Pushing the Barriers in Controller Synthesis for Cyber-Physical Systems

Vom Fachbereich Informatik der
Technischen Universität Kaiserslautern
zur Verleihung des akademischen Grades
Doktor der Naturwissenschaften (Dr. rer. nat.)

genehmigte Dissertation

von

Kaushik Mallik

Datum der wissenschaftlichen Aussprache	24.06.2022
Dekan	Prof. Dr. Jens Schmitt
Berichterstatter	Prof. Dr. Rupak Majumdar
	Prof. Dr. Necmiye Ozay
	Prof. Dr. Sadegh Soudjani

DE-386

Summary

Controller design for continuous dynamical systems is a core algorithmic problem in the design of cyber-physical systems (CPS). When the CPS application is safety critical, additionally we require the controller to have strong correctness guarantees. One approach for this design problem is to use simpler discrete abstraction of the original continuous system, on which known reactive synthesis methods can be used to design the controller. This approach is known as the abstraction-based controller design (ABCD) paradigm.

In this thesis, we build ABCD procedures which are faster and more modular compared to the state-of-the-art, and can handle problems which were beyond the scope of the existing techniques.

Usually, existing ABCD approaches use state space discretization for computing the abstractions, for which the procedures do not scale well for larger systems. Our first contribution is a *multi-layered* ABCD algorithm, where we combine coarse abstractions and *lazily* computed fine abstractions to improve scalability. So far, we only address reach-avoid and safety specifications, for which our prototype tool (called `Mascot`) showed up to an order of magnitude speedup on standard benchmark examples.

Second, we consider the problem of *modular* design of sound *local* controllers for a network of local discrete abstractions communicating via discrete/boolean variables and having local specifications. We propose a sound algorithm, where the systems *negotiate* a pair of local *assume-guarantee contracts*, in order to synchronize on a set of non-conflicting and correct behaviors. As a by-product, we also obtain a set of local controllers for the systems which ensure simultaneous satisfaction of the local specifications. We show the effectiveness of the our algorithm using a prototype tool (called `Agnes`) on a set of discrete benchmark examples.

Our third contribution is a novel ABCD algorithm for a more expressive model of nonlinear dynamical systems with *stochastic* disturbances and ω-regular specifications. This part has two subparts, which are of significant merits on their own rights. First, we present an abstraction algorithm for nonlinear stochastic systems using $2^1/_2$-player games (turn-based stochastic graph games). We show that an almost sure winning strategy in this abstract $2^1/_2$-player game gives us a sound controller for the original system for satisfying the specification with probability one. Second, we present symbolic algorithms for a seemingly different class of 2-player games with certain environmental fairness assumptions, which can also be used to efficiently compute winning strategies in the aforementioned abstract $2^1/_2$-player game. Using our prototype tool (`Mascot-SDS`), we show that our algorithm significantly outperforms the state-of-the-art implementation on standard benchmark examples from the literature.

Zusammenfassung

Ein zentrales algorithmisches Problem beim Entwurf sicherheitskritischer cyber-physikalischer Systeme (CPS) ist die vollautomatische Synthese von garantiert korrekter Regelsoftware für Systemkomponenten mit kontinuierlicher Dynamik und diskreten Spezifikationen. Ein vielversprechender Ansatz um dieses Problem zu lösen ist der sogenannte abstraktionsbasierte Reglerentwurf (engl. Abstraction-Based Control Design (ABCD)). In ABCD wird die kontinuierliche Systemdynamik zunächst durch diskrete Abstraktionen approximiert. Auf Basis dieser Abstraktionen wird dann mit Hilfe von Algorithmen aus der (diskreten) reaktiven Synthese ein korrekter Regler für die gegebene diskrete Spezifikation synthetisiert, welcher dann in einen Regler für das Orginalproblem verfeinert werden kann.

In dieser Arbeit befassen wir uns im Wesentlichen mit zwei Unzulänglichkeiten von ABCD. Zum einen nehmen wir uns der Herausforderungen bei der Skalierbarkeit von ABCD-Techniken an. Zum anderen erweitern wir ABCD auf ausdrucksstärkere Modelle, welche immer noch eine Synthese bezüglich ω-regulärer Spezifikationen zulassen. Insbesondere schlagen wir eine neue ABCD-Technik für kontinuierliche nichtlineare dynamische Systeme mit stochastischen Störungen vor. Unser Beitrag zu Skalierbarkeit und Expressivität von ABCD ist in drei Hauptteile gegliedert.

Unser erster Beitrag ist ein mehrschichtiger ABCD-Algorithmus für verbesserte Skalierbarkeit. Viele bestehende Ansätze verwenden eine Diskretisierung des Zustandsraums zur Berechnung der Abstraktion, was einen schwerwiegenden rechnerischen Engpass darstellt. In unserem mehrschichtigen Ansatz kombinieren wir grobe Abstraktionen und lokal berechnete feine Abstraktionen, um die Skalierbarkeit zu verbessern. Bislang haben wir diese Methode nur für eingeschränkten Spezifikationsklassen entwickelt, welche in einem Prototyp (genannt `Mascot`) implementiert sind. Im Vergleich zu anderen ABCD-Algorithmen ist `Mascot` bei der Reglerberechnung für standardisierte Benchmark-Beispiele bis zu eine Größenordnung schneller.

Der zweite Beitrag dient ebenfalls der besseren Skalierbarkeit, und schlägt einen Algorithmus für das (diskrete) verteilte reaktive Syntheseproblem vor. Dieser Algorithmus kann für den skalierbaren und modularen Entwurf lokaler Regler in einem Netzwerk lokaler diskreter Abstraktionen, die über diskrete/boolesche Variablen kommunizieren und lokale Spezifikationen haben, verwendet werden. Wir schlagen einen Algorithmus vor, bei dem die abstrakten Systeme lokal und dezentral Verträge untereinander aushandeln, und sich damit auf eine Reihe von konfliktfreien und korrekten Verhaltensweisen einigen. Wir zeigen die Anwendbarkeit unseres Algorithmus anhand eines Prototyps (genannt `Agnes`) an einer Reihe von diskreten Benchmark-Beispielen.

Der dritte Beitrag ist, wie bereits erwähnt, ein neuartiger ABCD-Algorithmus für nichtlinearer dynamischer Systeme mit stochastischen Störungen und ω-regulären Spezifikationen. Dieser Teil hat zwei Unterabschnitte, die für sich genommen von großem Wert sind. Zuerst stellen wir einen Abstraktionsalgorithmus für nichtlineare stochastische Systeme unter Verwendung von $2^1/_2$-Spieler-Spielen (rundenbasierte stochastische Graphenspiele) vor. Wir zeigen, dass eine Gewinnstrategie in diesem abstrakten $2^1/_2$-Spieler-Spiel in einen korrekten Regler für das Orginalsystem verfeinert werden kann. Danach entwickeln wir symbolische Algorithmen für eine scheinbar andere Klasse von 2-Spieler-Spielen mit bestimmten Fairness-Annahmen. Diese Algorithmen können allerdings auch zur effizienten Berechnung von Gewinnstrategien in dem oben erwähnten abstrakten $2^1/_2$-Spiel verwendet werden. Mit Hilfe unseres Prototyps (`Mascot-SDS`) zeigen wir, dass die Kombination beider Algorithmen den Stand der Technik deutlich übertrifft.

To *Jethu* (my uncle) . . .

Contents

Acknowledgments

This thesis is a result of many fruitful academic collaborations on a scientific level. Equally important was the support from my near and dear ones, who at the same time made the journey sustainable and enjoyable. I will try to convey my gratitude to all concerned, though it is beyond my capacity to do a complete justice in a short note like this one. I sincerely apologize in case I inadvertently missed anyone's name in this acknowledgement.

First and foremost, I would like to thank my advisor Rupak Majumdar. He has always been approachable and friendly, gave advice whenever I needed, comforted me with assurance whenever I met a failure, helped me shape my career path, and above all inspired me in every aspect of academia and many aspects of life while at MPI. I will cherish our long and engaging scientific discussions (which never felt long!), as well as our fun social interactions.

The other person to whom I am greatly indebted is Anne-Kathrin Schmuck. Had I not met her, I would most likely not end up in MPI. She is also the person who mentored me closely from the beginning, and with whom I spent most number of brainstorming hours during my PhD.

I was fortunate to have Daniel Neider, Sadegh Soudjani, and Damien Zufferey as mentors and collaborators. They played a crucial role in shaping the course of my PhD projects, and helped me and advised me whenever I needed.

I had the privilege to have excellent collaborators, including all the talented interns and master students who worked in our projects; thanks to all of them!

Sincere thanks to the thesis reviewers Rupak Majumdar, Necmiye Ozay, and Sadegh Soudjani for taking time to review the thesis, and the PhD committee members Rupak Majumdar, Anthony Lin (chair), and Klaus Schneider for smoothly conducting the thesis evaluation process.

Huge thanks to the amazing administrative staffs of MPI in Kaiserslautern, including Corinna, Geraldine, Mouna, Roslyn, Susanne, and Vera, and the super-efficient IT staffs, including Christian, Pascal, and Tobias, who made the non-scientific matters so easy and seamless that they never became any issue to be stressed about. Not only that, Corinna, Susanne, and Vera even helped me getting accustomed to the social life in Germany and were immensely helpful in settling down as a foreigner.

I am grateful to Mary-Lou, who not only helped wading through the administrative requirements of the university, but also showered encouragement on every occasion we met.

Many thanks to Rose, who taught us crucial skills for writing beautiful scientific texts and delivering fluent talks. I tried to implement her lessons in this thesis as well as I could!

The five odd years that I spent in Kaiserslautern have been the best five years of my life so far. The credit goes to the friends I made. With them, I spent a significant time of my leisure and created many prized memories of travel, sports, eat-outs, and fun experiences. In alphabetical order, they include: Abir, Aman, Amir, Andreea, Ana, Anne, Anthony, Arka, Arpan, Ashwani, Azalea, Burcu, Clothilde, Damien, Daniel, Dmitry, Eirene, Eleni,

Felix, Filip, Hasan, Iason, Isa, Ivan F., Ivan G., James, Johannes, Kata, Klara, Kyle, Laura, Leo, Lovro, Mahmoud, Manohar, Manuel, Marco M., Marco P., Marcus, Marko D., Marko H., Mateusz, Maziar, Mehrdad, Mia, Michalis, Mitra, Murat, Nastaran, Nina, Numair, Ori, Rajarshi, Ram, Rayna, Rosa, Sadegh, Satya, Simin, Soham, Stanly, Stratis, Utkarsh, Vinayak, Xuan, and Yunjun.

I would like to particularly mention Amir, Maziar, Michalis, Nastaran, and Rosa, because of whom I managed to effortlessly push through the difficult Covid-times. At a time when social life was almost nonexistent, with them I felt a deep sense of togetherness, helping to maintain balance and sanity in life.

I am eternally grateful to my parents for making me who I am, and to my sister for her love and support all along. They are my inner strengths.

I feel lucky that I shared a part of my life with Sohini. She offered unconditional help and support throughout, and with her I had the chance to create some of the best memories during my time in Germany.

Big thanks to all other family members, friends, and well-wishers whose encouragements and excitements meant a lot.

This thesis is dedicated to my uncle, who was himself a professor, a great scholar, a very respected educator, and a charming person. He was one of the biggest inspirations in my life for pursuing engineering studies followed by a career path in academia. He was probably the biggest patron of this thesis as well. He succumbed to Covid, before the thesis was finished.

1. Introduction

1.1. Background and Motivation

Throughout the last decade, we have seen great technological advancements in Cyber-Physical Systems (CPS) research. We have seen accomplishments like self-driving cars (Reese) and trucks (Kottasová) taking to the road, humanoid robots performing back-flips (Miller), and drones doing surveillance (Bisht). At the same time, we have also experienced the catastrophic effects of machines failing to operate safely, thereby putting human lives at risk. We have seen incidents like autonomous vehicles causing fatal accidents (Price), aircraft crashes costing hundreds of human lives (Gelles), and industrial robot accidentally crushing human operator to death (Associated Press).

As a result, providing strong correctness guarantee in modern CPS applications has become a problem of great importance. The general high-level goal is to formally certify that the machines only do what they are supposed to do. For instance, we want to certify that a self-driving car will always brake whenever there is an obstacle in the front, or that a flight control system will always perform a safe landing maneuver, or that an industrial robot will never act outside its safe operating protocols, etc.

Modern CPS applications, such as the ones mentioned above, operate through extremely complex orchestration of many dynamical and software components. In this thesis, we address the problem of designing verified *control software*, which is a central aspect in providing correctness guarantees in CPS.

At the heart of the verified control software design problem lies the core problem of synthesizing formally verified feedback controllers for nonlinear or hybrid dynamical systems. At its bare bones, a dynamical system is characterized by a set of states and inputs whose values change over time. The temporal change of state depends on its initial value and the input sequence given to the system. (The temporal change is usually modeled using a differential or a difference equation implicitly.) A (feedback) controller is a device that looks at the current state of the system and decides which input to pick, thereby influencing the state of the system in immediate future. The controller synthesis problem asks to find a controller such that the evolution of the state satisfies some given *specification*.

For instance, in a model of an autonomous vehicle, position and velocity will be part of the state, and throttle and steering angle will be the inputs. A controller for the vehicle needs to essentially mimic actions of a rational human driver by deciding values for the inputs based on the current state. A typical controller synthesis problem in this case could be: find a controller which would drive the vehicle to a selected destination while avoiding collisions on the road.

Traditionally, controller synthesis would often involve either an iterative trial-and-error search (as in PID tuning) or solving an optimization problem (as in the LQR problem). Unfortunately, the traditional approaches turn out to be inadequate for many CPS applications due to various reasons. First, they have very restricted support for perturbed nonlinear and hybrid dynamical systems with logical temporal specifications, as needed by modern CPS applications. Second, for complex and large systems, they either become infeasible or give up on the strong correctness guarantee.

