a_k, B_l x \leqslant b_l)$ of disjuncts there exists an interpolant θ that is localized w.r.t. $(X_0, X_1, \ldots, X_n)$ and satisfies $(\bigwedge \mathsf{LI})$ or $(\bigvee \mathsf{LI})$.*

To the best of our knowledge, no other interpolation procedure allows for requiring interpolants that are localized in the sense we discussed in this section. Given a pair of mixed systems and a variable partitioning, a localized interpolant may not always exist. Even if one does exist, it might not be of the conjunctive ($\bigwedge \mathsf{LI}$) or disjunctive ($\bigvee \mathsf{LI}$) forms. If this happens to be the case, the procedure LocalizedInterpolant$_{LRA}$ will detect this and return $\bot$. One option to address this situation is to try to use the outer refinement procedure to infer information from "partial results" of the call to LocalizedInterpolant$_{LRA}$ and the counterexample under scrutiny, and restart LocalizedInterpolant$_{LRA}$ with a "more localized" query. Such a procedure, however will be incomplete as it will not be able to detect if a localized interpolant of a general form does not exist. In Section 5.4.4 we describe such an approach that introduces disjunctions in an enumerative way and is guaranteed to produce a localized interpolant in a certain finite language, if such interpolant exists.

5.4 Abstraction Refinement Loop

The CEGAR-based method for solving games under incomplete information presented in this section follows the classical abstraction-refinement scheme. It employs the abstraction procedure from Section 2.3 and the refinement procedure from Section 5.3 that address the problems specific to games under incomplete information. Before we describe the CEGAR loop for games under incomplete information, we have to consider the case when the abstract counterexample tree contains an unconcretizable trace and provide a procedure for generating refinement predicates for this case.

5.4.1 Transition Relation Refinement

If for some trace $\tau_0 \in \mathit{Traces}(C_t^{\#})$ the formula $\varphi_{\mathit{trace}}(\tau_0)$ is unsatisfiable, then the occurrence of the spurious abstract counterexample tree $C_t^{\#}$ is due to the approximations of the transition relations. One way to compute refinement predicates sufficient to make these approximations more precise and eliminate $C_t^{\#}$ is to annotate each node n in $C_t^{\#}$ with a formula $\varphi_{\mathit{state}}(n, \tau_0)$, which denotes the subset of $\gamma({L_s}^{\#}(n))$ that consist of those states in G from which there exists a concrete counterexample path for the respective suffix of τ_0 in G that is contained in some path in the subtree of $C_t^{\#}$ rooted at n.

State formulas. For a trace $\tau_0 \in \mathit{Traces}(C_t^\#)$, the subtree of $C_t^\#$ induced by τ_0 is a labeled tree $\mathsf{subtree}(C_t^\#, \tau_0) = (N_0^\#, E_0^\#, {L_s}^\#, {L_a}^\#)$, where $N_0^\# \subseteq N^\#$ and $E_0^\# \subseteq E^\#$ are the smallest sets such that for every $\rho \in \mathit{Paths}(\tau_0)$ and every $0 \leq i < |\rho|$ it holds that $\rho[i] \in N_0^\#$ and if $i < |\rho| - 1$ then $(\rho[i], \rho[i+1]) \in E_0^\#$. For every $n \in N_0^\#$ with ${L_s}^\#(n) \in S_\exists^\#$ and $l < |\tau_0|$, where $l = |\mathit{path}(n)|_\exists$, we define $\mathsf{action}(n, \tau_0) = \tau_0[l]$ and for every $n \in N_0^\#$ with with ${L_s}^\#(n) \in S_\exists^\#$ and $|\mathit{path}(n)|_\exists \geq |\tau_0|$, we define $\mathsf{action}(n, \tau_0) = \epsilon$.

For a node n in $\mathsf{subtree}(C_t^\#, \tau_0)$ we recursively define the *formula* $\varphi_{\mathit{state}}(n, \tau_0)$:

- If n is a leaf node in $\mathsf{subtree}(C_t^\#, \tau_0)$ and ${L_s}^\#(n) \in S_\forall^\#$, or ${L_s}^\#(n) \in S_\exists^\#$ and $\mathsf{action}(n, \tau_0) = \epsilon$, then we define

$$\varphi_{\mathit{state}}(n, \tau_0) = [{L_s}^\#(n)] \wedge \varphi_{\mathit{Err}},$$

- Otherwise, if ${L_s}^\#(n) \in S_\exists^\#$ and $\mathsf{action}(n, \tau_0) = \sigma$ we let $N' = \{n' \mid (n, n') \in E_0^\#\}$ and define

$$\varphi_{\mathit{state}}(n, \tau_0) = [{L_s}^\#(n)] \wedge \Big(\neg\mathsf{Enabled}(\sigma) \vee \mathsf{Pre}_\exists\big(\bigvee_{n' \in N'} \varphi_{\mathit{state}}(n', \tau_0), \sigma\big) \vee \varphi_{\mathit{Err}}\Big),$$

- Otherwise, if ${L_s}^\#(n) \in S_\forall^\#$, then n has a (single) child n' in $C_t^\#$ and we define

$$\varphi_{\mathit{state}}(n, \tau_0) = [{L_s}^\#(n)] \wedge \big(\mathsf{Pre}_\forall(\varphi_{\mathit{state}}(n', \tau_0)) \vee \varphi_{\mathit{Err}}\big).$$

The following lemma establishes the connection between state formulas and trace formulas for a counterexample tree.

Lemma 5.4.1. *Let* $\mathsf{Safety}(\mathcal{G}, \varphi_{\mathit{Err}})$ *be a safety game under incomplete information,* $\mathcal{P} \subseteq \mathcal{AP}[V]$ *be a finite set of predicates that is precise w.r.t. the turn variable* t *and let* $G^\# = \mathsf{Abstract}(\mathcal{G}, \mathcal{P})$ *and* $\mathit{Err}^\# = \{s^\# \in S^\# \mid [\![s^\#]\!] \cap [\![\varphi_{\mathit{Err}}]\!] \neq \emptyset\}$.

Suppose that $C_t^\#$ *is an abstract counterexample tree in* $\mathsf{Safety}(G^\#, \mathit{Err}^\#)$ *and* $\tau \in \mathit{Traces}(C_t^\#)$. *If* n_0 *is the root of* $C_t^\#$, *then the formula* $\varphi_{\mathit{state}}(n_0, \tau) \wedge \varphi_{\mathit{Init}}$ *is satisfiable iff there exists a path* $\rho \in \mathit{Paths}(\tau)$ *and a prefix* $\pi \in \gamma(\mathit{pref}(\rho))$ *such that* $\pi \models \varphi_{\mathit{trace}}(\tau)$.

Refinement predicates. The procedure RefineTransitionRelations given as Algorithm 6 annotates in a bottom up manner each node of $\mathsf{subtree}(C_t^\#, \tau_0)$ with the respective formula $\varphi_{\mathit{state}}(n, \tau_0)$ and collects the predicates extracted from the computed formulas after renaming the indexed variables to the original variables form V. Additionally the set $\mathcal{R}$ also includes the predicates in $\mathsf{Predicates}_\Sigma(\tau_0)$ which will ensure that the refined abstraction is precise w.r.t. the outputs from τ_0.

Algorithm: RefineTransitionRelations

Input: safety game $\mathsf{Safety}(\mathcal{G}, \varphi_{Err})$, finite set of predicates $\mathcal{P}$,
abstract game $\mathsf{Safety}(\mathsf{Abstract}(\mathcal{G}, \mathcal{P}), Err^{\#})$, counterexample tree $C_t^{\#}$
trace $\tau_0 \in \mathit{Traces}(C_t^{\#})$ for which the formula $\varphi_{trace}(\tau_0)$ is `UNSAT`

Output: finite set $\mathcal{R}$ of refinement predicates

$\mathcal{R} := \mathsf{Predicates}_{\Sigma}(\tau_0)$;
$(N_0^{\#}, E_0^{\#}, {L_s}^{\#}, {L_a}^{\#}) = \mathsf{subtree}(C_t^{\#}, \tau_0)$;

forall $n \in N_0^{\#}$ **in a bottom-up manner do**
  construct $\varphi = \varphi_{state}(n, \tau_0)$;
  $\mathcal{R} := \mathcal{R} \cup \mathsf{ExtractPredicates}(\varphi)$;

return $\mathcal{R}$

Algorithm 6: Refinement Predicates for the Transition Relations.

Interpolation-based transition relation refinement. An alternative to the transition relation refinement described above, would be to use Craig interpolation in the classical way as done in verification. We apply this refinement when for some $\tau_0 \in \mathit{Traces}(C_t^{\#})$ the formula $\varphi_{trace}(\tau_0)$ is unsatisfiable, which means that for each $\rho \in \mathit{Paths}(\tau_0)$ the respective formula $\varphi_{path}(\tau_0, \rho)$ is unsatisfiable. As $\varphi_{path}(\tau, \rho_0)$ is a classical path formula, interpolation can be used to compute suitable predicates.

Example 5.4.1. For the game from Example 2.1.1 and the abstract counterexample tree shown in Figure 5.2, the trace formula $\varphi_{trace}(\tau)$ for the trace $\tau = N \cdot N$ is unsatisfiable. One way to compute a set of refinement predicates from $\varphi_{trace}(\tau)$ is to use the procedure RefineTransitionRelations as described above. Alternatively, we can use interpolation in this case as well. Here, in this way we can extract the predicates $(\mathit{steps} > 2)$ and $(y \geq 3)$, associated with nodes 0 and 1. These predicates suffice to eliminate the given abstract counterexample tree from subsequent abstractions. In order to reach a bad state with a concrete path corresponding to the abstract path for τ, at the second step of the path (corresponding to the abstract state at node 1) we should have $\mathit{steps} > 2$, which, together with $x < 6$ leads to $(\mathit{steps} > 3 \wedge x < 6)$ in the next step or, we should have $y \geq 3$ which would lead to $y \geq 4$ in the next state.

5.4.2 CEGAR Loop

The CEGAR loop for safety games under incomplete information is given as Algorithm 7. Its input is a symbolic safety game under incomplete information $\mathsf{Safety}(\mathcal{G}, \varphi_{Err})$ and if it terminates it returns either a finite-state winning strategy for $Player_\exists$ (as a memoryless abstract strategy) or a concretizable abstract counterexample tree.

The procedure InitialPredicates extracts the set of initial abstraction predicates for the game $\mathsf{Safety}(\mathcal{G}, \varphi_{Err})$. We let $\textsc{InitialPredicates}(\mathcal{G}, \varphi_{Err}) = \mathit{Preds}(\varphi_{Err}) \cup \{t = \exists, t = \forall\}$, i.e., the initial abstraction is precise w.r.t. the variable t and contains the predicates occurring in the formula φ_{Err} describing the error states.

The procedure AbstractGame constructs the abstraction $\mathsf{Abstract}(\mathcal{G}, \mathcal{P})$ of $\mathcal{G}$ w.r.t. $\mathcal{P}$ and the set of abstract error states $Err^\# = \{s^\# \in S^\# \mid [\![s^\#]\!] \cap [\![\varphi_{Err}]\!] \neq \emptyset\}$. The finite-state perfect-information safety game $\mathsf{Safety}(G^\#, Err^\#)$ is solved by the procedure SolveGame, which returns an abstract winning strategy for $Player_\exists$ if $Player_\exists$ wins the game $\mathsf{Safety}(G^\#, Err^\#)$, or an abstract counterexample tree otherwise. In the first case SolveIncompteteInformationGame returns the strategy for $Player_\exists$.

Otherwise, the counterexample tree $C_t^\# = \textsc{CounterexampleTree}(\mathit{strategy})$ is passed to the procedure Concretizable which constructs the tree formula $\mathsf{TF}(C_t^\#)$ for the given abstract counterexample tree, computing the set of traces $\mathit{Traces}(C_t^\#)$ and the set of trace formulas $\mathit{Formulas}(C_t^\#) = \{\varphi_{trace}(\tau) \mid \tau \in \mathit{Traces}(C_t^\#)\}$. If $C_t^\#$ is spurious, the procedure $\textsc{SpuriousTrace}(\mathit{Formulas}(C_t^\#), \mathit{Traces}(C_t^\#))$ checks if for some trace $\tau \in \mathit{Traces}(C_t^\#)$ the trace formula $\varphi_{trace}(\tau)$ is unsatisfiable and if this is the case, returns one such τ, and otherwise returns $\bot$. In the first case, the set of refinement predicates $\mathcal{R}$ is computed by the procedure RefineTransitionRelations, that takes the trace τ and $C_t^\#$ and returns predicates that refine the transition relations. In the latter case, the procedure RefineObservations computes from $\mathit{Formulas}(C_t^\#)$, using interpolation, a set of predicates that refine the abstract observation equivalence.

For Sections 5.4.2 and 5.4.3 we assume that RefineObservations does not abort.

Definition 5.4.1. Let $\mathsf{Safety}(\mathcal{G}, \varphi_{Err})$ be a safety game under incomplete information with $\mathcal{G} = (V_\exists, V_\forall, V_\forall^o, t, \varphi_{Init}, \mathcal{T}_\exists, \mathcal{T}_\forall)$ and $\mathcal{P}$ and $\mathcal{P}'$ be finite sets of predicates precise w.r.t. t such that $\mathcal{P} \subseteq \mathcal{P}'$. Let $\mathsf{Safety}(G^\#, Err^\#)$ and $\mathsf{Safety}(G', Err')$ be the safety games such that $G^\# = \mathsf{Abstract}(\mathcal{G}, \mathcal{P}) = (S_\exists^\#, S_\forall^\#, I^\#, =_\mathsf{o}^\#, \Sigma_\exists^\#, T_\exists^\#, T_\forall^\#)$, $Err^\# = \{s^\# \in S^\# \mid [\![s^\#]\!] \cap [\![\varphi_{Err}]\!] \neq \emptyset\}$, $G' = \mathsf{Abstract}(\mathcal{G}, \mathcal{P}') = (S_\exists', S_\forall', I', =_\mathsf{o}', \Sigma_\exists', T_\exists', T_\forall')$ and $Err' = \{s' \in S' \mid [\![s']\!] \cap [\![\varphi_{Err}]\!] \neq \emptyset\}$. If $C_t^\# = (N^\#, E^\#, {L_s}^\#, {L_a}^\#)$ and $C_t = (N', E', {L_s}', {L_a}')$

are abstract counterexample trees in $\mathsf{Safety}(G^{\#}, Err^{\#})$ and $\mathsf{Safety}(\mathcal{G}', Err')$ respectively, we write $C'_t \preceq C^{\#}_t$ iff for every path ρ' in C'_t there exists a path ρ in $C^{\#}_t$ such that: (i) $|\rho'| = |\rho|$, (ii) for every $0 \leq i < |\rho|$, $\gamma({L_s}'(\rho'[i])) \subseteq \gamma({L_s}^{\#}(\rho[i]))$ and (iii) for every $0 \leq i < |\rho| - 1$, $\gamma_\exists({L_a}'(\rho'[i], \rho'[i+1])) \subseteq \gamma_\exists({L_a}^{\#}(\rho[i], \rho[i+1]))$.

It is easy to see that the following properties hold for counterexample trees $C^{\#}_t = (N^{\#}, E^{\#}, {L_s}^{\#}, {L_a}^{\#})$ and $C'_t = (N', E', {L_s}', {L_a}')$ with $C'_t \preceq C^{\#}_t$.

Property 14. *If $n_1 \in N^{\#}$ and $n'_1 \in N'$ are such that $\gamma({L_s}'(n'_1)) \subseteq \gamma({L_s}^{\#}(n_1)) \subseteq S_\forall$, and there exist $(n_1, n_2) \in E^{\#}$ and $(n'_1, n'_2) \in E'$, then $\gamma({L_s}'(n'_2)) \subseteq \gamma({L_s}^{\#}(n_2))$.*

Property 15. *If $n_1 \in N^{\#}$ and $n'_1 \in N'$ are such that $\gamma({L_s}'(n'_1)) \subseteq \gamma({L_s}^{\#}(n_1)) \subseteq S_\exists$ and $(n_1, n_2) \in E^{\#}$, then for every $\sigma \in \gamma_\exists({L_a}^{\#}((n_1, n_2)))$, if there exists $(n'_1, n'_2) \in E'$ such that $\sigma \in \gamma_\exists({L_a}'((n'_1, n'_2)))$, then $\gamma({L_s}'(n'_2)) \subseteq \gamma({L_s}^{\#}(n_3))$ for some $(n_2, n_3) \in E^{\#}$.*

The above properties imply the following.

Property 16. *For every $\tau \in \mathit{Traces}(C^{\#}_t)$ there exists $\tau' \in \mathit{Traces}(C'_t)$ such that $\tau = \tau' \cdot \tau''$ for some $\tau'' \in \Sigma^*_\exists$ and such that for every $\rho' \in \mathit{Paths}(\tau')$ there exists $\rho \in \mathit{Paths}(\tau)$ such that $|\rho'| = |\rho|$ and for every $0 \leq i < |\rho|$ it holds that $\gamma({L_s}'(\rho'[i])) \subseteq \gamma({L_s}^{\#}(\rho[i]))$.*

Let $\mathcal{R}$ be the set of predicates returned by RefineObservations. In general, these predicates do not suffice to eliminate all counterexample trees $C'_t \preceq C^{\#}_t$ from the subsequent refined abstractions. Indeed, they allow for distinguishing the sets of concrete error paths for different traces, but may not be enough to precisely describe these sets. However, we show that the procedure RefineTransitionRelations can be used to compute additional predicates from these counterexample trees that suffice for successfully removing them. The procedure RefineTree computes the set *Trees* of counterexample trees C'_t in $\mathsf{Safety}(G', Err')$, where $G' = \mathsf{Abstract}(\mathcal{G}, \mathcal{P} \cup \mathcal{R}) = (S'_\exists, S'_\forall, I', ='_{\mathsf{o}}, \Sigma'_\exists, T'_\exists, T'_\forall)$ and $Err' = \{s' \in S' \mid [\![s']\!] \cap [\![\varphi_{Err}]\!] \neq \emptyset\}$ that are such that $C'_t \preceq C^{\#}_t$. By Lemma 5.4.2 below, for each $C'_t \in \mathit{Trees}$ there exists a trace $\tau' \in \mathit{Traces}(C'_t)$ such that $\varphi_{\mathit{trace}}(\tau')$ is unsatisfiable. For each C'_t and one such τ' the set of predicates computed by RefineTransitionRelations are included in the set $\mathcal{R}$.

The procedure RefineGame refines the game structure $G^{\#}$ and the set of error states $Err^{\#}$ w.r.t. the extended set $\mathcal{P}$ of predicates, such that $G^{\#} = \mathsf{Abstract}(\mathcal{G}, \mathcal{P})$.

The following lemma tells us that we can apply RefineTransitionRelations after RefineObservations as described above to ensure that all subsumed counterexample trees are successfully eliminated from subsequent abstractions.

Algorithm: SolveIncompleteteInformationGame

Input: safety game $\mathsf{Safety}(\mathcal{G}, \varphi_{Err})$ with $\mathcal{G} = (V_\exists, V_\forall, V_\forall^o, t, \varphi_{Init}, \mathcal{T}_\exists, \mathcal{T}_\forall)$
Output: $(winner, abstract\ strategy)$ or $(winner, abstract\ counterexample\ tree)$

$\mathcal{P} :=$ InitialPredicates$(\mathcal{G}, \varphi_{Err})$;
$(G^\#, Err^\#) :=$ AbstractGame$(\mathcal{G}, \varphi_{Err}, \mathcal{P})$;
$(winner, strategy) :=$ SolveGame$(G^\#, Err^\#)$;

while $winner = Player_\forall$ **do**
  $C_t^\# :=$ CounterexampleTree$(strategy)$;
  if Concretizable $(C_t^\#, G)$ **then**
    return $(winner, C_t^\#)$; `/* concretizable counterexample */`
  $\mathcal{R} := \emptyset$;
  $\tau :=$ SpuriousTrace$(Formulas(C_t^\#), Traces(C_t^\#))$;
  if $\tau \neq \bot$ **then** `/*` $\tau \in Traces(C_t^\#)$ `with` $\varphi_{trace}(\tau)$ `UNSAT */`
    $\mathcal{R} :=$ RefineTransitionRelations$(\tau, C_t^\#)$;
  else `/*` $\varphi_{trace}(\tau)$ `SAT for all` $\tau \in Traces(C_t^\#)$ `*/`
    $\mathcal{R} :=$ RefineObservations$(Formulas(C_t^\#), Traces(C_t^\#))$;
    $Trees :=$ RefineTree$(C_t^\#, \mathcal{P}, \mathcal{R})$;
    forall $C_t' \in Trees$ **do**
      $\tau' :=$ SpuriousTrace$(Formulas(C_t'), Traces(C_t'))$;
      $\mathcal{R} := \mathcal{R} \cup$ RefineTransitionRelations(τ', C_t');
  $\mathcal{P} := \mathcal{P} \cup \mathcal{R}$;
  $(G^\#, Err^\#) =$ RefineGame$(G^\#, \mathcal{G}, \mathcal{P})$;
  $(winner, strategy) =$ SolveGame$(G^\#, Err^\#)$;

return $(winner, strategy)$ `/* winning strategy for` $Player_\exists$ `*/`

Algorithm 7: CEAGR for games under incomplete information.

Lemma 5.4.2. *Let* $\mathsf{Safety}(\mathcal{G}, \varphi_{Err})$ *be a safety game under incomplete information with* $\mathcal{G} = (V_\exists, V_\forall, V_\forall^o, t, \varphi_{Init}, \mathcal{T}_\exists, \mathcal{T}_\forall)$ *and* $\mathcal{P}$ *and* $\mathcal{P}'$ *be finite sets of predicates precise w.r.t.* t *such that* $\mathcal{P} \subseteq \mathcal{P}'$. *Let* $\mathsf{Safety}(G^\#, Err^\#)$ *and* $\mathsf{Safety}(G', Err')$ *be the safety games such that* $G^\# = \mathsf{Abstract}(\mathcal{G}, \mathcal{P}) = (S_\exists^\#, S_\forall^\#, I^\#, =_\mathsf{o}^\#, \Sigma_\exists^\#, T_\exists^\#, T_\forall^\#)$, $Err^\# = \{s^\# \in S^\# \mid [\![s^\#]\!] \cap [\![\varphi_{Err}]\!] \neq \emptyset\}$, $G' = \mathsf{Abstract}(\mathcal{G}, \mathcal{P}') = (S_\exists', S_\forall', I', =_\mathsf{o}', \Sigma_\exists', T_\exists', T_\forall')$ *and* $Err' = \{s' \in S' \mid [\![s']\!] \cap [\![\varphi_{Err}]\!] \neq \emptyset\}$. *Suppose that* $C_t^\# = (N^\#, E^\#, {L_s}^\#, {L_a}^\#)$ *is a spurious counterexample tree in* $\mathsf{Safety}(G^\#, Err^\#)$ *such that each* $\varphi \in Formulas(C_t^\#)$

is satisfiable and $\mathcal{P}' \supseteq$ RefineObservations($Formulas(C_t^{\#}), Traces(C_t^{\#})$). *Then, if* $C_t' = (N', E', {L_s}', {L_a}')$ *is a counterexample tree in* $\mathsf{Safety}(G', Err')$ *such that* $C_t' \preceq C_t^{\#}$, *then there exists a* $\tau' \in Traces(C_t')$ *such that the formula* $\varphi_{trace}(\tau)$ *is unsatisfiable.*

Proof. Suppose that for each trace $\tau' \in Traces(C_t')$ the formula $\varphi_{trace}(\tau')$ is satisfiable.

Let $\tau \in Traces(C_t^{\#})$. By Property 16 there exists a trace $\tau' \in Traces(C_t')$ such that $\tau = \tau' \cdot \tau''$ for some $\tau'' \in \Sigma_\exists^*$ and for every path $\rho' \in Paths(\tau')$ there exists a path $\rho \in Paths(\tau)$ such that $|\rho'| = |\rho|$ and for every $i < |\rho|$, it holds that $\gamma({L_s}'(\rho'[i])) \subseteq \gamma({L_s}^{\#}(\rho^{\#}[i]))$. Since $\varphi_{trace}(\tau')$ is satisfiable, there exists a $\rho' \in Paths(\tau')$ such that $\varphi_{path}(\tau', \rho')$ is satisfiable. By the above, there exists a path $\rho \in Paths(\tau)$ such that $|\rho'| = |\rho|$ and for every $0 \leq i < |\rho|$, it holds that $\gamma({L_s}'(\rho'[i])) \subseteq \gamma({L_s}^{\#}(\rho^{\#}[i]))$. This implies that there exists a path π in G such that $\pi \in \gamma(\rho')$ and $\pi \models \varphi_{trace}(\tau)$.

Thus, we showed that for every $\tau \in Traces(C_t^{\#})$ there exist a prefix ρ_τ' in C_t' and prefix π_τ in G such that $\pi_\tau \in \gamma(\rho_\tau')$ and $\pi_\tau \models \varphi_{trace}(\tau)$. According to Lemma 5.3.1, since $\mathcal{P}' \supseteq$ RefineObservations($Formulas(C_t^{\#}), Traces(C_t^{\#})$), there exist traces $\tau_1, \tau_2 \in Traces(C_t^{\#})$ with $\tau_1 \neq \tau_2$ and $\mathsf{Predicates}_\Sigma(\{\tau_1, \tau_2\}) \subseteq \mathcal{P}'$, and a predicate $\psi \in \mathcal{P}'$, and a position $j < \mathsf{diff}(\pi_{\tau_1}, \pi_{\tau_2}, \tau_1, \tau_2)$, such that $\pi_{\tau_1}[j] \models \psi$ and $\pi_{\tau_2}[j] \models \neg\psi$.

Let ρ'_{τ_1} and ρ'_{τ_2} be the corresponding prefixes in C_t' for τ_1 and τ_2. Since $\pi_{\tau_1} \in \gamma(\rho'_{\tau_1})$, $\pi_{\tau_2} \in \gamma(\rho'_{\tau_2})$ and $\mathsf{Predicates}_\Sigma(\{\tau_1, \tau_2\}) \subseteq \mathcal{P}'$, it holds that for all $i < \mathsf{diff}(\pi_{\tau_1}, \pi_{\tau_2}, \tau_1, \tau_2)$, $\rho'_{\tau_1}[i] = \rho'_{\tau_2}[i]$. Thus, since $\psi \in \mathcal{P}'$ and ψ is an observable predicate, either $\pi_{\tau_1}[j] \models \psi$ and $\pi_{\tau_2}[j] \models \psi$ or $\pi_{\tau_1}[j] \models \neg\psi$ and $\pi_{\tau_2}[j] \models \neg\psi$, which contradicts the choice of j. □

5.4.3 Soundness and Progress

The soundness of the predicate abstraction of an incomplete information game and the correctness of the algorithm for checking an abstract counterexample for spuriousness imply the soundness of SolveIncompteteInformationGame.

Theorem 5.4.3. *The procedure* SolveIncompteteInformationGame *is sound: if it terminates for a game under incomplete information* $\mathsf{Safety}(G, \varphi_{Err})$ *and returns*

- $(\exists, f_\exists^{\#})$, *then there exists a winning strategy for* $Player_\exists$ *in* $\mathsf{Safety}(G, \varphi_{Err})$,
- $(\forall, C_t^{\#})$, *then there exists a counterexample tree* C_t *in* $\mathsf{Safety}(G, \varphi_{Err})$.

Proof. The first case follows from Theorem 5.1.1 and the second case follows from Theorem 5.2.1. □

The progress property of the refinement, namely, that the generated predicates suffice to eliminate from further abstractions the counterexample considered at the

current iteration of the CEGAR loop, relies on Lemma 5.4.2 and the following lemma, in which we formalize and prove the progress property of RefineTransitionRelations.

Lemma 5.4.4. *Let* $\mathsf{Safety}(\mathcal{G}, \varphi_{Err})$ *be a safety game under incomplete information with* $\mathcal{G} = (V_\exists, V_\forall, V_\forall^o, t, \varphi_{Init}, \mathcal{T}_\exists, \mathcal{T}_\forall)$ *and* $\mathcal{P}$ *and* $\mathcal{P}'$ *be finite sets of predicates precise w.r.t.* t *such that* $\mathcal{P} \subseteq \mathcal{P}'$. *Let* $\mathsf{Safety}(G^\#, Err^\#)$ *and* $\mathsf{Safety}(G', Err')$ *be the safety games such that* $G^\# = \mathsf{Abstract}(\mathcal{G}, \mathcal{P}) = (S_\exists^\#, S_\forall^\#, I^\#, =_\mathsf{o}^\#, \Sigma_\exists^\#, T_\exists^\#, T_\forall^\#)$, $Err^\# = \{s^\# \in S^\# \mid [\![s^\#]\!] \cap [\![\varphi_{Err}]\!] \neq \emptyset\}$, $G' = \mathsf{Abstract}(\mathcal{G}, \mathcal{P}') = (S'_\exists, S'_\forall, I', ='_\mathsf{o}, \Sigma'_\exists, T'_\exists, T'_\forall)$ *and* $Err' = \{s' \in S' \mid [\![s']\!] \cap [\![\varphi_{Err}]\!] \neq \emptyset\}$. *Suppose that* $C_t^\# = (N^\#, E^\#, {L_s}^\#, {L_a}^\#)$ *is a spurious counterexample tree in* $\mathsf{Safety}(G^\#, Err^\#)$ *and* $\tau \in Traces(C_t^\#)$ *is such that* $\varphi_{trace}(\tau)$ *is unsatisfiable and that* $\mathcal{P}' \supseteq \textsc{RefineTransitionRelations}(\tau, C_t^\#)$. *Then, in* $\mathsf{Safety}(C'_t, Err')$ *there does not exist a counterexample tree* C'_t *such that* $C'_t \preceq C_t^\#$.

Proof. Assume that $C'_t = (N', E', {L_s}', {L_a}')$ is a counterexample tree in $\mathsf{Safety}(C'_t, Err')$ such that $C'_t \preceq C_t^\#$. According to Property 16, there exists $\tau' \in Traces(C'_t)$ such that $\tau = \tau' \cdot \tau''$ for some $\tau'' \in \Sigma_\exists^*$ and such that for every $\rho' \in Paths(\tau')$ there exists $\rho \in Paths(\tau)$ such that $|\rho'| = |\rho|$ and for every $0 \leq i < |\rho|$ it holds that $\gamma({L_s}'(\rho'[i])) \subseteq \gamma({L_s}^\#(\rho[i]))$.

Since $\mathsf{Predicates}_\Sigma(\{\tau\}) \subseteq \textsc{RefineTransitionRelations}(\tau, C_t^\#) \subseteq \mathcal{P}'$ and there exists a $\rho' \in Paths(\tau')$, it holds that there exists a $s \in \gamma({L_s}'(n'_0))$ such that $s \models \varphi_{state}(n_0, \tau)$, where n'_0 is the root of C'_t and n_0 is the root of $C_t^\#$. That is, the formula $\varphi_{state}(n_0, \tau)$ is satisfiable, and hence $\varphi_{trace}(\tau)$ is too, which is a contradiction. □

Theorem 5.4.5. *Let* $\mathsf{Safety}(\mathcal{G}, \varphi_{Err})$ *be a safety game under incomplete information with* $\mathcal{G} = (V_\exists, V_\forall, V_\forall^o, t, \varphi_{Init}, \mathcal{T}_\exists, \mathcal{T}_\forall)$ *and* $\mathcal{P}$ *be a finite set of predicates precise w.r.t.* t. *Suppose that* $C_t^\#$ *is a spurious counterexample tree in* $\mathsf{Safety}(G^\#, Err^\#)$, *where* $G^\# = \mathsf{Abstract}(\mathcal{G}, \mathcal{P}) = (S_\exists^\#, S_\forall^\#, I^\#, =_\mathsf{o}^\#, \Sigma_\exists^\#, T_\exists^\#, T_\forall^\#)$ *and* $Err^\# = \{s^\# \in S^\# \mid [\![s^\#]\!] \cap [\![\varphi_{Err}]\!] \neq \emptyset\}$. *If* $\mathcal{R}$ *is the set of refinement predicates computed by* SolveIncompteteInformationGame, *and* $G' = \mathsf{Abstract}(\mathcal{G}, \mathcal{P} \cup \mathcal{R}) = (S'_\exists, S'_\forall, I', ='_\mathsf{o}, \Sigma'_\exists, T'_\exists, T'_\forall)$ *and* $Err' = \{s' \in S' \mid [\![s']\!] \cap [\![\varphi_{Err}]\!] \neq \emptyset\}$, *then in* $\mathsf{Safety}(C'_t, Err')$ *there does not exist a counterexample tree* C'_t *such that* $C'_t \preceq C_t^\#$.

Proof. The theorem is a direct consequence of Lemma 5.4.2 and Lemma 5.4.4. □

5.4.4 Relative Completeness

In this section we provide sufficient conditions for termination of the CEGAR schema given as Algorithm 7. We show that only finitely many different abstract states are generated during the execution of the procedure, provided that the concrete game

structure fulfills certain standard assumptions, as well as some conditions related to the presence of incomplete information, and the procedure for computing localized Craig interpolants obeys a given restriction regarding the resulting interpolants.

Sufficient Conditions for Termination

Definition 5.4.2. A *region algebra with observations* for a safety game $\mathsf{Safety}(G, Err)$ with game structure $G = (S_\exists, S_\forall, I, =_\mathsf{o}, \Sigma_\exists, T_\exists, T_\forall)$ and set of error states Err is a tuple $(R, R_{obs}, [\![\,]\!])$, consisting of a (possibly infinite) set of *regions* R, a (possibly infinite) set of *observable regions* $R_{obs} \subseteq R$, and a function $[\![\,]\!] : R \to 2^S$ that satisfy the conditions:

(i) R contains regions $\mathbb{T}$ and $\mathbb{F}$ such that $[\![\mathbb{T}]\!] = S$ and $[\![\mathbb{F}]\!] = \emptyset$,

(ii) R is closed under Boolean operations: for regions $r_1 \in R$ and $r_2 \in R$ there exist:

- region $r_1 \cup r_2$ such that $[\![r_1 \cup r_2]\!] = [\![r_1]\!] \cup [\![r_2]\!]$,
- region $r_1 \cap r_2$ such that $[\![r_1 \cap r_2]\!] = [\![r_1]\!] \cap [\![r_2]\!]$,
- region $r_1 \setminus r_2$ such that $[\![r_1 \setminus r_2]\!] = [\![r_1]\!] \setminus [\![r_2]\!]$,

that can be effectively computed,

(iii) R contains regions r_{Init} and r_{Err} such that $[\![r_{Init}]\!] = I$ and $[\![r_{Err}]\!] = Err$,

(iv) R_{obs} contains regions $r_\forall$ and $r_\exists$ such that $[\![r_\forall]\!] = S_\forall$ and $[\![r_\exists]\!] = S_\exists$,

(v) R is closed under predecessor operations: for every $r \in R$ and $\sigma \in \Sigma_\exists$ there exist:

- region $\mathsf{Pre}_\exists(r, \sigma)$ such that $\mathsf{Pre}_\exists(r, \sigma) = \mathsf{Pre}_\exists([\![r]\!], \sigma)$,
- region $\mathsf{Pre}_\forall(r)$ such that $\mathsf{Pre}_\forall(r) = \mathsf{Pre}_\forall([\![r]\!])$,

that can be effectively computed,

(vi) For every $\pi_1, \pi_2 \in \mathsf{Prefs}(G)$ such that each of $\mathsf{last}(\pi_1)$ and $\mathsf{last}(\pi_2)$ is a dead-end or error state, and for which there exists a j such that $\pi_1[j] \neq_\mathsf{o} \pi_2[j]$, there exist an index $0 \leq k \leq j$ and a region $r \in R_{obs}$ such that $\pi_1[k] \in [\![r]\!]$ and $\pi_2[k] \notin [\![r]\!]$.

Theorem 5.4.6. *Let* $\mathsf{Safety}(\mathcal{G}, \varphi_{Err})$ *be a symbolic safety game under incomplete information and let* $\mathsf{Safety}(G, Err)$ *be the induced explicit game under incomplete information. Assume that* $(R, R_{obs}, [\![\,]\!])$ *is region algebra for* $\mathsf{Safety}(G, Err)$ *such that:*

(i) R *is a* finite *subset of* $\mathcal{B}(\mathcal{AP})$*, where* $\mathcal{B}(\mathcal{AP})$ *is the closure of* $\mathcal{AP}$ *under* $\vee, \wedge, \neg$*,*

(ii) $\mathit{Preds}(\varphi_{\mathit{Init}}) \subseteq R$, $\mathit{Preds}(\varphi_{\mathit{Err}}) \subseteq R$, $(t = \exists) \in R$ *and* $(t = \forall) \in R$,

(iii) for every $r \in R$ *and* $\sigma \in \Sigma_\exists$, $\mathit{Preds}(\mathsf{Pre}_\exists(r, \sigma)) \subseteq R$ *and* $\mathit{Preds}(\mathsf{Pre}_\forall(r)) \subseteq R$,

(iv) for every $x \in V_\exists$ *and every* $c \in \mathsf{Dom}(x)$, *we have that* $(x = c) \in R_{\mathit{obs}}$.

Assume furthermore that LocalizedInterpolant *satisfies the following conditions:*

(LI1) if for the regions $r_1, r_2 \in R$ *and a variable partitioning* $(X_0, \ldots, X_n)$ *there exists an interpolant* θ *for* (r_1, r_2) *that is localized w.r.t.* $(X_0, \ldots, X_n)$ *and such that* $\mathit{Preds}(\theta) \subseteq R_{\mathit{obs}}$, *then* LocalizedInterpolant$(r_1, r_2, (X_0, \ldots, X_n)) \neq \bot$,

(LI2) if LocalizedInterpolant *returns a formula* θ, *then* $\mathit{Preds}(\theta) \subseteq R_{\mathit{obs}}$.

Then algorithm SolveIncompleteInformationGame *applied to* $\mathsf{Safety}(\mathcal{G}, \varphi_{\mathit{Err}})$ *is guaranteed to terminate.*

Proof. We first show that each call to the procedure LocalizedInterpolant during the execution of SolveIncompleteInformationGame on $\mathsf{Safety}(\mathcal{G}, \varphi_{\mathit{Err}})$ successfully computes an interpolant θ expressible over R_{obs}. Since R_{obs} is finite, condition (vi) from the definition of region algebra guarantees that for every two formulas with which LocalizedInterpolant is called and which are expressible over R, an interpolant expressible over R_{obs} exists. Thus, by condition (LI1), LocalizedInterpolant returns an interpolant θ. Condition (LI2) implies that $\mathit{Preds}(\theta) \subseteq R_{\mathit{obs}}$.

Thus, since each call to LocalizedInterpolant that returns an interpolant results in at least one refinement predicate which is not present in the current set of abstraction predicates, and since R_{obs} is finite, we have that LocalizedInterpolant, and hence also the procedure RefineObservations, is called only finitely many times during the execution of SolveIncompleteInformationGame on $\mathsf{Safety}(\mathcal{G}, \varphi_{\mathit{Err}})$.

Condition (ii) above implies that InitialPredicates$(\mathcal{G}, \mathit{Err}) \subseteq R$. Conditions (iii) and (iv) guarantee that each call to RefineTransitionRelations returns predicates that are elements of R. Therefore, since all refinement predicates generated by RefineObservations and RefineTransitionRelations are elements of R we conclude that for each of the constructed abstractions and state $s^\#$ in it, $[s^\#] \in R$.

According to the progress property, if SolveIncompleteInformationGame does not terminate, infinitely many spurious counterexamples are ruled out, which means that infinitely many times an abstract state is split. Since R is finite, this is not possible, and hence, SolveIncompleteInformationGame must terminate. □

Termination for Rectangular Games

Now we consider a specialization of the CEGAR schema Algorithm 7, obtained by instantiating LocalizedInterpolant with a procedure derived from the procedure LocalizedInterpolant$_{LRA}$, which in addition ensures that the computed Craig interpolants fall into a particular set of boolean combinations of inequalities. The resulting game-solving procedure is then guaranteed to terminate for all inputs where the game structure is defined by formulas in the theory of linear arithmetic and there exists a finite region algebra with observations for this game which satisfies the conditions of Theorem 5.4.6 and for which the set of observations consists of *rectangular predicates*.

Definition 5.4.3. A *rectangular predicate* φ over a set of variables X is defined by the grammar $\varphi := cx \lhd a \mid \varphi \wedge \varphi$, where $x \in X$, $c \in \{-1, 1\}$, $\lhd \in \{<, \leq\}$ and a is an integer constant. An inequality of the form $cx \lhd a$ is called *rectangular inequality*.

Definition 5.4.4. Given $m \in \mathbb{N}$, a *rectangular predicate* $\varphi = \bigwedge_{i=1}^{n} c_i x_i \lhd a_i$ is called m-bounded if $\mid c_i \mid \leq m$ for each $0 \leq i \leq n$. For a set X of variables, $\mathcal{RP}_m[X]$ is the set of all m-bounded rectangular predicates over X and $\mathcal{RP} = \bigcup_{m \in \mathbb{N}} \mathcal{RP}_m$.

We apply the standard technique (e.g, [JM06]) of restricting the language of the interpolants computed at each step to some finite language $\mathcal{L}_m$ and gradually enlarge the restriction language when necessary, in order to maintain completeness. In our case, the language at each step is defined by the m-bounded rectangular predicates, for some bound m, over the observable variables $Obs(V)$. To impose this restriction, we make use of the fact that the interpolation procedure reduces interpolant computation to constraint solving, which allows us to add constraints on the generated inequalities.

The procedure LIRect is a modification of `LILA` that receives as additional input a bound $m \in \mathbb{N}$. It ensures that in case the partitioning $(X_0, \ldots, X_n)$ consists of singleton sets, the returned interpolant θ is a rectangular predicate in $\mathcal{RP}_m$. This is achieved by adding the constraint $\chi_3 = \bigwedge_{j=0}^{n}(i_j \leq 1 \wedge i_j \geq -1 \wedge \delta_j \leq m \wedge \delta_j \geq -m)$ and requiting that the variables δ_j and the variables in the vector i_j assume integer values.

LocalizedInterpolant$_{Rect}$ is a modification of LocalizedInterpolant$_{LRA}$ that computes an interpolant θ where $Preds(\theta)$ consists of rectangular inequalities.

To achieve this, LocalizedInterpolant$_{Rect}$ first partitions the given sets of variables such that each set is a singleton, thus ensuring that only a single variable occurs in each atomic predicate. Then, it tries to compute an interpolant θ for (φ, ψ)

Algorithm:LIRect

Input: mixed systems $Ax \leqslant a$ and $Bx \leqslant b$ where $Ax \leqslant a \wedge Bx \leqslant b$ is UNSAT, partitioning $(X_0, X_1, \ldots, X_n)$ of the variables in the vector x, bound $m \in \mathbb{N}$

Output: interpolant $\bigwedge_{j=0}^{n} i_j x \lhd_j \delta_j$ with $\lhd_j \in \{\leq, <\}$ and $Vars(i_j x \lhd_j \delta_j) \subseteq X_j$ or $\bot$ if such an interpolant in $\mathcal{RP}_m$ could not be computed

$\chi_1 := \lambda \geq 0 \wedge \mu \geq 0 \wedge \lambda A + \mu B = 0;$
$\chi_2 := \lambda = \sum_{j=0}^{n} \lambda_j \wedge \bigwedge_{j=0}^{n} (\lambda_j \geq 0 \wedge i_j = \lambda_j A \wedge \delta_j = \lambda_j a \wedge \bigwedge_{k \in \overline{Ix(j)}} \lambda_j A_{|k} = 0);$
$\chi_3 = \bigwedge_{j=0}^{n} (i_j \leq 1 \wedge i_j \geq -1 \wedge \delta_j \leq m \wedge \delta_j \geq -m);$

if exist λ, μ, λ_j and integer i_j, δ_j, for $0 \leq j \leq n$
satisfying $\chi_1 \wedge \chi_2 \wedge \chi_3 \wedge \lambda a + \mu b \leq -1$
then return $\bigwedge_{j=0}^{n} i_j x \leq \delta_j$;

elif exist λ, μ, λ_j and integer i_j, δ_j, for $0 \leq j \leq n$
satisfying $\chi_1 \wedge \chi_2 \wedge \chi_3 \wedge \lambda a + \mu b \leq 0 \wedge \lambda^{\mathsf{lt}} \neq 0$
then return $\bigwedge_{0 \leq j \leq n, \lambda_j^{\mathsf{lt}} \neq 0} i_j x < \delta_j \wedge \bigwedge_{0 \leq j \leq n, \lambda_j^{\mathsf{lt}} = 0} i_j x \leq \delta_j$;

elif exist λ, μ, λ_j and integer i_j, δ_j, for $0 \leq j \leq n$
satisfying $\chi_1 \wedge \chi_2 \wedge \chi_3 \wedge \lambda a + \mu b \leq 0 \wedge \mu^{\mathsf{lt}} \neq 0$
then return $\bigwedge_{j=0}^{n} i_j x \leq \delta_j$;

else return $\bot$

Algorithm 8: Computing localized rectangular interpolants.

such that $Preds(\theta) \subseteq \mathcal{RP}_m$, where m is initially assigned the value of the global variable that stores the current bound and is incremented whenever an interpolant for the current m cannot be computed. To determine if an interpolant for the pair of mixed systems $A_k x \leqslant a_k$ and $B_l x \leqslant b_l$ can be computed with the current bound m, LocalizedInterpolant$_{Rect}$ proceeds as follows.

If LIRect does not succeed to compute an interpolant in $\mathcal{RP}_m$ neither for $(A_k x \leqslant a_k, B_l x \leqslant b_l)$ nor for $(B_l x \leqslant b_l, A_k x \leqslant a_k)$, then LocalizedInterpolant$_{Rect}$ replaces either $A_k x \leqslant a_k$ or $B_l x \leqslant b_l$ by a disjunction distinguishing the three possible cases with respect to a variable in $Vars(A_k x \leqslant a_k) \cap Vars(B_l x \leqslant b_l)$ and a constant in $\{-m, \ldots, m\}$. Then, procedure LIRect is called for each of the disjuncts. The process

is repeated until an interpolant is successfully computed for each disjunct, or no non-trivial splits are possible any more. In the latter case the bound m is increased.

Proposition 5.4.1. *Suppose that for LRA formulas φ and ψ whose conjunction is unsatisfiable there exists an interpolant θ with $Preds(\theta) \subseteq \mathcal{RP}$. Then, for any variable partitioning $(X_0, \ldots, X_n)$, $\textsc{LocalizedInterpolant}_{Rect}(\varphi, \psi, (X_0, \ldots, X_n))$ returns an interpolant θ' for (φ, ψ) localized w.r.t. $(X_0, \ldots, X_n)$ and such that $Preds(\theta') \subseteq \mathcal{RP}$. Furthermore, if $Preds(\theta) \subseteq \mathcal{RP}_m$ for some $m \geq 0$, then also $Preds(\theta') \subseteq \mathcal{RP}_m$.*

Proof. Clearly, if $\textsc{LocalizedInterpolant}_{Rect}$ returns a formula θ' it is a boolean combination of formulas returned by LIRect and hence $Preds(\theta') \subseteq \mathcal{RP}$. Since only a single variable occurs in each atom of θ, the formula θ is also localized w.r.t. $(X_0, \ldots, X_n)$.

Let θ be an interpolant for (φ, ψ) with $Preds(\theta) \subseteq \mathcal{RP}$ and m_θ be such that $Preds(\theta) \subseteq \mathcal{RP}_{m_\theta}$. Since for each bound the number of possible disjunctions introduced by $\textsc{LocalizedInterpolant}_{Rect}$ is finite, the current bound will eventually reach m_θ unless an interpolant is returned before that. When $m = m_\theta$, in the worst case all elements of $Preds(\theta)$ are considered for introducing disjunctions. Thus, at this point an interpolant in $\mathcal{RP}_m$ can be computed for each pair of disjuncts and the loop terminates. Since the process of introducing disjunctions replaces formulas with equivalent ones, the formula returned by $\textsc{LocalizedInterpolant}_{Rect}$ is an interpolant for (φ, ψ). □

Corollary 5.4.1. *Let $\mathsf{Safety}(\mathcal{G}, \varphi_{Err})$ be a symbolic safety game under incomplete information with formulas in the theory of linear rational arithmetic. Assume that there exists a region algebra $(R, R_{obs}, [\![]\!])$ for the corresponding explicit game under incomplete information $\mathsf{Safety}(G, Err)$ that satisfies the conditions (i)-(iv) from Theorem 5.4.6 and for which $R_{obs} = \mathcal{RP}_m$. Then algorithm* SolveIncompteteInformationGame*, in which* LocalizedInterpolant *is implemented by* $\textsc{LocalizedInterpolant}_{Rect}$ *applied to the game $\mathsf{Safety}(\mathcal{G}, \varphi_{Err})$ is guaranteed to terminate.*

Proof. The claim follows from Theorem 5.4.6 and Proposition 5.4.1. □

An noteworthy example of a class of infinite-state games that fulfill the conditions of the corollary above is the class of partial-observation timed games with fixed finite sets of observations which are rectangular predicates.

Algorithm: LocalizedInterpolant$_{Rect}$

Input: *LRA* formulas φ and ψ such that $\varphi \wedge \psi$ is UNSAT,
partitioning $(X_0, \ldots, X_n)$ of $Vars(\varphi) \cup Vars(\psi)$
Output: localized interpolant θ for (φ, ψ) or $\bot$

$\left(\bigvee_k A_k x \leqslant a_k\right) :=$ MixedSystems(φ);
$\left(\bigvee_l B_l x \leqslant b_l\right) :=$ MixedSystems(ψ);
$(X'_0, \ldots, X'_{n'}) :=$ SingletonSets$((X_0, \ldots, X_n))$;
$m :=$ *CurrentBound*;
foreach *k,l* **do**
  $\mathcal{A}_k := \{A_k x \leqslant a_k\}$;
  $\mathcal{B}_l := \{B_l x \leqslant b_l\}$;
  repeat
    repeat
      foreach $Ax \leqslant a \in \mathcal{A}_k$ **do**
        $\theta_{Ax \leqslant a} :=$ LIRect$(Ax \leqslant a, B_l x \leqslant b_l, (X'_0, \ldots, X'_{n'}), m)$;
      $\theta_{k,l} := \bigvee_{Ax \leqslant a \in \mathcal{A}_k} \theta_{Ax \leqslant a}$;
      if $\theta_{k,l} = \bot$ **then**
        foreach $Bx \leqslant b \in \mathcal{B}_l$ **do**
          $\theta_{Bx \leqslant b} :=$ LIRect$(Bx \leqslant b, A_k x \leqslant a_k, (X'_0, \ldots, X'_{n'}), m)$;
        $\theta_{k,l} := \bigwedge_{Bx \leqslant b \in \mathcal{B}_l} \neg \theta_{Bx \leqslant b}$;
      pick $x_A \in Vars(A_k x \leqslant a_k) \cap Vars(B_l x \leqslant b_l)$, $c_A \in \{-m, \ldots, m\}$ and
      $\varphi_A \in \mathcal{A}_k$ **such that** $\theta_{\varphi_A} = \bot$ and:
        $\varphi_A \wedge x_A < c_A$ is SAT and $\varphi_A \wedge x_A \geq c_B$ is SAT,or
        $\varphi_A \wedge x_A \leq c_A$ is SAT and $\varphi_A \wedge x_A > c_B$ is SAT;
      $\mathcal{A}_k := (\mathcal{A}_k \setminus \{\varphi_A\}) \cup \{\varphi_A \wedge (x_A \sim c_A) : \texttt{SAT} \mid \sim \in \{<, =, >\}\}$;
      pick $x_B \in Vars(A_k x \leqslant a_k) \cap Vars(B_l x \leqslant b_l)$, $c_B \in \{-m, \ldots, m\}$ and
      $\varphi_B \in \mathcal{B}_l$ **such that** $\theta_{\varphi_B} = \bot$ and:
        $\varphi_B \wedge x_B < c_B$ is SAT and $\varphi_B \wedge x_B \geq c_B$ is SAT, or
        $\varphi_B \wedge x_B \leq c_B$ is SAT and $\varphi_B \wedge x_B > c_B$ is SAT;
      $\mathcal{B}_l := (\mathcal{B}_l \setminus \{\varphi_B\}) \cup \{\varphi_B \wedge (x_B \sim c_B) : \texttt{SAT} \mid \sim \in \{<, =, >\}\}$;
    until $\theta_{k,l} \neq \bot$ *or both* $\mathcal{A}_k$ *and* $\mathcal{B}_l$ *did not change* ;
    if $\theta_{k,l} = \bot$ **then** $m := m + 1$;
  until $\theta_{k,l} \neq \bot$;
CurrentBound $:= m$;
return $\bigvee_k \bigwedge_l \theta_{kl}$;

Algorithm 9: Computing localized interpolants for pairs of formulas.

5.5 Experiments

5.5.1 Prototype implementation

We have implemented the approach described in this chapter as a prototype synthesis tool. The tool is written in C++ and uses the SMT solver Z3 [dMB08] as a computational engine in the construction of the abstract game, and the CUDD BDD library [Som09] for the symbolic representation of the states of the abstract game. The implementation of the interpolation-generation procedure for the theory of linear arithmetic described in Section 5.3.3 relies on a combination of the SMT solver CVC3 [BT07] and the linear programming solver glpk [GLP]. Linear arithmetic formulas with arbitrary boolean structure are handled by an extension of Pudlák's algorithm [Pud97, YM05]. CVC3 provides the resolution proof generated by the underlying SAT solver (in this case MiniSAT) and the correspondence between theory predicates an boolean variables in the conflict clauses. The systems of linear inequalities generated by algorithm `LILA` for the conflict clauses are solved by the LP solver glpk.

5.5.2 Experimental Results

Here we report on experimental results obtained with our prototype on several examples of infinite-state systems including two well-known mutual exclusion protocols, the task for which is to solve the program repair problem with respect to given safety and bounded liveness properties. The program augmentation and the encoding as a symbolic game structure in the required input format are done manually.

Bakery. This example is modeled after Lamport's Bakery algorithm [Lam74]. In this well known mutual exclusion protocol each thread that wants to access the critical section is assigned a unique positive number (ticket). Before entering the critical section each process has to check that it holds the minimal ticket among the processes waiting for access. This protocol is modeled as an infinite-state system, since the ticket variables are unbounded and their values can become arbitrarily large.

Figure 5.5 shows a *partial* program that consists of two processes A and B, where process A follows the Bakery protocol for accessing its critical section. In process B, on the other hand, the choices of how to update the variable `ticketB` at location `m2` and when to access the critical section are left unresolved. The task is to resolve the

Process A:

```
l0: ticketA := 0;
l1: while(true){
 l2: ticketA := ticketB + 1;
 l3: await(ticketB = 0 ||
           ticketA < ticketB);
 l4: critical;
 l5: ticketA := 0;
}
```

Process B:

```
m0: ticketB := 0;
m1: while(true){
 m2:  | ticketB := 0;
      | ticketB := ticketA;
      | ticketB := ticketA + 1;
 m3: await(?);
 m4: critical;
 m5: ticketB := 0;
}
```

Figure 5.5: Partial program for Bakery mutual exclusion protocol.

nondeterminism in process B (possibly by introducing additional program variables) such that the resulting program meets the safety property stating that it is never the case that process A is in location `l4` and process B is in location `m4`, and a bounded liveness property that states that every time process B enters location `m3`, after a certain number of steps of process A, process B allows the transition to location `m4`.

We model the partial program as a symbolic game structure with incomplete information and reduce the task above to the problem of finding an obs_s-consistent winning strategy for $Player_\exists$. The decisions of $Player_\exists$ are encoded via the variables $V_\exists = \{update, guard\}$ whose values determine the selected update and whether access to the critical section is allowed or not. $Player_\exists$ can observe the location of process B, as well as the shared variable `ticketB`. We model the fact that $Player_\exists$ can read the value of the shared variables that can be written by process A, in this case the variable `ticketA`, only when process B is scheduled, by introducing two $Player_\forall$ variables, namely $ticket_A$ and $ticket_A^o$, where only the latter one can be observed by $Player_\exists$. Thus, the set of variables observed by $Player_\exists$ is $V_\forall^o = \{pc_B, ticket_B, ticket_A^o\}$. In addition to that, the set $V_\forall$ of $Player_\forall$'s variables contains the unobservable variables $pc_A, ticket_A, scheduled, control, steps$. The value of *scheduled*, which is arbitrarily updated by $Player_\forall$, determines the scheduled process. The variables *control* and *steps* are used to encode the bounded liveness property that $Player_\exists$ must enforce.

In the final set of abstraction predicates we have the following predicates over the

Process A:

```
l0: while(true){
 l1: atomic{if(lock = 0)
            then x1 := 0;
            else goto l1;}
 // wait for delay x1 <= B1
 l2: atomic{if(x1 <= B1)
            then lock := 1; x1 := 0;
            else goto l1;}
 l3: await(x1 >= B2);
 l4: atomic{if(lock = 1)
            then goto l5;
            else goto l1;}
 l5: critical;
 l6: lock := 0;
}
```

Process B:

```
m0: while(true){
 m1: b := B1 | B2;
 m2: atomic{if(lock = 0)
            then x2 := 0;
            else goto m2;}
 // wait for delay x2 <= b
 m3: atomic{if(x2 <= b)
            then lock := 2; x2 := 0;
            else goto m1;}
 m4: b := B1 | B2;
 m5: await(x2 >= b);
 m6: atomic{if(lock = 2)
            then goto m7;
            else goto m1;}
 m7: critical;
 m8: lock := 0;
}
```

Figure 5.6: Partial program for Fischer's mutual exclusion protocol.

variables $ticket_B$ and $ticket_A^o$: $(ticket_B \leq ticket_A^o), (ticket_B > 0)$ and $(ticket_A^o > 0)$. The computed resulting strategy for $Player_\exists$ is memoryless and implements the Bakery protocol. Note that even if a memoryless concrete strategy exists, like in this case, the strategy returned by our approach does not necessarily have to be memoryless.

Fischer. Here we consider an example modeled after another mutual-exclusion algorithm, namely Fischer's real-time mutual-exclusion protocol, as described for example in [MP96]. As usual, the goal of the protocol is to guarantee that at most one process can be at the critical section at any given time. The protocol uses a single shared integer variable `lock` and two parameters `B1` and `B2`, which are fixed constants in $\mathbb{R}_{\geq 0}$. These constants determine the lengths of delays in the protocol. Figure 5.6 shows a partial program with two processes, where process A is running Fischer's protocol and in process B the use of the constants `B1` and `B2`, in this example 1 and 2, is not fixed.

	Iterations	Predicates	Max. States	Strategy	Time(s)
Bakery	5	25	897	Memoryless	10.98
Fischer	6	27	5559	Memoryless	18.58
Robot	16	30	1158	Memoryfull	31.48
Sensors	14	34	1088	Memoryfull	38.43

Table 5.1: Results from experiments with our prototype on mutex protocols and motion planning examples. We report on: number of iterations of the CEGAR loop, final number of abstraction predicates, maximal number of explored abstract states in some iteration of the CEGAR loop, the type of the computed strategy for $Player_\exists$ and running time.

We reduce the problem of resolving the nondeterminism in this partial program to the problem of finding an obs_s-consistent winning strategy for $Player_\exists$ in a safety game under incomplete information. $Player_\exists$ controls a finite-range variable b, whose value, either `B1` or `B2`, is used to bound the delays in process B. The result is a strategy for $Player_\exists$ that corresponds to Fischer's protocol, where before setting `lock` to 2 process B waits at most `B1` time units and after setting `lock` to 2 waits at least `B2` time units.

The next two examples are from the application domain of robot motion planning.

Robot. Here we considered the game formalized in Example 5.3.1.

Sensors. We looked at a variation of Example 2.1.1. First, here there is no uncertainty about the initial position of the robot, but the movement parallel to the y axis is "imprecise", that is, uncertainty is introduced by the transition relation. There is a sensor that gives an interval $[ys, yn]$ of the possible values of the coordinate y and the exact value of y is received on every second step of moving parallel to the y axis.

The game structure $\mathcal{G}' = (V_\exists', V_\forall', V_\forall^{o\prime}, t', \varphi'_{Init}, \mathcal{T}'_\exists, \mathcal{T}'_\forall)$ is defined as follows.

The output variables $V_\exists'$ are the same as in Example 2.1.1 and the observable and unobservable input ones are $V_\forall^{o\prime} = \{x, ys, yn\}$ and $V_\forall' \setminus V_\forall^{o\prime} = \{y, steps, err, sense\}$. The formulas describing the initial states and transition relations are shown in Fig. 5.7. The winning condition is defined by the formula φ_{Err} given in Example 2.1.1.

Table 5.1 shows the results from our experiments preformed on an Intel Core 2 Duo CPU at 2.53 GHz with 3.4 GB RAM, using a single core. In the mutual exclusion examples the computed respective abstract strategies are memoryless, which means that they can be implemented without adding extra state (program variables)

$$\varphi'_{Init} \;:=\; t = \exists \;\wedge\; move = 0 \;\wedge\; x = 4 \;\wedge\; y = 3 \;\wedge\; steps = 0 \;\wedge\; \neg err \;\wedge\; ys = 3 \;\wedge\; yn = 3 \;\wedge\; sense = true.$$

$$\begin{aligned}\mathcal{T}'_{\exists} \;:=\;& t = \exists \wedge t' = \forall \wedge (move' = N \vee move' = S \vee move' = E \vee move' = W) \wedge \\ & x' = x \;\wedge\; y' = y \;\wedge\; steps' = steps \;\wedge\; err' = err \;\wedge \\ & ys' = ys \;\wedge\; yn' = yn \;\wedge\; sense' = sense.\end{aligned}$$

$$\begin{aligned}\mathcal{T}'_{\forall} \;:=\;& t = \forall \wedge t' = \exists \wedge move' = move \wedge (\varphi'_N \vee \varphi'_S \vee \varphi'_E \vee \varphi'_W) \wedge \\ & steps' = steps + 1 \wedge \big((\varphi_{bad} \wedge err') \vee (\neg\varphi_{bad} \wedge err' = err)\big), \text{ where}\end{aligned}$$

$$\begin{aligned}\varphi'_N \;:=\;& y' \geq y + 1 \;\wedge\; y' \leq y + 1.5 \;\wedge\; x' = x \;\wedge \\ & \big(\neg sense \;\wedge\; ys' = y + 1 \;\wedge\; yn' = y + 1.5 \;\wedge\; sense' \vee \\ & sense \;\wedge\; ys' = y' \;\wedge\; yn' = y' \;\wedge\; \neg sense'\big),\end{aligned}$$

$$\begin{aligned}\varphi'_S \;:=\;& y' \geq y - 1.5 \;\wedge\; y' \leq y - 1 \;\wedge\; x' = x \;\wedge \\ & \big(\neg sense \;\wedge\; ys' = y - 1.5 \;\wedge\; yn' = y - 1 \;\wedge\; sense' \vee \\ & sense \;\wedge\; ys' = y' \;\wedge\; yn' = y' \;\wedge\; \neg sense'\big),\end{aligned}$$

$$\varphi'_E := x' = x + 2 \;\wedge\; y' = y \;\wedge\; ys' = ys \;\wedge\; yn' = yn \;\wedge\; sense' = sense,$$

$$\varphi'_W := x' = x - 2 \;\wedge\; y' = y \;\wedge\; ys' = ys \;\wedge\; yn' = yn \;\wedge\; sense' = sense,$$

$$\varphi_{bad} := \varphi_{hit-wall} \vee \varphi_{go-back}, \text{ where}$$

$$\varphi_{hit-wall} := (x < 5 \wedge x' \geq 5 \vee x > 5 \wedge x' \leq 5) \wedge (y \leq 0 \vee y \geq 2),$$

$$\varphi_{go-back} := x > 5 \wedge x' \leq 5,$$

Figure 5.7: Formulas defining the game structure for the modified robot example.

to the programs shown in Figure 5.5 and Figure 5.6. More specifically, they correspond to the classical Bakery and Fischer's protocols. The significant difference in the maximal number of explored abstract states for a similar number of predicates is partly due to the fact that the abstract game is constructed on-the-fly while also reusing results from previous iterations. One can observe that the running times are proportional to the number of iterations of the CEGAR loop and not to the size of the abstract games. This is the case when a significant amount of time is spent on the counterexample analysis and refinement operations. Although the goal of the procedure RefineTransitionRelations is to refine the abstract transition relation, the set of generated predicates can, of course, contain observable predicates, which refine the abstract transition relation as well. In the first two examples above it happens that RefineTransitionRelations succeeds in producing observable predicates that suffice to discover a consistent abstract winning strategy for $Player_{\exists}$. In the last two examples this is not the case, and applying the procedure RefineObservations is necessary for computing sufficient observations. In the Robot example this procedure generates the predicate $y_o \leq 0$, which allows the controller to decide correctly at the first step whether to make one step in direction north before heading east or not. This single predicate over y_o suffices for $Player_{\exists}$ to win the game. Even if we would change the transition relation such that y and y^o are equal in all reachable states, this is still the only predicate about y^o generated by the refinement. During the refinement of the transition relation a number of predicates over the unobservable variable y are also generated. Thus, if instead of having the extra variable y_o we declare variable y observable, a larger number of observable predicates are generated, which in this case are not necessary for the existence of a controller. In the last of the examples above, RefineObservations produces the predicates $y_s \leq \frac{3}{2}$ and $y_s \geq \frac{3}{2}$ which enable the controller to decide whether to make one or two steps in direction south before going east though the door.

5.5.3 Discussion

The synthesis problems considered in this section are out of the scope of other state-of-the-art synthesis tools that exist at the time of writing of this thesis. The only tool that takes both infinite state spaces and partial observation into account is SMACS [SMA10], which implements a technique presented in [KGMM09]. SMACS addresses the basic

(and the non-blocking) synthesis problem for *memoryless* controllers with partial observation for infinite-state systems, and is therefore restricted to memoryless strategies. More importantly, since it applies solely backward reasoning it is unable to discover strategies for the above examples. The reason is, that even when memoryless consistent winning strategies for $Player_\exists$ exist, establishing the existence of such a strategy relies on forward reasoning about the knowledge of $Player_\exists$ during the course of the game. Another tool for solving games under incomplete information is Alpaga [BCW+09], which is applicable to parity games on a *finite* game graph. The antichain approach [CDHR06] for finite-state incomplete information games on which Alpaga is based is also implemented as part of the tool GAVS+ [CKLB11]. Clearly, the problem that we addressed in this chapter is out of the scope of these tools.

Part III
Counterexample-Guided Abstraction Refinement for Synthesis of Observation Predicates for Timed Control

Chapter 6

Timed Control with Partial Observation

Controller synthesis, both in the discrete and in the timed setting, has been an active field of research in the last decades. The *timed controller synthesis* problem asks to automatically find a controller for an open plant such that the controlled closed loop system satisfies a given property. It naturally reduces to the problem of finding a winning strategy for the controller player in a two-player *timed game* between a controller and its environment (the plant). This problem is well-understood for the case that the controller can fully observe the state and evolution of the plant. In reality, however, this assumption is usually violated due to limited sensors or the inability to observe the internal behavior of the plant. The controller must therefore win the game under *partial observability*. The key challenge for timed controller synthesis methods is to find what sensor information, and more importantly, what timing information and precision is required by the controller in order to enforce the desired property.

Motivating Example. The example shown in Figure 6.1 demonstrates the role of observations in timed control under partial observability. The figure depicts a toy model of a production system. The role of the controller is to remove a box from a conveyor belt after the box is ready and before it reaches the end of the belt. The plant produces two types of boxes, and for each type the respective robot arm should be used to remove the box from the belt. The locations On, $\mathsf{Producing}_1$, $\mathsf{Producing}_2$, Sensed_1, Sensed_2, Ready_1, Ready_2, End and Off of the plant indicate the position of the

box (depending on its type) on the belt. Producing a box of Type I takes between 4 and 6 seconds and producing a box of Type II takes between 7 and 8 seconds. Regardless of its type, the box arrives at the respective location Ready_1 or Ready_2 between 9 and 10 seconds after the start of the current production phase. The goal of the controller is to avoid the error location End. For that, it has to execute the correct $\mathsf{arm}_1!$ or $\mathsf{arm}_2!$ action at the right time, namely when the box is in location Ready_1 or Ready_2.

The challenge is that locations On, $\mathsf{Produce}_1$ and $\mathsf{Produce}_2$ are indistinguishable by the controller, and so are locations Sensed_1, Sensed_2, Ready_1 and Ready_2. Thus, the controller cannot directly observe the box entering locations Ready_1 and Ready_2. The controller can only detect the presence of a box via a sensor (i.e, it observes the box entering locations Sensed_1 and Sensed_2) and can use timing information to determine the time-frame in which the box is in location Ready_1 (or Ready_2). The sensor cannot distinguish the type of the produced box, and hence the controller cannot observe if a box of Type I or a box of Type II is being produced. However, using the timing information, the controller can correctly infer the type of the box as well.

The controller cannot observe the plant's clock x. Thus, a solution to the synthesis problem is to use a clock y in the controller and activate the respective robot arm when $y = 21/2$, thus ensuring that the error location End is never reached, as the box is guaranteed to reach location Ready_1 or Ready_2 in 9 to 10 seconds after it is sensed and remains there at least until $y = 11$. Activating the correct robot arm can be done by checking whether or not the box has been sensed before the clock y reaches 7.

Given the two *observation predicates* $y = 21/2$ and $y \geq 7$, we can construct a correct controller. Clearly, both predicates are necessary: if the controller only observes one of them or only some other predicate, say, only $y = 30$, then it is impossible to enforce the specification. Note also that the two predicates play different roles in the control strategy. Predicate $y = 21/2$ identifies a particular point in time (out of the infinitely many) in which the controller may choose to *take an action*, while predicate $y \geq 7$ is needed in order to be able to *decide* on the right action. In the following, we distinguish these two types of observation predicates as *action points* and *decision predicates*.