As a promising alternative, recently, there have been a flurry of automated, correct-by-construction controller design techniques that on one hand support a broader class of systems and specifications, and on the other hand give us correct-by-design controllers in an automated way (Pola et al., 2008; Girard et al., 2010; Tabuada, 2009; Nilsson et al., 2017; Reissig et al., 2017; Prajna and Jadbabaie, 2004; Wieland and Allgöwer, 2007; Ravanbakhsh and Sankaranarayanan, 2017; Ames et al., 2019; Yang et al., 2020; Henrion and Korda, 2014; Chen et al., 2020; Bansal et al., 2017). We[1] build on a particular class of correct-by-construction techniques called *abstraction-based controller design (ABCD)*, which has found success in many areas of CPS in the last two decades (Tabuada, 2009; Nilsson et al., 2017; Reissig et al., 2017).

In ABCD, the "classical control" techniques for control of continuous systems are combined with the automata-theoretic techniques arising out of the control of discrete systems. The key to this combination is *abstraction*: a way to discretize and simplify the continuous dynamics so that one can apply automata-theoretic techniques on the discretization while maintaining a refinement back to the original continuous system. In this approach, a time-sampled version of the continuous dynamics, called the *concrete system*, is *abstracted* by a symbolic finite state model, called the *abstract system*. After that, automata-theoretic algorithms from *reactive synthesis* are used to synthesize a discrete abstract controller on the abstract system for a given temporal logic specification. The abstract controller can then be transferred to the sought *concrete controller* for the original concrete system via a mechanism called *refinement*. The correctness of the concrete controller will follow from the correctness of the reactive synthesis algorithms and a certain *feedback refinement relation* (FRR) between the concrete and the abstract system (Reissig et al., 2017).

The basic ideas of ABCD are well understood and supported by several tools—`SCOTS`, `CoSyMa`, `Pessoa`, `Tulip`, `ROCS`, and `PFACES`, to name a few (Rungger and Zamani, 2016; Mouelhi et al., 2013; Jr. et al., 2010; Khaled and Zamani, 2019; Li and Liu, 2018; Wongpiromsarn et al., 2011). Even though very versatile and completely automated, existing ABCD methods suffer from a huge computational bottleneck caused by the discrete abstraction, because the size of the abstraction grows exponentially with the number of state variables of the system. Besides, there is no existing technique that would satisfactorily solve the problem when the system is subjected to stochastic uncertainty (modeling uncertainty or external noise) and we want an ω-regular specification to be satisfied with maximum probability. These are some of the current active research

[1] Throughout this thesis, the pronoun of choice will be "we" (instead of "I") as the technical results are outcome of many fruitful collaborations with my colleagues.

directions in ABCD.

1.2. Contributions and Future Outlook in a Nutshell

We show that automated and correct-by-construction controller design techniques can be significantly faster, can be made modular, and can be applied to problems considerably broader in scope than what was possible before this dissertation. In particular, we develop new techniques that

(A) substantially improve the computational speed of the existing ABCD techniques by using multi-layered abstractions,

(B) compute controllers for stochastically perturbed continuous systems (controlled Markov processes) with guarantees on the minimum probability of satisfaction of ω-regular specifications,

(C) modularize the design of controllers using assume-guarantee contracts when the (discrete) abstract system has a certain decomposable structure, and

(D) efficiently synthesize controllers for discrete systems under the effect of a fair adversary.

We implemented our algorithms in several prototype tools, and empirically validated their effectiveness and improved performance over the state-of-the-art. The multi-layered ABCD (Part (A)) is implemented in `Mascot`, which in our experiments performed up to one order of magnitude faster for synthesis of controllers using ABCD. The stochastic controller synthesis algorithm (Part (B)) and the fair adversarial synthesis algorithm (Part (D)) are implemented in `Mascot-SDS` and `Fairsyn` respectively, and both perform significantly faster and significantly more memory-efficient than the state-of-the-art. The assume-guarantee synthesis algorithm (Part (C)) is implemented in `Agnes`, using which we show that distributed synthesis on the discrete systems can be performed within reasonable amount of time.

It took a number of years for the whole body of research presented in this thesis to take shape. And during this time, many significant fundamental improvements were accomplished (Gruber et al., 2017; Liu, 2017; Nilsson et al., 2017; Hussien and Tabuada, 2018; Girard and Gößler, 2020; Dutreix et al., 2020; Majumdar et al., 2020b) and powerful tools were built (Li and Liu, 2018; Khaled and Zamani, 2019; Lavaei et al., 2020a) with similar goals as ours. As a result, we have seen correct-by-construction synthesis procedures being applied more and more in real-life safety-critical applications (Ames et al., 2015; Yang et al., 2017), (Abate et al., 2020, Sec. 4.3). Yet, there are many challenges for a wide-scale use, such as scalability that is not yet up to the mark, lack of runtime adaptability to unexpected situations, lack of support for black-box systems, etc. We hope that extensions of our work will help overcoming some of these hurdles in future.

1.3. Intuitive Overview of the Results

In the following, we give an intuitive overview of our contributions. Different parts of our contributions (A), (B), (C), and (D), as mentioned in Sec. 1.2, are explained in sections 1.3.1, 1.3.3, 1.3.2, and 1.3.4 respectively.

1.3.1. Lazy Multi-Layered ABCD

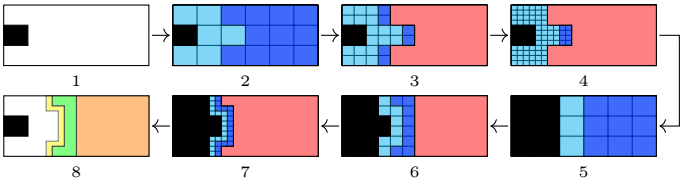
As our first contribution, we propose a more scalable approach for ABCD. The key computational bottleneck of the existing ABCD methods is the size of the abstraction, which grows exponentially with the system dimension. Tackling this state-space explosion is one of the main open directions of research. In ABCD, usually the abstract system is computed by first fixing a parameter τ for the sampling time and a parameter η for the state and the input spaces, and then representing the abstract state space as a set of *finitely many* hypercubes, each of diameter η. The success of ABCD depends on the choice of η and τ. Intuitively, increasing η (and τ)[1] results in fewer hypercubes but leads to a more imprecise abstract transition relation. Thus, one may not be able to find a controller for the abstract system. On the other hand, decreasing η (and τ) results in a more precise abstraction and a higher chance of successful controller synthesis. However, the larger state space can make the synthesis problem computationally intractable. This observation has led to an extension of ABCD to a *multi-layered* setting, where the algorithm maintains several "layers" of abstract systems by picking hypercube partitions of different resolutions. The resulting abstract controller synthesis procedure tries to find a controller for the coarsest abstraction whenever feasible, but adaptively considers finer abstractions when necessary. On a set of standard benchmark examples, we show that our algorithm is up to an order of magnitude faster compared to the baseline single-layered algorithm.

In the following, we intuitively explain the multi-layered ABCD algorithm using a simple example depicted in Fig. 1.1; the technical details are in Chap. 4.

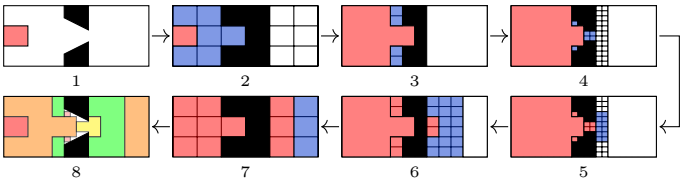
We consider the safety and the reach-avoid control problems. In reactive synthesis, these problems are solved using a maximal and a minimal fixed point computation, respectively (Maler et al., 1995). In particular, for safety, one starts with the maximal set of safe states and iteratively shrinks the latter until the remaining set, called the winning state set, does not change. That is, for all states in the winning state set, there is a control action which ensures that the system remains within this set for one step. For reachability, one starts with the set of target states as the winning state set and iteratively enlarges this set by adding all states which allow the system to surely reach the current winning state set, until no more states can be added. These differences in the underlying characteristics of the fixed points require different switching protocols when multiple abstraction layers are used.

The purpose of Fig. 1.1 is only to convey the basic idea of our algorithms in a visual and lucid way, without paying attention to the details of the underlying dynamics of the system. In our example, we use three layers of abstraction S_1, S_2 and S_3 with the

[1] Usually, τ is increased along with η to reduce non-determinism due to self loops.



(a) Illustration of lazy ABCD with a *safety* specification.



(b) Illustration of lazy ABCD with a *reach-avoid* specification.

Figure 1.1.: An illustration of the lazy ABCD algorithms for safety and reach-avoid specifications. In both scenarios, the solid black regions are the unsafe states which need to be avoided. In the reach-avoid problem, the system has to additionally reach the target (T) red square at the left of Pic. 1. Both figures show the sequence of synthesis stages across three abstraction layers: $l = 1$ (Pics. 4, 7), $l = 2$ (Pics. 3, 6), and $l = 3$ (Pics. 2, 5) for safety; and $l = 1$ (Pics. 4, 5), $l = 2$ (Pics. 3, 6), and $l = 3$ (Pics. 2, 7) for reach-avoid. Pic. 8 in both figures indicate the domains of the resulting controllers with different granularity: $l = 1$ (yellow), $l = 2$ (green), and $l = 3$ (orange). The red regions represent winning states, and the dark blue regions represent states added to the winning states in the present synthesis stage. Cyan regions represent "potentially losing" states in the safety synthesis. We set the parameter $m = 2$ for reach-avoid synthesis. The gridded regions in different layers represent the states where the transitions have been computed; large ungridded space in $l = 2$ and $l = 1$ signifies the computational savings of the lazy abstraction approach.

parameters (η, τ), $(2\eta, 2\tau)$ and $(4\eta, 4\tau)$. We refer to the steps of Fig. 1.1 as Pic. #.

For the *safety control problem* in Fig. 1.1a, we assume that a set of unsafe states are given (the black box in the left of Pic. 1). These need to be avoided by the system. For lazy ABCD, we first fully compute the abstract transition relation of the coarsest abstraction S_3, and find the states from where the unsafe states can be avoided for at least one time step of length 4τ (dark blue region in Pic. 2). Normally for a single-layered algorithm, the complement of the dark blue states would immediately be discarded as losing states. However, in the multi-layered approach, we treat these states as *potentially losing* (cyan regions), and proceed to S_2 (Pic. 3) to determine if some of these potentially losing states can avoid the unsafe states with the help of a more fine-grained controller.

However, we cannot perform any safety analysis on S_2 yet as the abstract transitions of S_2 have not been computed. Instead of computing all of them, as in a non-lazy approach, we only locally explore the outgoing transitions of the potentially losing states in S_2.

Then, we compute the subset of the potentially losing states in S_2 that can avoid the unsafe states for at least one time step (of length 2τ in this case). These states are represented by the dark blue region in Pic. 3, which get saved from being discarded as losing states in this iteration. Then we move to S_1 with the rest of the potentially losing states and continue similarly. The remaining potentially losing states at the end of the computation in S_1 are surely losing—relative to the finest abstraction S_1—and are permanently discarded. This concludes one "round" of exploration.

We restart the process from S_3. This time, the goal is to avoid reaching the unsafe states for at least two time steps of available lengths. This is effectively done by inflating the unsafe region with the discarded states from previous stages (black regions in Pics. 5, 6, and 7). The procedure stops when the combined winning regions across all layers do not change for two successive iterations.

In the end, the multi-layered safety controller is obtained as a collection of the safety controllers synthesized in different abstraction layers in the last round of fixed-point computations. The resulting safety controller domain is depicted in Pic. 8.

Now consider the *reach-avoid control problem* in Fig. 1.1b. The target set is shown in red, and the states to be avoided are shown in black. We start by computing the abstract transition system completely for the coarsest layer and solve the reachability fixpoint at this layer (until convergence) using under-approximations of the target and the safe states. The winning region is marked in blue (Pic. 2); note that the approximation of the bad states "cuts off" the possibility to reach the winning states from the states on the right. We store the representation of this winning region in the finest layer as the set Ψ_1.

Intuitively, we run the reachability fixpoint until convergence to enlarge the winning state set as much as possible using large cells. This is in contrast to the previous algorithm for safety control in which we performed just one iteration at each level. For safety, each iteration of the fixed-point shrinks the winning state set. Hence, running the safety fixpoint until convergence would only keep those coarse cells which form an invariant set by themselves. On the other hand, as we run just one iteration of the safety fixpoint at a time, hence the coarser layers in future iterations get a chance to benefit from the winning cells obtained in the finer layers. This allows the use of coarser control actions in larger parts of the state space (see Fig. 1 in Hsu et al. (2018a) for an illustrative example of this phenomenon).

To further extend the winning state set Ψ_1 for reach-avoid control, we proceed to the next finer layer $l = 2$ with the new target region (red) being the projection of Ψ_1 to $l = 2$. As in safety control, all the safe states in the complement of Ψ_l are potentially within the winning state set. The abstract transitions at layer $l = 2$ have not been computed at this point. We only compute the abstract transitions for the *frontier* states: these are all the cells that might contain layer 2 cells that can reach the current winning region within m steps (for some parameter m chosen in the implementation). The frontier is indicated for layer 2 by the small gridded part in Pic. 3.

We continue the backward reachability algorithm on this partially computed transition system by running the fixpoint algorithm for m steps. The projection of the resulting states to the finest layer is added to Ψ_1. In our example (Pic. 3), we reach a fixed point

just after one iteration implying that no more layer 2 (or layer 3) cells can be added to the winning region.

We now move to layer 1, compute a new frontier (the gridded part in Pic. 4), and run the reachability fixed point on Ψ_1 for m steps. We add the resulting winning states to Ψ_1 (the blue region in Pic. 4). At this point, we could keep exploring and synthesizing in layer 1, but in the interest of efficiency we want to give the coarser layers a chance to progress. This is the reason to only compute m steps of the reachability fixed point in any one iteration. Unfortunately, for our example, the attempt to go coarser fails as no new layer 2 cells can be added yet (see Pic. 3). We therefore fall back to layer 1 and make progress for m more steps (Pic. 5). At this point, the attempt to go coarser is successful (Pic. 6) as the right side of the small passage was reached.

We continue this movement across layers until synthesis converges in the finest layer. In Pic. 8, the orange, green and yellow colored regions are the controller domains obtained using $l = 3$, $l = 2$ and $l = 1$, respectively. Observe that we avoid computing transitions for a significant portion of layers 1 and 2 (the ungridded space in Pics. 5, 6, respectively).

1.3.2. Stochastic ABCD

Next, we extend (single-layered) ABCD to a class of stochastically perturbed systems known as *controlled Markov processes (CMPs)*. CMPs evolve over continuous state space and discrete time, and form a general model for temporal decision making under stochastic uncertainty. In recent years, the problem of finding or approximating optimal controllers in CMPs for specifications given in temporal logics or automata has received a lot of attention. While there is a steady progression towards more expressive models and properties (Tkachev et al., 2017; Haesaert and Soudjani, 2018; Svorenová et al., 2015; Dutreix and Coogan, 2019; Majumdar et al., 2020a; Dutreix et al., 2020), a satisfactory solution that can handle *nonlinear* models for general *ω-regular specifications* in a *symbolic* way has been open. In this thesis, we make progress towards a solution to this general problem. We show that the optimal controller synthesis problem for CMPs can be solved in two steps: In **step I**, We introduce a novel abstraction using $2^1/_2$-player games, whose solution under-approximates the set of *qualitative* winning states—called the *almost sure winning region*—from which the specification can be satisfied almost surely (i.e. with probability one). In **step II**, we address the *quantitative* aspect, where we show that the optimal satisfaction probability of the specification can be under-approximated using the optimal probability of reaching the almost sure winning region; for this part, we can use existing tools for synthesizing optimal controllers for reachability specifications.

One of the main contributions of this thesis is a novel ABCD procedure to approximate solution to **step I**. Just like established ABCD techniques for non-stochastic systems, we create a finite cover of the continuous state space using hyper-rectangular cover elements. Each cover element represents one abstract state, and the set of all cover elements form the state space of the finite abstraction. Our key insight in the abstraction phase is that the probabilistic disturbances naturally induce a *fairness* condition on the environment. We take this observation into account while creating the abstract transition(s). Firstly, like in regular (non-stochastic) ABCD, during the abstraction process we compute an abstract

transition relation that over-approximates the continuous transitions. In addition to that, we also maintain an *under-approximation* of the continuous transitions simultaneously, such that an under-approximating transition exists between two abstract states $\widehat{x}$ and $\widehat{y}$ if from every continuous state in $\widehat{x}$, the probability of reaching $\widehat{y}$ in one step is above some uniform positive threshold. Intuitively, if there is an under-approximating abstract transition between $\widehat{x}$ and $\widehat{y}$, then that transition will be eventually taken with probability one if $\widehat{x}$ is visited infinitely many times (probabilistic fairness).

This insight gets captured within the abstract $2^1/_2$-player game. We show that, for parity specifications, an almost sure winning controller for the abstract game can be mapped back to an almost sure winning controller for the original CMP. Since every ω-regular language can be represented using a canonical parity automaton, we obtain an ABCD procedure for CMPs with ω-regular control specifications. Part **(I)** of our ABCD procedure is symbolic, because the $2^1/_2$-player game can be solved using symbolic fixpoints; see the technical details of our proposed symbolic algorithm for $2^1/_2$-player games in Chap. 5.

We have implemented **step I** of our approach and evaluated it on the nonlinear model of a perturbed bistable switch from the literature. We show empirically that the lower bound on the winning region computed by our approach is precise, by comparing against an over-approximation of the qualitative winning region. Moreover, our symbolic implementation outperforms a recently proposed tool for solving this problem by a large margin. In fact, in many cases the existing tool crashed after consuming too much memory on a standard laptop, whereas our tool consumed small amount of memory and produced results within reasonable amount of time.

1.3.3. Assume-Guarantee Distributed Synthesis

We propose a *modular* solution to the *distributed* reactive synthesis problem of two systems connected in feedback and having *local* temporal specifications. In our setting, each of the two systems has its own private states and can partially observe the states of the other system. The dynamics of each system depends on its own state, control action, the observations of the other system, and the input from the environment. We want to synthesize *local* controllers for each system—those that pick control actions solely based on locally available information—such that each system satisfies its own local specification. We use *assume-guarantee contracts* to decompose the overall synthesis problem into local synthesis subproblems: Each system makes an *assumption* on the temporal behavior of the other system, and in return, in addition to satisfying its local specification, provides a *guarantee* on its own temporal behavior. The contracts are called compatible if the guarantee of each system implies the assumption made by the other system. It can be shown that compatible contracts result in simultaneous satisfaction of all the local specifications, leading to a solution of the distributed synthesis problem. We propose a *negotiation* procedure, where the two systems negotiate a pair of compatible assume-guarantee contracts among themselves. At the heart of our negotiation procedure lies the computation of minimally restrictive environment assumptions in reactive synthesis, which was proposed by Chatterjee et al. (2008). We explain the negotiation algorithm

using a simple mutual exclusion-style distributed synthesis problem; technical details can be found in Chap. 6.

Suppose we have a distributed architecture with two finite-state synchronous systems $\mathcal{S}_0$ and $\mathcal{S}_1$, where each system requires to transmit a single data packet through a shared bus within a certain deadline. Assume that sending each data packet takes one time unit for the bus. There is no handshaking involved in the transmission process: whenever a system wants to send the packet, it simply needs to write it at the sending end of the bus. However, if both systems write their data packets exactly at the same instant, the bus turns down the send request from both of the systems to avoid data corruption. When a send request is turned down by the bus, the systems can attempt to resend the failed data packet in the future. Clearly, if $\mathcal{S}_0$ and $\mathcal{S}_1$ keeps trying to send their packets all the time, they will never succeed.

The distributed synthesis problem involving $\mathcal{S}_0$ and $\mathcal{S}_1$ asks to compute *local* controllers for the two systems, which decide when to send their packets based on locally available information only. For instance, a possible solution in this case will be $\mathcal{S}_0$ only writing in the even cycles while assuming that $\mathcal{S}_1$ only writes in the odd cycles, and dually $\mathcal{S}_1$ only writing in the odd cycles while assuming that $\mathcal{S}_0$ only writes in the even cycles. Such type of synchronizations can be formalized using assume-guarantee contracts, where each system makes an assumption on the temporal behavior of the other system, and in exchange, provides a guarantee on its own behavior. If the systems have strategies to enforce their local specifications along with their guarantees under the hypothesis that the other system respects the assumptions, and moreover the guarantee of each system implies the assumption used by the other system, one can prove that the entire closed-loop system, using the synthesized strategies, satisfies all local specifications.

We propose an algorithm where the systems negotiate assume-guarantee contracts so that the specifications are fulfilled. Our algorithm is sound but necessarily incomplete, because this problem of distributed synthesis is known to be undecidable for the communication architecture that we consider (Pnueli and Rosner, 1990).

In the following we explain our algorithm on the packet sending problem involving $\mathcal{S}_0$ and $\mathcal{S}_1$. Figure 1.2 shows the structure of one system $\mathcal{S}_i$ in a guarded command language. The other system is similar. Each system has *state variables* (separate copies of s and t) that it reads and writes, *external variables* from the environment that it can read (env), and *output variables* that it writes and that provide its visible state to the environment (out). The external variables provide the visible state of the rest of the system—a system does not control their values. Additionally, the system has a number of input actions ($wait$ and wr) it can use to determine how its state is updated.

A state maps the state variables to values. Initially, the state is $(idle, 4)$: the system has 4 steps to send the packet. The transitions map the current state, current values of external variables, and current choice of control inputs to new values of the state variables. For example, when the state is $idle$, picking the $wait$ input action keeps the state idle but decreases t; picking the wr input action changes the state to write and also decreases t. The output variable is a function of the state; it is visible to other systems.

Intuitively, the system moves from *idle* to *writing* and then to *done*, once it successfully

state var $s \in \{idle, writing, done\}$,
$t \in \{1, 2, 3, 4\}$
external var $env \in \{idle, busy\}$
output var $out \in \{idle, busy\}$
input action $U = \{wait, wr\}$

init $s = idle, t = 4$
transition
$[\!] \; s = idle \xrightarrow{wait} s' = idle \wedge t' = t - 1$
$[\!] \; s = idle \xrightarrow{wr} s' = writing \wedge t' = t - 1$
$[\!] \; s = writing \wedge t \geq 2 \wedge env = idle \xrightarrow{wr} s' = done$
$[\!] \; s = writing \wedge t \geq 2 \wedge env = busy \xrightarrow{wr} s' = writing \wedge t' = t - 1$
output $out = busy$ if $s = writing$ and $idle$ otherwise

Figure 1.2.: The packet sender system. Our example has two such systems running synchronously in parallel.

sends the packet. However, if the bus is busy, it may fail to send the packet by the deadline. Each system wants to eventually successfully write its data packet. In terms of state variables, and using linear temporal logic (LTL) notation, the specification is $\Diamond(s = done)$.

We are looking for a *distributed* solution to the problem: each system must run a local controller that only sees the state of the system and the history of external inputs that it receives. Thus, we cannot simply take the product of the individual state spaces and run a reactive synthesis algorithm for the conjunction of the local specifications.

Let us first point out the issues with the naïve approach of assuming the worst-case and the best-case behaviors from the other systems.

Worst-Case Environment. Suppose we try to find a controller for each system, independently of the other. Unfortunately, we realize that there is no local controller without any assumptions on the behavior of the other system: in the worst case, the other system could be writing in all cycles, and our system will never be able to send its packet.

Assume the Specification of the Environment. Clearly, a "worst-case" behavior is too pessimistic. At least, the other system must satisfy its own specification. What if we assumed that the behavior of the other system is constrained by its own specification? Unfortunately, this is still not sufficient in our example: if we only know that the other system *eventually* does not write to the bus, we could still try to write in the same cycle. Moreover, both systems could end up waiting for each other.

Assumptions, Guarantees, and Negotiations. As already pointed out, an intuitive solution to the problem is that one system promises to write only in even cycles and the other only in odd cycles. Then, the first system can make progress towards its writing: it waits until the next even cycle and writes.

Our contribution is to show how we can automatically come up with such assume-guarantee pairs. In real life, whenever two entities, people or organizations, with their own sets of interests need to make an agreement for a peaceful co-existence, a *negotiation*

process is called for. Accordingly, we call our algorithm to iteratively compute assume-guarantee pairs a *negotiation*. We will use the above motivating example to give an informal description of our solution method.

The negotiation is an iterative procedure. Initially, we make no assumptions about the other system and check if perhaps each system can satisfy its specification no matter how the other one behaves; if so, we are done. On the other hand, if a system cannot satisfy its specification even while assuming full cooperation of the others, we can stop—certainly we shall not find any implementation in this case.

Otherwise, we proceed by finding an *environment assumption*: a minimal restriction on the behavior of the other system that enables our system to satisfy its specification.

Let us look at the negotiation from the perspective of system $\mathcal{S}_0$. For example, assuming $t = 4$, $\mathcal{S}_0$ would find an assumption $A_0 = (busy_1 + idle_1) \cdot (busy_1 + idle_1) \cdot idle_1^\omega$, which states that system $\mathcal{S}_1$ does not transmit *after* the second cycle. Under this assumption, $\mathcal{S}_0$ can satisfy its specification: simply send the packet after the second cycle.

Next, we check if system $\mathcal{S}_1$ can indeed satisfy its own specification while additionally guaranteeing the assumption A_0 of system $\mathcal{S}_0$. We check this by defining the guarantee G_1 as the projection of A_0 to the output variables of $\mathcal{S}_1$ and seeing if $\mathcal{S}_1$ has a winning strategy in the game $\diamond done \wedge G_1$. Unfortunately, since system $\mathcal{S}_1$ makes no assumptions (its current assumption is "true"), it cannot fulfill this specification. We find a tighter environment assumption that $\mathcal{S}_0$ must ensure in order for $\mathcal{S}_1$ to win; it is the language $A_1 = (busy_0 + idle_0) \cdot idle_0 \cdot (busy_0 + idle_0)^\omega$, which states that system $\mathcal{S}_0$ does not transmit *in* the second cycle.

In general, given a system's current assumption A_i and guarantee G_i, we check if it has a strategy to fulfill its specification $\Phi_i = \diamond done$ under the contract (A_i, G_i). If not, we find a new assumption and update the other system's guarantee, which starts a new round of negotiation.

We show this process is sound: if both systems can win the above game, then the current assumptions and guarantees form an assume-guarantee decomposition, and we can "read off" a distributed controller implementation. In our example, the negotiation terminates in the second round and outputs these final assumptions and guarantees:

$$A_0 = (busy_1 + idle_1) \cdot (busy_1 + idle_1) \cdot idle_1^\omega \tag{1.1}$$
$$G_0 = (busy_0 + idle_0) \cdot idle_0 \cdot (busy_0 + idle_0)^\omega \tag{1.2}$$
$$A_1 = (busy_0 + idle_0) \cdot idle_0 \cdot (busy_0 + idle_0)^\omega \tag{1.3}$$
$$G_1 = (busy_1 + idle_1) \cdot (busy_1 + idle_1) \cdot idle_1^\omega \tag{1.4}$$

We have built a prototype tool called `Agnes` that implements the negotiation algorithm. Using `Agnes`, we have empirically demonstrated the effectiveness of our proposed algorithm on two numerical examples.