Related Work. The timed controller synthesis problem, and hence computing a sufficient set of observation predicates, is undecidable under partial observability [BDMP03,

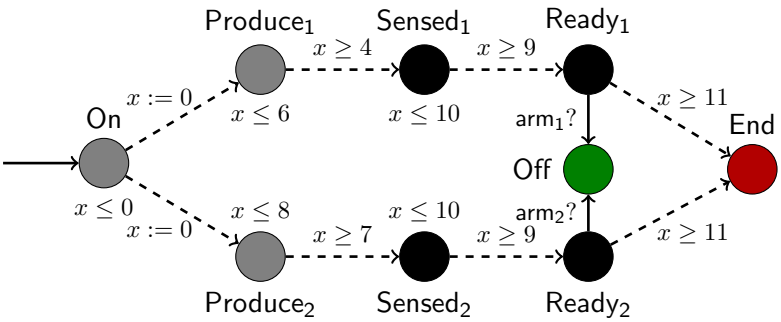


Figure 6.1: Example of a partially observable plant for a production system represented as a timed game automaton with a clock x. The plant has an uncontrollable action u (uncontrollable edges in the automaton are denoted with dashed lines). For readability, the edges with controllable actions $\mathsf{arm}_1?$ and $\mathsf{arm}_2?$ leading to End from locations On, $\mathsf{Produce}_1$, $\mathsf{Produce}_2$, Sensed_1, Sensed_2, End and locations Ready_2 and Ready_1 respectively, have been omitted from the figure. The equivalence relation between the locations of the plant has been encoded using colors depicting the equivalence classes.

BC06]. All previously known synthesis algorithms therefore rely on some *a-priori* restriction of the problem, such as fixing the resources of the controller [BDMP03], or fixing a template for the controller [FP12]. Alternatively, one can predefine the observations of the controller [CDL$^+$07, Cas07] in the form of a fixed finite set of predicates. The latter works were extended in [BCD$^+$12] to take into account the cost of observations and compute for a finite set of predicates a subset with minimal cost.

Contribution. In this part of the thesis we investigate the important research question of automatically discovering observation predicates for timed control. We present the first systematic method for the automatic synthesis of observation predicates, which was first published in [DF12]. The proposed method works by successively refining a finite set of observation predicates and is based on the analysis of spurious counterexamples. It builds on the CEGAR approach described in the second part of this thesis. For timed games, the main difficulty in the characterization of spurious counterexamples and refining the abstraction is caused by the fact that the number of moves available to the controller is infinite, corresponding to the infinite number of points in time when an action can be taken. To the best of our knowledge, prior to the work [DF12] of Dimitrova and Finkbeiner there was no approach to controller synthesis that can handle

partial observability for systems that allow for infinitely many choices of the controller, without fixing a priori a finite set of available observations.

6.1 Preliminaries

Let us recall some definitions and notations from the theory of timed automata [AD94].

For a set X of real-valued variables, $\mathbb{R}^X_{\geq 0} = X \to \mathbb{R}_{\geq 0}$ is the set of total functions from X to $\mathbb{R}_{\geq 0}$, also called *clock valuations*. For $v \in \mathbb{R}^X_{\geq 0}$, a set $Z \subseteq X$ of variables and a positive constant $\Delta \in \mathbb{R}_{>0}$ we denote with $v[0/Z]$ the valuation obtained from v by setting the values of the variables in Z to 0 and with $v + \Delta$ the valuation obtained from v by adding Δ to the value of each variable in X given by v. We let $\mathbf{0} \in \mathbb{R}^X_{\geq 0}$ be the function that maps each variable in X to the value 0.

Recall from Section 2.1.1, that for a set of real valued variables X, $\mathcal{B}[X]$ is the set of boolean combinations of formulas of the form $x \sim c$, where $x \in X$, $c \in \mathbb{Q}$ and $\sim \in \{<, \leq, >, \geq, =\}$, and $\mathcal{C}[X]$ consists of *true*, *false* and conjunctions of such atoms.

Definition 6.1.1 ([AD94])**.** A *timed automaton* is a tuple $\mathcal{A} = (\mathsf{Loc}, l_0, X, \Sigma, \mathsf{Inv}, \delta)$, where Loc is a finite set of locations, $l_0 \in \mathsf{Loc}$ is the initial location, X is a finite set of real-valued clocks, Σ is a finite set of actions, $\mathsf{Inv} : \mathsf{Loc} \to \mathcal{C}[X]$ maps each location in Loc to an invariant and $\delta \subseteq \mathsf{Loc} \times \Sigma \times \mathcal{B}[X] \times 2^X \times \mathsf{Loc}$ is a finite set of transitions.

Definition 6.1.2. The semantics of a timed automaton $\mathcal{A} = (\mathsf{Loc}, l_0, X, \Sigma, \mathsf{Inv}, \delta)$ is defined by a *timed transition system* $A = (S_{\mathcal{A}}, s^0_{\mathcal{A}}, \Sigma, \to)$, where $S_{\mathcal{A}} = \{(l, v) \in \mathsf{Loc} \times \mathbb{R}^X_{\geq 0} \mid v \models \mathsf{Inv}(l)\}$ is the set of states, $s^0_{\mathcal{A}} = (l_0, \mathbf{0})$ is the initial state, and the transition relation $\to \subseteq S_{\mathcal{A}} \times (\Sigma \cup \mathbb{R}_{>0}) \times S_{\mathcal{A}}$ is such that $((l, v), \sigma, (l', v')) \in \to$ (denoted $(l, v) \xrightarrow{\sigma} (l', v')$) iff $v \models \mathsf{Inv}(l)$, $v' \models \mathsf{Inv}(l')$ and one of the following conditions holds:

- $\sigma \in \Sigma$ and there exists $(l, \sigma, g, Z, l') \in \delta$ such that $v \models g$ and $v' = v[0/Z]$;
- $\sigma \in \mathbb{R}_{>0}$, $l' = l$, $v' = v + \sigma$, and $(v + \sigma') \models \mathsf{Inv}(l)$ for each $0 < \sigma' \leq \sigma$.

A timed automaton $\mathcal{A} = (\mathsf{Loc}, l_0, X, \Sigma, \mathsf{Inv}, \delta)$ is called *time-deterministic* iff for every state (l, v) the set of transitions enabled in (l, v) does not contain distinct transitions (σ, g', Z') and (σ, g'', Z'') such that the conjunction $g_1 \wedge g_2$ is satisfiable.

6.2 Timed Controller Synthesis

The timed controller synthesis problem in its general form asks to construct a controller (check if one exists) for a given plant, such that the controlled plant satisfies a given property. We study this problem under the partial observability hypothesis, where part of the input determines what aspects of the plant the controller is able to observe. We focus on the case when the clocks available to the controller are fixed a priori.

Partially Observable Plant

The plant given as input to the timed controller synthesis problem is represented as a timed automaton, in which the actions labeling the transitions are partitioned into controllable and uncontrollable. The partial observation of the plant by the controller is encoded by a partitioning of the clocks of the plant into observable and unobservable and specifying an equivalence relation on the set of locations of the plant. The controller can read the values only of the observable clocks and observes only the equivalence class of the location the plant is currently in, and not the actual location of the plant.

Definition 6.2.1. A *partially observable plant* is a tuple $\mathcal{N} = (\mathcal{A}, \Sigma_{\mathsf{c}}, \Sigma_{\mathsf{u}}, X_{\mathsf{o}}, X_{\mathsf{u}}, =_{\mathsf{o}}^{\mathsf{L}})$, where $\mathcal{A} = (\mathsf{Loc}, l_0, X, \Sigma, \mathsf{Inv}, \delta)$ is a timed automaton and:

- Σ_{c} and Σ_{u} partition the set Σ of actions of the automaton $\mathcal{A}$ into a set Σ_{c} of *controllable* actions and a set Σ_{u} of *uncontrollable* actions,
- X_{o} and X_{u} partition the set X of clocks of the automaton $\mathcal{A}$ into a set X_{o} of *observable* clocks and a set X_{u} of *unobservable* clocks,
- $=_{\mathsf{o}}^{\mathsf{L}} \subseteq \mathsf{Loc} \times \mathsf{Loc}$ is an *(observation) equivalence* relation on Loc.

We require that $\mathcal{N}$ is *input-enabled* and *deadlock-free*. A plant $\mathcal{N} = (\mathcal{A}, \Sigma_{\mathsf{c}}, \Sigma_{\mathsf{u}}, X_{\mathsf{o}}, X_{\mathsf{u}}, =_{\mathsf{o}}^{\mathsf{L}})$ with $\mathcal{A} = (\mathsf{Loc}, l_0, X, \Sigma, \mathsf{Inv}, \delta)$ is called *input-enabled* if for each controllable action $\sigma \in \Sigma_{\mathsf{c}}$ and configuration $(l, v) \in S_{\mathcal{A}}$ there exists a configuration $(l', v') \in S_{\mathcal{A}}$ such that $(l, v) \xrightarrow{\sigma} (l', v')$, that is, each controllable action is enabled in each configuration in $S_{\mathcal{A}}$. The plant $\mathcal{N}$ is *deadlock-free* if for each configuration $(l, v) \in S_{\mathcal{A}}$ there exist $\sigma \in \Sigma_{\mathsf{u}} \cup \mathbb{R}_{>0}$ and a configuration $(l', v') \in S_{\mathcal{A}}$ such that $(l, v) \xrightarrow{\sigma} (l', v')$, i.e., in each configuration from $S_{\mathcal{A}}$ an uncontrollable action can be performed or time can elapse.

Control Strategies

Let $\mathcal{N} = (\mathcal{A}, \Sigma_{\mathsf{c}}, \Sigma_{\mathsf{u}}, X_{\mathsf{o}}, X_{\mathsf{u}}, =_{\mathsf{o}}^{\mathsf{L}})$ be a partially observable plant with underlying timed automaton $\mathcal{A} = (\mathsf{Loc}, l_0, X, \Sigma, \mathsf{Inv}, \delta)$. A controller for $\mathcal{N}$ decides when and what controllable actions can be executed. The controller operates under incomplete information about the current location of the plant and the values of the plant's clocks. In a given configuration, the controller can only observe the equivalence class of the current location w.r.t. $=_{\mathsf{o}}^{\mathsf{L}}$ and the current values of the observable clocks X_{o}. The controller is equipped with a set of controllable clocks, whose values it can observe and set to 0.

Let X_{c} be a set of *controllable clocks* such that $X_{\mathsf{c}} \cap X = \emptyset$. We denote with $X_{\mathsf{o+c}} = X_{\mathsf{o}} \dot{\cup} X_{\mathsf{c}}$ the union of the observable clocks of the plant and the controllable clocks X_{c}. Let $\Sigma_{\mathsf{r}} = \{reset_Z \mid Z \in 2^{X_{\mathsf{c}}} \setminus \{\emptyset\}\}$ be a set of actions disjoint from Σ. Intuitively, the actions from Σ_{r} allow the controller to reset the controllable clocks. Let us define $\widetilde{S_{\mathcal{A}}} = \mathsf{Loc} \times \mathbb{R}_{\geq 0}^{X \,\dot{\cup}\, X_{\mathsf{c}}}$ and $\widetilde{\Sigma} = \Sigma \,\dot{\cup}\, \mathbb{R}_{>0} \,\dot{\cup}\, \Sigma_{\mathsf{r}}$. The set $\widetilde{S_{\mathcal{A}}}$ consists of the configurations of the plant extended with values for the clocks of the controller and the elements of $\widetilde{\Sigma}$ are the possible actions of the controlled plant.

A *run* of the controlled plant is a sequence $\pi \in (\widetilde{S_{\mathcal{A}}} \cdot (\widetilde{\Sigma} \cdot \widetilde{S_{\mathcal{A}}})^*) \cup (\widetilde{S_{\mathcal{A}}} \cdot (\widetilde{\Sigma} \cdot \widetilde{S_{\mathcal{A}}})^\omega)$.

A control strategy maps finite runs (the history of the execution so far) to decisions of the controller. The controller can decide to: (i) execute a controllable action σ when a clock $x \in X_{\mathsf{o+c}}$ reaches a value c, or (ii) execute a controllable action σ immediately, or (iii) remain idle and let time elapse, or (iv) reset a set of controllable clocks $Z \subseteq X_{\mathsf{c}}$. The function $Act_\exists : \widetilde{S_{\mathcal{A}}} \to 2^{(\Sigma_{\mathsf{c}} \cup (\Sigma_{\mathsf{c}} \times X_{\mathsf{o+c}} \times \mathbb{Q}_{>0}) \cup \{\flat\} \cup \Sigma_{\mathsf{r}})}$ maps a configuration $(l, \widetilde{v}) \in \widetilde{S_{\mathcal{A}}}$ to the set of possible choices of the controller in this configuration. Formally, the set $Act_\exists((l, \widetilde{v}))$ is the smallest subset of $\Sigma_{\mathsf{c}} \cup (\Sigma_{\mathsf{c}} \times X_{\mathsf{o+c}} \times \mathbb{Q}_{>0}) \cup \{\flat\} \cup \Sigma_{\mathsf{r}}$ such that:

(i) if $\sigma \in \Sigma_{\mathsf{c}}$ and $c > \widetilde{v}(x)$ for $c \in \mathbb{Q}_{>0}$, $x \in X_{\mathsf{o+c}}$, then $(\sigma, x, c) \in Act_\exists((l, \widetilde{v}))$ – the controller can choose to execute a controllable action when some clock in $X_{\mathsf{o+c}}$ reaches a value that is strictly greater than this clock's current value;

(ii) $\Sigma_{\mathsf{c}} \subseteq Act_\exists((l, \widetilde{v}))$ – the controller can always immediately execute a $\sigma \in \Sigma_{\mathsf{c}}$;

(iii) $\flat \in Act_\exists((l, \widetilde{v}))$ – the controller can always remain idle and let time elapse;

(iv) if $\widetilde{v}(x) > 0$ for each $x \in Z \subseteq X_{\mathsf{c}}$, then $reset_Z \in Act_\exists((l, \widetilde{v}))$ – the controller can reset clocks whose values in the current configurations are strictly positive.

A control strategy is a function $f_{\mathsf{c}} : \widetilde{S_{\mathcal{A}}} \cdot (\widetilde{\Sigma} \cdot \widetilde{S_{\mathcal{A}}})^* \to \Sigma_{\mathsf{c}} \cup (\Sigma_{\mathsf{c}} \times X_{\mathsf{o}+\mathsf{c}} \times \mathbb{Q}_{>0}) \cup \{\flat\} \cup \Sigma_{\mathsf{r}}$. This function must be consistent with what the controller can observe during the execution of the plant. To formalize this requirements we define a function

$$obs_r : \widetilde{S_{\mathcal{A}}} \cdot (\widetilde{\Sigma} \cdot \widetilde{S_{\mathcal{A}}})^* \to (Obs_{\mathsf{L}} \times \mathbb{R}_{\geq 0}^{X_{\mathsf{o}+\mathsf{c}}}) \cdot ((Obs_{\mathsf{L}} \times \mathbb{R}_{\geq 0}^{X_{\mathsf{o}+\mathsf{c}}}) \times (Obs_{\mathsf{L}} \times \mathbb{R}_{\geq 0}^{X_{\mathsf{o}+\mathsf{c}}}))^*,$$

where Obs_{L} is the set of equivalence classes of Loc w.r.t. $=_{\mathsf{o}}^{\mathsf{L}}$. The function obs_r maps a finite run to the sequence of observations the controller makes during this run. The controller observes controllable actions of the plant and discrete changes of the state-based observations, such as change of the equivalence class of the current location or reset of some clock in $X_{\mathsf{o}+\mathsf{c}}$. At such points the controller observes the equivalence class of the current location and the current values of the clocks in $X_{\mathsf{o}+\mathsf{c}}$.

First, we define the function $obs : \widetilde{S_{\mathcal{A}}} \to Obs_{\mathsf{L}} \times \mathbb{R}_{\geq 0}^{X_{\mathsf{o}+\mathsf{c}}}$, such that for $(l, \widetilde{v}) \in \widetilde{S_{\mathcal{A}}}$ we let $obs((l, \widetilde{v})) = ([l]_{=_{\mathsf{o}}^{\mathsf{L}}}, \widetilde{v}(X_{\mathsf{o}+\mathsf{c}}))$, where $[l]_{=_{\mathsf{o}}^{\mathsf{L}}}$ is the equivalence class that contains l. Now, for a finite run $\pi \in \widetilde{S_{\mathcal{A}}} \cdot (\widetilde{\Sigma} \cdot \widetilde{S_{\mathcal{A}}})^*$ we define:

$$obs_r(\pi) = \begin{cases} obs(\widetilde{s}) & \text{if } \pi = \widetilde{s} \in \widetilde{S_{\mathcal{A}}}, \\ obs_r(\pi' \cdot \widetilde{s}) \cdot (obs(\widetilde{s}), obs(\widetilde{s}')) & \text{if } \pi = \pi' \cdot \widetilde{s} \cdot \widetilde{\sigma} \cdot \widetilde{s}', \text{ and } \widetilde{\sigma} \in \Sigma_{\mathsf{c}} \cup \Sigma_{\mathsf{r}} \text{ or} \\ & l \neq_{\mathsf{o}}^{\mathsf{L}} l' \text{ or } \widetilde{v}(x) > \widetilde{v}'(x) \text{ for some } x \in X_{\mathsf{o}}, \\ & \text{where } \widetilde{s} = (l, \widetilde{v}) \text{ and } \widetilde{s}' = (l', \widetilde{v}'), \\ obs_r(\pi' \cdot \widetilde{s}) & \text{if } \pi = \pi' \cdot \widetilde{s} \cdot \widetilde{\sigma} \cdot \widetilde{s}' \text{ and } \widetilde{\sigma} \notin \Sigma_{\mathsf{c}} \cup \Sigma_{\mathsf{r}} \text{ and} \\ & l =_{\mathsf{o}}^{\mathsf{L}} l' \text{ and } \widetilde{v}(x) \leq \widetilde{v}'(x) \text{ for all } x \in X_{\mathsf{o}}, \\ & \text{where } \widetilde{s} = (l, \widetilde{v}) \text{ and } \widetilde{s}' = (l', \widetilde{v}'). \end{cases}$$

Definition 6.2.2. Let $\mathcal{N} = (\mathcal{A}, \Sigma_{\mathsf{c}}, \Sigma_{\mathsf{u}}, X_{\mathsf{o}}, X_{\mathsf{u}}, =_{\mathsf{o}}^{\mathsf{L}})$ be a partially observable plant with $\mathcal{A} = (\mathsf{Loc}, l_0, X, \Sigma, \mathsf{Inv}, \delta)$ and let X_{c} be a finite set of clocks with $X_{\mathsf{c}} \cap X = \emptyset$. A X_{c}-*control strategy for the partially observable plant* $\mathcal{N}$ is a total function $f_{\mathsf{c}} : \widetilde{S_{\mathcal{A}}} \cdot (\widetilde{\Sigma} \cdot \widetilde{S_{\mathcal{A}}})^* \to \Sigma_{\mathsf{c}} \cup (\Sigma_{\mathsf{c}} \times X_{\mathsf{o}+\mathsf{c}} \times \mathbb{Q}_{>0}) \cup \{\flat\} \cup \Sigma_{\mathsf{r}}$, that meets the following two requirements:

- $f_{\mathsf{c}}(\pi) \in Act_{\exists}(\mathsf{last}(\pi))$, for each $\pi \in \widetilde{S_{\mathcal{A}}} \cdot (\widetilde{\Sigma} \cdot \widetilde{S_{\mathcal{A}}})^*$,
- if $obs_r(\pi_1) = obs_r(\pi_2)$, then $f_{\mathsf{c}}(\pi_1) = f_{\mathsf{c}}(\pi_2)$, for each $\pi_1, \pi_2 \in \widetilde{S_{\mathcal{A}}} \cdot (\widetilde{\Sigma} \cdot \widetilde{S_{\mathcal{A}}})^*$.

A control strategy f_{c} for the plant $\mathcal{N}$ defines a set of *controlled runs* $\mathcal{CR}(f_{\mathsf{c}}, \mathcal{N}) \subseteq (\widetilde{S_{\mathcal{A}}} \cdot (\widetilde{\Sigma} \cdot \widetilde{S_{\mathcal{A}}})^*) \cup (\widetilde{S_{\mathcal{A}}} \cdot (\widetilde{\Sigma} \cdot \widetilde{S_{\mathcal{A}}})^\omega)$. For a run π we have $\pi \in \mathcal{CR}(f_{\mathsf{c}}, \mathcal{N})$ iff $\pi[0] = (l_0, \mathbf{0})$ and for every $0 < i < |\pi| - 1$ with $\pi[i-1] = (l, \widetilde{v})$, $\pi[i] = \widetilde{\sigma}$ and $\pi[i+1] = (l', \widetilde{v}')$:

(i) if $\widetilde{\sigma} \in \mathbb{R}_{>0}$, then $(l, \widetilde{v}(X)) \xrightarrow{\widetilde{\sigma}} (l', \widetilde{v}'(X))$, $\widetilde{v}'(X_{\mathsf{c}}) = \widetilde{v}(X_{\mathsf{c}}) + \widetilde{\sigma}$, and $f_{\mathsf{c}}(\pi[0, i-1]) = \flat$ or $f_{\mathsf{c}}(\pi[0, i-1]) = (\sigma, x, c)$ and $\widetilde{v}'(x) \leq c$ (either the controller chose to remain idle or time can only elapse while the value of clock x has not reached c);

(ii) if $\widetilde{\sigma} \in \Sigma_{\mathsf{c}}$, then $(l, \widetilde{v}(X)) \xrightarrow{\widetilde{\sigma}} (l', \widetilde{v}'(X))$, $\widetilde{v}'(X_{\mathsf{c}}) = \widetilde{v}(X_{\mathsf{c}})$, and $f_{\mathsf{c}}(\pi[0, i-1]) = \widetilde{\sigma}$ or $f_{\mathsf{c}}(\pi[0, i-1]) = (\widetilde{\sigma}, x, c)$ and $v(x) = c$ ($\widetilde{\sigma}$ is taken immediately, or x reached c);

(iii) if $\widetilde{\sigma} \in \Sigma_{\mathsf{u}}$, then $f_{\mathsf{c}}(\pi[0, i-1]) \notin \Sigma_{\mathsf{r}}$, $(l, \widetilde{v}(X)) \xrightarrow{\widetilde{\sigma}} (l', \widetilde{v}'(X))$ and $\widetilde{v}'(X_{\mathsf{c}}) = \widetilde{v}(X_{\mathsf{c}})$ (only resetting controllable clocks has priority over uncontrollable transitions);

(iv) if $\widetilde{\sigma} = \mathit{reset}_Z$, then $f_{\mathsf{c}}(\pi[0, i-1]) = \widetilde{\sigma}$, $(l', \widetilde{v}'(X)) = (l, \widetilde{v}(X))$ and $\widetilde{v}'(X_{\mathsf{c}}) = \widetilde{v}(X_{\mathsf{c}})[0/Z]$ (resetting clocks from X_{c} leaves the configuration of $\mathcal{N}$ unchanged).

Controller Synthesis Problem

A plant location $l \in \mathsf{Loc}$ is *reachable* in $\mathcal{CR}(f_{\mathsf{c}}, \mathcal{N})$ iff there exists a finite path $\pi \in \mathcal{CR}(f_{\mathsf{c}}, \mathcal{N})$ such that $\mathsf{last}(\pi) = (l, \widetilde{v})$ for some clock valuation $\widetilde{v} \in \mathbb{R}_{\geq 0}^{X \,\dot\cup\, X_{\mathsf{c}}}$.

Given

- a partially observable plant $\mathcal{N} = (\mathcal{A}, \Sigma_{\mathsf{c}}, \Sigma_{\mathsf{u}}, X_{\mathsf{o}}, X_{\mathsf{u}}, =_{\mathsf{o}}^{\mathsf{L}})$, $\mathcal{A} = (\mathsf{Loc}, l_0, X, \Sigma, \mathsf{Inv}, \delta)$,
- an *error location* $l_{Err} \in \mathsf{Loc}$, for which $[l_{Err}]_{=_{\mathsf{o}}^{\mathsf{L}}} = \{l_{Err}\}$, and
- a finite set X_{c} of controllable clocks such that $X_{\mathsf{c}} \cap X = \emptyset$,

the *timed safety control problem with partial observability* asks whether there exists a X_{c}-control strategy f_{c} for $\mathcal{N}$ such that l_{Err} is not reachable in $\mathcal{CR}(f_{\mathsf{c}}, \mathcal{N})$. The corresponding *controller synthesis problem* asks to compute such a strategy if one exists.

6.3 Observations for Timed Control

6.3.1 Undecidability Results

In [BDMP03] Bouyer et al. established undecidability of the timed control problem under partial observability for specifications given as deterministic timed automata. Later Bouyer and Chevalier [BC06] considered the special cases of reachability and safety specifications, given as a set of goal and error locations respectively. Under

partial observability, they show undecidability of the timed reachability control problem and the timed safety control problem under non-Zeno control strategies.

These undecidability results have initiated work on restrictions of the control problem under partial observability, where the observation power of the controller is fixed.

6.3.2 Timed Control with Fixed Observations

In their paper [BDMP03] Bouyer et al. investigated the case when the resources of the controller, i.e., its clocks and the constants these clocks can be compared to, are fixed. Formally, the input to the problem consists of a timed automaton describing the plant, a partitioning of the actions of the automaton into controllable, observable and unobservable actions, a partitioning of the set of clocks of the automaton into observable and unobservable clocks, and a *granularity* μ. The granularity $\mu = (X^r \cup X_{Cont}, m, max)$ consists of a set X_r of plant clocks which the controller can read and the set X_{Cont} of the controller's own clocks, a positive integer m and a function $max : X^r \cup X_{Cont} \to \mathbb{Q}_{>0}$. The constant m fixes all constants used in the controller to be integral multiples of m. The function max maps each clock in the granularity to the maximal constant this clock can be compared to. The granularity μ defines a finite subset of $\mathcal{B}[X^r \cup X_{Cont}]$. The timed controller synthesis problem with partial observability and with *fixed resources* μ, asks to decide if there exists a controller whose granularity is μ and such that the controlled plant satisfies the given specification.

In the framework of [BDMP03, BC06] the observations of the controller are event-based, i.e, the controller observes the controllable and a subset of the uncontrollable actions and uses in their guards the clocks from X_{Cont} and a subset of the plant's clocks. The observations used in [CDL$^+$07], on the other hand, are state-based. That is, each *observable predicate* is a pair (K, ϕ) consisting of a set K of plant locations and constraint $\phi \in \mathcal{C}[X]$, where X is the set of clocks. Intuitively, the controller observes a set of locations one of which is the plant's current location and observes the clocks not directly, but via the values of the given constraints. The synthesis problem considered in [CDL$^+$07] is to compute an observation- based strategy to control the given plant using a *an a priori fixed set of observable predicates.*

In both of its above variations the timed controller synthesis problem with fixed observations is decidable. The algorithm presented in [BDMP03] is based on the idea of the region graph construction, and the one developed in [CDL$^+$07] is zone-based.

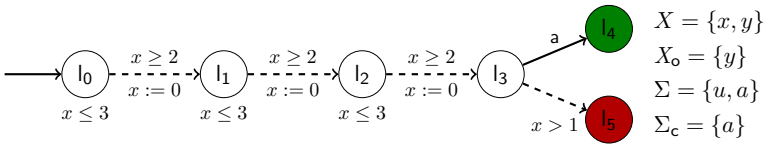


Figure 6.2: Partially observable plant for which the necessary action points for the controller are not bounded by the maximal constant occurring in a constraint of the plant automaton. The plant has two clocks x and y, and only y is observable by the controller. The plant has an uncontrollable action u (uncontrollable edges in the automaton are denoted with dashed lines). For readability, the edges with controllable action a leading to l_5 from locations l_0, l_1, l_2, l_4 and l_5 have been omitted from the figure. The equivalence relation between locations has been encoded using colors depicting the equivalence classes.

The challenge that remained, however, is to automatically find sufficient observations for the controller. According to the discussion in the beginning of this chapter, we distinguish between two types of observable predicates, namely action points and decision predicates, depending on the role they play in the resulting controller. In the following, we exemplify the fact that, in the general case, neither set of predicates can be directly derived from the syntactic description of the plant.

Let $\mathcal{A} = (\mathsf{Loc}, l_0, X, \Sigma, \mathsf{Inv}, \delta)$ be a timed automaton in which all constants are non-negative integers, and let c_{max} be the maximal constant in $\mathcal{A}$. We note with $\mathcal{D}[X, c_{max}]$ the subset of $\mathcal{D}[X]$, where all constants are non-negative integers bounded by c_{max}.

The *region automaton* [AD94] $\mathcal{A}_r$ for the timed automaton $\mathcal{A}$ is a finite automaton, whose states are the equivalence classes of an equivalence relation with finite index over the states of the timed transition system associated with $\mathcal{A}$. Each of these equivalence classes can be represented by a constraint in $\mathcal{D}[X, c_{max}]$. As the region graph is finite, it underlies most decidability results for timed automata and timed games.

The following example demonstrates that there exist partially observable plants with timed automata in which all constants are non-negative integers, where the necessary action points are not bounded by the maximal constant in the plant automaton.

Example 6.3.1. A controller for the plant depicted in Fig. 6.2 does not require any decision predicates. In order to avoid the error location l_5, the controller has to execute the controllable action a during the time the plant is in location l_3. It takes the plant between 6 and 9 time units from the start to enter location l_3, and thus it is guaranteed

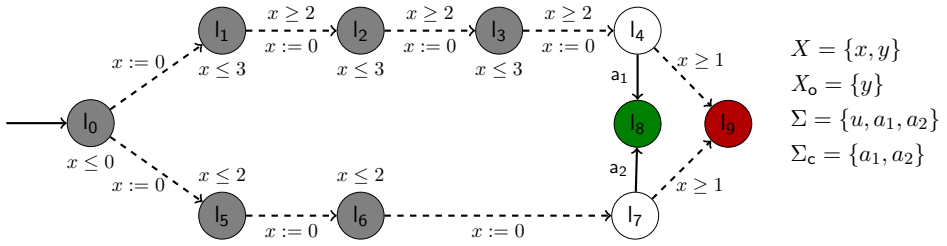


Figure 6.3: Partially observable plant for which the predicates describing the region automaton do not suffice as decision predicates. The plant has two clocks x and y, and only y is observable by the controller. The plant has an uncontrollable action u (uncontrollable edges in the automaton are denoted with dashed lines). For readability, the edges with controllable actions a_1 and a_2 leading to l_9 from locations $l_0, l_1, l_2, l_3, l_5, l_6, l_8, l_9$, and locations l_7 and l_4, respectively, have been omitted from the figure. The equivalence relation between locations has been encoded using colors depicting the equivalence classes.

to be in this location when $y \in (9, 10]$. Since y is the only observable clock, the winning action points are $y = c$, where $c \in (9, 10]$, for which clearly $y = c \notin \mathcal{D}[X_o, 3]$.

Now we give another example, in which no action points are needed by the controller, but there does not exist a controller whose decision predicates are in the set $\mathcal{D}[X_o, c_{max}]$, where c_{max} is the maximal constant in the plant automaton.

Example 6.3.2. For the plant in Fig. 6.3 no action points are necessary, as the controller can observe when the location changes from l_3 to l_4 and from l_6 to l_7, and executing the *correct* controllable action upon entering l_4 or l_7 results in a control strategy that avoids the error location l_9. However, in order to decide which controllable action to execute so as to to avoid location l_9, the controller has to be able to distinguish locations l_4 and l_7, which is not possible by using the constraints in $\mathcal{D}[X_o, 3]$. The decision predicate $y \leq 4$ suffices to distinguish the two locations, since when the plant enters l_4 we have $6 \leq y \leq 9$ and when it enters l_7 we have $0 \leq y \leq 4$.

6.3.3 Finite Control Strategies

We will be interested in controller strategies with finite memory, since such strategies can be implemented by timed automata, i.e., as *controller automata*.

Definition 6.3.1. Let $\mathcal{N} = (\mathcal{A}, \Sigma_c, \Sigma_u, X_o, X_u, =_o^l)$ be a partially observable plant with $\mathcal{A} = (\mathsf{Loc}, l_0, X, \Sigma, \mathsf{Inv}, \delta)$ and let X_c be a finite set of clocks with $X_c \cap X = \emptyset$.

A *finite controller automaton for* $\mathcal{N}$ *with clocks* X_{c} is a time-deterministic timed automaton $\mathcal{C} = (\widetilde{\mathsf{Loc}}, \widetilde{l}_0, \widetilde{X}, \widetilde{\Sigma}, \widetilde{\mathsf{Inv}}, \widetilde{\delta})$, where $\widetilde{X} = X_{\mathsf{o}} \dot\cup X_{\mathsf{c}}$, $\widetilde{\Sigma} = \widetilde{\Sigma}_1 \dot\cup \widetilde{\Sigma}_2$ and

$$\widetilde{\Sigma}_1 = \{(\sigma, o_1, o_2, Y) \mid \sigma \in \Sigma_{\mathsf{c}}, o_1, o_2 \in \mathit{Obs}_{\mathsf{L}} \text{ and } Y \subseteq X_{\mathsf{o}}\} \text{ and}$$

$$\widetilde{\Sigma}_2 = \{(\tau, o_1, o_2, Y) \mid o_1, o_2 \in \mathit{Obs}_{\mathsf{L}}, Y \subseteq X_{\mathsf{o}}, \text{ and it holds that } o_1 \neq o_2 \text{ or } Y \neq \emptyset\},$$

where τ is some fixed action label such that $\tau \notin \Sigma$. The mapping $\widetilde{\mathsf{Inv}}$ is such that for each $l \in \widetilde{\mathsf{Loc}}$, $\widetilde{\mathsf{Inv}}(l) = \mathit{true}$ and the transition relation $\widetilde{\delta}$ is such that

- $\widetilde{\delta} \subseteq \widetilde{\mathsf{Loc}} \times ((\widetilde{\Sigma}_1 \times \mathcal{C}[\widetilde{X}]) \cup (\widetilde{\Sigma}_2 \times \mathcal{B}[\widetilde{X}])) \times 2^{X_{\mathsf{c}}} \times \widetilde{\mathsf{Loc}}$,
- for each $l \in \widetilde{\mathsf{Loc}}$ and each $\sigma \in \Sigma_{\mathsf{c}}$ there exists a formula $(x \geq c) \in \mathcal{C}[\widetilde{X}]$ such that for each transition $(l, (\sigma, o_1, o_2, Y), g, Z, l') \in \widetilde{\delta}$ it holds that $g = (x \geq c)$, and
- for each $l \in \widetilde{\mathsf{Loc}}$ and each $\sigma \in \widetilde{\Sigma}_2$, $(\bigvee_{(l,\sigma,g,Z,l') \in \widetilde{\delta}} g) \equiv \mathit{true}$.

A set of transitions $\widetilde{\delta}' \subseteq \widetilde{\delta} \cap (\widetilde{\mathsf{Loc}} \times (\widetilde{\Sigma}_1 \times \mathcal{C}[\widetilde{X}]) \times 2^{X_{\mathsf{c}}} \times \widetilde{\mathsf{Loc}})$ in $\widetilde{\delta}$ labeled with controllable actions are *urgent*. This means that the respective source location must be left without (further) delay when the transition is enabled (i.e., when its guard is satisfied).

According to the above definition, the set of clocks of the controller automaton consists of the observable clocks of the plant and the controller's own clocks, where only the latter ones can be reset by the controller. The controller synchronizes with the plant on discrete controllable transitions and on the discrete changes in the state-based observation (i.e., observable location change or observable clock reset). The urgency condition for the controllable actions ensures these actions are taken as soon as they are enabled by the controller automaton. Notice that, as it is usually done, urgency can be encoded by adding invariants for the respective locations of the controller automaton (and possibly introducing a fresh clock variable to the controller automaton).

It is not difficult to see how a controller automaton for a partially observable plant defines a control strategy for this plant and yields the respective set of controlled paths.

In order to define the syntactic product of the timed automata for plant and controller, we would have to rename some of the actions in both automata. More precisely, for the transitions of the plant automaton that have observable effect (i.e., reset an observable clock or change the equivalence class of the plant location) we add this effect to the transition label. In addition, in the controller automaton, the transition labels

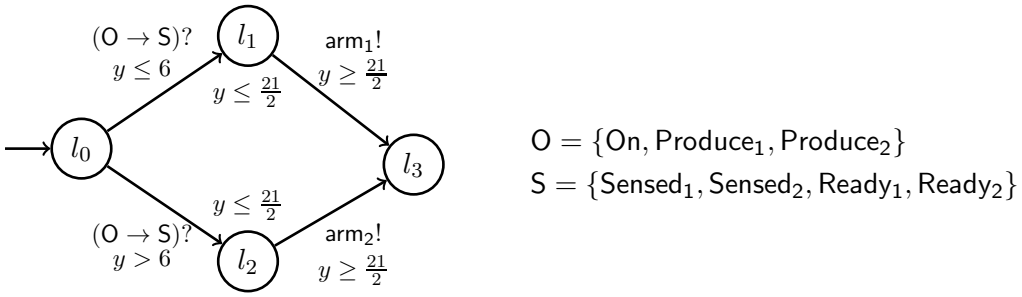


Figure 6.4: A controller automaton for the partially observable plant in Figure 6.1. The invariants at locations l_0 and l_1 ensure that the actions $\mathsf{arm}_1!$ and $\mathsf{arm}_2!$ are urgent.

are augmented with the actions of the plant automaton that have the respective observable effect. Then, the standard definition of product of timed automata yields a timed automaton in which the set of paths is exactly the set of controlled paths.

Figure 6.4 depicts a controller automaton for the partially observable plant shown in Figure 6.1, that enforces that in the controlled plant the location Err is not reachable. The transition relation of the controller automaton branches according to the value of the clock y when the equivalence class of the plant's location changes from O to S. As soon as y reaches the value $\frac{21}{2}$, the respective action $\mathsf{arm}_1!$ or $\mathsf{arm}_2!$ is executed.

An X_c-control strategy for a partially observable plant cannot always be represented as a finite automaton with clocks X_c. We give an example of a partially observable plant for which there does not exist a finite-state control strategy to avoid the error.

Example 6.3.3. For the partially observable plant shown in Figure 6.5, there does not exist a controller automaton (which by definition should be a finite timed automaton) that does not use additional clocks. In order for a control strategy to ensure that the error location l_3 is not reachable on any of the controlled paths, it must ensure that the controllable action a is taken when the location is l_1 and when the value of the clock x is 1. Since the clock x cannot be observed by the controller, the value of the observable (but not controllable) clock y when the action must be taken depends on its value at the time when the plant's location changes from l_0 to l_1. Hence, in a controller automaton that does have any controllable clocks, we would need one location for each such value, which means that there must be infinitely many locations in the automaton.

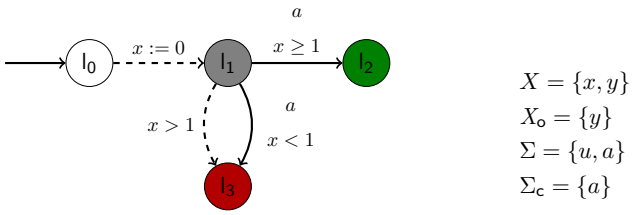


Figure 6.5: Partially observable plant for which there does not exist a finite controller automaton. The plant has an uncontrollable action u (uncontrollable edges in the automaton are denoted with dashed lines) and a controllable action a. The location l_3 is an error location. For readability we have omitted the transitions labeled with a and with guard *true* from locations l_0, l_2 and l_3 to l_3. The equivalence relation between locations has been encoded using colors depicting the equivalence classes.

Chapter 7

Synthesis of Observation Predicates for Timed Control

7.1 Await-Time Games

In this section we introduce *await-time games* and show that the timed safety controller synthesis problem with partial observability reduces to the problem of finding a winning strategy for the controller (the existential player) in an await-time game against the plant (the universal player). As we will see, await-time games are a convenient and natural representation of the timed safety controller synthesis problem with partial observability. They allow us to represent the infinite number of choices available to the controller in a suitable way using the variables in the symbolic game structure.

Construction of the Await-Time Game

We fix a partially observable plant $\mathcal{N} = (\mathcal{A}, \Sigma_{\mathsf{c}}, \Sigma_{\mathsf{u}}, X_{\mathsf{o}}, X_{\mathsf{u}}, =_{\mathsf{o}}^{\mathsf{L}})$ with timed automaton $\mathcal{A} = (\mathsf{Loc}, l_0, X, \Sigma, \mathsf{Inv}, \delta)$, and an error location $l_{Err} \in \mathsf{Loc}$ with $[l_{Err}]_{=_{\mathsf{o}}^{\mathsf{L}}} = \{l_{Err}\}$.

Let X_{c} be a fixed finite set of controllable clocks such that $X_{\mathsf{c}} \cap X = \emptyset$.

An await-time game models the interaction between a X_{c}-control strategy ($Player_{\exists}$), and the partially observable plant $\mathcal{N}$, i.e, the controller's environment, ($Player_{\forall}$), in a turn-based manner. Whenever it is his turn, $Player_{\exists}$ has the possibility to propose what controllable action should be executed and when. Then, $Player_{\forall}$ can do one or more transitions executing the actual actions of the plant, i.e., updating the location and all clocks, respecting the choice of $Player_{\exists}$. Since the controller and the plant

synchronize when a controllable action is executed or a discrete change in the state-based observation has occurred, the turn is back to $Player_\exists$ as soon as this happens.

The *await-time game* for $\mathcal{N}$, l_{Err} and X_c is the game $\mathsf{Safety}(\mathcal{G}(\mathcal{N}, l_{Err}, X_\mathsf{c}), \varphi_{Err})$, where the symbolic game structure $\mathcal{G}(\mathcal{N}, l_{Err}, X_\mathsf{c}) = (V_\exists, V_\forall, V_\forall^o, t, \varphi_{Init}, \mathcal{T}_\exists, \mathcal{T}_\forall)$ and the formula φ_{Err} are defined as described in the remainder of this section.

We denote the corresponding explicit game structure with $G(\mathcal{N}, l_{Err}, X_\mathsf{c})$.

Variables. As usual, we let $V = V_\exists \dot\cup V_\forall \dot\cup \{t\}$ be the set of all variables in $\mathcal{G}$.

$Player_\exists$ updates the variables in $V_\exists$, which model the decisions of the controller. We partition the set $V_\exists$ into two sets: the set $V_\exists^f = \{v \in V_\exists \mid |\mathsf{Dom}(v)| < \infty\}$ consists of the variables from $V_\exists$ that have finite domains, and $SC = \{v \in V_\exists \mid |\mathsf{Dom}(v)| = \mathbb{Q}_{\geq 0}\}$ consists of the variables whose domains are infinite. Thus, $V_\exists = V_\exists^f \dot\cup SC$, where:

- $V_\exists^f = \{act, wait, reset\}$, $\mathsf{Dom}(act) = \Sigma_\mathsf{c} \cup \{\flat\}$, $\mathsf{Dom}(wait) = \mathbb{B}$, $\mathsf{Dom}(reset) = 2^{X_\mathsf{c}}$,
- $SC = \{c_x \mid x \in X_{\mathsf{o}+\mathsf{c}}\}$ consists of variables called *symbolic constants* and contains exactly one symbolic constant c_x for each $x \in X_{\mathsf{o}+\mathsf{c}}$, where $\mathsf{Dom}(c_x) = \mathbb{Q}_{\geq 0}$.

When $Player_\exists$ choses to execute a controllable action from Σ_c, the variable *act* indicates the selected action. $Player_\exists$ sets *wait* to *true* if he wants to let time elapse (before executing a controllable action or when remaining idle). The variable *reset* stores a set of controllable clocks $Player_\exists$ wants to reset. The values of the symbolic constants determine when $Player_\exists$ chose to execute the selected controllable action.

$Player_\forall$ updates the variables in $V_\forall$, which encode the state of the plant. Recall that the set $V_\forall^o \subseteq V_\forall$ consists of the variables belonging to $Player_\forall$ that $Player_\exists$ can read. Here, the set $V_\forall^o$ is defined according to what the controller is allowed to observe. Thus, we have $V_\forall = V_\forall^o \dot\cup V_\forall^u$, where we have defined:

- $V_\forall^o = \{oloc, er\} \dot\cup X_{\mathsf{o}+\mathsf{c}}$, where $\mathsf{Dom}(oloc) = Obs_\mathsf{L}$ and $\mathsf{Dom}(er) = \mathbb{B}$,
- $V_\forall^u = \{loc, et\} \dot\cup X_\mathsf{u}$, where $\mathsf{Dom}(loc) = \mathsf{Loc}$ and $\mathsf{Dom}(et) = \mathbb{B}$.

The variables *loc* and *oloc* store the plant's current location and the equivalence class of the current location. The auxiliary boolean variable *er* indicates in which states $Player_\exists$ can choose to reset clocks from X_c. The unobservable auxiliary variable *et* has value *false* in the states in which $Player_\forall$ has disabled further delay transitions.

Initial states. The set of initial states of $\mathcal{G}(\mathcal{N}, l_{Err}, X_{\mathsf{c}})$ is characterized by the formula

$$\begin{aligned}\varphi_{Init} \;:=\; & t = \exists \wedge act = \flat \wedge wait \wedge reset = \emptyset \wedge \bigwedge_{x \in X_{\mathsf{o}+\mathsf{c}}} c_x = 0 \wedge \\ & loc = l_0 \wedge oloc = [l_0]_{=^{\mathsf{L}}_{\mathsf{o}}} \wedge \bigwedge_{x \in X \cup X_{\mathsf{c}}} x = 0 \wedge \neg er \wedge et.\end{aligned}$$

Thus, in the unique initial state, which belongs to $Player_\exists$ all variables of $Player_\exists$ have some default values, the plant is in its initial location and all clocks of the plant and the controller are 0. Thus, $Player_\exists$ is not allowed to reset the controller's clocks.

Transition relation of $Player_\exists$. The transitions of $Player_\exists$ correspond to the possible choices of the controller. We distinguish four types of transitions, according to the different cases in the definition of $Act_\exists$ in Section 6.2. In case (i) $Player_\exists$ chooses an action $\sigma \in \Sigma_{\mathsf{c}}$ to be executed after a positive delay, in case (ii) $Player_\exists$ chooses an action $\sigma \in \Sigma_{\mathsf{c}}$ to be executed immediately, in case (iii) $Player_\exists$ chooses to remain idle, and in case (iv) $Player_\exists$ selects a set of clocks from X_{c} to be reset. Thus, we define

$$\mathcal{T}_\exists[V, V'] := t = \exists \wedge t' = \forall \wedge \mathsf{preserve}(V_\forall) \wedge \Big(\bigwedge_{x \in X_{\mathsf{o}+\mathsf{c}}} c'_x \geq 0 \Big) \wedge \big(\mathcal{T}_\exists^{(\mathrm{i})} \vee \mathcal{T}_\exists^{(\mathrm{ii})} \vee \mathcal{T}_\exists^{(\mathrm{iii})} \vee \mathcal{T}_\exists^{(\mathrm{iv})} \big),$$

where for a set of variables $U \subseteq V$ the formula $\mathsf{preserve}(U)$ is a shortcut for the formula $\bigwedge_{u \in U}(u' = u)$ and the formulas $\mathcal{T}_\exists^{(\mathrm{i})}$, $\mathcal{T}_\exists^{(\mathrm{ii})}$, $\mathcal{T}_\exists^{(\mathrm{iii})}$ and $\mathcal{T}_\exists^{(\mathrm{iv})}$ are defined as follows:

$$\begin{aligned}\mathcal{T}_\exists^{(\mathrm{i})} \;:=\; & \big(\textstyle\bigvee_{\sigma \in \Sigma_{\mathsf{c}}} act' = \sigma \big) \wedge wait' \wedge reset' = \emptyset \wedge \big(\textstyle\bigvee_{x \in X_{\mathsf{o}+\mathsf{c}}} c'_x > 0 \big) \wedge \\ & \qquad \textstyle\bigwedge_{x \in X_{\mathsf{o}+\mathsf{c}}} (c'_x > 0 \rightarrow c'_x > x), \\ \mathcal{T}_\exists^{(\mathrm{ii})} \;:=\; & \big(\textstyle\bigvee_{\sigma \in \Sigma_{\mathsf{c}}} act' = \sigma \big) \wedge \neg wait' \wedge reset' = \emptyset \wedge \textstyle\bigwedge_{x \in X_{\mathsf{o}+\mathsf{c}}} c'_x = 0, \\ \mathcal{T}_\exists^{(\mathrm{iii})} \;:=\; & act' = \flat \wedge wait' \wedge reset' = \emptyset \wedge \textstyle\bigwedge_{x \in X_{\mathsf{o}+\mathsf{c}}} c'_x = 0, \\ \mathcal{T}_\exists^{(\mathrm{iv})} \;:=\; & \textstyle\bigvee_{\substack{Z \subseteq X_{\mathsf{c}}, \\ Z \neq \emptyset}} \big(er \wedge act' = \flat \wedge \neg wait' \wedge reset' = Z \wedge \textstyle\bigwedge_{x \in X_{\mathsf{o}+\mathsf{c}}} c'_x = 0 \big).\end{aligned}$$

We call the positive values of variables from SC in case (i) *await points*.

Transition relation of $Player_\forall$. The transitions of $Player_\forall$ correspond to the possible transitions of the plant, respecting the choices of the controller. According to the definition of controlled run in Section 6.2 we consider the following types of transitions:

(i) delay transitions of the plant, (ii) discrete controllable transitions, (iii) discrete uncontrollable transitions and (iv) transitions that reset clocks selected by $Player_\exists$. An additional type of transitions (v) allow $Player_\forall$ to give the turn back to $Player_\exists$ at any point after letting some time elapse, in case $Player_\exists$ has chosen to remain idle.

First, with each $(l, \sigma, g, Y, l') \in \delta$ in $\mathcal{N}$ we associate a formula $\varphi_{(l,\sigma,g,Y,l')}$ over the variables in V and V' that describes the corresponding transition relation:

$$\begin{aligned}\varphi_{(l,\sigma,g,Y,l')}[V,V'] \;:=\; & loc = l \wedge \mathsf{Inv}(l)[X] \wedge g[X] \wedge \bigwedge_{y\in Y} y' = 0 \wedge \\ & loc' = l' \wedge oloc' = [l']_{=^{\mathsf{L}}_{\mathsf{o}}} \wedge \mathsf{Inv}(l')[X'/X] \wedge \mathsf{preserve}(X \setminus Y).\end{aligned}$$

The transition relation formula $\mathcal{T}_\forall$ contains a disjunct for each type of transitions:

$$\mathcal{T}_\forall[V, V'] := t = \forall \wedge \mathsf{preserve}(V_\exists) \wedge \big(\mathcal{T}_\forall^{(\mathrm{i})} \vee \mathcal{T}_\forall^{(\mathrm{ii})} \vee \mathcal{T}_\forall^{(\mathrm{iii})} \vee \mathcal{T}_\forall^{(\mathrm{iv})} \vee \mathcal{T}_\forall^{(\mathrm{v})}\big), \text{ where}$$

$$\begin{aligned}\mathcal{T}_\forall^{(\mathrm{i})} \;:=\; & (wait \wedge et) \wedge \Big(\exists \Delta.\, \Delta > 0 \wedge \textstyle\bigwedge_{x\in X \,\dot\cup\, X_{\mathsf{c}}} (x' = x + \Delta)\Big) \wedge \\ & \textstyle\bigwedge_{l\in \mathsf{Loc}} \big(loc = l \to \mathsf{Inv}(l)[X']\big) \wedge \mathsf{preserve}\big(\{oloc, loc, t, er\}\big) \wedge \\ & \Big(act \neq \flat \to \textstyle\bigwedge_{x\in X_{\mathsf{o}+\mathsf{c}}} (x < c_x \to x' \leq c_x)\Big) \wedge \\ & \Big(act \neq \flat \to \big(et' \leftrightarrow \textstyle\bigwedge_{x\in X_{\mathsf{o}+\mathsf{c}}} (c_x = 0 \vee x \geq c_x \vee x' < c_x)\big)\Big),\end{aligned}$$

$$\begin{aligned}\mathcal{T}_\forall^{(\mathrm{ii})} \;:=\; & (\neg wait \vee \neg et) \wedge \Big(\bigvee_{\substack{\sigma\in\Sigma_{\mathsf{c}},\\ (l,\sigma,g,Y,l')\in\delta}} act = \sigma \wedge \varphi_{(l,\sigma,g,Y,l')} \wedge \mathsf{preserve}(X_{\mathsf{c}})\Big) \wedge \\ & t' = \exists \wedge er' \wedge et',\end{aligned}$$

$$\begin{aligned}\mathcal{T}_\forall^{(\mathrm{iii})} \;:=\; & reset = \emptyset \wedge \Big(\bigvee_{\substack{\sigma\in\Sigma_{\mathsf{u}},\\ (l,\sigma,g,Y,l')\in\delta}} \varphi_{(l,\sigma,g,Y,l')} \wedge \mathsf{preserve}(X_{\mathsf{c}})\Big) \wedge \\ & \Big(t' = \exists \leftrightarrow \big(oloc' \neq oloc \vee \textstyle\bigvee_{x\in X_{\mathsf{o}}} (x > 0 \wedge x' = 0)\big)\Big) \wedge \\ & \Big(t' = \exists \to er' \wedge et'\Big) \wedge \Big(t' = \forall \to er' = er \wedge et' = et\Big),\end{aligned}$$

$$\begin{aligned}\mathcal{T}_\forall^{(\mathrm{iv})} \;:=\; & reset \neq \emptyset \wedge \Big(\textstyle\bigwedge_{x\in reset} x' = 0\Big) \wedge \mathsf{preserve}\big(\{loc, oloc\} \cup ((X \,\dot\cup\, X_{\mathsf{c}}) \setminus reset)\big) \wedge \\ & t' = \exists \wedge \neg er' \wedge et',\end{aligned}$$

$$\begin{aligned}\mathcal{T}_\forall^{(\mathrm{v})} \;:=\; & act = \flat \wedge wait \wedge \neg et \wedge \mathsf{preserve}\big(\{loc, oloc\} \cup (X \,\dot\cup\, X_{\mathsf{c}})\big) \wedge \\ & t' = \exists \wedge \neg er' \wedge et'.\end{aligned}$$

In (i), $Player_\forall$ can let time elapse, if delay transitions are enabled, that is $Player_\exists$ has set *wait* to *true*, and $Player_\forall$ has not disabled further delay transitions. The latter happens upon reaching an await point, as $Player_\forall$ cannot let time pass beyond an await point. The controllable action selected by $Player_\exists$ can be executed only

when time-elapse transitions are disabled (i.e., $Player_\exists$ wanted to execute the transition immediately or an await point has been reached). Once a controllable transition is taken the turn is back to $Player_\exists$. In (iii), an enabled uncontrollable transition can be executed if $Player_\exists$ did not choose to reset controllable clocks. The turn is back to $Player_\exists$ if and only if the transition changes the state-based observation. If $Player_\exists$ chose a non-empty set of controllable clocks to reset, $Player_\forall$ can set these clocks to 0 and give the turn back to $Player_\exists$, disabling consecutive resets of controllable clocks.

Error states. The set of error states in the game $\mathsf{Safety}(\mathcal{G}(\mathcal{N}, l_{Err}, X_\mathsf{c}), \varphi_{Err})$ is described by the formula $\varphi_{Err} := t = \forall \wedge \mathit{oloc} = [l_{Err}]_{=_{\mathsf{L}_\mathsf{o}}}$.

Correctness of the Reduction

As we mentioned in the definition of the observation function obs_r in Section 6.2, the controller observes only those transitions of the plant that correspond to controllable actions or to a discrete change of the state-based observation, i.e., change the equivalence class of the current location or reset clocks that the controller can observe. Thus, the controller does not observe the intermediate sequences of transitions of the plant, nor their number. In particular, the controller does not observe the number or duration of individual delay transitions, nor when and if an uncontrollable action is taken.

Therefore, in order to ensure that the considered strategies of $Player_\exists$ in the corresponding await-time game correspond to control strategies, we define an observation function $obs_r : \mathsf{Prefs}(G(\mathcal{N}, l_{Err}, X_\mathsf{c})) \to Obs^*$, which maps a prefix π to the sequence of observations made by $Player_\exists$ according to what the controller can observe. The function $obs : S \to Obs$ maps a state s to the corresponding equivalence class.

$$obs_r(\pi) = \begin{cases} obs(s) & \text{if } \pi = s \in S_\exists, \\ obs_r(\pi') \cdot obs(s) \cdot obs(s') & \text{if } \pi = \pi' \cdot s \cdot s' \text{ and } s' \in S_\exists, \\ obs_r(\pi') & \text{if } \pi = \pi' \cdot s \text{ and } s \in S_\forall. \end{cases}$$

Unlike in the asynchronous observation function obs_a defined in Section 2.2, we include the observations of the $Player_\forall$ states preceding $Player_\exists$ states, since the controller (and hence $Player_\exists$) can observe the values of the clocks in X_o at the point when an observed transition occurs. We do not use the stuttering-free observation function obs_f defined in Section 2.2, since $Player_\exists$ can read the values of the clocks in $X_{\mathsf{o}+\mathsf{c}}$, but should not observe their intermediate values (changed by unobserved transitions).

The following proposition states that we can reduce the timed safety controller synthesis problem with partial observability to finding an obs_r-consistent winning strategy for $Player_\exists$ in the corresponding await-time game.

Proposition 7.1.1. *If $\mathcal{N} = (\mathcal{A}, \Sigma_\mathsf{c}, \Sigma_\mathsf{u}, X_\mathsf{o}, X_\mathsf{u}, =_\mathsf{o}^\mathsf{L})$ is a partially observable plant, $l_{Err} \in \mathsf{Loc}$ is an error location such that $[l_{Err}]_{=_\mathsf{o}^\mathsf{L}} = \{l_{Err}\}$ and X_c is a finite set of controllable clocks such that $X_\mathsf{c} \cap X = \emptyset$, then there exists a X_c-control strategy f_c for $\mathcal{N}$ such that l_{Err} is not reachable in $\mathcal{CR}(f_\mathsf{c}, \mathcal{N})$ iff $Player_\exists$ has an obs_r-consistent winning strategy in the await-time game $\mathsf{Safety}(\mathcal{G}(\mathcal{N}, l_{Err}, X_\mathsf{c}), \varphi_{Err})$.*

Proof. Let us denote $G = G(\mathcal{N}, l_{Err}, X_\mathsf{c}) = (S_\exists, S_\forall, I, =_\mathsf{o}, \Sigma_\exists, T_\exists, T_\forall)$.

Suppose that f_c is a X_c-control strategy for $\mathcal{N}$ such that l_{Err} is not reachable in $\mathcal{CR}(f_\mathsf{c}, \mathcal{N})$. We define a function $f_\exists : \mathsf{Prefs}_\exists(G) \to \Sigma_\exists$ by first fixing a function $g_1 : \mathsf{Prefs}(G) \to \widetilde{S_\mathcal{A}} \cdot (\widetilde{\Sigma} \cdot \widetilde{S_\mathcal{A}})^*$ such that if $obs_r(\pi_1) = obs_r(\pi_2)$, then $obs_r(g_1(\pi_1)) = obs_r(g_1(\pi_2))$. For $\pi = s$ we define $g_1(\pi) = (s(loc), s(X \dot\cup X_\mathsf{c}))$ and if $\pi = \pi' \cdot s \cdot s'$, then we fix a value $g_1(\pi)$ such that one of the following conditions holds:

- $g_1(\pi) = g_1(\pi' \cdot s)$ and $s \in S_\exists$ or $(s, s') \models \mathcal{T}_\forall^{(v)}$;
- $g_1(\pi) = g_1(\pi' \cdot s) \cdot s(act) \cdot (s'(loc), s'(X \dot\cup X_\mathsf{c}))$ and $(s, s') \models \mathcal{T}_\forall^{(ii)}$;
- $g_1(\pi) = g_1(\pi' \cdot s) \cdot \widetilde{\sigma} \cdot (s'(loc), s'(X \dot\cup X_\mathsf{c}))$, $(s, s') \models \mathcal{T}_\forall^{(i)} \vee \mathcal{T}_\forall^{(iii)}$ and $(s(loc), s(X)) \xrightarrow{\widetilde{\sigma}} (s'(loc), s'(X))$;
- $g_1(\pi) = g_1(\pi' \cdot s) \cdot s(reset) \cdot (s'(loc), s'(X \dot\cup X_\mathsf{c}))$ and $(s, s') \models \mathcal{T}_\forall^{(iv)}$.

Now, let $\pi \in \mathsf{Prefs}_\exists$ and $f_\mathsf{c}(g_1(\pi)) = \sigma'$. We then define $f_\exists(\pi) = \sigma$, where:

- if $\sigma' \in \Sigma_\mathsf{c}$, then $\sigma(act) = \sigma'$, $\sigma(wait) = \mathit{false}$, $\sigma(reset) = \emptyset$, $\sigma(c_x) = 0$ for $c_x \in SC$,
- if $\sigma' = (\sigma, x, c) \in \Sigma_\mathsf{c} \times X_{\mathsf{o}+\mathsf{c}} \times \mathbb{Q}_{>0}$, then $\sigma(act) = \sigma$, $\sigma(wait) = \mathit{true}$, $\sigma(reset) = \emptyset$, $\sigma(c_x) = c$ and $\sigma(c_y) = 0$ for every $y \in X_{\mathsf{o}+\mathsf{c}}$ that is such that $y \neq x$,
- if $\sigma' = \flat$, then $\sigma(act) = \flat$, $\sigma(wait) = \mathit{true}$, $\sigma(reset) = \emptyset$, $\sigma(c_x) = 0$ for $c_x \in SC$,
- if $\sigma' \subseteq X_\mathsf{c}$, then $\sigma(act) = \flat$, $\sigma(wait) = \mathit{false}$, $\sigma(reset) = \sigma'$, $\sigma(c_x) = 0$ for $c_x \in SC$.

By the definition of g_1 and since f_c is a control strategy, we have that $f_\exists$ is an obs_r-consistent strategy for $Player_\exists$. Since l_{Err} is not reachable in $\mathcal{CR}(f_\mathsf{c}, \mathcal{N})$, it is easy to see from the definition of $f_\exists$ that $f_\exists$ is a wining strategy for $Player_\exists$.

For the other direction, suppose that $f_\exists$ is an obs_r-consistent winning strategy in the await-time game $\mathsf{Safety}(\mathcal{G}(\mathcal{N}, l_{Err}, X_\mathsf{c}), \varphi_{Err})$. Similarly to above we define a total

function $f_{\mathsf{c}} : \widetilde{S_{\mathcal{A}}} \cdot (\widetilde{\Sigma} \cdot \widetilde{S_{\mathcal{A}}})^* \to \Sigma_{\mathsf{c}} \cup (\Sigma_{\mathsf{c}} \times X_{\mathsf{o}+\mathsf{c}} \times \mathbb{Q}_{>0}) \cup \{\flat\} \cup \Sigma_{\mathsf{r}}$ by first fixing a partial function $g_2 : \widetilde{S_{\mathcal{A}}} \cdot (\widetilde{\Sigma} \cdot \widetilde{S_{\mathcal{A}}})^* \to \mathsf{Prefs}(G)$ such that if $obs_r(\pi_1) = obs_r(\pi_2)$ and $g_2(\pi_1)$ and $g_2(\pi_2)$ are defined, then $obs_r(g_2(\pi_1)) = obs_r(g_2(\pi_2))$. If $\pi = \widetilde{s}$, then we define $g_2(\pi) = s_0$, where $I = \{s_0\}$. Now let $\pi = \pi' \cdot \widetilde{s} \cdot \widetilde{\sigma} \cdot \widetilde{s}'$. Then $g_2(\pi)$ is defined iff $g_2(\pi' \cdot \widetilde{s})$ is defined and there exist states s' and s'' with the properties stated below and then

$$g_2(\pi) = \begin{cases} g_2(\pi' \cdot \widetilde{s}) \cdot s'' \cdot s' & \text{if } \mathsf{last}(g_2(\pi' \cdot \widetilde{s})) \in S_\exists \\ g_2(\pi' \cdot \widetilde{s}) \cdot s' & \text{if } \mathsf{last}(g_2(\pi' \cdot \widetilde{s})) \in S_\forall. \end{cases}$$

For s'' we let $s'' = s'''$ if $s''' = \mathsf{last}(g_2(\pi' \cdot \widetilde{s})) \in S_\forall$. Otherwise, we let $s''(V_\forall) = s'''(V_\forall)$, $s''(t) = \forall$ and $s''(V_\exists) = f_\exists(g_2(\pi' \cdot \widetilde{s}))$. The state s' must satisfy the conditions:

- $s'(loc) = l$ and $s'(X \cup X_{\mathsf{c}}) = v$, where $\widetilde{s}' = (l, v)$,
- if $\widetilde{\sigma} \in \mathbb{R}_{>0}$, then $(s'', s') \models \mathfrak{T}_\forall^{(i)}$ and then, if $s''(act) = \flat$, then $s'(et) = true$;
- if $\widetilde{\sigma} \in \Sigma_{\mathsf{c}}$, then $(s'', s') \models \mathfrak{T}_\forall^{(ii)}$;
- if $\widetilde{\sigma} \in \Sigma_{\mathsf{u}}$, then $(s'', s') \models \mathfrak{T}_\forall^{(iii)}$;
- if $\widetilde{\sigma} \subseteq X_{\mathsf{c}}$, then $(s'', s') \models \mathfrak{T}_\forall^{(iv)}$.

By the definition of $T_\forall$ and the above requirements, if s' exists, it is unique.

Consider $\pi \in \widetilde{S_{\mathcal{A}}} \cdot (\widetilde{\Sigma} \cdot \widetilde{S_{\mathcal{A}}})^*$. If $g_2(\pi')$ is undefined for all π' with $obs_r(\pi) = obs_r(\pi')$, we let $f_{\mathsf{c}}(\pi) = \flat$. Otherwise, let $\sigma = f_\exists(g_2(\pi'))$, where $\pi' \in \widetilde{S_{\mathcal{A}}} \cdot (\widetilde{\Sigma} \cdot \widetilde{S_{\mathcal{A}}})^*$ is some for which $g_2(\pi')$ is defined and $obs_r(\pi) = obs_r(\pi')$. We then define $f_{\mathsf{c}}(\pi) = \widetilde{\sigma}$, where:

- if $\sigma(act) \in \Sigma_{\mathsf{c}}$ and $\sigma(wait) = false$, then $\widetilde{\sigma} = \sigma(act)$,
- if $\sigma(act) \in \Sigma_{\mathsf{c}}$ and $\sigma(wait) = true$, then $\widetilde{\sigma} = (\sigma(act), x, \sigma(c_x))$, where $x \in X_{\mathsf{o}+\mathsf{c}}$ is such that $\sigma(c_x) > 0$ and for each $y \in X_{\mathsf{o}+\mathsf{c}}$ that is such that $y \neq x$ and $\sigma(c_y) > 0$, it holds that $\sigma(c_x) - s(x) \leq \sigma(c_y) - s(y)$, where $s = \mathsf{last}(g_2(\pi'))$,
- if $\sigma(\Sigma_{\mathsf{c}}) = \flat$ and $wait = true$, then $\widetilde{\sigma} = \flat$,
- if $\sigma(reset) = Z \neq \emptyset$, then if $Z' = \{z \in Z \mid s(z) > 0\} \neq \emptyset$, where $s = \mathsf{last}(g_2(\pi'))$, we let $\widetilde{\sigma} = Z'$ and otherwise we define $\widetilde{\sigma}$ using $f_\exists(\pi'')$, where $\pi'' = g_2(\pi') \cdot s' \cdot s''$, s' is the unique state in G such that $(s, \sigma, s') \in T_\exists$ and s'' is the unique according to the definition of $T_\forall$ in the game structure $G(\mathcal{N}, l_{Err}, X_{\mathsf{c}})$ successor of s'.

By the definition of g_2 and since $f_\exists$ is an obs_r-consistent strategy, we have that f_{c} is a control strategy for the plant $\mathcal{N}$. Since the strategy $f_\exists$ is winning for $Player_\exists$ in the game $\mathsf{Safety}(\mathcal{G}(\mathcal{N}, l_{Err}, X_{\mathsf{c}}), \varphi_{Err})$, we have that l_{Err} is not reachable in $\mathcal{CR}(f_{\mathsf{c}}, \mathcal{N})$. □

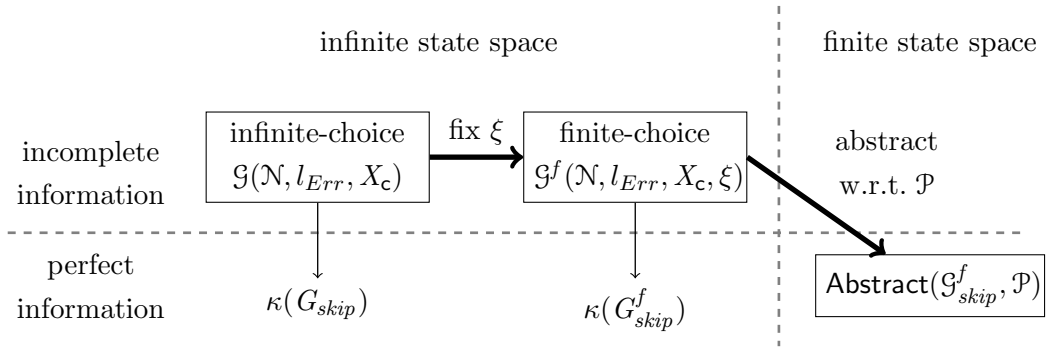


Figure 7.1: Overview of the abstraction process. An await-time game structure $\mathcal{G}(\mathcal{N}, l_{Err}, X_c)$ is first abstracted into a finite-choice game structure $\mathcal{G}^f(\mathcal{N}, l_{Err}, X_c, \xi)$, and then into a finite-state prefect-information game structure $\mathsf{Abstract}(\mathcal{G}^f_{skip}, \mathcal{P})$.

7.2 Fixed-Observations Abstraction

This section describes an abstraction-based approach to solving await-time games. We develop a two-step abstraction which transforms an await-time game, which is an infinite-state game under incomplete information with a safety winning condition, into a finite-state perfect-information game with a safety winning condition. In the first step we construct for a given partially observable plant, error location and a set of controllable clocks, an abstraction of the corresponding await-time game w.r.t. a *fixed finite set of action points*. In the second step the resulting game is abstracted w.r.t. a *fixed finite set of predicates* following the approach described in Chapter 5. Thus, by fixing also the decision predicates for the controller, in the second step we completely fix the set of observation predicates that can be used by the control function.

Figure 7.1 gives an overview of the abstraction process. The original await-time game is *infinite-choice*, i.e., in each state there is an infinite number of choices available to the controller. After fixing the action points, we obtain a *finite-choice* game, which is still an infinite-state game under incomplete information. The final game resulting from applying predicate abstraction is a finite-state game with perfect information.

7.2.1 Abstraction with Fixed Action Points

The symbolic constants in the await-time game structure for given partially observable plant and set of controllable clocks are abstracted away using an *action-point function*.