1.3.4. Symbolic Algorithms for Fair Adversarial Games

Our second contribution in the discrete reactive synthesis domain is a symbolic algorithm for solving 2-player graph games under a fairness assumption on the environment. 2-player

graph games are at the heart of many core problems in the synthesis of correct-by-construction hardware, software, and cyber-physical systems. In fact, through a series of reductions, the reactive synthesis problem for ω-regular specifications reduces to finding winning strategies in 2-player graph games. In practice, it is often the case that no solution exists to a given synthesis problem, but for "uninteresting" reasons. For example, consider synthesizing a mutual exclusion protocol from a specification that requires (1) that at most one of two processes can be in the critical section at any time and (2) that a process wishing to enter the critical section is eventually allowed to do so. As stated, there may not be a feasible solution to the problem because a process within the critical section may decide to stay there forever. Fairness assumptions rule out such uninteresting conditions by constraining the possible behaviors of the environment. The updated winning condition under fairness is of the form

$$\text{Fairness Assumption} \;\Rightarrow\; \omega\text{-regular Specification.} \tag{1.5}$$

We consider a particular form of fairness, called the *strong transition fairness* (Queille and Sifakis, 1983; Francez, 1986; Baier and Katoen, 2008), which naturally models many practical instances of fairness like processes eventually leaving critical section, fair scheduling of threads, etc. A strong transition fairness assumption can be modeled by a set of *live* environment transitions in the underlying two-player game graph. Whenever the source vertex of a live transition is visited infinitely often, the transition will be taken infinitely often by the environment. We call the resulting game a *fair adversarial game*. Unfortunately, despite the widespread prevalence of strong transition fairness, current symbolic algorithms for solving games do not take advantage of their special structure in the winning condition (1.5) and no algorithm better than those for general (Streett) liveness conditions is known. We propose symbolic fixpoint algorithms for Rabin and several other ω-regular fair adversarial games, which are significantly faster compared to the naïve treatment of the live edges using regular Streett condition. Our fixpoints are obtained through simple syntactic transformations of the regular fixpoint algorithms for 2-player games without strong transition fairness assumption. Since $2^1/_2$-player games are special instances of fair adversarial games, so, as a byproduct, we obtain direct symbolic algorithms for solving $2^1/_2$-player games which are significantly faster than the state-of-the-art approaches.

In the following, we give an intuitive overview of fair adversarial games and our algorithm for solving such games; the details are in Chap. 7.

Consider a game between 2 players, called *Player* 0 and *Player* 1, played on the graph shown in Fig. 1.3: The game starts by placing a token on one of the vertices, called the initial vertex. When the token is on a circular vertex, *Player* 0 moves the token to one of the successors. Similarly, when the token is on a rectangular vertex, *Player* 1 moves the token to one of the successors. The dotted edge (v_1, v_3) is a *live* edge, and the *strong transition fairness* condition states: If the vertex v_1 is visited infinitely many times then *Player* 1 must take the edge (v_1, v_3) infinitely many times. Suppose the winning condition for *Player* 0 is to make the token eventually reach vertex v_3. Then *Player* 0 *wins the fair*

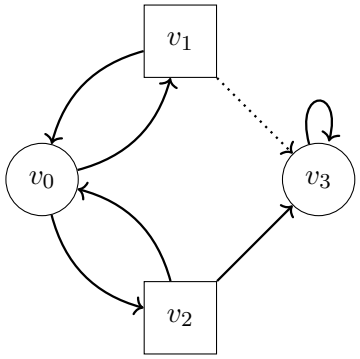


Figure 1.3.: An example of a 2-player fair adversarial game graph. The circular vertices are controlled by *Player* 0, and the rectangular ones are controlled by *Player* 1. The edge (v_1, v_3) (dotted) is the live edge.

adversarial game if the following conditional winning condition is satisfied:

$$\text{Strong transition fairness} \;\Rightarrow\; \text{eventually reach } v_3.$$

As already mentioned before, in practice, fairness assumptions like the one above help us rule out "uninteresting" reasons for which no solution to the game exists. For the game in Fig. 1.3, without the fairness assumption on the live edge (v_1, v_3), no winning strategy exists for *Player* 0 when the initial vertex is v_0. (Because *Player* 1 would be able to prevent the token from reaching v_3, by sending it back to v_0, each time the token is in either v_1 or v_2.) However, *Player* 0 *can win the fair adversarial game* simply by moving the token to v_1 each time the token comes back to v_0, just because eventually *Player* 1 needs to take the live edge (v_1, v_3) as per the strong transition fairness assumption.

This class of games, with strong transition fairness assumptions on *Player* 1 edges, are called (2-player) *fair adversarial games*. In this thesis, we present a symbolic algorithm for computing winning regions and winning strategies for fair adversarial games. We show a surprising syntactic transformation that modifies well-known symbolic fixpoint algorithms for solving two-player games on graphs *without* fairness assumptions, such that the modified fixed point solves the game for the winning condition (1.5) whenever the given fairness assumption can be specified as *strong transition fairness*. To appreciate the simplicity of our modification, let us consider the well-known fixpoint algorithms for Büchi and co-Büchi games—particular classes of Rabin games—given by the following μ-calculus formulas:

$$\begin{array}{ll} \textbf{Büchi:} & \nu Y.\ \mu X.\ (G \cap \mathit{Cpre}(Y)) \cup (\mathit{Cpre}(X))\,, \\ \textbf{Co-Büchi:} & \mu X.\ \nu Y.\ (G \cup \mathit{Cpre}(X)) \cap (\mathit{Cpre}(Y))\,. \end{array}$$

where $\mathit{Cpre}(\cdot)$ denotes the controllable predecessor operator and G denotes the set of goal states that should be visited recurrently. In the presence of strong transition fairness, the

new algorithms become

$$\begin{array}{ll} \textbf{Büchi:} & \nu Y.\ \mu X.\ (G \cap \mathit{Cpre}(Y)) \cup (\mathit{Apre}(Y, X)), \\ \textbf{Co-Büchi:} & \nu W.\ \mu X.\ \nu Y.\ (G \cup \mathit{Apre}(W, X)) \cap (\mathit{Cpre}(Y)). \end{array}$$

The only syntactic change (highlighted in blue) we make is to substitute the controllable predecessor for the μ variable X by a new *almost sure predecessor operator* $\mathit{Apre}(Y, X)$ incorporating also the previous ν variable Y; if the fixpoint starts with a μ variable (with no previous ν variable), like for co-Büchi games, we introduce one additional ν variable in the front. For the general class of Rabin specifications, with a more involved fixpoint and with arbitrarily high nesting depth depending on the number of Rabin pairs, we need to perform this substitution for every such $\mathit{Cpre}(\cdot)$ operator for every μ variable.

In a nutshell, our results show that one can solve games under strong transition fairness assumptions on environment behaviors *while retaining* the algorithmic characteristics of known symbolic fixpoint algorithms when fairness assumptions are not considered. We prove the correctness of our syntactic fixpoint transformation for solving Rabin games (Rabin, 1969; Piterman and Pnueli, 2006) and *generalized Rabin games*. Further, we also show its correctness for Reachability, Safety, (generalized) Büchi, (generalized) co-Büchi, Rabin-chain, parity (Emerson and Jutla, 1991; Maler et al., 1995), and GR(1) games (Piterman et al., 2006) as special cases. Moreover, through a simple reduction, we obtain a direct symbolic algorithm for almost sure winning in $2^1/_2$-player games, which is significantly faster than the state-of-the-art. While our proofs are subtle, symbolic implementations of our algorithms require very small changes to existing code. Moreover, our empirical evaluation demonstrates that our new algorithm can be multiple orders of magnitude more efficient than previous algorithms.

We have implemented our algorithms in a synthesis engine based on BDDs (Binary Decision Diagrams). We show on a set of synthetic and real benchmarks that our algorithm is scalable, parallelizable, and outperforms previous algorithms by orders of magnitude.

1.4. Organization of the Contents

In Fig. 1.4, we show a dependency graph between various parts of the thesis. In Chap. 2, we introduce the basic notation and summarize various concepts related to temporal logics and graph games. All the other chapters use concepts which are defined in Chap. 2. Chap. 3, Chap. 4, and Chap. 5 form the Part I of this thesis, dealing with the abstraction aspect of the ABCD procedure: In Chap. 3, we recapitulate the basic concepts of ABCD; most of the results in this chapter uses the feedback refinement relations-based ABCD (Reissig et al., 2017). In Chap. 4, we present our work on lazy multi-layered abstractions, and in Chap. 5, we present our work on ABCD for controlled Markov processes. Chap. 6 and Chap. 7 form the Part II of this thesis, dealing with our contributions in reactive synthesis towards better scalability: In Chap. 6, we present our negotiation algorithm for the distributed reactive synthesis problem, and in Chap. 7, we present the symbolic algorithmic solution of fair adversarial games.

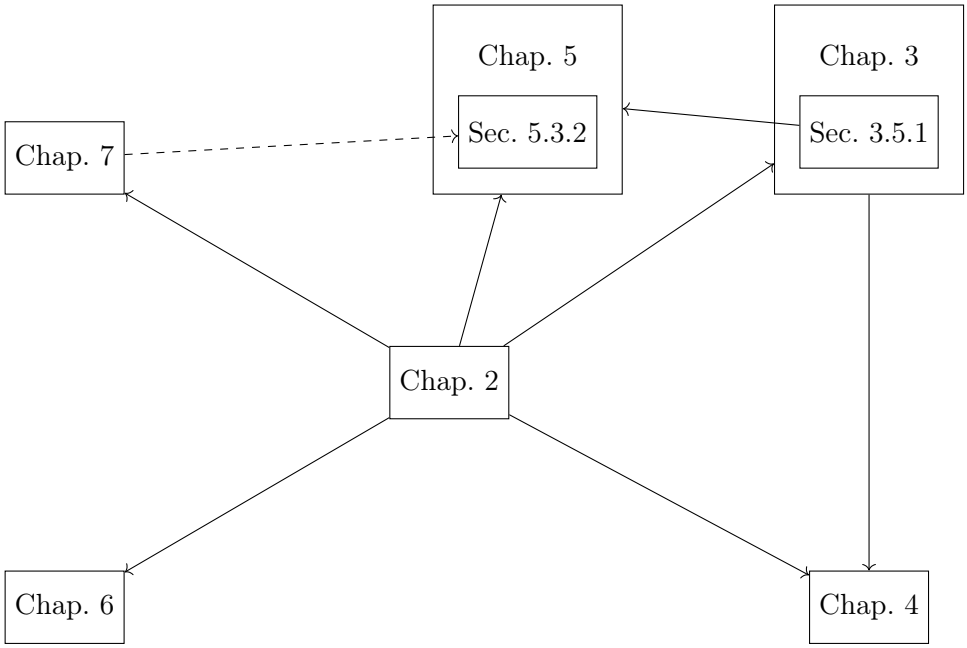


Figure 1.4.: A dependency graph between the chapters. A solid arrow from a node p to a node q means understanding p is necessary for understanding q. A dashed arrow from a node p to a node q means p is used in q, but p can be treated as a blackbox without much sacrifice in understanding.

2. Preliminaries: Temporal Logics, $2^1/_2$-Player Games

We introduce general mathematical notation and concepts related to temporal logics and turn-based graph games, which will be used abundantly in the thesis.

Primarily, we will use temporal logics to specify properties of systems, and we will restrict ourselves to only linear temporal logic and certain fragments of it.

We will use two types of turn-based graph games. The first one is called $2^1/_2$-player games, and will model interaction between two adversarial agents in the presence of probabilistic environment. The second one is called 2-player games, which is a particular restricted subclass of $2^1/_2$-player games, and will model the interaction between two adversarial agents only (no probabilistic environment). These games will emerge as abstractions of CPS synthesis problems, and their solution procedures will serve as the algorithmic basis for correct-by-construction controller synthesis for CPS.

2.1. Notation

Numbers and intervals. We use the symbols $\mathbb{N}$, $\mathbb{R}$, $\mathbb{R}_{\geq 0}$, $\mathbb{R}_{>0}$, $\mathbb{Z}$, and $\mathbb{Z}_{>0}$ to denote the sets of natural numbers including zero, reals, non-negative reals, positive reals, integers, and positive integers, respectively. Given $a, b \in \mathbb{R}$ such that $a \leq b$, we denote by $[a, b]$ a closed interval and define $[a; b] = [a, b] \cap \mathbb{Z}$ as its discrete counterpart. Given $a, b \in \mathbb{R}^n$, we denote by a_i and b_i their i-th element and write $[\![a, b]\!]$ for the closed hyper-interval $\mathbb{R}^n \cap ([a_1, b_1] \times \ldots \times [a_n, b_n])$. We define the relations $<, \leq, \geq, >$ on a, b component-wise.

Sets. Given a set A, we use $|A|$ to denote the cardinality of A. For any set $A \subseteq U$ defined on the universe U, we use the notation $\overline{A}$ to denote the complement of A.

Probability space. For any set A, a sigma-algebra on A comprises subsets of A as events that includes A itself and is closed under complement and countable unions. We consider a probability space $(\Omega, \mathcal{F}_\Omega, P_\Omega)$, where Ω is the sample space, $\mathcal{F}_\Omega$ is a sigma-algebra on Ω, and P_Ω is a probability measure that assigns probabilities to events. An ($(S, \mathcal{F}_S)$-valued) random variable X is a measurable function of the form $X : (\Omega, \mathcal{F}_\Omega) \to (S, \mathcal{F}_S)$, where S is the codomain of X and $\mathcal{F}_S$ is a sigma-algebra on S. Any random variable X induces a probability measure on its space $(S, \mathcal{F}_S)$ as $P(\{A\}) = P_\Omega\{X^{-1}(A)\}$ for any $A \in \mathcal{F}_S$. We often directly discuss the probability measure on $(S, \mathcal{F}_S)$ without explicitly mentioning the underlying probability space $(\Omega, \mathcal{F}_\Omega, P_\Omega)$ and the function X itself.