The abstraction leaves $\mathit{Player}_\exists$ with a finite state of possible choices in each of his states, and lets $\mathit{Player}_\forall$ resolve the resulting nondeterminism.

Definition 7.2.1. An *action-point function* $\xi : X_{\mathsf{o}+\mathsf{c}} \to 2^{\mathbb{Q}_{>0}}$ for a partially observable plant $\mathcal{N} = (\mathcal{A}, \Sigma_\mathsf{c}, \Sigma_\mathsf{u}, X_\mathsf{o}, X_\mathsf{u}, =^{\mathsf{L}}_{\mathsf{o}})$, with $\mathcal{A} = (\mathsf{Loc}, l_0, X, \Sigma, \mathsf{Inv}, \delta)$, and a finite set of controllable clocks X_c, with $X_\mathsf{c} \cap X = \emptyset$, is a function $\xi : X_{\mathsf{o}+\mathsf{c}} \to 2^{\mathbb{Q}_{>0}}$ that maps each $x \in X_{\mathsf{o}+\mathsf{c}}$ to a *finite* set of positive rational constants called *action points for* x.

We fix a partially observable plant $\mathcal{N} = (\mathcal{A}, \Sigma_\mathsf{c}, \Sigma_\mathsf{u}, X_\mathsf{o}, X_\mathsf{u}, =^{\mathsf{L}}_{\mathsf{o}})$ with timed automaton $\mathcal{A} = (\mathsf{Loc}, l_0, X, \Sigma, \mathsf{Inv}, \delta)$, an error location $l_{Err} \in \mathsf{Loc}$ with $[l_{Err}]_{=^{\mathsf{L}}_{\mathsf{o}}} = \{l_{Err}\}$ and a finite set of controllable clocks X_c such that $X_\mathsf{c} \cap X = \emptyset$.

Given an action point function ξ, the *finite-choice await-time game* for the game $\mathsf{Safety}(\mathcal{G}(\mathcal{N}, l_{Err}, X_\mathsf{c}), \varphi_{Err})$ is $\mathsf{Safety}(\mathcal{G}^f(\mathcal{N}, l_{Err}, X_\mathsf{c}, \xi), \varphi_{Err})$, where the symbolic game structure $\mathcal{G}^f(\mathcal{N}, l_{Err}, X_\mathsf{c}, \xi)$ is obtained from $\mathcal{G}(\mathcal{N}, l_{Err}, X_\mathsf{c})$ by replacing the symbolic constant c_x for each $x \in X_{\mathsf{o}+\mathsf{c}}$ with the action points $\xi(x)$ as described below.

We let $\mathcal{G}^f(\mathcal{N}, l_{Err}, X_\mathsf{c}, \xi) = (V^f_\exists, V_\forall, V^o_\forall, t, \varphi^f_{Init}, \mathcal{T}^f_\exists, \mathcal{T}^f_\forall)$, where the sets $V_\forall$ and $V^o_\forall$ of $\mathit{Player}_\forall$ variables are the ones from $\mathcal{G}(\mathcal{N}, l_{Err}, X_\mathsf{c})$. Let $V^f = V^f_\exists \,\dot\cup\, V_\forall \,\dot\cup\, \{t\}$.

We denote the corresponding explicit game structure with $G^f(\mathcal{N}, l_{Err}, X_\mathsf{c}, \xi)$.

Variables for $\mathit{Player}_\exists$. The set of $\mathit{Player}_\exists$ variables in $\mathcal{G}^f(\mathcal{N}, l_{Err}, X_\mathsf{c}, \xi)$ is the set $V^f_\exists$ of finite-range variables from the game structure $\mathcal{G}(\mathcal{N}, l_{Err}, X_\mathsf{c})$.

Initial states. The set of initial states is defined by the formula

$$\begin{aligned}\varphi^f_{Init} \quad := \quad & t = \exists \wedge \mathit{act} = \flat \wedge \mathit{wait} \wedge \mathit{reset} = \emptyset \wedge \\ & \mathit{loc} = l_0 \wedge \mathit{oloc} = [l_0]_{=^{\mathsf{L}}_{\mathsf{o}}} \wedge \bigwedge_{x \in X \cup X_\mathsf{c}} x = 0 \wedge \neg \mathit{er} \wedge \mathit{et}.\end{aligned}$$

The formula φ^f_{Init} differs from the formula φ_{Init} in $\mathcal{G}(\mathcal{N}, l_{Err}, X_\mathsf{c})$ in that it does not assert initial values for the variables from SC, which are not part of $\mathcal{G}^f(\mathcal{N}, l_{Err}, X_\mathsf{c}, \xi)$.

Transition relation of $\mathit{Player}_\exists$. As in $\mathcal{G}(\mathcal{N}, l_{Err}, X_\mathsf{c})$, we distinguish four types of transitions for $\mathit{Player}_\exists$. The only difference is in case (i), where $\mathit{Player}_\exists$ chooses an action $\sigma \in \Sigma_\mathsf{c}$ to be executed after a positive delay. The difference lies in the fact that now, the exact point when the selected controllable action is taken will be determined

by $Player_\forall$, since the symbolic constants are not part of $Player_\exists$'s variables any more. Thus, the transition relation of $Player_\exists$ is described by the formula

$$\mathcal{T}^f_\exists[V, V'] := t = \exists \wedge t' = \forall \wedge \mathsf{preserve}(V_\forall) \wedge \big(\mathcal{T}^{(\mathrm{i})f}_\exists \vee \mathcal{T}^{(\mathrm{ii})f}_\exists \vee \mathcal{T}^{(\mathrm{iii})f}_\exists \vee \mathcal{T}^{(\mathrm{iv})f}_\exists\big),$$

where the formulas $\mathcal{T}^{(\mathrm{i})f}_\exists$, $\mathcal{T}^{(\mathrm{ii})f}_\exists$, $\mathcal{T}^{(\mathrm{iii})f}_\exists$ and $\mathcal{T}^{(\mathrm{iv})f}_\exists$ are defined as follows.

$$\begin{aligned}
\mathcal{T}^{(\mathrm{i})f}_\exists &:= \big(\textstyle\bigvee_{\sigma\in\Sigma_\mathsf{c}} act' = \sigma\big) \wedge wait' \wedge reset' = \emptyset,\\
\mathcal{T}^{(\mathrm{ii})f}_\exists &:= \big(\textstyle\bigvee_{\sigma\in\Sigma_\mathsf{c}} act' = \sigma\big) \wedge \neg wait' \wedge reset' = \emptyset,\\
\mathcal{T}^{(\mathrm{iii})f}_\exists &:= act' = \flat \wedge wait' \wedge reset' = \emptyset,\\
\mathcal{T}^{(\mathrm{iv})f}_\exists &:= \textstyle\bigvee_{\substack{Z\subseteq X_\mathsf{c},\\ Z\neq\emptyset}} \big(er \wedge act' = \flat \wedge \neg wait' \wedge reset' = Z\big).
\end{aligned}$$

Transition relation of $Player_\forall$. The transition relation for $Player_\forall$ again contains a disjunct for each type of transitions:

$$\mathcal{T}^f_\forall[V, V'] = t = \forall \wedge \mathsf{preserve}(V^f_\exists) \wedge \big(\mathcal{T}^{(\mathrm{i})f}_\forall \vee \mathcal{T}^{(\mathrm{ii})}_\forall \vee \mathcal{T}^{(\mathrm{iii})}_\forall \vee \mathcal{T}^{(\mathrm{iv})}_\forall \vee \mathcal{T}^{(\mathrm{v})}_\forall\big), \text{ where}$$

$$\begin{aligned}
\mathcal{T}^{(\mathrm{i})f}_\forall := {} & (wait \wedge et) \wedge \Big(\exists\Delta.\, \Delta > 0 \wedge \textstyle\bigwedge_{x\in X \,\dot\cup\, X_\mathsf{c}} (x' = x + \Delta)\Big) \wedge\\
& \textstyle\bigwedge_{l\in\mathsf{Loc}} \big(loc = l \rightarrow \mathsf{Inv}(l)[X']\big) \wedge \mathsf{preserve}(\{oloc, loc, t, er\}) \wedge\\
& \Big(\textstyle\bigwedge_{x\in X_{\mathsf{o}+\mathsf{c}}, c\in\xi(x)} (x < c \rightarrow x' \leq c)\Big) \wedge\\
& \Big(et' \rightarrow \textstyle\bigwedge_{x\in X_{\mathsf{o}+\mathsf{c}}, c\in\xi(x)} (x \geq c \vee x' < c)\Big).
\end{aligned}$$

The nondeterminism resulting from replacing $\mathcal{T}^{(\mathrm{i})}_\exists$ by $\mathcal{T}^{(\mathrm{i})f}_\exists$, i.e., how much time should elapse before the selected controllable action is executed, is resolved by $Player_\forall$. The controllable action can now be fired at any time *up to (and including) the first action point* is reached, after a positive amount of time has elapsed. This is achieved by replacing $\mathcal{T}^{(\mathrm{i})}_\forall$ by $\mathcal{T}^{(\mathrm{i})f}_\forall$, which differs in the following. First, $\mathcal{T}^{(\mathrm{i})f}_\forall$ allows $Player_\forall$ to disable further delay transitions at any point. Second, $\mathcal{T}^{(\mathrm{i})f}_\forall$ requires that the duration of the time-elapse transitions is constrained by the action points, regardless of whether $Player_\exists$ has chosen to execute a controllable action after a delay or to remain idle.

Finite-choice await-time games as abstraction of await-time games. We will now establish the connection between the two game structures $G = G(\mathcal{N}, l_{Err}, X_\mathsf{c}) = (S_\exists, S_\forall, I, =_\mathsf{o}, \Sigma_\exists, T_\exists, T_\forall)$ and $G^f = G^f(\mathcal{N}, l_{Err}, X_\mathsf{c}, \xi) = (S^f_\exists, S^f_\forall, I^f, =^f_\mathsf{o}, \Sigma^f_\exists, T^f_\exists, T^f_\forall)$.

We define functions $\gamma^f : S^f \to 2^S$ and $\gamma^f_\exists : \Sigma^f_\exists \to 2^{\Sigma_\exists}$ that map a state of G^f to a set of states of G and a $Player_\exists$ action in G^f to a set of actions in G.

First, we define the function $AwaitPoints_\xi : S^f \to 2^{Vals(SC)}$, such that if $s^f \in S^f$ and $v \in Vals(SC)$, then $v \in AwaitPoints_\xi(s)$ iff the following conditions are satisfied:

(1) for some $x \in X_{\mathsf{o+c}}$, $v(c_x) > 0$, and

(2) for all $x \in X_{\mathsf{o+c}}$, if $v(c_x) > 0$, then $v(c_x) \geq s^f(x)$,

(3) if $\Delta = \min\{v(c_x) - s^f(x) \mid v(c_x) \geq s^f(x)\}$, then for all $x \in X_{\mathsf{o+c}}$ and all $c \in \xi(x)$ with $c > s^f(x)$ it holds that $s^f(x) + \Delta \leq c$.

Thus, $AwaitPoints_\xi(s)$ consists of those valuations of the symbolic constants that are valid choices for $Player_\exists$ in G (conditions (1) and (2)) which are reached not later than the time when the first action point determined by ξ is reached (condition (3)).

Now, for $s^f \in S^f$ we define

$$\gamma^f(s^f) = \begin{cases} \{s \in S \mid s(V^f) = s^f(V^f) \wedge s(SC) \in AwaitPoints_\xi(s^f)\} & s^f(act) \in \Sigma_\exists \text{ and } s^f(wait) = true \\ \{s \in S \mid s(V^f) = s^f(V^f)\} \cup & \\ \{s \in S \mid s(V^f \setminus \{act\}) = s^f(V^f \setminus \{act\}) \wedge s(act) \in \Sigma_\exists \wedge & \\ \qquad s(SC) \notin AwaitPoints_\xi(s^f)\} & s^f(act) = \flat \text{ and } s^f(wait) = true, \\ \{s \in S \mid s(V^f) = s^f(V^f)\} & \text{otherwise.} \end{cases}$$

Each state s^f in G^f in which $Player_\exists$ has chosen to take a controllable action immediately or to reset some controllable clocks (i.e., where $s^f(wait) = false$), is mapped to the set of states in G each of which agrees with s^f on the values of the variables from V^f and gives arbitrary values to the symbolic constants. The concretization of a state s^f in which $Player_\exists$ has chosen to execute a controllable action after letting time elapse, consists of all states in G which agree with s^f on the values of the variables from V^f and whose valuation of the symbolic constants is a valid choice for $Player_\exists$ in the game structure G and do not exceed the next action point. The concretization of a state s^f in which $Player_\exists$ has chosen to remain idle is the union of two sets. The first set consists of those states in G that agree with s^f on the values of the variables from V^f and give arbitrary values to the symbolic constants. The second set consists of the states in G in which $Player_\exists$ has chosen to execute a controllable action after a

delay, but the valuation of the symbolic constants is not a valid choice for $Player_\exists$ or exceeds the next action point. Thus, intuitively, states in G in which $Player_\exists$ chose to execute a controllable action after the next action-point determined by ξ, correspond to abstract states in which $Player_\exists$ chose to remain idle (until the next action point is reached).

For $\sigma^f \in \Sigma^f_\exists$, we let $\gamma^f_\exists(\sigma^f) = \{\sigma \in \Sigma_\exists \mid \sigma(V^f_\exists) = \sigma^f(V^f_\exists)\}$.

The function γ^f maps each of the finitely many possible actions $\Sigma^f_\exists$ of $Player_\forall$ in the game structure G^f to the corresponding set of infinitely many possible choices $Player_\exists$ has in the game structure G (resulting from the valuations of the variables SC).

It is easy to verify the conditions of Definition 2.3.1 and establish that the game structure $G^f(\mathcal{N}, l_{Err}, X_\mathsf{c}, \xi)$ together with the concretization functions γ^f and $\gamma^f_\exists$ is an abstraction of the game structure $G(\mathcal{N}, l_{Err}, X_\mathsf{c})$.

Soundness of the abstraction. We claim that in the game structure G^f $Player_\forall$ is more powerful than in G, while $Player_\exists$ is less powerful, i.e., the abstraction is sound.

We first extend the function γ^f to play prefixes in G^f, that is, we define the function $\gamma^f : \mathsf{Prefs}(G^f) \to 2^{\mathsf{Prefs}(G)}$. For $\pi^f \in \mathsf{Prefs}(G^f)$, $\gamma^f(\pi^f) \subseteq \mathsf{Prefs}(G)$ is such that $\pi \in \gamma^f(\pi^f)$ iff there exists a total function $idx : [0, |\pi| - 1] \to [0, |\pi^f| - 1]$ such that:

- $idx(0) = 0$, $idx(|\pi| - 1) = |\pi^f| - 1$, $idx(i) < idx(i + 1)$ for each $0 \leq i < |\pi| - 1$,
- for each $0 \leq i < |\pi|$, it holds that $\pi[i] \in \gamma^f(\pi^f[idx(i)])$,
- for each $0 \leq i^f < |\pi^f|$ for which there does not exist $0 \leq i < |\pi|$ such that $idx(i) = i^f$, it holds that there exist $x \in X_{\mathsf{o}+\mathsf{c}}$ and $c \in \xi(x)$, such that $\pi^f[i^f](x) = c$.

The concretization function for prefixes has the following properties:

Property 17. *For $\pi \in \mathsf{Prefs}(G)$, there is exactly one $\pi^f \in \mathsf{Prefs}(G^f)$ with $\pi \in \gamma^f(\pi^f)$.*

Property 18. *For every $\pi_1, \pi_2 \in \mathsf{Prefs}(G)$ with $obs_r(\pi_1) = obs_r(\pi_2)$, and $\pi^f_1, \pi^f_2 \in \mathsf{Prefs}(G^f)$ with $\pi_1 \in \gamma^f(\pi^f_1)$ and $\pi_2 \in \gamma^f(\pi^f_2)$, it holds that $obs_r(\pi^f_1) = obs_r(\pi^f_2)$.*

Intuitively, for $\pi \in \mathsf{Prefs}(G)$ the prefix $\pi^f \in \mathsf{Prefs}(G^f)$ with $\pi \in \gamma^f(\pi^f)$ is obtained by first mapping π to the sequence of states in G^f that correspond to the sequence of states in π, and then extending the resulting sequence with the intermediate states

necessary to obtain a prefix π^f in G^f, that is, the $Player_\exists$ and $Player_\forall$ states corresponding to reaching intermediate action points from ξ. This extension is uniquely determined by the definition of γ^f, which also entails Property 18 above.

Now, consider an obs_r-consistent strategy $f^f_\exists$ for $Player_\exists$ in G^f. We define a function $Await_{f^f_\exists} : \mathsf{Prefs}_\exists(G^f) \to \mathsf{Prefs}_\exists(G^f)$ as follows. Let $\pi^f \in \mathsf{Prefs}_\exists(G^f)$. Note that by construction of the game structures $\mathcal{G}(\mathcal{N}, l_{Err}, X_\mathsf{c})$ and $\mathcal{G}^f(\mathcal{N}, l_{Err}, X_\mathsf{c}, \xi)$, for each $s^f \in S^f_\exists$ it holds that $\mathsf{Enabled}(s^f, G^f) \neq \emptyset$. Thus, $f^f_\exists(\pi^f)$ is defined. If $f^f_\exists(\pi^f)(wait) = false$, we let $Await_{f^f_\exists}(\pi^f) = \pi^f$. If $f^f_\exists(\pi^f)(wait) = true$ and $f^f_\exists(\pi^f)(act) \in \Sigma_\mathsf{c}$, then we also let $Await_{f^f_\exists}(\pi^f) = \pi^f$. Otherwise, we distinguish the following two cases.

Case 1. There exists a prefix $\pi^f_1 \in \mathsf{Prefs}(f^f_\exists) \cap \mathsf{Prefs}_\exists(G^f)$ such that:

- there exists an index $i_0 \geq 0$ such that $\pi^f = \pi^f_1[0, i_0]$,
- for each $i_0 < i < |\pi^f_1|$, $\pi^f_1[i](oloc) = \pi^f_1[i_0](oloc)$ and $\pi^f_1[i](x) \geq \pi^f_1[i-1](x)$ for each $x \in X_{\mathsf{o}+\mathsf{c}}$ (i.e., there is no discrete change of the observation),
- for each $i > i_0$ with $\pi^f_1[i] \in S^f_\exists$ there are $x \in X_{\mathsf{o}+\mathsf{c}}$ and $c \in \xi(x)$ with $\pi^f_1[i](x) = c$,
- for each $i_0 < i < |\pi^f_1| - 1$ with $\pi^f_1[i] \in S^f_\exists$ it holds that $f^f_\exists(\pi^f_1[0, i])(act) = \flat$ and $f^f_\exists(\pi^f_1[0, i])(wait) = true$, and for π^f_1 it holds that $f^f_\exists(\pi^f_1)(act) \in \Sigma_\mathsf{c}$.

Intuitively, the prefix π^f_1 extends π^f and is such that no discrete observation change occurs between π^f and π^f_1, and all intermediate $Player_\exists$ states correspond to reaching intermediate action points, and are such that for the corresponding prefix, the strategy $f^f_\exists$ for $Player_\exists$ is to remain idle. Thus, all other prefixes with the same properties are obs_r-equivalent to π^f_1. In this case we let $Await_{f^f_\exists}(\pi^f) = \pi^f_1$.

Case 2. There exists no prefix in $\mathsf{Prefs}(f^f_\exists)$ with the properties required in *Case 1* above. In this case we let $Await_{f^f_\exists}(\pi^f) = \pi^f$.

The function $Await_{f^f_\exists}$ has the following property:

Property 19. *If $\pi^f_1, \pi^f_2 \in \mathsf{Prefs}(G^f)$ are such that $obs_r(\pi_1) = obs_r(\pi_2)$, $Await_{f^f_\exists}(\pi^f_1) = \widetilde{\pi}^f_1$, $Await_{f^f_\exists}(\pi^f_2) = \widetilde{\pi}^f_2$, $\widetilde{\pi}^f_1 \neq \pi^f_1$ and $\widetilde{\pi}^f_2 \neq \pi^f_2$, then it holds that $obs_r(\widetilde{\pi}^f_1) = obs_r(\widetilde{\pi}^f_2)$.*

We are now ready to prove the following proposition.

Proposition 7.2.1. *Given a partially observable plant $\mathcal{N} = (\mathcal{A}, \Sigma_\mathsf{c}, \Sigma_\mathsf{u}, X_\mathsf{o}, X_\mathsf{u}, =^\mathsf{L}_\mathsf{o})$, an error location $l_{Err} \in \mathsf{Loc}$ such that $[l_{Err}]_{=^\mathsf{L}_\mathsf{o}} = \{l_{Err}\}$ and a finite set of controllable*

clocks X_{c} *such that* $X_{\mathsf{c}} \cap X = \emptyset$*, it holds for every action-point function* $\xi : X_{\mathsf{o+c}} \to 2^{\mathbb{Q}_{>0}}$ *that if* $Player_\exists$ *has an* obs_r*-consistent winning strategy in the finite-choice await-time game* $\mathsf{Safety}(\mathcal{G}^f(\mathcal{N}, l_{Err}, X_{\mathsf{c}}, \xi), \varphi_{Err})$*, then* $Player_\exists$ *has an* obs_r*-consistent winning strategy in the await-time game* $\mathsf{Safety}(\mathcal{G}(\mathcal{N}, l_{Err}, X_{\mathsf{c}}), \varphi_{Err})$*.*

Proof. Let $f^f_\exists$ be an obs_r-consistent winning strategy for $Player_\exists$ in the finite-choice await-time game $\mathsf{Safety}(\mathcal{G}^f(\mathcal{N}, l_{Err}, X_{\mathsf{c}}, \xi), \varphi_{Err})$. We define $f_\exists : \mathsf{Prefs}_\exists(G(\mathcal{N}, l_{Err}, X_{\mathsf{c}})) \to \Sigma_\exists$ and show that it is an obs_r-consistent winning strategy for $Player_\exists$.

Let $\pi \in \mathsf{Prefs}(G)$ and $\pi^f \in \mathsf{Prefs}(G^f)$ be the unique prefix such that $\pi \in \gamma^f(\pi^f)$. We have to consider the following four cases.

Case 1. $f^f_\exists(\pi^f)(wait) = false$. Then we define $f_\exists(\pi) = \sigma$, where $\sigma(v) = f^f_\exists(\pi^f)(v)$ for each $v \in V^f$, and $\sigma(c_x) = 0$ for each $x \in X_{\mathsf{o+c}}$.

Case 2. $f^f_\exists(\pi^f)(wait) = true$ and $f^f_\exists(\pi^f)(act) \in \Sigma_{\mathsf{c}}$. Then we define $f_\exists(\pi)(\pi^f) = \sigma$, where $\sigma(v) = f^f_\exists(\pi^f)(v)$ for each $v \in V^f$, and for each $x \in X_{\mathsf{o+c}}$ we let

$$\sigma(c_x) = \begin{cases} c & \text{if } c \text{ is the smallest element of } \xi(x) \text{ such that } \mathsf{last}(\pi^f)(x) < c \\ 0 & \text{if no } c \in \xi(x) \text{ such that } \mathsf{last}(\pi^f)(x) < c \text{ exists.} \end{cases}$$

Case 3. $f^f_\exists(\pi^f)(wait) = true$ and $f^f_\exists(\pi^f)(act) = \flat$, and there exists a prefix $\widetilde{\pi}$ such that $obs_r(\widetilde{\pi}) = obs_r(\pi)$ and for the unique prefix $\widetilde{\pi}^f$ in G^f with $\widetilde{\pi} \in \gamma^f(\widetilde{\pi}^f)$ it holds that $f^f_\exists(\pi^f_1)(act) \in \Sigma_{\mathsf{c}}$, where $\pi^f_1 = Await_{f^f_\exists}(\widetilde{\pi}^f)$. In this case we let $f_\exists(\pi) = \sigma$, where $\sigma(v) = f^f_\exists(\pi^f_1)(v)$ for each $v \in V^f$. If $f^f_\exists(\pi^f_1)(wait) = false$ we let $\sigma(c_x) = 0$ for each $x \in X_{\mathsf{o+c}}$, and otherwise for each $x \in X_{\mathsf{o+c}}$ we let

$$\sigma(c_x) = \begin{cases} c & \text{if } c \text{ is the smallest element of } \xi(x) \text{ such that } \mathsf{last}(\pi^f_1)(x) < c \\ 0 & \text{if no } c \in \xi(x) \text{ such that } \mathsf{last}(\pi^f_1)(x) < c \text{ exists.} \end{cases}$$

Case 4. $f^f_\exists(\pi^f)(wait) = true$ and $f^f_\exists(\pi^f)(act) = \flat$, and for every prefix $\widetilde{\pi}$ such that $obs_r(\widetilde{\pi}) = obs_r(\pi)$ and for prefix $\widetilde{\pi}^f$ in G^f with $\widetilde{\pi} \in \gamma^f(\widetilde{\pi}^f)$ it holds that $f^f_\exists(\pi^f_1)(wait) = true$ and $f^f_\exists(\pi^f_1)(act) = \flat$, where $\pi^f_1 = Await_{f^f_\exists}(\widetilde{\pi}^f)$. In this case we let $f_\exists(\pi) = \sigma$, where $\sigma(v) = f^f_\exists(\pi^f_1)(v)$ for each $v \in V^f$ and $\sigma(c_x) = 0$ for each $x \in X_{\mathsf{o+c}}$.

By the definition of $f_\exists$ and the fact that $f^f_\exists$ is a strategy we have that $f_\exists$ is also a strategy for $Player_\exists$. Property 17, Property 18 and the definition of $Await_{f^f_\exists}$ together with the definition of $f_\exists$ and the fact that $f^f_\exists$ is obs_r-consistent imply that for prefixes $\pi_1, \pi_2 \in \mathsf{Prefs}(G)$ with $obs_r(\pi_1) = obs_r(\pi_2)$ it holds that $f_\exists(\pi_1) = f_\exists(\pi_2)$.

It is easy to see that from the definitions of $f_\exists$ and γ^f it follows that for each prefix $\pi \in \mathsf{Prefs}(f_\exists)$ for the prefix $\pi^f \in \mathsf{Prefs}(G^f)$, for which $\pi \in \gamma^f(\pi^f)$, it holds that $\pi^f \in \mathsf{Prefs}(f^f_\exists)$. Thus, since $f^f_\exists$ is winning for $Player_\exists$, $f_\exists$ is winning for $Player_\exists$. □

Await Points as Action Points We now establish a connection between await-time games and await-time games with fixed observations in the other direction. For a strategy for $Player_\exists$ in the await-time game structure $\mathcal{G}(\mathcal{N}, l_{Err}, X_\mathsf{c})$ that uses a finite set of await points, we can give an action-point function ξ that uses these await points as action points and a $Player_\exists$ strategy in the resulting game structure with fixed action points $\mathcal{G}^f(\mathcal{N}, l_{Err}, X_\mathsf{c}, \xi)$ whose set of outcomes is the same as the set of outcomes of the original strategy. Intuitively, the strategy in the abstraction takes controllable actions precisely at the respective action points. This is formalized in the following proposition.

Proposition 7.2.2. *Let $\mathcal{N} = (\mathcal{A}, \Sigma_\mathsf{c}, \Sigma_\mathsf{u}, X_\mathsf{o}, X_\mathsf{u}, =^{\mathsf{L}}_\mathsf{o})$ be a partially observable plant with timed automaton $\mathcal{A} = (\mathsf{Loc}, l_0, X, \Sigma, \mathsf{Inv}, \delta)$, $l_{Err} \in \mathsf{Loc}$ be an an error location such that $[l_{Err}]_{=^{\mathsf{L}}_\mathsf{o}} = \{l_{Err}\}$ and X_c be a finite set of controllable clocks such that $X_\mathsf{c} \cap X = \emptyset$.*

Let $f_\exists$ be an obs_r-consistent strategy for $Player_\exists$ in $G = G(\mathcal{N}, l_{Err}, X_\mathsf{c})$, for which there exists a finite set $C \subseteq \mathbb{Q}_{>0}$ such that for each $\pi \in \mathsf{Prefs}_\exists(G)$ it holds that $f_\exists(\pi)(c_x) = 0$ or $f_\exists(\pi)(c_x) \in C$ for each $x \in X_{\mathsf{o}+\mathsf{c}}$. Let ξ be an action-point function such that $\xi(x) \supseteq \{c \in \mathbb{Q}_{>0} \mid \exists \pi \in \mathsf{Prefs}_\exists(G).\ f_\exists(\pi)(c_x) = c\}$, for each $x \in X_{\mathsf{o}+\mathsf{c}}$. Then, there exists an obs_r-consistent $Player_\exists$ strategy $f^f_\exists$ in $G^f = G^f(\mathcal{N}, l_{Err}, X_\mathsf{c}, \xi)$ such that for each play $\pi^f \in \mathsf{Outcome}(f^f_\exists)$ there exists a play $\pi \in \mathsf{Outcome}(f_\exists)$ such that for each $0 \leq i < |\pi^f|$ there exists $0 \leq j < |\pi|$ such that $\pi[j](loc) = \pi^f[i](loc)$.

Proof. In order to define a strategy $f^f_\exists$ for $Player_\exists$ in G^f, we first give a function $AwaitPref : \mathsf{Prefs}(G^f) \to \mathsf{Prefs}(G)$ that maps each prefix in G^f to a prefix in G. The function is defined recursively as follows. For $\pi^f \in \mathsf{Prefs}(G^f)$ we let:

$$AwaitPref(\pi^f) = \begin{cases} s_0 & \text{if } \pi^f = s^f_0 \in I^f \text{ and where } I = \{s_0\}, \\ \pi_1 \cdot s_1 \cdot s_2 & \text{if } \pi^f = \pi^f_1 \cdot s^f_1 \cdot s^f_2, AwaitPref(\pi^f_1 \cdot s^f_1) = \pi_1 \cdot s_1, \\ & s_1(t) = s^f_1(t), s_2 = Match(\pi_1 \cdot s_1, s^f_2), (s_1, s_2) \in T, \\ AwaitPref(\pi^f_1 s^f_1) & \text{otherwise,} \end{cases}$$

where the function $Match$ maps a state $s^f \in S^f$ to a $s' \in S$ w.r.t. a given prefix $\pi \cdot s \in \mathsf{Prefs}(G)$. For $\pi \cdot s \in \mathsf{Prefs}(G)$ and $s_f \in S^f$ we have $Match(\pi \cdot s, s^f) = s'$ iff

- $s'(v) = s^f(v)$, for each $v \in V^f \setminus \{act, wait, et\}$,

- $$s'(act) = \begin{cases} s^f(act) & \text{if } s^f(act) \in \Sigma_{\mathsf{c}}, \\ s(act) & \text{if } s^f(act) = \flat \text{ and } s(t) = \forall, \\ \sigma(act) & \text{if } s^f(act) = \flat, s^f(wait) = true, s(t) = \exists \text{ and } f_\exists(\pi \cdot s) = \sigma, \\ s^f(act) & \text{otherwise.} \end{cases}$$

- $$s'(wait) = \begin{cases} s^f(wait) & \text{if } s^f(wait) = true, \\ s(wait) & \text{if } s^f(wait) = false \text{ and } s(t) = \forall, \\ s^f(wait) & \text{otherwise.} \end{cases}$$

- for each $x \in X_{\mathsf{o}+\mathsf{c}}$,

$$s'(c_x) = \begin{cases} s(c_x) & \text{if } s(t) = \forall, \\ \sigma(c_x) & \text{if } s^f(act) = \flat, s^f(wait) = true, s(t) = \exists \text{ and } f_\exists(\pi \cdot s) = \sigma, \\ s(x) + 1 & \text{if } s^f(act) \in \Sigma_{\mathsf{c}}, s^f(wait) = true \text{ and } s(t) = \exists, \\ 0 & \text{otherwise.} \end{cases}$$

- $$s'(et) = \begin{cases} s^f(et) & \text{if } s^f(et) = true, \\ true & \text{if } s^f(et) = false, s(act) \in \Sigma_{\mathsf{c}} \text{ and} \\ & \text{for all } x \in X_{\mathsf{o}+\mathsf{c}} : s(x) \geq s(c_x) \text{ or } s^f(x) \neq s(c_x), \\ s^f(et) & \text{otherwise.} \end{cases}$$

Now we are ready to define the function $f^f_\exists : \mathsf{Prefs}_\exists(G^f) \to \Sigma^f_\exists$. For $\pi^f \in \mathsf{Prefs}_\exists(G^f)$, let π be the longest prefix of $AwaitPref(\pi^f)$ with $\pi \in \mathsf{Prefs}_\exists(G)$ and

$$f^f_\exists(\pi^f) = \begin{cases} \sigma(V^f_\exists) & \text{if } f_\exists(\pi) = \sigma \text{ and } \sigma(act) = \flat \text{ or } \sigma(wait) = false, \\ \sigma^f & \text{if } f_\exists(\pi) = \sigma,\ \sigma(act) \in \Sigma_{\mathsf{c}},\ \sigma(wait) = true \text{ and} \\ & \text{for all } x \in X_{\mathsf{o}+\mathsf{c}}, \mathsf{last}(\pi)(x) \neq \sigma(c_x),\ \text{and} \\ & \sigma^f(act) = \flat \text{ and } \sigma^f(v) = \sigma(v) \text{ for each } v \in V^f \setminus \{act\}, \\ \sigma^f & \text{if } f_\exists(\pi) = \sigma,\ \sigma(act) \in \Sigma_{\mathsf{c}},\ \sigma(wait) = true \text{ and} \\ & \text{for some } x \in X_{\mathsf{o}+\mathsf{c}}, \mathsf{last}(\pi)(x) = \sigma(c_x),\ \text{and} \\ & \sigma^f(wait) = false \text{ and } \sigma^f(v) = \sigma(v) \text{ for each } v \in V^f \setminus \{wait\}. \end{cases}$$

Since $f_\exists$ is a strategy for $Player_\exists$, the function $f_\exists^f$ is one as well. As for prefixes $\pi_1^f, \pi_2^f \in \mathsf{Prefs}(G^f)$ with $obs_r(\pi_1^f) = obs_r(\pi_2^f)$ it holds that $obs_r(AwaitPref(\pi_1)) = obs_r(AwaitPref(\pi_2))$, and the strategy $f_\exists$ is obs_r-consistent, we have that $f_\exists^f$ is also obs_r-consistent. It remains to show that for each $\pi^f \in \mathsf{Outcome}(f_\exists^f)$ there exists $\pi \in \mathsf{Outcome}(f_\exists)$ such that or each $0 \leq i < |\pi^f|$ there exists $0 \leq j < |\pi|$ such that $\pi[j](loc) = \pi^f[i](loc)$. We will show that for each $\pi^f \in \mathsf{Outcome}(f_\exists^f)$ there exists a $\pi \in \mathsf{Outcome}(f_\exists)$ such that for each $0 \leq i < |\pi^f|$, $AwaitPref(\pi^f[0,i])$ is a prefix of π, which entails the above property. First, note that $f_\exists^f$ has the following properties.