Let A be a finite set. A *probability distribution* over A is a probability measure on the space $(A, 2^A)$. We use the notation $\mathrm{dist}(A)$ to denote the set of all probability

distributions on A. Given any distribution $f \in \mathrm{dist}(A)$, we define the support of f as: $\mathsf{supp}\ f \coloneqq \{a \in A \mid f(a) > 0\}$. For any given element $a \in A$, the Dirac delta distribution on a is denoted as $\delta_a \in \mathrm{dist}(A)$, and is defined as:

$$\delta_a(x) \coloneqq \begin{cases} 1 & x = a \\ 0 & x \neq a. \end{cases}$$

Borel space. A topological space S is called a Borel space if it is homeomorphic to a Borel subset of a Polish space (i.e., a separable and completely metrizable space). Examples of a Borel space are the Euclidean spaces $\mathbb{R}^n$, its Borel subsets endowed with a subspace topology, as well as hybrid spaces. Any Borel space S is assumed to be endowed with a Borel sigma-algebra (i.e., the one generated by the open sets in the topology), which is denoted by $\mathcal{B}(S)$. We say that a map $f\colon S \to Y$ is measurable whenever it is Borel measurable.

Sequences and languages. For an alphabet Σ, we write Σ^*, Σ^+, and Σ^ω for the sets of finite words, non-empty finite words, and infinite words over Σ, respectively. We define $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. For $w \in \Sigma^*$, we write $|w|$ for the length of Σ; the length of $w \in \Sigma^\omega$ is ∞. We define $\mathrm{dom}(w) = [0; |w| - 1]$ if $w \in \Sigma^*$, and $\mathrm{dom}(w) = \mathbb{N}$ if $w \in \Sigma^\omega$. We denote by $\mathrm{dom}^+(w) = \mathrm{dom}(w) \setminus \{0\}$ the positive domain of w. For $k \in \mathrm{dom}(w)$ we sometimes (when the symbols in the sequence are not made explicit) write w^k for the k-th symbol of w and $w|_{[0;k]}$ for the restriction of w to the domain $[0; k]$. If $\Sigma = A \times B$, the projection of $w \in \Sigma^\infty$ on A is denoted by $\mathbf{proj}_A(w)$. Given two words $u \in \Sigma^*$ and $v \in \Sigma^\infty$, we write uv to denote the concatenation of u and v. (Note that if u is allowed to be an infinite word, then uv is not well-defined.) The empty word is denoted using ε, and for any given word $u \in \Sigma^\infty$, $\varepsilon u = u$, and for any given word $u \in \Sigma^*$, $u\varepsilon = u$. Moreover, given two languages $M \subseteq \Sigma^*$ and $N \subseteq \Sigma^\infty$, we write MN to denote the concatenation of M and N, i.e. $MN = \{uv \mid u \in M \wedge v \in N\}$, and write M^ω to denote the concatenation of infinitely many copies of M.

We define the prefix relation on words over an alphabet Σ as $u \leq w$ if there exists v such that $uv = w$. Note that w can be an ω-word. We extend the notion to languages: the prefix of a language L, written $\mathrm{pref}(L)$, is the set $\{w \in \Sigma^* \mid \exists u \in L\ .\ w \leq u\}$. A language L is *prefix-closed* if, whenever $w \in L$ and $u < w$, then $u \in L$. Given a ($*$-)language $L \subseteq \Sigma^*$, we define the *limit* $\lim(L)$ of L as the ω-language

$$\{u \in \Sigma^\omega \mid \exists \text{ infinitely many } n \text{ such that } u_{[0;n]} \in L\}.$$

Thus, an infinite word belongs to the limit of a $*$-language L iff infinitely many of its prefixes belong to L. If L is prefix-closed, this implies that an infinite word belongs to $\lim(L)$ iff all its finite prefixes belongs to L. An (ω-)language L is a *safety* language if $L = \lim(\mathrm{pref}(L))$ and a *liveness* language if $\mathrm{pref}(L) = \Sigma^*$.

Functions and relations. Given two sets A and B, $f : A \to 2^B$ and $f : A \to B$ denote a set-valued and ordinary functions, respectively. The function f is called *total* if f maps every point in A to some point in 2^B (or B for ordinary f); otherwise f is called *partial*. When f is partial, for every $a \in A$ either $f(a)$ is in 2^B (or B for ordinary f) or $f(a)$ is

undefined. We identify set-valued functions with their respective binary relation over $A \times B$, i.e., $(a, b) \in f$ iff $b \in f(a)$. The inverse function $f^{-1} : B \to 2^A$ is defined via its respective binary relation: $f^{-1}(b) = \{a \in A \mid b \in f(a)\}$. For the sets A, B, and C, and for the functions $f\colon A \to B$ and $g\colon B \to C$, the composition of f and g is the function $f \circ g\colon A \to C$, defined as: $f \circ g(a) = f(g(a))$.

Let X be a set and $R \subseteq X \times Y$ be a relation. For simplicity, let us assume that $\mathrm{dom}(R) \coloneqq \{x \in X \mid \exists y \in Y \,.\, (x, y) \in R\} = X$. For any given $x \in X$, we use the notation $R(x)$ to denote the set $\{y \in Y \mid (x, y) \in R\}$. We extend this notation to sets: For any given $Z \subseteq X$, we use the notation $R(Z)$ to denote the set $\cup_{z \in Z} R(z)$. The inverse relation of R is the relation $R^{-1} \subseteq Y \times X$ defined as $R^{-1} \coloneqq \{(y, x) \mid (x, y) \in R\}$.

The identity function, the one mapping every element to itself, is denoted by id.

2.2. Temporal Logics

Throughout this thesis, we will use various different logical formalisms to express properties of systems. We summarize them in the following. The semantics of all the properties will be interpreted over sets of infinite words of a given finite alphabet Σ.

2.2.1. Linear Temporal Logic

Linear Temporal Logic (LTL) was introduced by Gabbay et al. (1980) as an extension of the propositional logic by the temporal operators "next" and "until," denoted by the symbols "$\bigcirc$" and "$\,\mathsf{U}$" respectively. The semantics of various LTL formulas, which are to be introduced in the following, are illustrated in Fig. 2.1 in Page 22.

Definition 2.1 (Linear Temporal Logic) Let Σ be any alphabet and $AP \subseteq 2^\Sigma$ be a set of atomic propositions. An LTL formula φ over AP has the following syntax:

$$\varphi ::= \mathsf{true} \mid p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \bigcirc \varphi \mid \varphi_1 \,\mathsf{U}\, \varphi_2, \qquad p \in AP,$$

where φ_1 and φ_2 are also LTL formulas.

The boolean disjunction operator "$\vee$" and the truth value "false" can be derived from the basic syntax in the usual way: $\varphi_1 \vee \varphi_2 \coloneqq \neg(\neg\varphi_1 \wedge \neg\varphi_2)$ and $\mathsf{false} \coloneqq \neg\mathsf{true}$. We also use the well-known derived temporal operators "eventually," "always," and "release," denoted respectively by "$\Diamond$," "$\Box$," and "$\,\mathsf{R}$," and defined as $\Diamond\varphi \coloneqq \mathsf{true}\,\mathsf{U}\varphi$, $\Box\varphi \coloneqq \neg\Diamond\neg\varphi$, and $\varphi_1 \,\mathsf{R}\, \varphi_2 \coloneqq \neg(\neg\varphi_1 \,\mathsf{U}\, \neg\varphi_2)$.

The semantics of an LTL formula φ are defined over the set of infinite words over Σ. Let $w \in \Sigma^\omega$ be an infinite word over the alphabet Σ. The satisfaction of φ by w, written

as $w \models \varphi$, is defined inductively as follows:

$$
\begin{aligned}
&w \models \mathsf{true}, && \\
&w \models p && \text{if } w^0 \in p, \\
&w \models \neg\varphi && \text{if } w \not\models \varphi, \\
&w \models \varphi_1 \wedge \varphi_2 && \text{if } w \models \varphi_1 \text{ and } w \models \varphi_2, \\
&w \models \bigcirc\varphi && \text{if } w|_{[1;\infty)} \models \varphi, \\
&w \models \varphi_1 \,\mathsf{U}\, \varphi_2 && \text{if } \exists t \in [0;\infty) \cdot w|_{[t;\infty)} \models \varphi_2 \text{ and } \forall t' \in [0;t) \cdot w|_{[t',\infty)} \models \varphi_1.
\end{aligned}
$$

The set of every infinite word over Σ satisfying φ is denoted as $\langle\!\langle\varphi\rangle\!\rangle_\Sigma$, i.e. $\langle\!\langle\varphi\rangle\!\rangle_\Sigma \coloneqq \{w \in \Sigma^\omega \mid w \models \varphi\}$.

Notational convention for bounded-horizon LTL property. We adopt a notational convention for expressing bounded-horizon temporal behaviors in LTL. Let $k \in \mathbb{N}$ be a nonnegative integer. We use the notation $\bigcirc^k$ and $\mathsf{U}^{\leq k}$ to respectively denote the k-step next operator and the k-step bounded until operator, which are shorthand notation for the following expanded formulas:

$$
\begin{aligned}
\bigcirc^k\psi &\equiv \bigcirc\bigcirc\ldots\langle k \text{ times } \rangle\,\psi \\
\psi_1 \,\mathsf{U}^{\leq k}\psi_2 &\equiv (\psi_1 \,\mathsf{U}\psi_2) \wedge \textstyle\bigvee_{l\in[0;k]} \bigcirc^l\psi_2.
\end{aligned}
$$

The bounded eventually, the bounded always, and the bounded release operators are defined as $\Diamond^{\leq k}\psi = \mathsf{true}\,\mathsf{U}^{\leq k}\psi = \bigvee_{l\in[0;k]} \bigcirc^l\psi$, $\Box^{\leq k}\psi = \neg\Diamond^{\leq k}\neg\psi = \bigwedge_{l\in[0;k]} \bigcirc^l\psi$, and $\psi_1 \,\mathsf{R}^{\leq k}\psi_2 = \neg(\neg\psi_1 \,\mathsf{U}^{\leq k}\neg\psi_2)$.

The positive normal form. An LTL formula is in *Positive Normal Form* (PNF) if all the negation symbols "¬" appear only adjacent to the atomic propositions. Every arbitrary LTL formula φ can be transformed into an equivalent and canonical LTL-formula in PNF by pushing the negations through the boolean and the temporal operators using the following rewrite rules (Baier and Katoen, 2008, Sec. 5.1.5, pp. 258):

$$
\begin{aligned}
\neg\mathsf{true} &\rightsquigarrow \mathsf{false} \\
\neg\neg\varphi &\rightsquigarrow \varphi \\
\neg(\varphi_1 \wedge \varphi_2) &\rightsquigarrow \neg\varphi_1 \vee \neg\varphi_2 \\
\neg \bigcirc \varphi &\rightsquigarrow \bigcirc\neg\varphi \\
\neg(\varphi_1 \,\mathsf{U}\varphi_2) &\rightsquigarrow \neg\varphi_1 \,\mathsf{R}\neg\varphi_2
\end{aligned}
$$

The size of the transformed formula is $\mathcal{O}(|\varphi|)$, where $|\varphi|$ is the size of the original formula φ (Baier and Katoen, 2008, Thm. 5.24).

2.2.2. A Next Operator-Free Fragment of LTL

While dealing with continuous-time systems, the notion of the next operator will turn out to be ill-defined. For such systems, we will consider the next operator-free fragment of LTL—denoted as $\mathrm{LTL}_{\setminus\bigcirc}$—that disallows the use of the next operator.

Definition 2.2 (The LTL$_{\setminus\bigcirc}$ fragment of LTL) The syntax of any formula in LTL$_{\setminus\bigcirc}$ is given as:

$$\varphi ::= \mathsf{true} \mid p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \,\mathsf{U}\, \varphi_2, \qquad\qquad p \in AP,$$

where φ_1 and φ_2 are also LTL$_{\setminus\bigcirc}$ formulas.

The semantics of LTL$_{\setminus\bigcirc}$ are directly inherited from the semantics of LTL.

2.2.3. Regular Properties

Fix an alphabet Σ and let $AP \subseteq 2^\Sigma$ be a set of atomic propositions. A property φ is called regular if its semantics, i.e. the set of (finite) words that satisfies φ, represent a regular language. The two regular properties which we will consider often are the reach-avoid and the safety properties, both of which are fragments of LTL. The reach-avoid property holds true in a sequence over Σ if a given target proposition is met before a given unsafe proposition holds true. The safety property holds true if a given unsafe proposition never holds true, or dually if a given safe proposition always holds true.

Definition 2.3 (Reach-Avoid Property) A reach-avoid property ψ involves two predicates $T, O \in 2^\Sigma$, and in LTL it is written as $\psi \coloneqq \neg O \,\mathsf{U}\, T$. The semantics can be derived from the semantics of LTL formulas in Sec. 2.2.1, which can be written down as follows:

$$\llbracket \neg O \,\mathsf{U}\, T \rrbracket_\Sigma \coloneqq \{w \in \Sigma^* \mid \exists k \in \mathrm{dom}(w) \,.\, w^k \in T \;\wedge\; \forall k' < k \,.\, w^{k'} \notin O\}.$$

Definition 2.4 (Safety Property) A safety property ψ involves a single predicate $B \in 2^\Sigma$, and in LTL it is written as $\psi \coloneqq \Box B$. The semantics can be derived from the semantics of LTL formulas in Sec. 2.2.1, which can be written down as follows:

$$\llbracket \Box B \rrbracket_\Sigma \coloneqq \{w \in \Sigma^* \mid \forall k \in \mathbb{N} \,.\, w^k \in O\}.$$

2.2.4. ω-Regular Properties

We introduce several ω-regular acceptance conditions for formally specifying properties of systems. Traditionally these acceptance conditions are used in ω-automata, with finitely many states and a transition relation. We will always assume that the underlying automata structure has already been incorporated within the system under consideration, possibly through a synchronous product construction.