Property 20. *If $\pi^f \in \mathsf{Prefs}(f_\exists^f)$, $\pi = AwaitPref(\pi^f)$, $s^f = \mathsf{last}(\pi^f)$, $s = \mathsf{last}(\pi)$, then:*

(1) if $s(act) \neq s^f(act)$, then $s(act) \in \Sigma_\mathsf{c}$ and $s^f(act) = \flat$,

(2) if $s(wait) \neq s^f(wait)$, then $s(wait) = true$ and $s^f(wait) = false$,

(3) if $s(et) \neq s^f(et)$, then $s(et) = true$ and $s^f(et) = false$,

(4) if $s(t) \neq s^f(t)$, then $s(t) = \forall$ and $s^f(t) = \exists$,

(5) $s(v) = s^f(v)$ for $v \in V^f \setminus \{t, act, wait, et\}$.

Property 20 follows from the definition of the function *AwaitPrefs*, by induction on the length of the prefix π^f. Also by induction we can show the following.

Property 21. *If $\pi^f \in \mathsf{Outcome}(f_\exists^f)$ and $i_0 < i_1 < i_2 < \ldots$ are all the indices such that $i_j = 0$ or $AwaitPref(\pi^f[0, i_j - 1]) \neq AwaitPref(\pi^f[0, i_j])$, then:*

- *if π^f is infinite, then the sequence $i_0 < i_1 < i_2 < \ldots$ is also infinite, and*
- *if $s_j = \mathsf{last}(AwaitPref(\pi^f[0, i_j]))$, then $\pi = s_0 \cdot s_1 \cdot s_2 \ldots \in \mathsf{Outcome}(f_\exists)$.*

Thus, according to Property 20, for each $\pi^f \in \mathsf{Outcome}(f_\exists^f)$ the prefix defined using Proposition 21 satisfies the desired requirements. □

7.2.2 Predicate Abstraction

Abstraction predicates. Let $\mathcal{P} \subseteq \mathcal{AP}[V^f]$ be a finite set of predicates. In this part of the thesis we consider sets of predicates that meet the following requirements:

(i) $Preds(\varphi_{Init}) \subseteq \mathcal{P}$, $Preds(\varphi_{Err}) \subseteq \mathcal{P}$,

(ii) $(x \leq c) \in \mathcal{P}$ and $(x \geq c) \in \mathcal{P}$ for each $x \in X_{\mathsf{o+c}}$ and $c \in \xi(x)$,

(iii) $\mathcal{P}$ is precise w.r.t. the set of variables $\{v \in \mathit{Obs}(V^f) \mid |\mathsf{Dom}(v)| < \infty\}$,

(iv) $\mathcal{P}$ is precise w.r.t. the set $\Sigma^f_\exists$ of $\mathit{Player}_\exists$ actions in $\mathcal{G}^f(\mathcal{N}, l_{Err}, X_\mathsf{c}, \xi)$,

(v) for all $p \in \mathcal{P}$, either $\mathit{Vars}(p) \subseteq V^f_\exists$ or $\mathit{Vars}(p) \cap V^f_\exists = \emptyset$.

Condition (i) ensures that the predicates in $\mathcal{P}$ suffice to express φ_{Init} and φ_{Err}. Intuitively, (ii) ensures that the abstraction is precise w.r.t. the action-point function ξ. Since our goal is to synthesize observation predicates over the clock variables $X_{\mathsf{o}+\mathsf{c}}$, we require in (iii) for simplicity that $\mathcal{P}$ is precise w.r.t. all observable variables with finite domains. Finally, (iv) requires that $\mathcal{P}$ is precise w.r.t. the actions of $\mathit{Player}_\exists$ in G^f. Requirement (v), which ensures that no predicate refers to variables from $V_\exists$ and from $V_\forall$, can be satisfied by substituting variables in $V^f_\exists$ with their possible values.

The observable predicates in $\mathcal{P}$ (i.e., those referring only to variables in $\mathit{Obs}(V)$) are exactly the observation predicates the controller can track in the current abstraction.

Game structure transformation. The abstraction defined in Section 5.1.2 is sound w.r.t. obs_s-consistent strategies for $\mathit{Player}_\exists$ in the concrete game structure. In order to obtain an abstract game that soundly abstracts $(\mathsf{Safety}(G^f, Err^f), obs_r)$, we will apply predicate abstraction to a modified game structure G^f_{skip} that will have the property that $\mathit{Player}_\exists$ has an obs_r-consistent winning strategy in $\mathsf{Safety}(G^f, Err^f)$ iff he has an obs_s-consistent winning strategy in $\mathsf{Safety}(G^f_{skip}, Err^f)$. We will check concretizability of abstract counterexample trees in the game structures G^f_{skip} (and G_{skip}) respectively.

The symbolic game structure $\mathcal{G}_{skip}$ for $\mathcal{G} = (V_\exists, V_\forall, V^o_\forall, t, \varphi_{Init}, \mathcal{T}_\exists, \mathcal{T}_\forall)$ is the tuple $\mathcal{G}_{skip} = (V_\exists, V_\forall, V^o_\forall, t, \varphi_{Init}, \mathcal{T}_\exists, \mathcal{T}_{\forall skip})$, where $\mathcal{T}_{\forall skip} = \mathcal{T}_\forall \vee (t = \forall \wedge \mathsf{preserve}(V))$. We denote with G_{skip} the corresponding explicit game structure with skip transition. Similarly, the symbolic game structure $\mathcal{G}^f_{skip}$ for $\mathcal{G}^f = (V^f_\exists, V_\forall, V^o_\forall, t, \varphi^f_{Init}, \mathcal{T}^f_\exists, \mathcal{T}^f_\forall)$ is the tuple $\mathcal{G}^f_{skip} = (V^f_\exists, V_\forall, V^o_\forall, t, \varphi^f_{Init}, \mathcal{T}^f_\exists, \mathcal{T}^f_{\forall skip})$, where $\mathcal{T}^f_{\forall skip} = \mathcal{T}^f_\forall \vee (t = \forall \wedge \mathsf{preserve}(V^f))$. We denote with G^f_{skip} the corresponding explicit game structure with skip transition.

Proposition 7.2.3. *If we are given game structures* $\mathcal{G} = (V_\exists, V_\forall, V^o_\forall, t, \varphi_{Init}, \mathcal{T}_\exists, \mathcal{T}_\forall)$ *and* $\mathcal{G}_{skip} = (V_\exists, V_\forall, V^o_\forall, t, \varphi_{Init}, \mathcal{T}_\exists, \mathcal{T}_{\forall skip})$, *where* $\mathcal{T}_{\forall skip} = \mathcal{T}_\forall \vee (t = \forall \wedge \mathsf{preserve}(V))$, *and* $Err \subseteq S$, *then if* $Player_\exists$ *has an* obs_r*-consistent winning strategy in the game* $\mathsf{Safety}(G, Err)$, *then he has an* obs_s*-consistent winning strategy in* $\mathsf{Safety}(G_{skip}, Err)$.

Proof. To see that for each obs_r-consistent winning strategy $f_\exists$ for $Player_\exists$ in the game $\mathsf{Safety}(G, Err)$, we can define a obs_s-consistent winning strategy $f'_\exists$ for $Player_\exists$ in $\mathsf{Safety}(G_{skip}, Err)$, consider the function $delete_{skip} : \mathsf{Prefs}_\exists(G_{skip}) \to \mathsf{Prefs}(G)$ that maps a prefix in G_{skip} to the prefix in G obtained by deleting all *skip* steps. Then, we define $f'_\exists(\pi') = f_\exists(delete_{skip}(\pi'))$. Since for every $\pi'_1, \pi'_2 \in \mathsf{Prefs}_\exists(G_{skip})$ it holds that $obs_s(\pi'_1) = obs_s(\pi'_2)$ implies $obs_r(delete_{skip}(\pi'_1)) = obs_r(delete_{skip}(\pi'_2))$, $f'_\exists$ is obs_s-consistent. Since π' and $delete_{skip}(\pi')$ visit the same states, $f'_\exists$ is winning. □

Proposition 7.2.4. *If we are given game structures* $\mathcal{G} = (V^f_\exists, V_\forall, V^o_\forall, t, \varphi_{Init}, \mathcal{T}_\exists, \mathcal{T}_\forall)$ *and* $\mathcal{G}_{skip} = (V^f_\exists, V_\forall, V^o_\forall, t, \varphi_{Init}, \mathcal{T}_\exists, \mathcal{T}_{\forall skip})$, *where* $\mathcal{T}_{\forall skip} = \mathcal{T}_\forall \vee (t = \forall \wedge \mathsf{preserve}(V))$, *and* $Err \subseteq S$, *then if* $Player_\exists$ *has an* obs_s*-consistent winning strategy* $f_\exists$ *in the game* $\mathsf{Safety}(G_{skip}, Err)$ *such that the set* $\{\sigma \in \Sigma_\exists \mid \exists \pi \in \mathsf{Prefs}_\exists(G).\ f_\exists(\pi) = \sigma\}$ *is finite, then he has an* obs_r*-consistent winning strategy in* $\mathsf{Safety}(G, Err)$.

Proof. Let us suppose that $Player_\exists$ has an obs_s-consistent winning strategy in the game $\mathsf{Safety}(G_{skip}, Err)$. Consider the game structure $G^k_{skip} = (S^k_\exists, S^k_\forall, I^k, \Sigma^k_\exists, T^k_\exists, T^k_\forall)$ defined by the knowledge-based subset construction from Section 2.2.3 for G_{skip}. According to our hypothesis, $Player_\exists$ has a memoryless winning strategy in the perfect information game $\mathsf{Safety}(G^k_{skip}, Err^k)$. Let $f^k_\exists$ be such a strategy, and let $f_\exists$ be the corresponding obs_s-consistent winning strategy for $Player_\exists$ in $\mathsf{Safety}(G_{skip}, Err)$.

For $\pi, \pi' \in \mathsf{Prefs}(G_{skip})$, we write $\pi \leq \pi'$ if $obs_r(\pi_1) = obs_r(\pi_2)$ and there exists a function $idx : [0, |\pi| - 1] \to [0, |\pi'| - 1]$ such that $idx(0) = 0$, $idx(|\pi| - 1) = |\pi'| - 1$, $\mathsf{id}(i_1) < \mathsf{id}(i_2)$ for each $0 \leq i_1 < i_2 < |\pi|$, and $\pi[i] = \pi'[idx(i)]$ for each $0 \leq i < |\pi|$.

For any prefixes $\pi_1, \pi_2 \in \mathsf{Prefs}(G_{skip})$ with $obs_r(\pi_1) = obs_r(\pi_2)$ there exist $\pi'_1, \pi'_2 \in \mathsf{Prefs}(G_{skip})$ such that $obs_s(\pi'_1) = obs_s(\pi'_2)$, $\pi_1 \leq \pi'_1$ and $\pi_2 \leq \pi'_2$.

Let $\Sigma_f = \{\sigma \in \Sigma_\exists \mid \exists \pi \in \mathsf{Prefs}_\exists(G).\ f_\exists(\pi) = \sigma\}$. By our hypothesis, Σ_f is finite.

We will show that there exists a function $f'_\exists : \mathsf{Prefs}_\exists(G) \to \Sigma_f$ such that for each $\Pi \subseteq \mathsf{Prefs}_\exists(G)$ that satisfies the following two properties:

- if $\pi_1, \pi_2 \in \Pi$ then $obs_r(\pi_1) = obs_r(\pi_2)$,
- if $\pi_1 \in \Pi$ and $obs_r(\pi_1) = obs_r(\pi_2)$, then $\pi_2 \in \Pi$,

it holds that the following conditions are satisfied:

(1) there exists $\sigma \in \Sigma_f$ such that for each $\pi \in \Pi \cap \mathsf{Prefs}_\exists(G)$ it holds that $f'_\exists(\pi) = \sigma$ and $\sigma \in \mathsf{Enabled}(\mathsf{last}(\pi))$,

(2) if for each $\pi \in \Pi$ there exists a $\pi' \in \mathsf{Prefs}(f_\exists)$ such that $\pi \leq \pi'$, then for each $\pi \in \Pi \cap \mathsf{Prefs}_\exists(G)$ there exists $\pi' \in \mathsf{Prefs}(f_\exists)$ such that $\pi \leq \pi'$ and $f_\exists(\pi') = f'_\exists(\pi)$.

Assume for contradiction that a function $f'_\exists$ with the above properties does not exist. Thus, there exists $\Pi \subseteq \mathsf{Prefs}_\exists(G)$ that satisfies the preconditions above, such that for each $\pi \in \Pi$ there exists a $\pi' \in \mathsf{Prefs}(f_\exists)$ such that $\pi \leq \pi'$, but for each σ there exists a $\pi_\sigma \in \Pi$, such that for every $\pi' \in \mathsf{Prefs}(f_\exists)$ with $\pi \leq \pi'$ it holds that $f_\exists(\pi') \neq \sigma$.

Since $obs_r(\pi_{\sigma_1}) = obs_r(\pi_{\sigma_1})$ for any $\sigma_1, \sigma_2 \in \Sigma_f$, there exist $\pi''_\sigma \in \mathsf{Prefs}_\exists(G)$ for each $\sigma \in \Sigma_f$ such that $\pi_\sigma \leq \pi''_\sigma$ for each $\sigma \in \Sigma_f$ and $obs_s(\pi_{\sigma_1}) = obs_s(\pi_{\sigma_2})$ for each $\sigma_1, \sigma_2 \in \Sigma_f$. According to our assumption, for each $\sigma \in \Sigma_f$, there exists a $\pi'_\sigma \in \mathsf{Prefs}(f_\exists)$ such that $\pi''_\sigma \leq \pi'_\sigma$. Since the strategy $f_\exists$ is obs_s-consistent, we can choose π'_σ for $\sigma \in \Sigma_f$ such that $obs_s(\pi'_{\sigma_1}) = obs_s(\pi'_{\sigma_2})$ for each $\sigma_1, \sigma_2 \in \Sigma_f$. Furthermore, there exists $\sigma' \in \Sigma_f$ such that $f_\exists(\pi'_\sigma) = \sigma'$ for each $\sigma \in \Sigma_f$. Therefore, $f_\exists(\pi'_{\sigma'}) = \sigma'$, which contradicts our assumption, i.e., the choice of $\pi_{\sigma'}$, which concludes the proof by contradiction.

A function $f'_\exists$ that has the property stated above is clearly an obs_r-consistent strategy for $Player_\exists$ in the game structure G. What remains to show is that the the strategy $f'_\exists$ is winning for $Player_\exists$ in $\mathsf{Safety}(G, Err)$. To this end, we can show by induction on the number of $Player_\exists$ states on a prefix that for each prefix $\pi \in \mathsf{Prefs}(f'_\exists)$ there exists a prefix $\pi' \in \mathsf{Prefs}(f_\exists)$ such that $\pi \leq \pi'$. Then, by the definition of the relation $\leq$ on the set $\mathsf{Prefs}(G^f_{skip})$ of prefixes, and since $f_\exists$ is winning for $Player_\exists$ in $\mathsf{Safety}(G_{skip}, Err)$, we have that $f'_\exists$ is winning for $Player_\exists$ in $\mathsf{Safety}(G, Err)$.

□

Since the set of $Player_\exists$ actions $\Sigma^f_\exists$ in the game structure G^f_{skip} is finite, each strategy for $Player_\exists$ has finite co-domain. Thus, we obtain the following corollary.

Corollary 7.2.1. *If we are given the game structures* $\mathcal{G}^f = (V^f_\exists, V_\forall, V^o_\forall, t, \varphi^f_{Init}, \mathcal{T}^f_\exists, \mathcal{T}^f_\forall)$ *and* $\mathcal{G}^f_{skip} = (V^f_\exists, V_\forall, V^o_\forall, t, \varphi^f_{Init}, \mathcal{T}^f_\exists, \mathcal{T}^f_{\forall\, skip})$, *where* $\mathcal{T}^f_{\forall\, skip} = \mathcal{T}^f_\forall \vee (t = \forall \wedge \mathsf{preserve}(V^f))$, *and* $Err^f \subseteq S^f$, *then* $Player_\exists$ *has an* obs_r*-consistent winning strategy in the game* $\mathsf{Safety}(G^f, Err^f)$ *iff he has an* obs_s*-consistent winning strategy in* $\mathsf{Safety}(G^f_{skip}, Err^f)$.

Following the same reasoning as in the proof of Proposition 7.2.2 we can prove an analogous property for the game structure $G_{skip}(\mathcal{N}, l_{Err}, X_\mathsf{c})$.

Proposition 7.2.5. *Let* $\mathcal{N} = (\mathcal{A}, \Sigma_\mathsf{c}, \Sigma_\mathsf{u}, X_\mathsf{o}, X_\mathsf{u}, =^\mathsf{L}_\mathsf{o})$ *be a partially observable plant with timed automaton* $\mathcal{A} = (\mathsf{Loc}, l_0, X, \Sigma, \mathsf{Inv}, \delta)$, $l_{Err} \in \mathsf{Loc}$ *be an an error location such that* $[l_{Err}]_{=^\mathsf{L}_\mathsf{o}} = \{l_{Err}\}$ *and* X_c *be a finite set of controllable clocks such that* $X_\mathsf{c} \cap X = \emptyset$.

Let $f_\exists$ *be an* obs_s*-consistent strategy for* $Player_\exists$ *in* $G_{skip} = G_{skip}(\mathcal{N}, l_{Err}, X_\mathsf{c})$, *for which there exists a finite set* $C \subseteq \mathbb{Q}_{>0}$ *such that for each* $\pi \in \mathsf{Prefs}_\exists(G_{skip})$ *it holds that* $f_\exists(\pi)(c_x) = 0$ *or* $f_\exists(\pi)(c_x) \in C$ *for each* $x \in X_{\mathsf{o}+\mathsf{c}}$. *Let* ξ *be an action-point function such that* $\xi(x) \supseteq \{c \in \mathbb{Q}_{>0} \mid \exists \pi \in \mathsf{Prefs}_\exists(G_{skip}).\ f_\exists(\pi)(c_x) = c\}$, *for each* $x \in X_{\mathsf{o}+\mathsf{c}}$.

Then, there exists an obs_s-consistent $Player_\exists$ strategy $f^f_\exists$ in $G^f_{skip} = G^f_{skip}(\mathcal{N}, l_{Err}, X_\mathsf{c}, \xi)$ such that for each $\pi^f \in \mathsf{Outcome}(f^f_\exists)$ there exists a play $\pi \in \mathsf{Outcome}(f_\exists)$ such that for each $0 \leq i < |\pi^f|$ there exists $0 \leq j < |\pi|$ such that $\pi[j](loc) = \pi^f[i](loc)$.

Soundness of the abstraction. Given a finite set of predicates $\mathcal{P}$, we construct the finite-state game structure $\mathsf{Abstract}(\mathcal{G}^f_{skip}(\mathcal{N}, l_{Err}, X_\mathsf{c}, \xi), \mathcal{P})$ with perfect information, which abstracts $G^f_{skip}(\mathcal{N}, l_{Err}, X_\mathsf{c}, \xi)$ w.r.t. $\mathcal{P}$. Theorem 5.1.1 directly implies that the game structure $\mathsf{Abstract}(\mathcal{G}^f_{skip}(\mathcal{N}, l_{Err}, X_\mathsf{c}, \xi), \mathcal{P})$ soundly abstracts $\mathcal{G}^f_{skip}(\mathcal{N}, l_{Err}, X_\mathsf{c}, \xi)$.

Proposition 7.2.6. *Given a partially observable plant $\mathcal{N} = (\mathcal{A}, \Sigma_\mathsf{c}, \Sigma_\mathsf{u}, X_\mathsf{o}, X_\mathsf{u}, =^\mathsf{L}_\mathsf{o})$, an error location $l_{Err} \in \mathsf{Loc}$ such that $[l_{Err}]_{=^\mathsf{L}_\mathsf{o}} = \{l_{Err}\}$ and a finite set of controllable clocks X_c such that $X_\mathsf{c} \cap X = \emptyset$, and an action-point function $\xi : X_{\mathsf{o}+\mathsf{c}} \to 2^{\mathbb{Q}_{>0}}$, it holds for every finite set of predicates $\mathcal{P} \subseteq \mathcal{AP}(V^f)$ that is precise w.r.t. t that $(\mathsf{Safety}(G^\#, Err^\#), obs^\#_s)$ soundly abstracts $(\mathsf{Safety}(G^f_{skip}(\mathcal{N}, l_{Err}, X_\mathsf{c}, \xi), Err), obs_s)$, where $G^\# = \mathsf{Abstract}(\mathcal{G}^f_{skip}(\mathcal{N}, l_{Err}, X_\mathsf{c}, \xi), \mathcal{P})$, $Err^\# = \{s^\# \in S^\# \mid [\![s^\#]\!] \cap [\![\varphi_{Err}]\!] \neq \emptyset\}$.*

7.2.3 Existence of Finite-State Strategies

Since $G^\#$ is a finite-state game structure with perfect information, if $Player_\exists$ has a winning strategy in $\mathsf{Safety}(G^\#, Err^\#)$, then he also has a memoryless one. According to the results in Section 5.1.4, for each such winning strategy for $Player_\exists$ in $\mathsf{Safety}(G^\#, Err^\#)$ there exists a finite semi-symbolic strategy automaton that represents an obs_s-consistent winning strategy for $Player_\exists$ in $\mathsf{Safety}(G^f_{skip}, Err^f)$, and this automaton can be effectively constructed. Furthermore, we can use the idea of the proof of Proposition 7.2.4 to directly construct a finite semi-symbolic strategy automaton $\mathcal{M}^f$, which represents an obs_r-consistent winning strategy for $Player_\exists$ in $\mathsf{Safety}(\mathcal{G}^f, Err^f)$. Now, using the idea of the proof of Proposition 7.2.1 we can construct a finite semi-symbolic strategy automaton $\mathcal{M}$ from $\mathcal{M}^f$, which represents an obs_r-consistent winning strategy for $Player_\exists$ in $\mathsf{Safety}(G, Err)$.

7.3 Observation Refinement

In this section we describe a nested CEGAR loop for the automatic discovery of observation predicates, which is based on the two-step fixed observation abstraction presented in Section 7.2. If the loop terminates, it either yields a finite set of observation predicates that suffice to construct a finite-state X_c-control strategy for the given partially

observable plant and set X_{c} of controllable clocks, that avoids the given error location, or determines that an X_{c}-control strategy to avoid the error location does not exist.

7.3.1 CEGAR Loop

The CEGAR loop for timed control with partial observability is given as Algorithm 10. Its input consists of a partially observable plant $\mathcal{N} = (\mathcal{A}, \Sigma_{\mathsf{c}}, \Sigma_{\mathsf{u}}, X_{\mathsf{o}}, X_{\mathsf{u}}, =^{\mathsf{L}}_{\mathsf{o}})$ with timed automaton $\mathcal{A} = (\mathsf{Loc}, l_0, X, \Sigma, \mathsf{Inv}, \delta)$, an error location $l_{Err} \in \mathsf{Loc}$ with $[l_{Err}]_{=^{\mathsf{L}}_{\mathsf{o}}} = \{l_{Err}\}$, and a finite set of controllable clocks X_{c} with $X_{\mathsf{c}} \cap X = \emptyset$. If the procedure terminates it returns either a finite-state winning strategy for $Player_{\exists}$ (as a memoryless abstract strategy) or an abstract counterexample tree concretizable in $\mathcal{G}(\mathcal{N}, l_{Err}, X_{\mathsf{c}})$.

The initial action-point function $\xi = \{(x, \emptyset) \mid x \in X_{\mathsf{o+c}}\}$ maps each clock in $X_{\mathsf{o+c}}$ to the empty set, that is, initially there are no action points. The procedure InitialPredicates returns a set of initial abstraction predicates that satisfy the conditions listed in Section 7.2.2. We let InitialPredicates$(\mathcal{G}, \varphi_{Err}, \xi) = \mathit{Preds}(\varphi_{\mathit{Init}}) \cup \mathit{Preds}(\varphi_{Err}) \cup \{t = \exists, t = \forall\} \cup \{x = d \mid x \in \mathit{Obs}(V^f) \wedge |\mathsf{Dom}(x)| < \infty \wedge d \in \mathsf{Dom}(x)\}$.

The procedure FixActionPoints constructs the abstraction $\mathcal{G}^f(\mathcal{N}, l_{Err}, X_{\mathsf{c}}, \xi)$ of $\mathcal{G}(\mathcal{N}, l_{Err}, X_{\mathsf{c}})$ w.r.t. the current action-point function ξ. Then, AbstractGame constructs the abstraction $\mathsf{Abstract}(\mathcal{G}^f_{\mathit{skip}}, \mathcal{P})$ of $\mathcal{G}^f_{\mathit{skip}}$ w.r.t. $\mathcal{P}$ and the set of abstract error states $Err^{\#} = \{s^{\#} \in S^{\#} \mid [\![s^{\#}]\!] \cap [\![\varphi_{Err}]\!] \neq \emptyset\}$. As in Algorithm 7, the game $\mathsf{Safety}(G^{\#}, Err^{\#})$ is solved by the procedure SolveGame. If this game is not won by $Player_{\exists}$, the counterexample tree $C_t^{\#} =$ CounterexampleTree$(\mathit{strategy})$ is analyzed for concretizability in two steps, corresponding to the two levels of abstraction.

The procedure Concretizable applied to $C_t^{\#}$ and $\mathcal{G}^f_{\mathit{skip}}$ checks as in Section 5.4 if $C_t^{\#}$ is concretizable in the game $\mathcal{G}^f_{\mathit{skip}}$. Thus, the inner loop is the CEGAR loop of Algorithm 7. The difference is, that an abstract counterexample tree that is concretizable in $\mathcal{G}^f_{\mathit{skip}}$ is passed to the concretizability check of the outer refinement loop.

In the outer loop, the procedure Concretizable applied to an *extended* tree $C_t^{\#}$ and the await-time game structure $\mathcal{G}_{\mathit{skip}}(\mathcal{N}, l_{Err}, X_{\mathsf{c}})$ constructs the *quantified tree formula* $\mathsf{QTF}(C_t^{\#})$ for the given abstract counterexample tree, as described below, and checks if it is satisfiable. If this formula is satisfiable, the abstract counterexample tree, which is concretizable in the concrete await-time game, is returned. Otherwise, the procedure RefineActionPoints described later in this section is called to refine the action-point function ξ and compute a corresponding set of refinement predicates $\mathcal{R}$.

Algorithm: SynthesizeObservationPredicates

Input: partially observable plant $\mathcal{N} = (\mathcal{A}, \Sigma_{\mathsf{c}}, \Sigma_{\mathsf{u}}, X_{\mathsf{o}}, X_{\mathsf{u}}, =^{\mathsf{L}}_{\mathsf{o}})$
with timed automaton $\mathcal{A} = (\mathsf{Loc}, l_0, X, \Sigma, \mathsf{Inv}, \delta)$,
error location $l_{Err} \in \mathsf{Loc}$ with $[l_{Err}]_{=^{\mathsf{L}}_{\mathsf{o}}} = \{l_{Err}\}$,
finite set of controllable clocks X_{c} with $X_{\mathsf{c}} \cap X = \emptyset$

Output: (*winner*, *abstract strategy*) or (*winner*, *abstract counterexample tree*)

$\xi = \{(x, \emptyset) \mid x \in X_{\mathsf{o}+\mathsf{c}}\}$;
$\mathcal{P} :=$ InitialPredicates$(\mathcal{G}^f_{skip}, \varphi_{Err}, \xi)$;

while *winner* $= Player_\forall$ **do**
  $\mathcal{G}^f :=$ FixActionPoints$(\mathcal{G}(\mathcal{N}, l_{Err}, X_{\mathsf{c}}), \xi)$;
  $(G^\#, Err^\#) :=$ AbstractGame$(\mathcal{G}^f_{skip}, \varphi_{Err}, \mathcal{P})$;
  $(winner, strategy) :=$ SolveGame$(G^\#, Err^\#)$;

  while *winner* $= Player_\forall$ **do**
    $C^\#_t :=$ CounterexampleTree$(strategy)$;
    if Concretizable $(C^\#_t, \mathcal{G}^f_{skip})$ **then break;**
    $\mathcal{R} := \emptyset$;
    $\tau :=$ SpuriousTrace$(Formulas(C^\#_t), Traces(C^\#_t))$;
    if $\tau \neq \bot$ **then** `/* ` $\tau \in Traces(C^\#_t)$ `with` $\varphi_{trace}(\tau)$ `UNSAT */`
      $\mathcal{R} :=$ RefineTransitionRelations$(\tau, C^\#_t)$;
    else `/* ` $\varphi_{trace}(\tau)$ `SAT for all` $\tau \in Traces(C^\#_t)$ `*/`
      $\mathcal{R} :=$ RefineObservations$(Formulas(C^\#_t), Traces(C^\#_t))$;
      $Trees :=$ RefineTree$(C^\#_t, \mathcal{P}, \mathcal{R})$;
      forall $C'_t \in Trees$ **do**
        $\tau' :=$ SpuriousTrace$(Formulas(C'_t), Traces(C'_t))$;
        $\mathcal{R} := \mathcal{R} \cup$ RefineTransitionRelations(τ', C'_t);
    $\mathcal{P} := \mathcal{P} \cup \mathcal{R}$;
    $G^\# =$ RefineGame$(G^\#, \mathcal{G}, \mathcal{P})$;
    $(winner, strategy) =$ SolveGame$(G^\#, Err^\#)$;

  if *winner* $= Player_\exists$ **then return** $(winner, strategy)$;
  $C^\#_t := Extend(C^\#_t)$;
  if Concretizable $(C^\#_t, \mathcal{G}_{skip}(\mathcal{N}, l_{Err}, X_{\mathsf{c}}))$ **then return** $(winner, C^\#_t)$;
  $(\xi, \mathcal{R}) :=$ RefineActionPoints$(C^\#_t)$;
  $\mathcal{P} := \mathcal{P} \cup \mathcal{R}$;

return $(winner, strategy)$; `/* winning strategy for` $Player_\exists$ `*/`

Algorithm 10: CEGAR for timed control with partial observability.

7.3.2 Concretizability Characterization

Concretization Functions

The concretization functions relating the game structures $G^{\#}$ and G^{f}_{skip} are the ones defined in Section 5.1. The concretization functions relating $G^{\#}$ and G_{skip} are as follows: $\gamma : S^{\#} \to 2^{S}$ is such that $\gamma(s^{\#}, G_{skip}) = \gamma^{f}(\gamma(s^{\#}, G^{f}_{skip}))$ and for $\gamma_{\exists} : \Sigma^{\#} \to 2^{\Sigma_{\exists}}$ we have $\gamma_{\exists}(\sigma^{\#}, G_{skip}) = \{\sigma \in \Sigma_{\exists} \mid \exists s \in S.\ (\forall v \in V^{f}_{\exists}.\ s(v) = \sigma(v)) \wedge (\forall \varphi \in Obs(\mathcal{P}).\ s \models \varphi \Leftrightarrow \sigma^{\#} \models \varphi)\}$. While the concretization of an abstract action $\sigma^{\#}$ in G^{f}_{skip} is a singleton set $\{\sigma^{f}\}$, the concretization in G_{skip} is $\{\sigma \in \Sigma_{\exists} \mid \forall v \in V^{f}_{\exists}.\ \sigma(v) = \sigma^{f}(v)\} = \gamma^{f}_{\exists}(\sigma^{f})$.

The construction of $\mathsf{Abstract}(\mathcal{G}^{f}_{skip}(\mathcal{N}, l_{Err}, X_{\mathsf{c}}, \xi), \mathcal{P})$ together with the definition of $\mathcal{G}^{f}_{skip}(\mathcal{N}, l_{Err}, X_{\mathsf{c}}, \xi)$ and the requirements on $\mathcal{P}$ listed in Section 7.2.2, entail the following properties of $\mathsf{Abstract}(\mathcal{G}^{f}_{skip}(\mathcal{N}, l_{Err}, X_{\mathsf{c}}, \xi), \mathcal{P})$.

Property 22. *For a state $s^{\#}$ in $G^{\#} = \mathsf{Abstract}(\mathcal{G}^{f}_{skip}(\mathcal{N}, l_{Err}, X_{\mathsf{c}}, \xi), \mathcal{P})$ it holds that:*

- *There exists an action $\sigma \in \Sigma^{f}_{\exists}$ such that for each state $s^{f} \in \gamma(s^{\#}, G^{f}_{skip})$ and each state $s \in \gamma(s^{\#}, G_{skip}) \cap [\![[s^{\#}]]\!]$ it holds that $s^{f}(V^{f}_{\exists}) = \sigma$ and $s(V^{f}_{\exists}) = \sigma$.*
- *Either for each state $s^{f} \in \gamma(s^{\#}, G^{f}_{skip})$ and each state $s \in \gamma(s^{\#}, G_{skip})$ it holds that $s^{f}(er) = true$ and $s(er) = true$, or for each state $s^{f} \in \gamma(s^{\#}, G^{f}_{skip})$ and each state $s \in \gamma(s^{\#}, G_{skip})$ it holds that $s^{f}(er) = false$ and $s(er) = false$.*
- *If $s^{\#} \in S^{\#}_{\exists}$, then for each $\sigma \in \Sigma^{f}_{\exists}$ such that we have either $s(er) = true$ for each $s \in \gamma(s^{\#}, G_{skip})$ or $\sigma(reset) = \emptyset$, there exists a unique state $s^{\#\prime}$ such that for each $s^{f} \in \gamma(s^{\#}, G^{f}_{skip})$ and each $s \in \gamma(s^{\#}, G_{skip}) \cap [\![[s^{\#}]]\!]$ we have $s^{f}(V^{f}_{\exists}) = \sigma$ and $s(V^{f}_{\exists}) = \sigma$ and it holds thats $(s^{\#}, \sigma^{\#}, s^{\#\prime}) \in \mathcal{T}^{\#}_{\exists}$ for some $\sigma^{\#} \in \Sigma^{\#}_{\exists}$.*

Extended Counterexample Tree

In order to ensure that the values computed to refine the action-point function are suitable for eliminating the analyzed abstract counterexample tree, we have to take into account the connection between the possible choices (i) and (iii) by the controller, namely executing a controllable action after certain delay or letting time elapse. More specifically, in order for $Player_{\exists}$ to spoil a given counterexample by taking a controllable action after some delay, i.e., case (i), within the duration of this delay $Player_{\forall}$ should not be able to enforce an error node following the subtree corresponding to case (iii).