In the following, we use the symbols "$\in_{even}$" and "$\in_{odd}$" to denote memberships in the set of even and odd integers within a given set of integers: For example, for a given set of natural numbers $M \subseteq \mathbb{N}$, the notation $n \in_{even} M$ is equivalent to $n \in M \cap \{0, 2, 4, \ldots\}$, and the notation $n \in_{odd} M$ is equivalent to $n \in M \cap \{1, 3, 5, \ldots\}$. Let Σ be a finite alphabet. We define the operator $\mathrm{Rec}\colon \Sigma^\omega \to 2^\Sigma$ that maps every infinite word over Σ to the set of infinitely recurring symbols on that sequence: $\mathrm{Rec}(w) \coloneqq \{\sigma \in \Sigma \mid \forall i \in \mathbb{N} \,.\, \exists j > i \,.\, w^j = \sigma\}$.

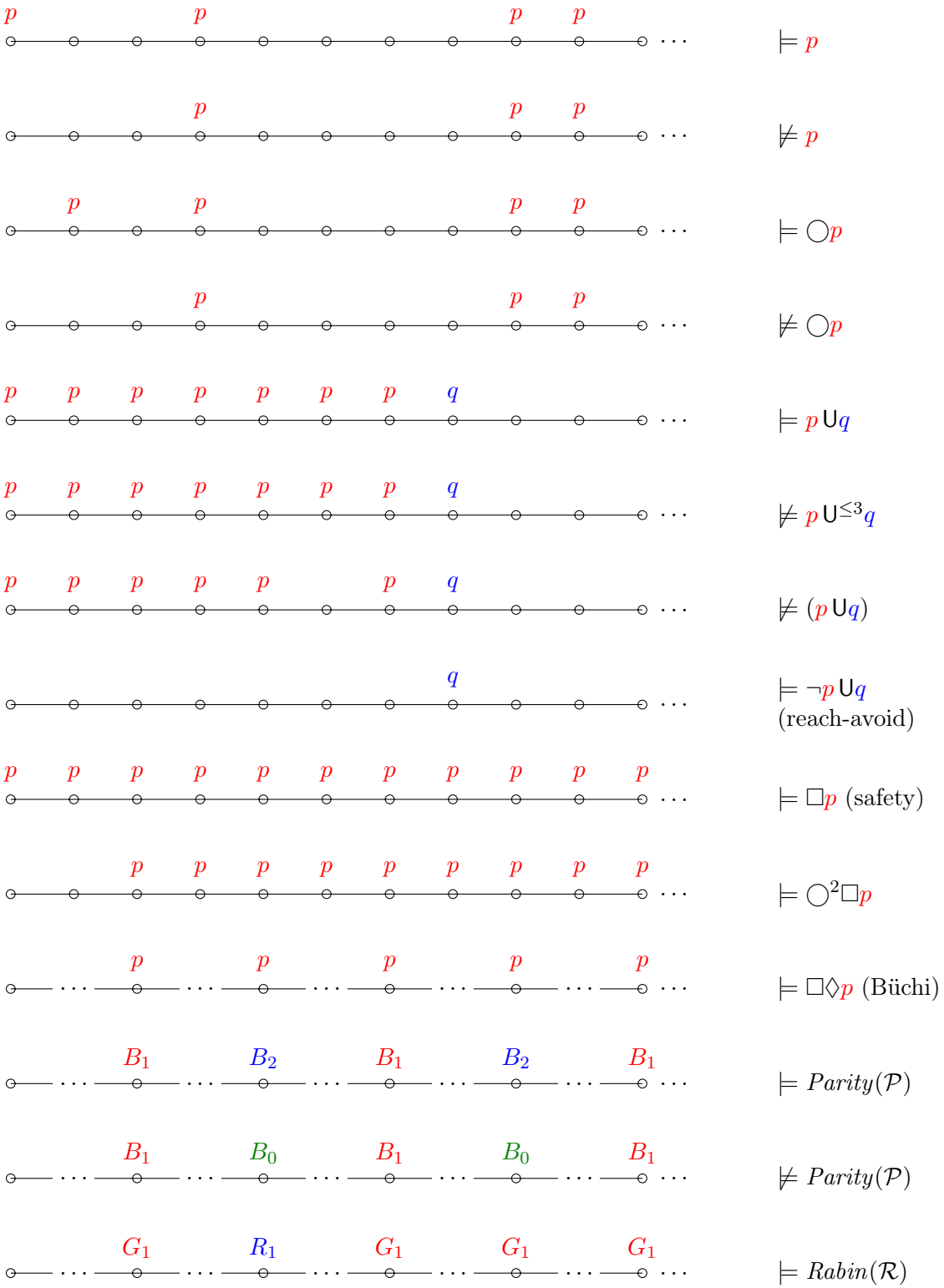


Figure 2.1.: Illustration of the semantics of various Linear Temporal Logic formulas on infinite words.

Definition 2.5 (Büchi Properties) Consider an alphabet Σ. A Büchi property can be expressed using an LTL formula of the form $\square\lozenge B$ for some atomic proposition $B \subseteq \Sigma$. The semantics can be derived from the semantics of LTL formulas in Sec. 2.2.1, and can be written down as:

$$\llbracket \square\lozenge B \rrbracket_\Sigma \coloneqq \{w \in \Sigma^\omega \mid \mathrm{Rec}(w) \cap B \neq \emptyset\}. \tag{2.1}$$

Intuitively, the Büchi property holds along an infinite word if certain atomic proposition B holds true infinitely often.

Definition 2.6 (Parity Properties) Consider a set of atomic propositions $\mathcal{P} = \{B_0, B_1, \ldots, B_{2k}\}$ whose elements partition the alphabet Σ. The corresponding parity property $\mathit{Parity}(\mathcal{P})$ can be expressed using the following LTL formula:

$$\mathit{Parity}(\mathcal{P}) \coloneqq \bigwedge_{i \in_{odd} [1;2k]} \left(\square\lozenge B_i \rightarrow \bigvee_{j \in_{even} [i+1;2k]} \square\lozenge B_j \right). \tag{2.2}$$

We say that a symbol $\sigma \in \Sigma$ has *priority* i if $\sigma \in B_i$. The even priority sets are all B_i-s with even i and the odd priority sets are all B_i-s with odd i. The semantics can be derived from the semantics of LTL formulas in Sec. 2.2.1, and can be written down as:

$$\llbracket \mathit{Parity}(\mathcal{P}) \rrbracket_\Sigma \coloneqq \{w \in \Sigma^\omega \mid \max(\{i \mid \mathrm{Rec}(w) \cap B_i \neq \emptyset\}) = \mathit{even}\}. \tag{2.3}$$

Intuitively, for a given set of priority sets $\mathcal{P}$, the parity property holds along an infinite word if the largest infinitely occurring priority is even.

Definition 2.7 (Rabin Properties) Consider a set of atomic propositions $G_1, R_1, \ldots, G_k, R_k \in 2^\Sigma$. Each pair $\langle G_i, R_i \rangle$ for $i \in [1;k]$ is called a Rabin pair, and we denote the set of Rabin pairs as $\mathcal{R} \coloneqq \{\langle G_1, R_1 \rangle, \ldots, \langle G_k, R_k \rangle\}$. The corresponding Rabin property $\mathit{Rabin}(\mathcal{R})$ can be expressed using the following LTL formula:

$$\mathit{Rabin}(\mathcal{R}) \coloneqq \bigvee_{i \in [1;k]} (\square\lozenge G_i \wedge \lozenge\square\neg R_i). \tag{2.4}$$

The semantics can be derived from the semantics of LTL formulas in Sec. 2.2.1, and can be written down as:

$$\llbracket \mathit{Rabin}(\mathcal{R}) \rrbracket_\Sigma \coloneqq \{w \in \Sigma^\omega \mid \exists i \in [1;k] \,.\, \mathrm{Rec}(w) \cap G_i \neq \emptyset \wedge \mathrm{Rec}(w) \cap R_i = \emptyset\}. \tag{2.5}$$

Intuitively, for a given set of Rabin pairs $\{\langle G_i, R_i \rangle\}_{i \in [1;k]}$, the Rabin property holds along an infinite word if there exists an index i for which the proposition G_i holds infinitely often and the proposition R_i holds only finitely often.

2.3. Games on Finite Graphs

At the heart of all the synthesis algorithms, that we present later in this thesis, lie finite graph games. There are various forms of finite graph games in the literature. For our purpose, we summarize the traditional $2^1/_2$-player games, also known as simple (turn-based) stochastic games, played between *Player* 0, *Player* 1, and a third player representing *environmental randomness* and being treated as a "half player." Parallel to this, we introduce 2-player turn-based games as special cases of the $2^1/_2$-player games.

The first ingredient for formalization of $2^1/_2$-player games is the underlying arena or the game graph, that is defined as follows:

Definition 2.8 ($2^1/_2$-player game graphs) A $2^1/_2$-player game graph is a tuple $\mathcal{G} = \langle V, V_0, V_1, V_r, E\rangle$ where

(i) V is a finite set of vertices,

(ii) V_0, V_1, and V_r are subsets of V which form a partition of V, i.e. V_0, V_1, and V_r are pairwise disjoint and $V_0 \cup V_1 \cup V_r = V$, and

(iii) $E \subseteq V \times V$ is the set of directed edges.

The vertices in V_0 and V_1 are respectively called the *Player* 0 *vertices* and the *Player* 1 *vertices*, and the edges originating in a *Player* 0 vertex and a *Player* 1 vertex are respectively called *Player* 0 *edges* and *Player* 1 *edges*. The vertices in V_r are called *random vertices* and the edges originating in a random vertex are called *random edges*. The set of all random edges is denoted by $E_r \coloneqq E(V_r)$.

A 2-player game graph on the other hand is a special case of the $2^1/_2$-player game graph that does not have any random vertex in it:

Definition 2.9 (2-player game graphs) A $2^1/_2$-player game graph with no random vertices (i.e. $V_r = \emptyset$) is called a 2-player game graph. We simply omit V_r in this case and represent the game graph as the tuple $\mathcal{G} = \langle V, V_0, V_1, E\rangle$.

The second ingredient for formalization of both $2^1/_2$-player games and 2-player games is the winning condition for *Player* 0, which is the set of valid infinite sequences of vertices visited in $\mathcal{G}$.

Definition 2.10 (Winning Conditions) A *winning condition* φ for *Player* 0 in a game graph $\mathcal{G}$ is an LTL formula over a set of atomic propositions over the alphabet V.

Unless otherwise mentioned, we will use the convention that the given winning condition is a winning condition for *Player* 0.

Finally, we define the $2^1/_2$-player game and the 2-player game as pairs of the respective game graphs together with some winning conditions:

Definition 2.11 ($2^1/_2$-Player Game and 2-Player Game) A $2^1/_2$-*player game* is a pair $\langle \mathcal{G}, \varphi \rangle$ of a $2^1/_2$-player game graph $\mathcal{G}$ and a winning condition φ for $\mathcal{G}$. A 2-*player game* is a pair $\langle \mathcal{G}, \varphi \rangle$ of a 2-player game graph $\mathcal{G}$ and a winning condition φ for $\mathcal{G}$.

The games are played as follows: The game starts from an arbitrary *initial vertex*; we implicitly assume that every vertex in the game graph can be an initial vertex. Then in every step, the player controlling the current vertex chooses a successor and the game moves to this successor vertex in the next step. The choice of the successors from the random vertices is performed uniformly at random, wheres the choice by *Player* 0 and *Player* 1 are governed by *strategies*.

Definition 2.12 (Strategies) Let $\mathcal{G} = \langle V, V_0, V_1, V_r, E \rangle$ be a $2^1/_2$-player game graph. A strategy of *Player* 0 is a function $\pi_0 \colon V^* V_0 \to V$ with the constraint $\pi_0(wv) \in E(v)$ for every $wv \in V^* V_0$. Likewise, a strategy of *Player* 1 is a function $\pi_1 \colon V^* V_1 \to V$ with the constraint $\pi_1(wv) \in E(v)$ for every $wv \in V^* V_1$. We denote the set of strategies of *Player* 0 and *Player* 1 on the game graph $\mathcal{G}$ by $\Pi_0(\mathcal{G})$ and $\Pi_1(\mathcal{G})$, respectively.

Of special interest is the class of memoryless strategies for *Player* 0 and *Player* 1: a strategy π_0 of *Player* 0 is *memoryless* if for every $w_1 v, w_2 v \in V^* V_0$, we have $\pi_0(w_1 v) = \pi_0(w_2 v)$. We use the notation $\Pi_i^{\mathrm{DM}}(\mathcal{G})$ to denote the set of all deterministic memoryless strategies of *Player* i. Observe that $\Pi_i^{\mathrm{DM}}(\mathcal{G}) \subseteq \Pi_i(\mathcal{G})$.

Whenever the underlying game graph is clear from the context, we simply write Π_i and Π_i^{DM} for the set of strategies and the set of deterministic memoryless strategies of *Player* i.

In general strategies of *Player* 0 and *Player* 1 can be randomized, meaning the choice of the next state can be based on a probability distribution on the possible successors. In contrast, we only use *deterministic* strategies, since it is known that randomized strategies are no more powerful than deterministic ones for $2^1/_2$-player Büchi, parity, and Rabin games (Chatterjee et al., 2003, 2005).

Definition 2.13 (Plays) Consider an infinite sequence of vertices $\rho = v^0 v^1 v^2 \ldots \in V^\omega$. The sequence ρ is called a *play* over $\mathcal{G}$ starting at the vertex v^0 if for every $i \in \mathbb{N}_0$, we have $v^i \in V$ and $(v^i, v^{i+1}) \in E$. A play is *finite* if it is of the form $v^0 v^1 \ldots v^n$ for some finite $n \in \mathbb{N}_0$.

In our convention for denoting vertices, superscripts (ranging over $\mathbb{N}_0$) will denote the position of a vertex within a given play, whereas subscripts, either 0, 1, or r, will denote the membership of a vertex in the sets V_0, V_1, or V_r respectively.