To take into account this relation between different branches of $C_t^\#$, we consider an *extended counterexample tree* $Extend(C_t^\#) = (\widetilde{N}^\#, \widetilde{E}^\#, \widetilde{L_s}^\#, \widetilde{L_a}^\#)$ and a relation $R \subseteq (N^\# \cup \widetilde{N}^\#) \times N^\#$, which are the smallest tree and binary relation such that:

- $C_t^\#$ is a connected subgraph of $Extend(C_t^\#)$,
- $Extend(C_t^\#)$ satisfies conditions (i), (ii), (iii), (iv), (v) and (vii) of Definition 2.2.4,
- if $n \in N^\#$, ${L_s}^\#(n) \in S_\exists^\#$, $m_1, m_2 \in \mathit{Children}(n)$, $\sigma_\exists(m_1)(act) \in \Sigma_\mathsf{c}$, $\sigma_\exists(m_2)(act) = \flat$, $\sigma_\exists(m_1)(wait) = true$ and $\sigma_\exists(m_2)(wait) = true$, then $(m_1, m_2) \in R$,
- if $(n_1, n_2) \in R$ and $\sigma_\exists(n_2)(wait) = true$ then $\sigma_\exists(n_2)(act) = \flat$,
- if $(n_1, n_2) \in R$, $s_1^\# = {L_s}^\#(n_1)$ and $s_2^\# = {L_s}^\#(n_2)$, then for every $s_1 \in [\![s_1^\#]\!]$ there exists $s_2 \in [\![s_2^\#]\!]$ such that $s_1(V^f \setminus \{t, act\}) = s_2(V^f \setminus \{t, act\})$ and:
  - if $s_1(t) \neq s_2(t)$, then $s_1(act) \in \Sigma_\mathsf{c}$, $s_1(wait) = true$, $s_1(t) = \forall$ and $s_2(t) = \exists$,
  - if $s_1(act) \neq s_2(act)$, then $s_1(act) \in \Sigma_\mathsf{c}$ and $s_2(act) = \flat$,
- if $n_1, n_2 \in \widetilde{N}^\#$, $(n_1, n_2) \in R$ and ${L_s}^\#(n_1), {L_s}^\#(n_2) \in S_\exists^\#$, then for each $m_2 \in \mathit{Children}(n_2)$ with $\sigma_\exists(act) = \flat$ there exists $m_1 \in \mathit{Children}(n_1)$ with $(m_1, m_2) \in R$,
- if $n_1, n_2 \in \widetilde{N}^\#$, $(n_1, n_2) \in R$ and ${L_s}^\#(n_1), {L_s}^\#(n_2) \in S_\forall^\#$, then for each $m_2 \in \mathit{Children}(n_2)$ there exists $m_1 \in \mathit{Children}(n_1)$ such that $(m_1, m_2) \in R$,
- if $n_1, n_2 \in \widetilde{N}^\#$, $(n_1, n_2) \in R$, ${L_s}^\#(n_1) \in S_\forall^\#$ and ${L_s}^\#(n_2) \in S_\exists^\#$, then for each $m_2 \in \mathit{Children}(n_2)$ for which $\sigma_\exists(act) = \flat$ and $\sigma_\exists(wait) = true$ there exists $m_1 \in \mathit{Children}(n_1)$ such that $(m_1, m_2) \in R$.

The existence of the tree $Extend(C_t^\#)$ is guaranteed by the properties of the transition relation of $Player_\forall$ in the game structure G_{skip}^f. More precisely, if $s_1, s_2 \in S_\forall^f$ are such that $s_1(V^f \setminus \{t, act\}) = s_2(V^f \setminus \{t, act\})$, $s_1(wait) = s_2(wait) = true$, $s_1(act) \in \Sigma_\mathsf{c}$ and $s_2(act) = \flat$, then $Player_\forall$ has from state s_1 all the options that he has from state s_2, except for giving the turn back to $Player_\exists$ without executing σ, which however can be matched by an idle transition from s_1. Thus, there exist $s_1', s_2' \in S^f$ such that $(s_1, s_1') \in T_\forall^f$, $(s_2, s_2') \in T_\forall^f$, $s_1'(V^f \setminus \{t, act\}) = s_2'(V^f \setminus \{t, act\})$, $s_1'(wait) = s_2'(wait) = true$, $s_1'(act) \in \Sigma_\mathsf{c}$ and $s_2'(act) = \flat$. Furthermore, the tree $Extend(C_t^\#)$ is finite and can be constructed from $C_t^\#$. Thus, for the rest of this section we assume that $C_t^\#$ is the extended counterexample tree defined above.

Counterexample Concretization

Definition 5.2.1 yields concretization functions for (extended) abstract counterexample trees, where $\gamma(C_t^{\#}, G_{skip}^f)$ and $\gamma(C_t^{\#}, G_{skip})$ are sets of knowledge-based counterexample trees in the games $\mathsf{Safety}(G_{skip}^f, Err^f)$ and $\mathsf{Safety}(G_{skip}, Err)$ respectively.

Note that if the original counterexample tree is concretizable in some of the game structures G_{skip} and G_{skip}^f, so is the extended one. Note also that Proposition 7.2.1 and Theorem 2.2.2 imply that if $\gamma(C_t^{\#}, G_{skip}) \neq \emptyset$ then also $\gamma(C_t^{\#}, G_{skip}^f) \neq \emptyset$.

If $\gamma(C_t^{\#}, G_{skip}) \neq \emptyset$, then there exists a knowledge-based counterexample tree C_k in the game $\mathsf{Safety}(G_{skip}, Err)$. Therefore, by Theorem 2.2.1, $Player_{\exists}$ does not have an obs_s-consistent winning strategy in the game $\mathsf{Safety}(G_{skip}, Err)$. By Proposition 7.2.3, $Player_{\exists}$ does not have an obs_r-consistent winning strategy in the game $\mathsf{Safety}(G, Err)$.

Quantified Tree Formula

We now provide a logical characterization of concretizability of an abstract counterexample $C_t^{\#}$ in the game $\mathsf{Safety}(G_{skip}, Err)$. To this end, we construct a *quantified tree formula* $\mathsf{QTF}(C_t^{\#})$ that evaluates to *true* iff the abstract counterexample tree is concretizable in $\mathsf{Safety}(G_{skip}, Err)$. On a high-level the construction resembles the one described in Section 5.1. Here, however, we cannot construct the formula as a conjunction over a set of trace formulas, since the set $\Sigma_{\exists}$ of $Player_{\exists}$ actions is infinite. Thus, here the result is a linear arithmetic formula with alternating universal and existential quantifiers corresponding to the alternating choices of the two players. The variables $V_{\forall} \dot{\cup} \{t\}$ updated by $Player_{\forall}$ are existentially quantified, and the variables $V_{\exists} \dot{\cup} \{t\}$ updated by $Player_{\exists}$, including the symbolic constants, are universally quantified.

First, for a path $\rho = n_0 n_1 \dots n_k$ in $C_t^{\#}$, we define the formula $\varphi_{path}(\rho)$, which characterizes the set of concrete paths in G_{skip} corresponding to ρ, as follows:

$$\begin{aligned}\varphi_{path}(\rho)[V^{n_0}, \dots, V^{n_k}] \;=\; & \big(\textstyle\bigwedge_{i=0}^{k} [{L_s}^{\#}(n_i)][V^{n_i}/V]\big) \wedge \\ & \big(\textstyle\bigwedge_{i<k, {L_s}^{\#}(n_i) \in S_{\exists}^{\#}} \mathcal{T}_{\exists}[V^{n_i}/V, V^{n_{i+1}}/V]\big) \wedge \\ & \big(\textstyle\bigwedge_{i<k, {L_s}^{\#}(n_i) \in S_{\forall}^{\#}} \mathcal{T}_{\forall}[V^{n_i}/V, V^{n_{i+1}}/V]\big).\end{aligned}$$

Since the abstraction is precise w.r.t. the choices of $Player_{\exists}$ in G_{skip}^f, for each node n in $C_t^{\#}$, ${L_s}^{\#}(n)$ defines a valuation $\sigma^f(n)$ of the variables in the set $V_{\exists}^f$. Based on this valuation we define the substitution $\eta_n = \{\sigma^f(n)(x)/x^n \mid x \in V_{\exists}^f \cup \{t\}\}$ for each n.

Now, for a node n in $C_t^{\#}$ we define a formula $\psi_{node}(n)$ that characterizes the set of sets of observationally equivalent prefixes in $\mathcal{G}_{skip}$ that are subsumed by the prefix leading to n and lead to a set of states from which there is a knowledge-based counterexample tree in $\mathcal{G}_{skip}$ contained in the subtree of $C_t^{\#}$ rooted at the node n.

The formula $\psi_{node}(n)$ is defined recursively as follows.

- If n is a leaf node in $C_t^{\#}$ and $\rho = path(n) = n_0 n_1 \dots n_k$ then we define

$$\psi_{node}(n) = \exists V_\forall^{o\,n} \exists (V_\forall \setminus V_\forall^o)^{n_0} \dots \exists (V_\forall \setminus V_\forall^o)^{n_k}.\ \big(\varphi_{path}(\rho) \wedge loc^n = l_{Err}\big)\eta_n.$$

- Otherwise, if ${L_s}^{\#}(n) \in S_\forall^{\#}$ we let $N' = \{n' \mid (n, n') \in E^{\#}\}$ and define

$$\psi_{node}(n) = \exists V_\forall^{o\,n}.\ \bigvee_{m \in N'} \big(\psi_{node}(m)[SC^n/SC^m]\eta_n\big).$$

- Otherwise, if ${L_s}^{\#}(n) \in S_\exists^{\#}$ we let $N' = \{n' \mid (n, n') \in E^{\#}\}$ and define

$$\psi_{node}(n) = \exists V_\forall^{o\,n}.\ \bigwedge_{m \in N'} \big(\psi(n, m)\eta_m\big),$$

where for each $m \in N'$, the formula $\psi(n, m)$ is defined depending on $\sigma^f(m)$, that is, depending on the choice made by $Player_\exists$ at node n in the game $G^{\#}$.

For the successor corresponding to case (i), i.e., executing a controllable action after a positive delay, we quantify universally over the variables in SC^m, adding a condition which restricts their values to ones that are valid choices of await points for $Player_\exists$ in $\mathcal{G}_{skip}$. We further require that these await points correspond to intermediate action points. This gives a condition $\theta(n, m)$ on the symbolic constants SC^m at node m and we define $\psi(n, m) = \forall SC^m.(\theta(n, m) \rightarrow \psi_{node}(m))$, where

$$\begin{aligned}\theta(n, m) \;=\; & \big(\textstyle\bigvee_{x \in X_{\mathsf{o}+\mathsf{c}}} c_x^m > 0\big) \wedge \textstyle\bigwedge_{x \in X_{\mathsf{o}+\mathsf{c}}} (c_x^m > 0 \rightarrow c_x^m > x^n) \wedge \\ & \textstyle\bigwedge_{\substack{x,y \in X_{\mathsf{o}+\mathsf{c}} \\ c \in \xi(x)}} \big((x^n < c \wedge c_y^m > 0) \rightarrow x^n + (c_y^m - y^n) < c\big).\end{aligned}$$

For the successors corresponding to cases (ii), (iii) or (iv), i.e, executing a controllable action immediately, letting time elapse or resetting a set of controllable clocks, we substitute the symbolic constants SC^m with $\mathbf{0}$ (according to the transition relation $\mathcal{T}_\exists$ in $\mathcal{G}_{skip}$). Thus, $\psi(n, m) = \psi_{node}(m)[\mathbf{0}/SC^m]$.

For a node n in $C_t^{\#}$ with $path(n) = n_0 n_1 \dots n_k$ we have $Vars(\psi_{node}(n)) \subseteq SC^n \cup \bigcup_{i=0}^{k-1}(Obs(V^{n_i}) \cup SC^{n_i})$. Thus, for the root node n_0, $Vars(\psi_{node}(n_0)) \subseteq SC^{n_0}$.

The formula $\mathsf{QTF}(C_t^{\#})$ is constructed by annotating in a bottom-up manner each node n in $C_t^{\#}$ with the formula $\psi_{node}(n)$ and letting $\mathsf{QTF}(C_t^{\#}) = \psi_{node}(n_0)[\mathbf{0}/SC^{n_0}]$ where n_0 is the root of $C_t^{\#}$. Thus, $\mathsf{QTF}(C_t^{\#})$ is a closed formula.

The formula $\mathsf{QTF}(C_t^{\#})$ characterizes the concretizability of $C_t^{\#}$ in $\mathcal{G}_{skip}$.

Theorem 7.3.1. *Let $\mathcal{N} = (\mathcal{A}, \Sigma_{\mathsf{c}}, \Sigma_{\mathsf{u}}, X_{\mathsf{o}}, X_{\mathsf{u}}, =_{\mathsf{o}}^{\mathsf{L}})$ be a partially observable plant, $l_{Err} \in \mathsf{Loc}$ be an error location such that $[l_{Err}]_{=_{\mathsf{o}}^{\mathsf{L}}} = \{l_{Err}\}$, X_{c} be a finite set of controllable clocks such that $X_{\mathsf{c}} \cap X = \emptyset$, $\xi : X_{\mathsf{o}+\mathsf{c}} \to 2^{\mathbb{Q}_{>0}}$ be an action-point function and $\mathcal{P} \subseteq \mathcal{AP}(V^f)$ be a finite set of predicates satisfies the requirements listed in Section 7.2.2.*

Then, if $G^{\#} = \mathsf{Abstract}(\mathcal{G}_{skip}^f(\mathcal{N}, l_{Err}, X_{\mathsf{c}}, \xi), \mathcal{P})$, $Err^{\#} = \{s^{\#} \in S^{\#} \mid [\![s^{\#}]\!] \cap [\![\varphi_{Err}]\!] \neq \emptyset\}$ and $C_t^{\#}$ is an abstract counterexample tree in $\mathsf{Safety}(G^{\#}, Err^{\#})$, then $\gamma(C_t^{\#}, \mathcal{G}_{skip}) \neq \emptyset$ iff the quantified tree formula $\mathsf{QTF}(C_t^{\#})$ is satisfiable.

Proof. First, consider the case when $\mathsf{QTF}(C_t^{\#})$ is satisfiable, and let M be a model of the prenex normal form Φ of $\mathsf{QTF}(C_t^{\#})$, obtained from $\mathsf{QTF}(C_t^{\#})$ by first replacing the formula $\psi_{node}(n)$ for each leaf node n by the formula

$$\exists V_{\forall}^{o\,n} \exists (V_{\forall} \setminus V_{\forall}^o)^{n_0, path_{\exists}(n)} \dots \exists (V_{\forall} \setminus V_{\forall}^o)^{n_k, path_{\exists}(n)}. \big(\varphi_{path}(\rho) \wedge loc^n = l_{Err}\big)\eta'_n,$$

where $path_{\exists}(n)$ is the subsequence of $path(n)$ consisting of the indices n_i with ${L_s}^{\#}(n_i) \in S_{\exists}^{\#}$, and the substitution η'_n is $\eta_n \circ \big\{x^{n_i, path_{\exists}(n)}/x^{n_i} \mid x \in V_{\forall} \setminus V_{\forall}^o, n_i \in path(n)\big\}$.

For a node $n \in N^{\#}$ we denote with $num_{\exists}(n)$ the number of nodes n' on $path(n)$ such that ${L_s}^{\#}(Parent(n')) \in S_{\exists}^{\#}$, $\sigma^f(n)(act) \in \Sigma_{\mathsf{c}}$ and $\sigma^f(n)(wait) = true$.

For each $n \in N^{\#}$ and $v \in V_{\forall}^o$, $M(v^n)$ is a function $M(v^n) : \Sigma_{\exists}^{num_{\exists}(n)} \to \mathsf{Dom}(v)$.

For each $n \in N^{\#}$ and each leaf node $m \in N^{\#}$, where n is a node on $path(m)$, and every $v \in V_{\forall} \setminus V_{\forall}^o$, $M\big(v^{n, path_{\exists}(m)}\big)$ is a function $M\big(v^{n, path_{\exists}(m)}\big) : \Sigma_{\exists}^{num_{\exists}(m)} \to \mathsf{Dom}(v)$.

We define the labeled tree $C_k = (N, E, K_s, L_a)$ with $N \subseteq N^{\#} \times \Sigma_{\exists}^*$ as the smallest graph that satisfies the following conditions:

- $(n_0^{\#}, \epsilon) \in N^{\#}$ and $K_s((n_0^{\#}, \epsilon)) = I \cap obs(s_0)$, where $I = \{s_0\}$;
- if $(n, \tau) \in N$, $K_s((n, \tau)) \cap Err = \emptyset$, ${K_s}^{\#}(n) \subseteq S_{\exists}^{\#}$, $(n, n') \in E^{\#}$, $\sigma \in \Sigma_{\exists}$, $\sigma(V_{\exists}^f) = \sigma^f(n')(V_{\exists}^f)$, then let $S' = \mathsf{Post}_{\exists}(K_s((n, \tau)), \sigma)$ and if $S' \neq \emptyset$, then $(n', \tau \cdot \sigma) \in N$, $((n, \tau), (n', \tau \cdot \sigma)) \in E$, $K_s((n', \tau \cdot \sigma)) = S'$, $L_a(((n, \tau), (n', \tau \cdot \sigma))) = \sigma$,
- if $(n, \tau) \in N$, $K_s((n, \tau)) \cap Err = \emptyset$, ${K_s}^{\#}(n) \subseteq S_{\forall}^{\#}$ and $(n, n') \in E^{\#}$ then let $S' = \mathsf{Post}_{\forall}(K_s(n, \tau)) \cap o$, where $o = \{s \in S \mid \forall v \in V_{\forall}^o.\, s(v) = M(v^{n'})(\tau)\}$, and if $S' \neq \emptyset$, then $(n', \tau) \in N$, $((n, \tau), (n', \tau)) \in E$, $K_s((n', \tau)) = S'$, $L_a(((n, \tau), (n', \tau))) = \epsilon$.

Conditions (i), (ii), (iii), (iv) and (v) from the definition of knowledge-based counterexample tree are implied by the definition of C_k. Condition (vi) holds, since (i) is satisfied and it holds that if $s_1, s_2 \in S_\exists$, $s_1', s_2' \in S$, $\sigma \in \Sigma_\exists$, $obs(s_1) = obs(s_2)$, $(s_1, \sigma, s_1') \in T_\exists$ and $(s_2, \sigma, s_2') \in T_\exists$, then $obs(s_1') = obs(s_2')$. Since M is a model of $\mathsf{QTF}(C_t^\#)$, we can choose for each node $(n, \tau) \in N$ with $K_s((n, \tau)) \subseteq S_\forall$ a node $(n', \tau) \in \mathit{Children}((n, \tau))$ such that the resulting tree C_k' satisfies condition (viii). By the definition of C_k and C_k', condition (vii) is satisfied by the tree C_k'. Thus, C_k' is a knowledge-based counterexample tree in $\mathsf{Safety}(G_{skip}, Err)$. Taking into account that M is a model of $\mathsf{QTF}(C_t^\#)$, the definition of C_k also implies that $C_k' \in \gamma(C_t^\#, G_{skip})$.

For the other direction, suppose that $\gamma(C_t^\#, G_{skip}) \neq \emptyset$ and let $C_k = (N, E, K_s, L_a) \in \gamma(C_t^\#, G_{skip})$. For a node $n \in N$ we denote with $\mathit{trace}(n)$ the sequence of elements of $\Sigma_\exists$ labeling the edges on $\mathit{path}(n)$. By the properties of C_k, for each leaf node $m \in N$ with $\mathit{path}(m) = n_0 n_1 \dots n_k$, there exists a path $\pi_m = s_0 s_1 \dots s_k \in \mathsf{Prefs}(G_{skip})$ such that $s_i \in K_s(n_i)$ for each $0 \leq i \leq k$. For each $n^\# \in N^\#$ in $C_t^\#$, we denote with $N_{n^\#}$ the set of corresponding nodes in the tree C_k, and with $M_{n^\#}$ the set of pairs (n, m), where $n \in N_{n^\#}$ and m is a leaf node in C_k such that n is a node on $\mathit{path}(m)$. By Definition 2.2.5, for each $\tau \in \Sigma_\exists^*$ there exists at most one $n \in N$ such that $\mathit{trace}(n) = \tau$.

For each $n^\# \in N^\#$ and $v \in V_\forall^o$, we define the function $M(v^{n^\#}) : \Sigma_\exists^{num_\exists(n^\#)} \to \mathsf{Dom}(v)$, such that for $\tau \in \Sigma_\exists^{num_\exists(n^\#)}$, $M(v^{n^\#})(\tau) = s(v)$, if there exists $n \in N_{n^\#}$ such that $\mathit{trace}(n) = \tau$ and $s \in K_s(n)$, and $M(v^{n^\#})(\tau)$ is arbitrary if there is no such n.

For each $n^\# \in N^\#$ and each leaf node $m^\# \in N^\#$, where $n^\#$ is a node on $\mathit{path}(m^\#)$, and every $v \in V_\forall \setminus V_\forall^o$, we define the function $M\big(v^{n^\#, path_\exists(m^\#)}\big) : \Sigma_\exists^{num_\exists(m^\#)} \to \mathsf{Dom}(v)$ such that for $\tau \in \Sigma_\exists^{num_\exists(m^\#)}$, $M\big(v^{n^\#, path_\exists(m^\#)}\big)(\tau) = s(v)$, if there exists a pair $(n, m) \in M_{n^\#, m^\#}$ and s is the state on the path π_m corresponding to the node n, and $M\big(v^{n^\#, path_\exists(m^\#)}\big)(\tau)$ is arbitrarily fixed if no such pair (n, m) exists.

The functions defined above constitute a model M for $\mathsf{QTF}(C_t^\#)$. □

7.3.3 Computing Observation Predicates

At each iteration of the inner or outer loop when the respective formula characterizing the concretizability of the abstract counterexample tree is unsatisfiable, i.e., the counterexample is not concretizable, the fixed-observation abstraction of the considered await-time game is refined by introducing new observation predicates. The new observation predicates are computed using the respective tree formula.

7.3.3.1 Computing Decision Predicates

When the concretizability check of the inner loop determines that the abstract counterexample tree is not concretizable in the game structure G^f_{skip}, we refine the abstract game using predicates computed by the procedures described in Section 5.3 and Section 5.4. The new observable predicates are the new decision predicates. In particular, the interpolation-based procedure from Section 5.3 yields decision predicates that allow $Player_\exists$ to distinguish states in which he has to make different choices.

As additional decision predicates we obtain the predicates $x \geq c$ and $x \leq c$ corresponding to each new action point c for each clock variable x from $X_{\mathsf{o+c}}$. New action points are computed during the refinement phase of the outer loop as described below.

7.3.3.2 Computing Action Points

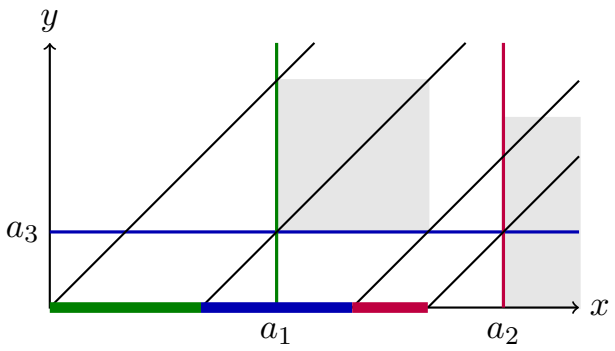
The action-point function ξ is refined in the cases when the abstract counterexample tree is concretizable in the game structure G^f_{skip}, but the extended abstract counterexample tree is not concretizable in G_{skip}. The procedure RefineActionPoints, computes new action points for ξ based on the unsatisfiable formula $\mathsf{QTF}(C_t^\#)$.

If the closed formula $\mathsf{QTF}(C_t^\#)$ evaluates to *false*, its negation $\neg\mathsf{QTF}(C_t^\#)$ evaluates to *true*, i.e., is satisfiable. Let Ψ be the negation of the prenex normal form of the formula $\mathsf{QTF}(C_t^\#)$, constructed as described in the proof of Theorem 7.3.1. In Ψ, after pushing the negation through the quantifiers, all symbolic constants (indexed accordingly) are existentially quantified. Consider a node $n \in N^\#$ such that ${L_s}^\#(Parent(n)) \in S^\#_\exists$, and let $path(n) = n_0 n_1 \dots n_r$. In Ψ there exists a quantifier block $\exists SC^n$, which is preceded by the blocks of universal quantifiers $\forall V^{on_i}_\forall$ for $i = 0, \dots, r$. Thus, if M is a model of Ψ, then for each $x \in X_{\mathsf{o+c}}$ we have that the witness $M(c^n_x)$ for c^n_x is a function $M(c^n_x) : Vals(V^{on_0}_\forall) \times \dots \times Vals(V^{on_r}_\forall) \to \mathbb{Q}_{\geq 0}$.

Our goal is to compute witnesses for the symbolic constants that can be used for refining the action-point function ξ. Assume for now, that in a model M we have a tuple of witness functions for the variables in SC^n of the following form.

For some $k \in \mathbb{N}_{>0}$, there are positive rational constants $a_1, \dots, a_k \in \mathbb{Q}_{>0}$ and a function $b : Vals(V^{on_0}_\forall) \times \dots \times Vals(V^{on_r}_\forall) \to \mathbb{N} \cap [1, k]$, such that:

- each a_i is associated with some variable $x_i \in X_{\mathsf{o+c}}$,
- for each $\overline{s} \in Vals(V^{on_0}_\forall) \times \dots \times Vals(V^{on_r}_\forall)$ it holds that:



$$a_1 = 3, \quad x_1 = x$$
$$a_2 = 6, \quad x_2 = x$$
$$a_3 = 1, \quad x_3 = y$$

$$x^n \in [0,2) \mapsto c_x^n = 3, c_y^n = 0$$
$$x^n \in [2,4) \mapsto c_x^n = 0, c_y^n = 1$$
$$x^n \in [4,5] \mapsto c_x^n = 6, c_y^n = 0$$

Figure 7.2: Example of witnesses for symbolic constants c_x^n and c_y^n.

- if $b(\overline{s}) = i$ and $x = x_i$, then $M(c_x) = a_i$,
- if $b(\overline{s}) = i$ and $x \neq x_i$, then $M(c_x) = 0$.

That is, we have a case split with k cases according to the valuation of the observable variables along the prefix, such that in each case exactly one of the symbolic constants (the one corresponding to the clock variable associated with this case) is assigned the constant corresponding to this case, and to all other symbolic constants 0 is assigned.

Example 7.3.1. Consider an example with $X_{\mathsf{o+c}} = \{x, y\}$, where in the abstract state $L_s(n)$ we know that the value of x^n is in $[0, 5]$ and the value of y^n is 0, and the "good" values for SC^n are depicted as the gray sets on Fig. 7.2. The figure shows an example where $k = 3$ and each a_i is associated with the variable x_i shown in the figure.

Given such functions, we can refine the action-point function ξ as follows. For each $x \in X_{\mathsf{o+c}}$ we add to the set $\xi(x)$ all positive values a such that $M(c_x^n)(\overline{s}) = a$ for some node n and valuation $\overline{s} \in \mathit{Vals}(V_\forall^{on_0}) \times \ldots \times \mathit{Vals}(V_\forall^{on_r})$. The assumption that we made about the model M ensures that the number of these values is finite.

Now, given a natural number $k \in \mathbb{N}_{>0}$ we can restrict the possible witness functions for the indexed symbolic constants to the form discussed above by introducing additional variables to encode functions of this form and strengthening the formula Ψ.

In the strengthened formula Ψ_k we use the following fresh additional variables.

- A set $A_k^n = \{a_1^n, \ldots, a_k^n\}$ of k rational variables.
- An integer variable b^n with domain $\mathbb{N} \cap [1, k]$.

- A set $D_k^n = \{d_1^n, \ldots, d_k^n\}$ of k integer variables with domain $\mathbb{N} \cap [1, |X_{\mathsf{o+c}}|]$.

The formula Ψ_k is obtained from Ψ by replacing for each $n \in N^{\#}$ the condition θ^n for SC^n that was used in the construction of Ψ by the stronger one $\theta^n \wedge \theta_k^n$, where:

$$\theta_k^n = \exists b^n \bigwedge_{i=1}^{k} \Big(b^n = i \rightarrow \bigwedge_{x \in X_{\mathsf{o+c}}} \big((\mathsf{id}(x) = d_i^n \rightarrow c_x^n = a_i^n) \wedge (\mathsf{id}(x) \neq d_i^n \rightarrow c_x^n = 0) \big) \Big),$$

and $\mathsf{id} : X_{\mathsf{o+c}} \rightarrow \mathbb{N} \cap [1, |X_{\mathsf{o+c}}|]$ is an indexing function for the clock variables $X_{\mathsf{o+c}}$.

In the formula Ψ_k, the variable b^n is existentially quantified in the same quantifier block as the symbolic constants in SC^n. The variables from the sets A_k^n and D_k^n are free in Ψ_k (existentially quantified on the outermost level).

The formula Ψ_k has only models in which the functions assigned to the symbolic constants are of the form we discussed above. Formally, it has the following property.

Property 23. *If the formula Ψ_k is satisfiable and M is a model of Ψ_k, then M is a model of Ψ as well, and for each $x \in X_{\mathsf{o+c}}$ and $n \in N^{\#}$ it holds that:*

$$M(c_x^n)(v) = \begin{cases} M(a_1^n) & \text{if } M(b^n)(v) = 1 \wedge M(d_1^n) = \mathsf{id}(x), \\ \ldots & \\ M(a_k^n) & \text{if } M(b^n)(v) = k \wedge M(d_k^n) = \mathsf{id}(x), \\ 0 & \text{otherwise}, \end{cases}$$

where $path(n) = n_0 n_1 \ldots n_r$ and $v \in Vals(V_\forall^{on_0}) \times \ldots \times Vals(V_\forall^{on_r})$.

The refinement procedure RefineActionPoints iterates over the values of $k \geq 1$, at each step constructing the formula Ψ_k as described above. Ψ_k is a strengthening of Ψ and Ψ_{k+1} is weaker than Ψ_k. The procedure terminates if a k for which the formula Ψ_k is satisfiable is reached. In this case, we use the values in the resulting model of the newly introduced variables from A_k^n to refine the action-point function ξ.

If it terminates, Algorithm 11 returns a new action-point function ξ' such that for every $x \in X_{\mathsf{o+c}}$, we have $\xi'(x) \supseteq \xi(x)$. The new action points for x are extracted from a model for Ψ_k and are the values of those variables a_i^n for which d_i^n is equal to $\mathsf{id}(x)$.

Example 7.3.2. We now provide an example, for which there does not exists a k that is such that there exist witness function of the desired form. In the case we consider, for each $k > 0$ the formula Ψ_k is unsatisfiable, although the formula Ψ is satisfiable.

Here, $X_{\mathsf{o+c}} = \{x\}$ and $X_{\mathsf{u}} = \{y\}$, where in the abstract state $L_s(n)$ we know that the value of x^n is in $[0, 5]$ and the value of y^n is 0, and the "good" values for SC^n (i.e., when $y = 1$) are depicted as the gray line in Figure 7.3.

Algorithm: RefineActionPoints

Input: satisfiable formula Ψ with conditions θ^n for SC^n, for nodes $n \in N^{\#}$;
action-point function $\xi : X_{\mathsf{o+c}} \to 2^{\mathbb{Q}_{>0}}$

Output: refined action-point function $\xi' : X_{\mathsf{o+c}} \to 2^{\mathbb{Q}_{>0}}$

$\xi'(x) := \xi(x)$ for every $x \in X_{\mathsf{o+c}}$;
$\mathcal{R} = \emptyset$;
$k := 0$;
$sat := false$;
while $sat = false$ **do**
  $k++$;
  $\Psi_k := Strengthen(\Psi, k)$;
  $sat := \mathsf{CheckSat}(\Psi_k)$;

$M := \mathsf{Model}(\Psi_k)$;
foreach $(n, i) \in (N^{\#} \times \{1, ..k\})$ ***with*** $M(a_i^n) > 0$ **do**
  foreach $x \in X_{\mathsf{o+c}}$ ***with*** $\mathsf{id}(x) = M(d_i^n)$ **do**
    $\xi'(x) := \xi'(x) \cup \{M(a_i^n)\}$;
    $\mathcal{R} := \mathcal{R} \cup \{x \leq M(a_i^n), x \geq M(a_i^n)\}$;

return $(\xi', \mathcal{R})$;

Algorithm 11: Computation of refinement action points.

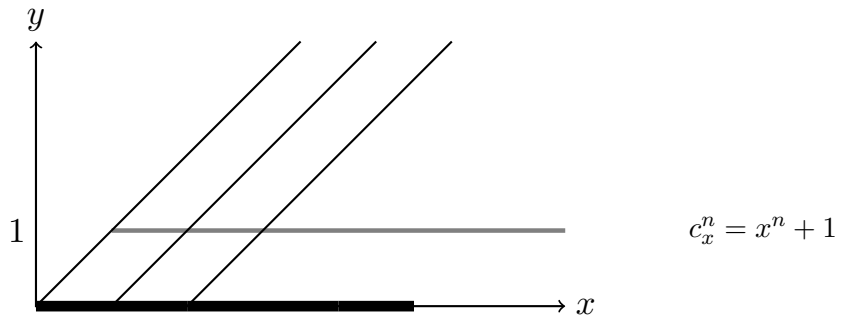


Figure 7.3: Example of witnesses for symbolic constant c_x^n.

7.3.4 Progress

Now we will establish a progress property of the procedure RefineActionPoints, namely we will prove that the new action points suffice to eliminate all knowledge-based counterexample trees C_k^f in G_{skip}^f that are in the concretization of $C_t^\#$.

Theorem 7.3.2. *Let $\mathcal{N} = (\mathcal{A}, \Sigma_\mathsf{c}, \Sigma_\mathsf{u}, X_\mathsf{o}, X_\mathsf{u}, =_\mathsf{o}^\mathsf{L})$ be a partially observable plant, $l_{Err} \in \mathsf{Loc}$ be an error location such that $[l_{Err}]_{=_\mathsf{o}^\mathsf{L}} = \{l_{Err}\}$, X_c be a finite set of controllable clocks such that $X_\mathsf{c} \cap X = \emptyset$, $\xi : X_{\mathsf{o}+\mathsf{c}} \to 2^{\mathbb{Q}_{>0}}$ be an action-point function and $\mathcal{P} \subseteq \mathcal{AP}(V^f)$ be a finite set of predicates satisfies the requirements listed in Section 7.2.2.*

Let $G^\# = \mathsf{Abstract}(\mathcal{G}_{skip}^f(\mathcal{N}, l_{Err}, X_\mathsf{c}, \xi), \mathcal{P})$, $Err^\# = \{s^\# \in S^\# \mid [\![s^\#]\!] \cap [\![\varphi_{Err}]\!] \neq \emptyset\}$ and suppose that $C_t^\#$ is an extended counterexample tree in $\mathsf{Safety}(G^\#, Err^\#)$ such that the quantified tree formula $\mathsf{QTF}(C_t^\#)$ is unsatisfiable. If ξ' is the refined action-point function and $\mathcal{R}$ is the set of refinement predicates computed by RefineActionPoints*, then for the await-time game with fixed action points $\mathsf{Safety}(\mathcal{G}_{skip}^f(\mathcal{N}, l_{Err}, X_\mathsf{c}, \xi'), \varphi_{Err})$, it holds that $\gamma(C_t^\#, \mathcal{G}_{skip}^f(\mathcal{N}, l_{Err}, X_\mathsf{c}, \xi')) = \emptyset$.*

Proof. By our hypothesis the formula $\mathsf{QTF}(C_t^\#)$ is unsatisfiable, which means that the respective negated formula Ψ is satisfiable. We are considering the case when the procedure RefineActionPoints terminates and returns a refined action-point function ξ'. Let $k > 0$ be the smallest positive natural number such that the corresponding formula Ψ_k is satisfiable. By the construction of Ψ_k, there exist a finite set $C \subseteq \mathbb{Q}_{>0}$ and an obs_s-consistent strategy $f_\exists$ for $Player_\exists$ in the game structure G_{skip} such that:

- For every $\pi \in \mathsf{Prefs}(f_\exists)$ for which there exists a path ρ in $C_t^\#$ such that $\pi[i] \in \gamma({L_s}^\#(\rho[i]))$ for each $0 \leq i < |\pi|$, it holds that $\pi[i] \notin Err$ for each $0 \leq i < |\pi|$.
- For each $\pi \in \mathsf{Prefs}_\exists(G_{skip})$ and $x \in X_{\mathsf{o}+\mathsf{c}}$, it holds that $f_\exists(\pi)(\sigma(c_x)) \in C \cup \{0\}$.

Thus, since the set $\Sigma_f = \{\sigma \in \Sigma_\exists \mid \exists \pi \in \mathsf{Prefs}_\exists(G_{skip}).\ f_\exists(\pi) = \sigma\}$ is finite, by Proposition 7.2.5 and the construction of ξ' we have that there exists an obs_s-consistent strategy $f_\exists^f$ for $Player_\exists$ in the game structure ${G^f}'_{skip} = G_{skip}^f(\mathcal{N}, l_{Err}, X_\mathsf{c}, \xi')$ such that for each $\pi^f \in \mathsf{Outcome}(f_\exists^f)$ there exists a play $\pi \in \mathsf{Outcome}(f_\exists)$ such that for each $0 \leq i < |\pi^f|$ there exists $0 \leq j < |\pi|$ such that $\pi[j](loc) = \pi^f[i](loc)$.

Furthermore, by the definition of $f_\exists^f$ in the proof of Proposition 7.2.5 (i.e., the proof of Proposition 7.2.2) and the fact that $C_t^\#$ is an extended counterexample tree, we have that if $\pi^f \in \mathsf{Prefs}(f_\exists^f)$ and there exists a path ρ in $C_t^\#$ such that for each $0 \leq i < |\pi^f|$, $\pi^f[i] \in \gamma({L_s}^\#(\rho[i]), {G^f}'_{skip})$, then there exists $\pi \in \mathsf{Prefs}(f_\exists)$ such that

- there exists a path ρ' in $C_t^\#$ such that for each $0 \leq i < |\pi|$, $\pi[i] \in \gamma({L_s}^\#(\rho'[i]), G)$,

- for each $0 \leq i < |\pi^f|$ there exists $0 \leq j < |\pi|$ such that $\pi[j](loc) = \pi^f[i](loc)$.

Now, suppose that $C_k \in \gamma(C_t^{\#}, {G^f}'_{skip})$ is a knowledge-based counterexample tree in $\mathsf{Safety}({G^f}'_{skip}, Err^f)$. Since $f_\exists^f$ is an obs_s-consistent strategy for $Player_\exists$ in ${G^f}'_{skip}$, there exists a path ρ in C_k such that for each $0 \leq i < |\rho|$ and each $s \in K_s(\rho[i])$ it holds that there exists a prefix $\pi^f \in \mathsf{Prefs}(f_\exists^f)$ such that $\mathsf{last}(\pi^f) = s$. By the above properties of $f_\exists^f$, we have that $s \notin Err$, which contradicts to the choice of C_k.

□

7.4 Experiments

7.4.1 Prototype implementation

We have developed a version of the prototype tool described in Section 5.5.1 that implements the approach described in this chapter. Here we rely on the ability of the recent versions of the Z3 SMT solver to handle formulas with quantifiers as the ones generated in the outer abstraction-refinement loop of SynthesizeObservationPredicates.

7.4.2 Experimental Results

We applied our prototype to the safety controller synthesis problem for the partially observable plant shown in Figure 6.1 and the Box Painting Production System and the Timed Game For Sorting Bricks examples from [Cas07]. The synthesis problems were encoded as await-time games. We applied our method to compute, starting with an empty set of action points and the respective set of initial predicates, sets of observation predicates for which the given plants are controllable, that is the sets of controlled paths fulfill the corresponding safety requirements.

Production System This is the example presented in the introduction to Chapter 6, described by the timed game automaton shown in Figure 6.1. Here, the controller needs observation predicates in the role of decision predicates and observation predicates in the role of action points in order to ensure that the error location End is never reached.

Box Painting The timed game automaton for the Box Painting Production System example is shown in Figure 1 in [Cas07]. Here the task of the controller is to execute

	OBS Iter.	CEGAR Iter.	Act. Points	Predicates	A. States	Time(s)	TIGA(s)
Prod. Syst.	2	4	3	35	3944	87.16	0.23
Paint	2	8	2	33	560	54.48	0.21
Paint-100	2	5	2	29	573	29.31	3.93
Paint-1000	2	5	2	29	573	29.18	339.95
Paint-10000	2	7	2	34	560	52.72	TO
Paint-100000	2	7	2	34	560	50.93	TO
Bricks	3	3	3	28	1175	24.68	0.04
Bricks-100	3	3	3	28	1175	25.22	2.56
Bricks-1000	3	3	3	28	1175	23.75	303.79
Bricks-10000	3	3	3	28	1175	24.21	TO
Bricks-100000	3	3	3	28	1175	25.33	TO

Table 7.1: Results from experiments with our prototype: number of iterations of the respective refinement loops, number of action points in the final abstraction, total number of predicates in the final abstraction, largest number of states explored in an intermediate abstraction, and running time. Results from experiments with UPPAAL-TIGA with fixed observations: running time, where timeout (TO) was set to 30 minutes.

the single controllable action at the right time to avoid the error location despite of the timing uncertainty coming from the variable duration of the processing phases.

Bricks Sorting The Timed Game For Sorting Bricks is given in Figure 3 in [Cas07]. It is similar to our first example, the difference being that the controller does not need decision predicates over clock variables, as observing the equivalence class of the plant's location suffices for selecting the correct controllable action to be executed.

Results In order to demonstrate that our method performs well in situations where fine granularity is needed to win the game, i.e., when the constraints occurring in the plant involve large constants and the differences between certain guards and invariants are small, we constructed multiple instances of each of the last two examples. Instances Paint$-N$ and Bricks$-N$, where $N \in \{100, 1000, 10000, 100000\}$ were obtained by adding the constant N to all positive constants occurring in the plants.

In Table 7.1 we report on the results from our experiments preformed on an Intel Core 2 Duo CPU at 2.53 GHz with 3.4 GB RAM, using a single core. We give the number of refinement iterations for the await-time game with fixed action points and the number of iterations of the inner loop, the sizes of the final sets of action points and abstraction predicates and the maximal number of explored states in the intermediate

$\mathcal{G}$	CEGAR	$\mathcal{P}(\mathcal{G})$	$\mathcal{P}_{\min}(\mathcal{G})$	$\mathcal{P}_s(\mathcal{G})$
Paint-10000	52.78s	2.77s	0.23s	MemoryError (35m30.402s)
Bricks-10000	24.21s	1.382s	0.78s	MemoryError (17m4.781s)

Table 7.2: Comparison of the running times of counterexample-guided synthesis of observations and solving respective timed games with fixed observations using [BCD+12]. The first column shows the running time of our prototype and the remaining three columns give the running time of [BCD+12] on sets of observation predicates corresponding respectively to observable clock predicates computed by the CEGAR approach, a minimal subset of this set and a set corresponding to an a priori fixed fine granularity.

abstractions. The results show that the size of the abstract games generated by our approach depend on the number of action points and predicates and not on the size of the constants in the plant. This is in contrast with approaches based on fixed granularity, where strategies involve counting modulo the given granularity.

Since the problem of synthesizing observation predicates for timed games under incomplete information is out of the scope of existing synthesis tools, a relevant comparison is not possible. However, we used the tool Uppaal-TIGA version 4.1.4-0.16, which supports timed games with partial observability and fixed observations, on the problem instances constructed as explained above. For the Box Painting Production System we used the observation predicates describing the equivalence relation on the set of locations, together with the observation $0 \leq y < 1$ and for the Timed Game For Sorting Bricks we used the location observations plus the observation $0 \leq y < \frac{1}{2}$ (given also as $0 \leq y \in 1$ by scaling the constants in the model accordingly). One can see in Table 7.1 that, although on the small instances the running times are better compared to our approach, on instances where fine granularity is needed, our approach synthesizes the sufficient observation predicates considerably faster than it takes Uppaal-TIGA to solve the game with a fixed set of observation predicates.

7.4.3 Discussion

Based on the results from the previous paragraph we can make a conclusion about the importance of finding appropriate observation predicates both for the feasibility of the synthesis procedure and for the quality of the resulting controller. Clearly, the counterexample-guided search for observation predicates can be combined with techniques for a posteriori minimization of the set of observations, such as [BCD+12].

To further support our conclusion, we compared the results from our implementation also with the prototype implementation[1] of the technique [BCD+12] which in turn builds on the algorithm from [CDL+07], which is the one implemented also in Uppaal-TIGA. We ran their prototype on two of the examples above, with the sets of observations corresponding to the predicates generated by our implementation, with a minimal subset of each of this sets that suffices for controllability, and with the single predicate over the observable clock used above for the comparison with Uppaal-TIGA.

The results shown in Table 7.2 are obtained for the following sets of predicates: For *Paint-10000* we use $\mathcal{P} = \{loc = \mathsf{Sensor}, loc = \mathsf{Off}, y < 10008, y < 10011, y < 20021\}$, $\mathcal{P}_{\min} = \{loc = \mathsf{Sensor}, y < 20021\}$ and $\mathcal{P}_s = \{loc = \mathsf{Sensor}, loc = \mathsf{Off}, y < 1\}$, and for *Bricks-10000* we use $\mathcal{P} = \{loc = H, loc = L, y < 20011, y < 20012, y < 20021\}$, $\mathcal{P}_{\min} = \{loc = H, y < 20011, y < 20021\}$ and $\mathcal{P}_s = \{loc = H, y < 1\}$. The results demonstrate that albeit computationally expensive, the counterexample-guided generation of observation predicates can immensely outperform solving a timed game with badly chosen set of observation predicates. One can also see that the set of observations computed by our approach is not minimal, which is due to the choice of counterexamples. This means that the approach can clearly benefit from techniques for further minimization of the set of observations, such as for example the ones described in [BCD+12] that are implemented in the aforementioned prototype[1].

[1] `https://launchpad.net/pytigaminobs`

Chapter 8

Conclusion & Outlook

In this thesis we have studied infinite-state two-player games under incomplete information as they naturally arise in the synthesis of reactive systems with partial observation. We develop game solving methods for a number of fundamental types of infinite-state games under incomplete information, which are classified according to the resources available to the synthesized component. More precisely, this classification is based on the interface between the synthesized component and its environment, comprising the possible observations which this component can make and the actions which it controls. The methods described in this thesis are applicable to systems which are beyond the scope of other currently existing approaches to synthesis of reactive systems.

We established a generic class of infinite-state game structures with incomplete information for which games with safety and reachability winning conditions are decidable. The class is characterized by a better-quasi ordering on the set of states and the requirement that the sets of possible observations and possible actions of the synthesized system component are finite. As a particular instance, we provided symbolic algorithms for solving lossy channel games under incomplete information, where the hostile environment can nondeterministically drop messages from the communication buffers. This environment model, however, is often too conservative in practical situations. It is well known that, unfortunately, adding a fairness assumption on the channels makes the verification of eventuality properties of lossy channel systems undecidable [AJ96]. We have seen that this negative result extends to the game model considered in this thesis. One approach to circumvent this problem taken in verification [Sch04] is to consider probabilistic lossy channel systems. Recently, perfect-information games with

(generalized) Büchi objectives on probabilistic lossy channel systems have been shown to be decidable [AHdA$^+$08, BS13]. One interesting direction of future research is to investigate incomplete-information games for systems with probabilistic message losses. Another possible direction is to identify other types of systems and applications for which the synthesis problem under partial observability reduces to the game solving problem for game structures that fall into the class we established here.

In some of the target applications of synthesis it is undesirable to make the assumption that a finite set of possible observation is provided as input to the synthesis procedure. In robotics, for example, the task one might be actually interested in can be to determine a (possibly small) set of sensors and precision of these sensors that enable a robot controller to enforce the given specification. Here, as a first step, we demonstrated on a couple of examples that our abstraction-refinement based method for solving games under incomplete information can potentially be applied to such problems. In particular, modeling the given sensors as observable variables in the synthesis games allows us to use refinement to infer sufficient precision of these sensors. A promising direction of research is to explore further these ideas to develop methods to discover additional sensors that are necessary and to reduce the number of used sensors or their precision. As demonstrated by the examples, identifying the role that different observable predicates play in the abstract game is essential for determining if they are necessary for the controller. A further possibility in this direction can be to combine counterexample analysis with domain knowledge that might aid the sensor discovery.

In the real-time setting, the set of generated observation predicates also plays a crucial role in the quality of the synthesized controller. Developing methods to minimize the set of generated observation predicates during the refinement phase will allow for reducing the size and number of clock constraints of the resulting controller. Additionally, the application of such techniques will increase the practicability of our approach. An important criterion for the quality of a real-time controller strategy is its robustness [CHP08, SBMR13]. Since our approach tightly relates the time delays chosen by the controller (i.e., the times when the controller makes a discrete transition) and the observation predicates, it will be interesting to investigate the connection between the robustness of control strategies and the observation predicates and derive synthesis methods that guarantee some notion of robustness of their result.

References

[AAB99] Parosh Aziz Abdulla, Aurore Annichini, and Ahmed Bouajjani, *Symbolic verification of lossy channel systems: An application to the bounded retransmission protocol*, Proc. TACAS '99-5th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems, Lecture Notes in Computer Science, vol. 1579, 1999.

[ABd08] Parosh Aziz Abdulla, Ahmed Bouajjani, and Julien d'Orso, *Monotonic and downward closed games*, J. Log. Comput. **18** (2008), no. 1, 153–169.

[Abd10] Parosh Aziz Abdulla, *Well (and better) quasi-ordered transition systems*, Bulletin of Symbolic Logic **16** (2010), no. 4, 457–515.

[ABJ98] Parosh Aziz Abdulla, Ahmed Bouajjani, and Bengt Jonsson, *On-the-fly analysis of systems with unbounded, lossy FIFO channels*, Proc. 10th Int. Conf. on Computer Aided Verification, Lecture Notes in Computer Science, vol. 1427, 1998, pp. 305–318.

[ACABJ04] Parosh Aziz Abdulla, Aurore Collomb-Annichini, Ahmed Bouajjani, and Bengt Jonsson, *Using forward reachability analysis for verification of lossy channel systems*, FMSD **25** (2004), no. 1, 39–65.

[ACMS13] Parosh Aziz Abdulla, Lorenzo Clemente, Richard Mayr, and Sven Sandberg, *Stochastic parity games on lossy channel systems*, CoRR **abs/1305.5228** (2013).

[AD94] Rajeev Alur and David L. Dill, *A theory of timed automata*, Theoretical Computer Science **126** (1994), no. 2, 183–235.

REFERENCES

[AHdA+08] Parosh Aziz Abdulla, Noomene Ben Henda, Luca de Alfaro, Richard Mayr, and Sven Sandberg, *Stochastic games with lossy channels*, FoSSaCS (Roberto M. Amadio, ed.), Lecture Notes in Computer Science, vol. 4962, Springer, 2008, pp. 35–49.

[AHKV98] Rajeev Alur, Thomas A. Henzinger, Orna Kupferman, and Moshe Y. Vardi, *Alternating refinement relations*, In Proceedings of the Ninth International Conference on Concurrency Theory (CONCUR'98), volume 1466 of LNCS, Springer-Verlag, 1998, pp. 163–178.

[AJ93] Parosh Aziz Abdulla and Bengt Jonsson, *Verifying programs with unreliable channels*, Proc. LICS '93-8th IEEE Int. Symp. on Logic in Computer Science, 1993, pp. 160–170.

[AJ96] ———, *Undecidable verification problems for programs with unreliable channels*, Inf. Comput. **130** (1996), no. 1, 71–90.

[BBS06] Christel Baier, Nathalie Bertrand, and Philippe Schnoebelen, *On computing fixpoints in well-structured regular model checking, with applications to lossy channel systems*, Proceedings of the 13th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'06), Lecture Notes in Artificial Intelligence, vol. 4246, Springer, November 2006, pp. 347–361.

[BC06] Patricia Bouyer and Fabrice Chevalier, *On the control of timed and hybrid systems*, Bulletin of the EATCS **89** (2006), 79–96.

[BCD+12] Peter Bulychev, Franck Cassez, Alexandre David, Kim Guldstrand Larsen, Jean-François Raskin, and Pierre-Alain Reynier, *Controllers with minimal observation power (application to timed systems)*, Proc. ATVA, LNCS, vol. 7561, Springer, 2012.

[BCW+09] Dietmar Berwanger, Krishnendu Chatterjee, Martin De Wulf, Laurent Doyen, and Thomas A. Henzinger, *Alpaga: A tool for solving parity games with imperfect information*, TACAS, LNCS, vol. 5505, Springer, 2009, pp. 58–61.

[BDFW07] Ingo Brückner, Klaus Dräger, Bernd Finkbeiner, and Heike Wehrheim, *Slicing abstractions*, FSEN (Farhad Arbab and Marjan Sirjani, eds.), Lecture Notes in Computer Science, vol. 4767, Springer, 2007, pp. 17–32.

[BDMP03] Patricia Bouyer, Deepak D'Souza, P. Madhusudan, and Antoine Petit, *Timed control with partial observability*, Proc. CAV'03, LNCS, vol. 2725, Springer, 2003.

[BK06] Thomas Ball and Orna Kupferman, *An abstraction-refinement framework for multi-agent systems*, Proc. LICS, IEEE Computer Society, 2006, pp. 379–388.

[BMMR01] Thomas Ball, Rupak Majumdar, Todd Millstein, and Sriram K. Rajamani, *Automatic predicate abstraction of c programs*, Proceedings of the ACM SIGPLAN 2001 conference on Programming language design and implementation (New York, NY, USA), PLDI '01, ACM, 2001, pp. 203–213.

[BS13] Nathalie Bertrand and Philippe Schnoebelen, *Solving stochastic büchi games on infinite arenas with a finite attractor*, Proceedings of the 11th International Workshop on Quantitative Aspects of Programming Languages (QAPl'13) (Luca Bortolussi and Herbert Wiklicky, eds.), Electronic Proceedings in Theoretical Computer Science, vol. 117, June 2013, pp. 116–131.

[BT07] Clark Barrett and Cesare Tinelli, *CVC3*, Proceedings of the 19^{th} International Conference on Computer Aided Verification (CAV '07) (Werner Damm and Holger Hermanns, eds.), Lecture Notes in Computer Science, vol. 4590, Springer-Verlag, July 2007, Berlin, Germany, pp. 298–302.

[Cac02] Thierry Cachat, *Symbolic strategy synthesis for games on pushdown graphs*, Proceedings of the 29th International Colloquium on Automata, Languages and Programming (London, UK, UK), ICALP '02, Springer-Verlag, 2002, pp. 704–715.

[Cas07] Franck Cassez, *Efficient on-the-fly algorithms for partially observable timed games*, Proc. FORMATS'07, LNCS, vol. 4763, Springer, 2007.

REFERENCES

[CC00] Patrick Cousot and Radhia Cousot, *Temporal abstract interpretation*, POPL '00: Proceedings of the 27th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (New York, NY, USA), ACM Press, 2000, pp. 12–25.

[CCGS03] Sagar Chaki, Edmund Clarke, Alex Groce, and Ofer Strichman, *Predicate abstraction with minimum predicates*, Proceedings of 12th Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME), 2003.

[CCH+11] Pavol Cerný, Krishnendu Chatterjee, Thomas A. Henzinger, Arjun Radhakrishna, and Rohit Singh, *Quantitative synthesis for concurrent programs*, CAV, Lecture Notes in Computer Science, vol. 6806, Springer, 2011, pp. 243–259.

[CDHR06] K. Chatterjee, L. Doyen, T. A. Henzinger, and J.-F. Raskin, *Algorithms for omega-regular games of incomplete information*, Proc. CSL'06, LNCS, vol. 4207, Springer, 2006.

[CDL+07] Franck Cassez, Alexandre David, Kim Guldstrand Larsen, Didier Lime, and Jean-François Raskin, *Timed control with observation based and stuttering invariant strategies*, Proc. ATVA'07, LNCS, vol. 4762, Springer, 2007.

[CE81] Edmund M. Clarke and E. Allen Emerson, *Design and synthesis of synchronization skeletons using branching-time temporal logic*, Logic of Programs (Dexter Kozen, ed.), Lecture Notes in Computer Science, vol. 131, Springer, 1981, pp. 52–71.

[CGJ+00] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith, *Counterexample-guided abstraction refinement*, Proc. CAV'00, LNCS, vol. 1855, Springer, 2000, pp. 154–169.

[CGL94] Edmund M. Clarke, Orna Grumberg, and David E. Long, *Model checking and abstraction*, ACM Transactions on Programming Languages and Systems **16** (1994), no. 5, 1512–1542.

[CHP08] Krishnendu Chatterjee, Thomas A. Henzinger, and Vinayak S. Prabhu, *Timed parity games: Complexity and robustness*, FORMATS (Franck Cassez and Claude Jard, eds.), Lecture Notes in Computer Science, vol. 5215, Springer, 2008, pp. 124–140.

[Chu63] Alonzo Church, *Logic, arithmetic, and automata*, Proceedings of the International Congress of Mathematicians (ICM'62), 1963, pp. 23–35.

[CKLB11] Chih-Hong Cheng, Alois Knoll, Michael Luttenberger, and Christian Buckl, *GAVS+: an open platform for the research of algorithmic game solving*, Proceedings of the 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'11), LNCS, Springer, 2011.

[Cra57] William Craig, *Linear reasoning. a new form of the herbrand-gentzen theorem*, J. Symb. Log. **22** (1957), no. 3, 250–268.

[CRKB11] Chih-Hong Cheng, Harald Rueß, Alois Knoll, and Christian Buckl, *Synthesis of fault-tolerant embedded systems using games: From theory to practice*, VMCAI, Lecture Notes in Computer Science, vol. 6538, Springer, 2011, pp. 118–133.

[dAGJ04] Luca de Alfaro, Patrice Godefroid, and Radha Jagadeesan, *Three-valued abstractions of games: Uncertainty, but with precision*, Proc. LICS'04, IEEE Computer Society, 2004.

[dAR07] Luca de Alfaro and Pritam Roy, *Solving games via three-valued abstraction refinement*, Proc. CONCUR'07, LNCS, vol. 4703, Springer, 2007, pp. 74–89.

[DD02] Satyaki Das and David L. Dill, *Counter-example based predicate discovery in predicate abstraction*, Proc. FMCAD'02, Springer-Verlag, 2002, pp. 19–32.

[De 06] Martin De Wulf, *From timed models to timed implementations*, Thèse de doctorat, Département d'Informatique, Université Libre de Bruxelles, Belgium, December 2006.

[DF08] Rayna Dimitrova and Bernd Finkbeiner, *Abstraction refinement for games with incomplete information*, IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS), LIPIcs, vol. 2, Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2008.

[DF12] ———, *Counterexample-guided synthesis of observation predicates*, 10th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS) (Marcin Jurdzinski and Dejan Nickovic, eds.), LNCS, vol. 7595, Springer, 2012, pp. 107–122.

[DF13] ———, *Lossy channel games under incomplete information*, 1st International Workshop on Strategic Reasoning (SR) (Fabio Mogavero, Aniello Muran, and Moshe Y. Vardi, eds.), EPTCS, vol. 112, 2013, pp. 43–51.

[DGG97] Dennis Dams, Rob Gerth, and Orna Grumberg, *Abstract interpretation of reactive systems*, ACM Trans. Program. Lang. Syst. **19** (1997), no. 2, 253–291.

[DKFW10] Klaus Dräger, Andrey Kupriyanov, Bernd Finkbeiner, and Heike Wehrheim, *Slab: A certifying model checker for infinite-state concurrent systems*, Proceedings of the 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Lecture Notes in Computer Science, Springer-Verlag, 2010.

[dMB08] Leonardo Mendonça de Moura and Nikolaj Bjørner, *Z3: An efficient smt solver*, TACAS (C. R. Ramakrishnan and Jakob Rehof, eds.), Lecture Notes in Computer Science, vol. 4963, Springer, 2008, pp. 337–340.

[DR11] L. Doyen and J.-F. Raskin, *Games with imperfect information: Theory and algorithms*, Lectures in Game Theory for Computer Scientists, 2011, pp. 185–212.

[DWDR06] M. De Wulf, L. Doyen, and J.-F. Raskin, *A lattice theory for solving games of imperfect information*, Proc. HSCC'06, LNCS, Springer, 2006, pp. 153–168.

[FP12] Bernd Finkbeiner and Hans-Jörg Peter, *Template-based controller synthesis for timed systems*, TACAS, LNCS, vol. 7214, Springer, 2012, pp. 392–406.

[FS01] Alain Finkel and Ph. Schnoebelen, *Well-structured transition systems everywhere!*, Theor. Comput. Sci. **256** (2001), no. 1-2, 63–92.

[GBC06] Andreas Griesmayer, Roderick Bloem, and Byron Cook, *Repair of boolean programs with an application to c*, CAV, LNCS, vol. 4144, Springer, 2006, pp. 358–371.

[GLP] *GNU Linear Programming Kit*, `http://www.gnu.org/software/glpk`.

[GRV04] G. Geeraerts, J.-F. Raskin, and L. Van Begin, *Expand, Enlarge and Check: new algorithms for the coverability problem of WSTS*, Proceedings of FSTTCS'04, 24th International Conference on Foundations of Software Technology and Theoretical Computer Science, Chennai, India (Kamal Lodoya and Meena Mahajan, eds.), Lecture Notes in Computer Science, vol. 3328, Springer-Verlag, 2004, pp. 287–298.

[GRV05] _____, *Expand, enlarge and check... made efficient*, Lecture Notes in Computer Science, no. 3576, Springer Verlag, 2005, pp. 394–404.

[HJM03] T.A. Henzinger, R. Jhala, and R. Majumdar, *Counterexample-guided control*, Proc. ICALP'03, LNCS, vol. 2719, Springer, 2003, pp. 886–902.

[HJMM04] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Kenneth L. McMillan, *Abstractions from proofs*, Proc. POPL'04 (New York, NY, USA), ACM Press, 2004, pp. 232–244.

[HJMS03] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Grégoire Sutre, *Software verification with blast*, Model Checking Software, 10th International SPIN Workshop. Portland, OR, USA, May 9-10, 2003, Proceedings (Thomas Ball and Sriram K. Rajamani, eds.), Lecture Notes in Computer Science, vol. 2648, Springer, 2003, pp. 235–239.

[HJS01] Michael Huth, Radha Jagadeesan, and David A. Schmidt, *Modal transition systems: A foundation for three-valued program analysis*, ESOP'01 (David Sands, ed.), Lecture Notes in Computer Science, vol. 2028, Springer, 2001, pp. 155–169.

[HMMR00] Thomas A. Henzinger, Rupak Majumdar, Freddy Y. C. Mang, and Jean-François Raskin, *Abstract interpretation of game properties*, Proc. SAS'00, Springer-Verlag, 2000, pp. 220–239.

[JGB05] Barbara Jobstmann, Andreas Griesmayer, and Roderick Bloem, *Program repair as a game*, CAV, 2005, pp. 226–238.

[JM06] Ranjit Jhala and Kenneth L. McMillan, *A practical and complete approach to predicate refinement*, Proc. TACAS'06, vol. 3920, Springer-Verlag, 2006, pp. 459–473.

[JSGB12] Barbara Jobstmann, Stefan Staber, Andreas Griesmayer, and Roderick Bloem, *Finding and fixing faults*, J. Comput. Syst. Sci. **78** (2012), no. 2, 441–460.

[KG05] R. Kumar and V.K. Garg, *On computation of state avoidance control for infinite state systems in assignment program framework*, Automation Science and Engineering, IEEE Transactions on **2** (2005), no. 1, 87–91.

[KGFP09] Hadas Kress-Gazit, Georgios E. Fainekos, and George J. Pappas, *Temporal-logic-based reactive mission and motion planning*, IEEE Transactions on Robotics **25** (2009), no. 6, 1370–1381.

[KGM93] Ratnesh Kumar, Vijay Garg, and Steven I. Marcus, *Predicates and predicate transformers for supervisory control of discrete event dynamical systems*, IEEE Transactions on Automatic Control **38** (1993), 232–247.

[KGMM09] Gabriel Kalyon, Tristan Le Gall, Hervé Marchand, and Thierry Massart, *Control of infinite symbolic transition systems under partial observation*, European Control Conference, 2009, pp. 1456–1462.

[KGMM12] ———, *Symbolic supervisory control of infinite transition systems under partial observation using abstract interpretation*, Discrete Event Dynamic Systems **22** (2012), no. 2, 121–161.

[KV97] O. Kupferman and M.Y. Vardi, *Synthesis with incomplete informatio*, 2nd International Conference on Temporal Logic (Manchester), July 1997, pp. 91–106.

[Lam74] Leslie Lamport, *A new solution of dijkstra's concurrent programming problem*, Commun. ACM **17** (1974), no. 8, 453–455.

[LBBO01] Y. Lakhnech, S. Bensalem, S. Berezin, and S. Owre, *Incremental verification by abstraction*, Lecture Notes in Computer Science **2031** (2001), 98+.

[Mar75] Donald A. Martin, *Borel determinacy*, Annals of Mathematics **102** (1975), no. 2, 363–371.

[Mar01] Alberto Marcone, *Fine analysis of the quasi-orderings on the power set*, Order **18** (2001), no. 4, 339–347.

[McM06] Kenneth L. McMillan, *Lazy abstraction with interpolants*, CAV (Thomas Ball and Robert B. Jones, eds.), Lecture Notes in Computer Science, vol. 4144, Springer, 2006, pp. 123–136.

[Mil85] E. C. Milner, *Basic wqo- and bqo-theory*, Graphs and order, 1985, pp. 487–502.

[MP96] Z Manna and Amir Pnueli, *Clocked transition systems*, Tech. report, Stanford, CA, USA, 1996.

[MPS95] Oded Maler, Amir Pnueli, and Joseph Sifakis, *On the synthesis of discrete controllers for timed systems*, Proc. STACS'95, LNCS, vol. 900, Springer, 1995.

[MW80] Zohar Manna and Richard Waldinger, *A deductive approach to program synthesis*, ACM Trans. Program. Lang. Syst. **2** (1980), no. 1, 90–121.

REFERENCES

[Nas65] C. Nash-Williams, *On well-quasi ordering infinite trees*, Proceedings of the Cambridge Philosophical Society **61** (1965), 697–720.

[PAHSv02] Roberto Passerone, Luca De Alfaro, Thomas A. Henzinger, and Alberto L. Sangiovanni-vincentelli, *Convertibility verification and converter synthesis: Two faces of the same coin*, In International Conference on Computer Aided Design ICCAD, ACM, 2002.

[PPS06] Nir Piterman, Amir Pnueli, and Yaniv Sa'ar, *Synthesis of reactive(1) designs*, VMCAI (E. Allen Emerson and Kedar S. Namjoshi, eds.), Lecture Notes in Computer Science, vol. 3855, Springer, 2006, pp. 364–380.

[PR89a] Amir Pnueli and Roni Rosner, *On the synthesis of a reactive module*, POPL (New York, NY, USA), ACM Press, 1989, pp. 179–190.

[PR89b] Amir Pnueli and Roni Rosner, *On the synthesis of an asynchronous reactive module*, Proceedings of the 16th International Colloquium on Automata, Languages and Programming (London, UK, UK), ICALP '89, Springer-Verlag, 1989, pp. 652–671.

[PR90] Amir Pnueli and Roni Rosner, *Distributed reactive systems are hard to synthesize*, FOCS'90, IEEE Computer Society, 1990, pp. 746–757.

[PR07] Andreas Podelski and Andrey Rybalchenko, *Armc: The logical choice for software model checking with abstraction refinement*, PADL (Michael Hanus, ed.), Lecture Notes in Computer Science, vol. 4354, Springer, 2007, pp. 245–259.

[PRSV98] Roberto Passerone, James A. Rowson, and Alberto Sangiovanni-Vincentelli, *Automatic synthesis of interfaces between incompatible protocols*, Proceedings of the 35th annual Design Automation Conference (New York, NY, USA), DAC '98, ACM, 1998, pp. 8–13.

[Puc10] Bernd Puchala, *Asynchronous omega-regular games with partial information*, Proc. MFCS, LNCS, vol. 6281, Springer, 2010, pp. 592–603.

[Pud97] Pavel Pudlák, *Lower bounds for resolution and cutting plane proofs and monotone computations*, J. Symb. Log. **62** (1997), no. 3, 981–998.

[Rei84] John H. Reif, *The complexity of two-player games of incomplete information*, J. Comput. Syst. Sci. **29** (1984), no. 2, 274–301.

[RSS07] Andrey Rybalchenko and Viorica Sofronie-Stokkermans, *Constraint solving for interpolation*, Proc. VMCAI'07, LNCS, vol. 4349, Springer-Verlag, 2007, pp. 346–362.

[RW89] Peter J. Ramadge and W. Murray Wonham, *The control of discrete event systems*, Proceedings of the IEEE **77** (1989), no. 1, 81–98.

[SBMR13] Ocan Sankur, Patricia Bouyer, Nicolas Markey, and Pierre-Alain Reynier, *Robust controller synthesis in timed automata*, CONCUR (Pedro R. D'Argenio and Hernán C. Melgratti, eds.), Lecture Notes in Computer Science, vol. 8052, Springer, 2013, pp. 546–560.

[Sch86] Alexander Schrijver, *Theory of linear and integer programming*, John Wiley & Sons, Inc., New York, NY, USA, 1986.

[Sch04] Ph. Schnoebelen, *The verification of probabilistic lossy channel systems*, Validation of Stochastic Systems (Christel Baier, Boudewijn R. Haverkort, Holger Hermanns, Joost-Pieter Katoen, and Markus Siegle, eds.), Lecture Notes in Computer Science, vol. 2925, Springer, 2004, pp. 445–466.

[SF06] Sven Schewe and Bernd Finkbeiner, *Synthesis of asynchronous systems*, 16th International Symposium on Logic Based Program Synthesis and Transformation (LOPSTR 2006), Springer Verlag, 2006, pp. 127–142.

[SF07] ———, *Bounded synthesis*, ATVA (Kedar S. Namjoshi, Tomohiro Yoneda, Teruo Higashino, and Yoshio Okamura, eds.), Lecture Notes in Computer Science, vol. 4762, Springer, 2007, pp. 474–488.

[SG04] Sharon Shoham and Orna Grumberg, *Monotonic abstraction-refinement for ctl*, TACAS'04 (Kurt Jensen and Andreas Podelski, eds.), Lecture Notes in Computer Science, vol. 2988, Springer, 2004, pp. 546–560.

[SMA10] *The SMACS Tool*, `http://www.smacs.be`, 2010.

[Som09] Fabio Somenzi, *CUDD: CU decision diagram package, release 2.4.2*, University of Colorado at Boulder, 2009.

[Tri04] Stavros Tripakis, *Undecidable problems of decentralized observation and control on regular languages*, Inf. Process. Lett. **90** (2004), no. 1, 21–28.

[vEJ13] Christian von Essen and Barbara Jobstmann, *Program repair without regret*, Computer Aided Verification (CAV) (Springer, ed.), 2013, To appear.

[VYY09] Martin T. Vechev, Eran Yahav, and Greta Yorsh, *Inferring synchronization under limited observability*, TACAS (Stefan Kowalewski and Anna Philippou, eds.), Lecture Notes in Computer Science, vol. 5505, Springer, 2009, pp. 139–154.

[WTM10] Tichakorn Wongpiromsarn, Ufuk Topcu, and Richard M. Murray, *Receding horizon control for temporal logic specifications*, HSCC (Karl Henrik Johansson and Wang Yi, eds.), ACM ACM, 2010, pp. 101–110.

[YM05] Greta Yorsh and Madanlal Musuvathi, *A combination method for generating interpolants*, CADE, LNCS, vol. 3632, Springer, 2005, pp. 353–368.