Let π_0 and π_1 be a given pair of strategies of *Player* 0 and *Player* 1, respectively, and let v^0 be a given initial vertex. An infinite play $\rho = v^0 v^1 \ldots$ *complies to* π_0 and π_1 if for every finite prefix $\rho|_{[0;n]} = v^0 v^1 \ldots v^n$ of ρ, the next vertex v^{n+1} in the play ρ is obtained using the following rules: (a) If $v^n \in V_0$ then $v^{n+1} = \pi_0(v^0 \ldots v^n)$, (b) if $v^n \in V_1$ then $v^{n+1} = \pi_1(v^0 \ldots v^n)$, and (c) if $v^n \in V_r$ then v^{n+1} is chosen uniformly at random from

the set $E(v^n)$. The random choice in the random vertices induce a probability measure $P_{v^0}^{\pi_0,\pi_1}$ on the sample space of plays.[1]

For a 2-player game graph $\mathcal{G}$, for any given pair of strategies π_0 and π_1 and for any given initial vertex v^0, the resulting play is *unique*.

Let $\langle \mathcal{G}, \varphi \rangle$ be a $2^1/_2$-player game where $\mathcal{G} = \langle V, V_0, V_1, V_r, E \rangle$ is a $2^1/_2$-player game graph and φ is a winning condition. Let us denote the event that the plays of $\mathcal{G}$ satisfies φ using the symbol $\mathcal{G} \models \varphi$. For a given initial vertex $v^0 \in V$ and for a given pair of strategies π_0 and π_1 of *Player* 0 and *Player* 1, we denote the probability of the occurrence of the event $\mathcal{G} \models \varphi$ by $P_{v^0}^{\pi_0,\pi_1}(\mathcal{G} \models \varphi)$.

In this thesis, we will be interested only in the *qualitative analysis* of $2^1/_2$-player games, which deal with the following two modes of winning: (a) The game is sure winning for *Player* 0 if she can satisfy the specification on *every* compliant play for every (adversarial) *Player* 1 strategy, and (b) the game is almost sure winning for *Player* 0 if she can satisfy the specification with *probability one* against every (adversarial) *Player* 1 strategy. In the following, we formalize the objects associated to the two winning modes.

Definition 2.14 (Sure Winning Region) Let $\mathcal{G} = \langle V, V_0, V_1, V_r, E \rangle$ be a $2^1/_2$-player game graph and φ be a winning condition. A vertex $v \in V$ is called *sure winning* for *Player* 0 if there *exists* a *Player* 0 strategy $\pi_0 \in \Pi_0$, such that for *every Player* 1 strategy $\pi_1 \in \Pi_1$, *every* play starting at v and complying to π_0 and π_1 is in ψ. The strategy π_0 is called the *sure winning strategy* from the vertex v. The *sure winning region* of *Player* 0 is the set of every sure winning vertices of *Player* 0, and is denoted as $\mathcal{W}^{sure}(\mathcal{G}, \varphi)$, or just $\mathcal{W}^{sure}$ if the game $\langle \mathcal{G}, \varphi \rangle$ is clear from the context. The sure winning region and the sure winning strategy of *Player* 1 can be defined analogously.

Definition 2.15 (Almost Sure Winning Region) Let $\mathcal{G} = \langle V, V_0, V_1, V_r, E \rangle$ be a $2^1/_2$-player game graph and φ be a winning condition. A vertex $v \in V$ is called *almost sure winning* for *Player* 0 if

$$\sup_{\pi_0 \in \Pi_0} \inf_{\pi_1 \in \Pi_1} P_v^{\pi_0,\pi_1}(\mathcal{G} \models \varphi) = 1. \tag{2.6}$$

The strategy π_0 is called the almost sure winning strategy from the vertex v. The *almost sure winning region* of *Player* 0 is the set of every almost sure winning vertices of *Player* 0, and is denoted as $\mathcal{W}^{a.s.}(\mathcal{G}, \varphi)$, or just $\mathcal{W}^{a.s.}$ if the game $\langle \mathcal{G}, \varphi \rangle$ is clear from the context. The almost sure winning region and the sure winning strategy of *Player* 1 can be defined analogously.

It follows from the definition that the notion of almost sure winning is weaker than the notion of sure winning: $\mathcal{W}^{a.s.}(\mathcal{G}, \varphi) \subseteq \mathcal{W}^{sure}(\mathcal{G}, \varphi)$ for every $\mathcal{G}$ and φ. Additionally, when $\mathcal{G}$ is a 2-player game graph *or* when φ is a safety winning condition, the two notions of winning coincide: $\mathcal{W}^{a.s.}(\mathcal{G}, \varphi) = \mathcal{W}^{sure}(\mathcal{G}, \varphi)$.

[1] The unique measure $P_{v^0}^{\pi_0,\pi_1}$ is obtained through Carathéodory's extension theorem by extending the pre-measure on every infinite extension—called the cylinder set—of every finite play; see (Baier and Katoen, 2008, pp. 757) for details.

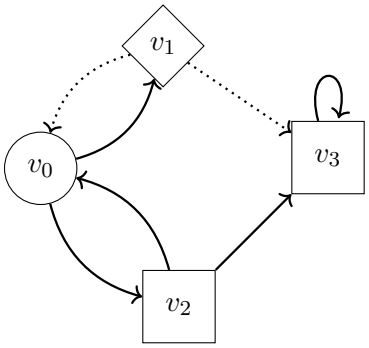


Figure 2.2.: An example of a $2^1/_2$-player game graph. The *Player* 0 vertices (V_0) are in circles, the *Player* 1 vertices (V_1) are in squares, and the random vertices (V_r) are in diamond, respectively. The random edges (i.e. the ones originating from the random vertices) are represented using dotted arrows. We will use this same convention while depicting $2^1/_2$-game graphs throughout this thesis.

Example 2.1 We illustrate a $2^1/_2$-player game graph $\mathcal{G}$ in Fig. 2.2. Consider the game $\langle \mathcal{G}, \Diamond v_3 \rangle$, where the winning condition requires *Player* 0 to eventually reach vertex v_3. A strategy for *Player* 0 decides the next state from the only *Player* 0 vertex v_0 whenever the game reaches v_0. Here are some examples of *Player* 0 strategies, given any finite play $\rho \subseteq V^* v_0$:

- $\pi_0^a(\rho) = v_1$ for all ρ,
- $\pi_0^b(\rho) = v_2$ for all ρ,
- $\pi_0^c(\rho) = v_1$ if there are *odd* number of v_0 in ρ, and $\pi_0^c(\rho) = v_2$ if there are *even* number of v_0 in ρ,
- $\pi_0^d(\rho) = v_1$ if v_1 appeared i-times in ρ and v_2 appeared exactly 2^i-times in ρ,
- ...

The strategies π_0^a and π_0^b are memoryless, π_0^c requires one bit of memory (the memory bit gets flipped every time v_0 is seen), and π_0^d requires infinite memory (to keep track of the ever growing sequence $(2^0, 2^1, 2^2, \ldots)$ of "trigger points" when v_1 will be chosen from v_0). It can be shown that except for the strategy π_0^b all the three other strategies are almost sure winning strategies for *Player* 0 from every vertex in the game graph $\mathcal{G}$. In other words, the almost sure winning region $\mathcal{W}^{a.s.}$ is the entire set of vertices. The strategy π_0^b is not almost sure winning from the vertex v_0, because there exists a *Player* 1 strategy π_1 with $\pi_1(\rho) = v_0$ for every finite run ρ ending in v_2, such that $P_{v_0}^{\pi_0^b, \pi_1}(\mathcal{G} \models \Diamond v_3) = 0$. On the other hand, the only sure winning vertex is the trivial one v_3, because no matter what strategy *Player* 0 picks, there will always be an infinite play (possibly with probability 0) that never reaches v_3.

Of special interests are the parity and the Rabin games, because every other ω-regular games can be transformed into a parity or a Rabin game by taking synchronous product of the game graph with the parity or the Rabin automaton accepting the original ω-regular specification (similar to the construction in the book by Gradel and Thomas (2002, Proof of Thm. 2.7, pp. 28)). It can be shown that a sure/almost sure winning strategy in the transformed game can be mapped back to a (possibly memory-dependent) sure/almost sure winning strategy in the original game.

A $2^1/_2$-player game $\langle \mathcal{G}, \varphi \rangle$ is called a Borel game if φ is any Borel winning condition, which is a family of winning conditions subsuming every winning conditions used in this thesis.

Proposition 2.1 (Determinacy for sure winning) *Every vertex of a 2-player Borel game is sure winning for one of the two players (Martin, 1975). For 2-player parity games, both the players have deterministic memoryless sure winning strategies (Emerson and Jutla, 1991). For 2-player Rabin games, Player 0 has deterministic memoryless sure winning strategies and Player 1 has deterministic finite-memory sure winning strategies (Gurevich and Harrington, 1982; Emerson and Jutla, 1988).*

$2^1/_2$-player games, on the other hand, have no determinacy for sure winning, though we can state a similar theorem in terms of the almost sure winning mode. For this we need to introduce the dual of almost sure winning, called positive winning. Given a $2^1/_2$-player game, a vertex v is *positive winning* for *Player* 0 if *Player* 0 has a strategy π_0 such that

$$\sup_{\pi_0 \in \Pi_0} \inf_{\pi_1 \in \Pi_1} P_v^{\pi_0, \pi_1}(\mathcal{G} \models \varphi) > 0. \tag{2.7}$$

Proposition 2.2 (Determinacy for almost sure winning) *Every vertex of a $2^1/_2$-player Borel game is either almost sure winning for one of the players, or is positive winning for the other player (Martin, 1998). For $2^1/_2$-player parity games, both the players have deterministic memoryless almost sure winning strategies (Chatterjee et al., 2003). For $2^1/_2$-player Rabin games, Player 0 has deterministic memoryless almost sure winning strategies (Chatterjee et al., 2005).*

2.4. Symbolic Fixpoint Algorithms for Computation of Sure Winning Regions

We use the μ-calculus (Kozen, 1983) as a convenient logical notation used to define a symbolic algorithm (i.e., an algorithm that manipulates sets of states rather then individual states) for computing a set of states with a particular property over a given game graph $\mathcal{G} = \langle V, V_0, V_1, V_r, E \rangle$. Let $pre \colon 2^V \to 2^V$ be a function mapping every set of vertices to another set of vertices; we call the function pre a *set transformer*. In addition, pre is a *monotone* set transformer if it is monotonic with respect to set inclusion, i.e. for every $U, U' \subseteq V$ with $U \subseteq U'$, we have $pre(U) \subseteq pre(U')$. The formulas of the μ-calculus, interpreted over a 2-player game graph $\mathcal{G}$, are given by the grammar

$$\varphi \ ::= \ p \mid X \mid \varphi \cup \varphi \mid \varphi \cap \varphi \mid pre(\varphi) \mid \mu X.\varphi \mid \nu X.\varphi$$

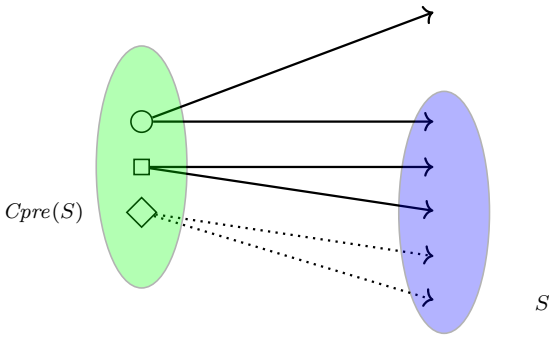


Figure 2.3.: Illustration of the *Cpre* operator.

where p ranges over subsets of V, X ranges over a set of formal variables, *pre* is any monotone set transformer, and μ and ν denote, respectively, the least and the greatest fixed point of the functional defined as $X \mapsto \varphi(X)$. Since the operations $\cup$, $\cap$, and the set transformers *pre* are all monotonic, the fixed points are guaranteed to exist. A μ-calculus formula evaluates to a set of states over $\mathcal{G}$, and the set can be computed by induction over the structure of the formula, where the fixed points are evaluated by iteration. We omit the (standard) semantics of formulas (see Kozen (1983)).

Let $\langle \mathcal{G}, \psi \rangle$ be a $2^1/_2$-player game. Our goal is to develop symbolic fixpoint algorithms to characterize the *sure* winning region of the game $\langle \mathcal{G}, \psi \rangle$. As a first step, given $\mathcal{G} = \langle V, V_0, V_1, V_r, E \rangle$, we define the required monotonic set transformers. We define the existential, two types of universal, and controllable predecessor operators as follows. For $S \subseteq V$, we have

$$\mathrm{Pre}_0^{\exists}(S) \coloneqq \{v \in V_0 \mid E(v) \cap S \neq \emptyset\}, \tag{2.8a}$$

$$\mathrm{Pre}_1^{\forall}(S) \coloneqq \{v \in V_1 \mid E(v) \subseteq S\}, \tag{2.8b}$$

$$\mathrm{Pre}_r^{\forall}(S) \coloneqq \{v \in V_r \mid E(v) \subseteq S\}, \text{ and} \tag{2.8c}$$

$$\mathit{Cpre}(S) \coloneqq \mathrm{Pre}_0^{\exists}(S) \cup \mathrm{Pre}_1^{\forall}(S) \cup \mathrm{Pre}_r^{\forall}(S). \tag{2.8d}$$

Intuitively, the controllable predecessor operator $\mathit{Cpre}(S)$ computes the set of all states that can be controlled by *Player* 0 to stay in S after one step regardless of the strategy of *Player* 1 and regardless of the probabilistic choices made in the random vertices. When $\mathcal{G}$ is a 2-player game graph, then $\mathit{Cpre}(S) = \mathrm{Pre}_0^{\exists}(S) \cup \mathrm{Pre}_1^{\forall}(S)$.

2.4.1. Bounded-Horizon Reach-Avoid Games

Suppose $k \in \mathbb{N}$ is a nonnegative time bound and $\varphi = S \,\mathsf{U}^{\leq k} T$ is a bounded-horizon reach-avoid specification for given safe states $S \subseteq \mathcal{X}$ and target states $T \subseteq \mathcal{X}$. Then the winning region $\mathcal{W}^{sure}$ for the $2^1/_2$-player game $\langle \mathcal{G}, \varphi \rangle$ is given by W^k, where W^k can be computed using the following algorithm:

1: $W^0 \leftarrow \emptyset$
2: **repeat for** $l = 0, 1, \ldots$ **do**
3: $\quad W^{l+1} \leftarrow T \cup \big(Cpre(W^l) \cap S\big)$
4: **until** $l = k$
5: $\mathcal{W}^{sure} \leftarrow W^k$

An optimal memoryless winning strategy $\pi_0^* \in \Pi_0^{\mathrm{DM}}$ can be extracted by stitching together the choices made by the *Cpre* operator in each iteration. For every $v \in W^k \cap V_0$, $\pi_0^*(v) = v'$ where (i) if $v \in T$ then v' is any arbitrary vertex in $E(v)$ and (ii) if $v \in W^k \setminus T$ then $v' \in W^{i^*}$ where $i^* = \min\{i \mid v \in W^i \setminus T\} - 1$. Intuitively, the index i^* is a distance-like metric that represents the maximum number of steps it would take for *Player* 0 to reach T from a given vertex $v \in W^{i^*}$, if it uses the optimal strategy π_0^*. The optimal strategy π_0^* ensures that this distance is decreased at every step. Thus, for every vertex in the set $W^k \setminus W^{k-1}$, i.e. when $i^* = k$, it will take at most k steps for the optimal strategy π_0^* to make the game reach the target T.

2.4.2. Reach-Avoid Games

Suppose $\varphi = S \,\mathsf{U}\, T$ is a reach-avoid specification for given safe states $S \subseteq V$ and target states $T \subseteq V$. The winning region $\mathcal{W}^{sure}$ of $\langle \mathcal{G}, \varphi \rangle$ can be computed by repeating the loop for the bounded-horizon reach-avoid game until convergence (the difference is highlighted using the blue box):

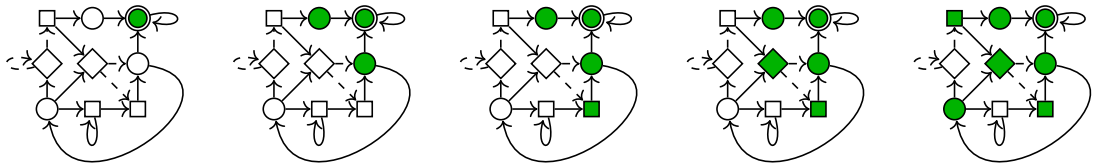
1: $W^0 \leftarrow \emptyset$
2: **repeat for** $l = 0, 1, \ldots$ **do**
3: $\quad W^{l+1} \leftarrow T \cup \big(Cpre(W^l) \cap S\big)$
4: **until** $\boxed{W^l = W^{l+1}}$
5: $\mathcal{W}^{sure} \leftarrow W^l$

The above algorithm is illustrated in Fig. 2.4b. The special case when the set S equals V, known as the *reachability game*, is illustrated in Fig. 2.4a. Note that for the bounded horizon reach-avoid games, we will simply need to terminate the fixpoint iteration after the given bound k. For instance, the green vertices in the third figure give us the winning region for the winning condition $S \,\mathsf{U}^{\leq 2} T$.

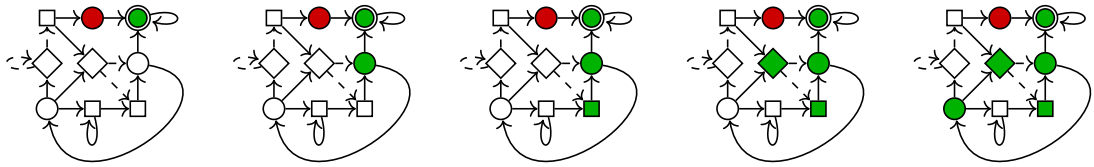
The algorithm for reach-avoid games can be alternatively represented using the following μ-calculus expression:

$$\mathcal{W}^{sure} = \mu X \,.\, T \cup (Cpre(X) \cap S)\,. \tag{2.9}$$

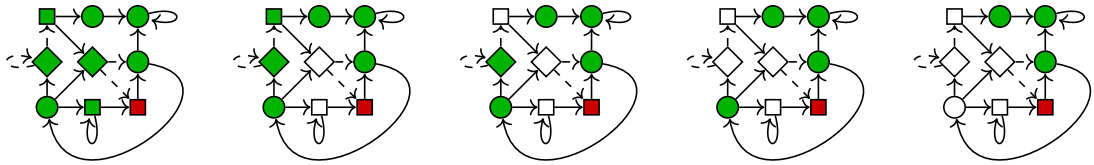
By using the monotonicity property of the fixpoint expression in (2.9) and the finiteness of the set of vertices V, it can be shown that the sequence $\{W^l\}$ converges within finite number of steps. The winning strategy for *Player* 0 can be extracted the same way as described for the bounded-horizon case above.



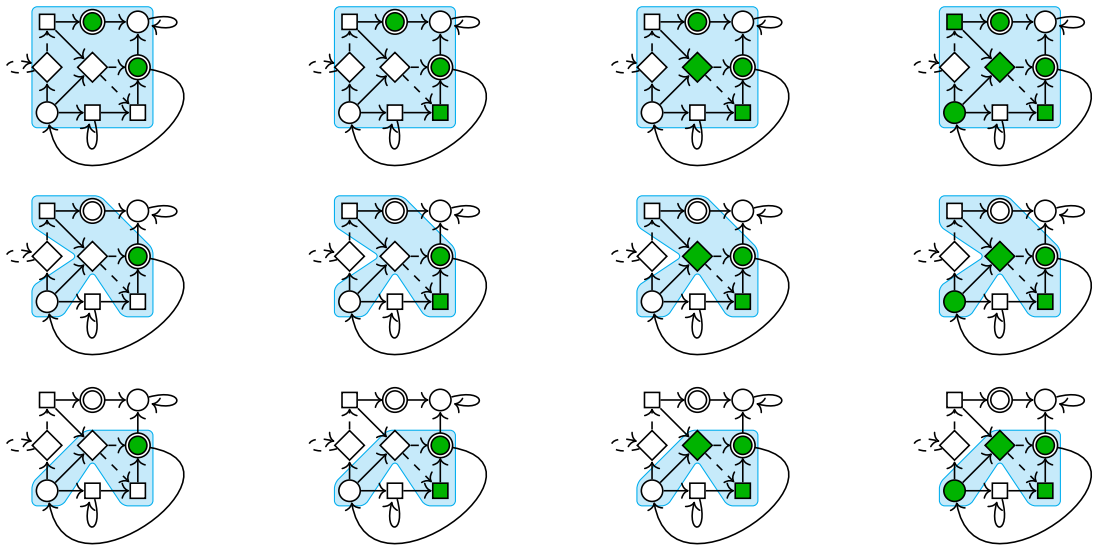
(a) Reachability game (specification: $\Diamond T$), i.e. reach-avoid game with no obstacles, where the target set T equals the vertex with double circle.



(b) Reach-avoid game (specification: $\neg O \,\mathsf{U}\, T$), where the obstacle set O equals the vertex in red and the target set T equals the vertex with double circle.



(c) Safety game (winning condition: $\Box B$), where the safe set B equals vertices *not* in red.



(d) Büchi game (winning condition: $\Box\Diamond B$), where the target set B equals the vertices in double circles. The states inside the blue background constitute the current value of the outer-fixpoint variable.

Figure 2.4.: Illustration of the steps in the iterative symbolic computation of sure winning regions for various $2\frac{1}{2}$-player games. In all the diagrams, the vertices in green represent the current value of the respective fixpoint variable.

2.4.3. Safety Games

Suppose $\varphi = \Box S$ is a safety specification for given safe states $S \subseteq V$. The winning region $\mathcal{W}^{sure}$ for the 2-player game $\langle \mathcal{G}, \varphi \rangle$ can be computed using the following iterative procedure:

1: $W^0 \leftarrow V$
2: **repeat for** $l = 0, 1, \ldots$ **do**
3: $\quad W^{l+1} \leftarrow S \cap Cpre(W^l)$
4: **until** $W^l = W^{l+1}$
5: $\mathcal{W}^{sure} \leftarrow W^l$

The above algorithm is illustrated in Fig. 2.4c. It can be alternatively expressed using the following μ-calculus expression:

$$\mathcal{W}^{sure} = \nu Y \,.\, S \cap Cpre(Y). \tag{2.10}$$

By using the monotonicity property of the fixpoint expression in (2.10) and the finiteness of the set of vertices V, it can be shown that the sequence $\{W^l\}$ converges in finite number of steps. An optimal memoryless strategy $\pi_0^* \in \Pi_0^{\mathrm{DM}}$ can be extracted from $\mathcal{W}^{sure}$ by assigning to every $v \in \mathcal{W}^{sure}$ a successor in $\mathcal{W}^{sure}$, i.e. $\pi_0^*(v) = v' \in E(v) \cap \mathcal{W}^{sure}$.

2.4.4. Büchi Games

Suppose $\varphi = \Box\Diamond B$ is a Büchi specification for a given set of target states $B \subseteq V$. For $l \in \mathbb{N}$, let W^l be a set of vertices and Y^l be a function mapping a set of vertices to another set of vertices. The winning region $\mathcal{W}^{sure}$ for the 2-player game $\langle \mathcal{G}, \varphi \rangle$ can be computed using the following algorithm:

1: $W^0 \leftarrow V$
2: **repeat for** $l = 0, 1, \ldots$ **do**
3: $\quad Y^0 \leftarrow \emptyset$
4: $\quad$ **repeat for** $j = 0, 1, \ldots$ **do**
5: $\quad\quad Y^{j+1} \leftarrow \left(B \cap Cpre(W^l)\right) \cup Cpre\left(Y^j\right)$
6: $\quad$ **until** $Y^j = Y^{j+1}$
7: $\quad W^{l+1} \leftarrow Y^j$
8: **until** $W^l = W^{l+1}$
9: $\mathcal{W}^{sure} \leftarrow W^l$

The above algorithm is illustrated in Fig. 2.4d. It can be equivalently expressed using the following μ-calculus expression:

$$\mathcal{W}^{sure} = \nu W \,.\, \mu Y \,.\, (B \cap Cpre(W)) \cup Cpre(Y). \tag{2.11}$$

By using the monotonicity property of the fixpoint expression in (2.10) and the finiteness of the set of vertices V, it can be shown that (1) for a given W^l, the sequence $\{Y^j\}$ converges in finite number of steps, and (2) the sequence $\{W^l\}$ converges in finite number of steps. An optimal memoryless strategy $\pi_0^* \in \Pi_0^{\mathrm{DM}}$ can be extracted from $\mathcal{W}^{sure}$ by

assigning to every $v \in \mathcal{W}^{sure} \cap B$ a successor in $\mathcal{W}^{sure}$, i.e. $\pi_0^*(v) = v' \in E(v) \cap \mathcal{W}^{sure}$, and to every $v \in \mathcal{W}^{sure} \setminus B$ a successor that reduces the distance to $\mathcal{W}^{sure} \cap B$ by 1 step (similar to the bounded-horizon reach-avoid game).

2.4.5. Parity Games

Let $\mathcal{P} = \{B_0, B_1, \ldots, B_{2k}\}$ be a set of atomic propositions, representing priorities, and $Parity(\mathcal{P})$ be the respective parity condition. The parity game can be solved using the following $2k$-nested iterative procedure:

1: $Y_{2k}^0 \leftarrow V$
2: **repeat for** $l_{2k} = 0, 1, \ldots$ **do**
3: $\quad X_{2k-1}^0 \leftarrow \emptyset$
4: $\quad$ **repeat for** $l_{2k-1} = 0, 1, \ldots$ **do**
$\quad\quad \vdots$
5: $\quad\quad Y_2^0 \leftarrow V$
6: $\quad\quad$ **repeat for** $l_2 = 0, 1, \ldots$ **do**
7: $\quad\quad\quad X_1^0 \leftarrow \emptyset$
8: $\quad\quad\quad$ **repeat for** $l_1 = 0, 1, \ldots$ **do**
9: $\quad\quad\quad\quad X_1^{l_1+1} \leftarrow (C_1 \cap Cpre(X_1^{l_1})) \cup (C_2 \cap Cpre(Y_2^{l_2})) \cup (C_3 \cap Cpre(X_3^{l_3})) \ldots \cup (C_{2k} \cap Cpre(Y_{2k}^{l_{2k}}))$
10: $\quad\quad\quad$ **until** $X_1^{l_1} = X_1^{l_1+1}$
11: $\quad\quad\quad Y_2^{l_2+1} \leftarrow X_1^{l_1+1}$
12: $\quad\quad$ **until** $Y_2^{l_2} = Y_2^{l_2+1}$
$\quad\quad \vdots$
13: $\quad$ **until** $X_{2k-1}^{l_{2k-1}} = X_{2k-1}^{l_{2k-1}+1}$
14: $\quad Y_{2k}^{2k} \leftarrow X_{2k-1}^{l_{2k-1}}$
15: **until** $Y_{2k}^{2k} = Y_{2k}^{2k+1}$
16: $\mathcal{W}^{sure} \leftarrow Y_{2k}^{2k}$

The above algorithm can be equivalently expressed using the following μ-calculus expression:

$$\mathcal{W}^{sure} = \nu Y_{2k}.\ \mu X_{2k-1} \ldots \nu Y_2.\ \mu X_1. \quad (2.12)$$
$$(C_1 \cap Cpre(X_1)) \cup (C_2 \cap Cpre(Y_2)) \cup (C_3 \cap Cpre(X_3)) \ldots \cup (C_{2k} \cap Cpre(Y_{2k})).$$

The extraction of winning strategies for parity games has been considered by Bruse et al. (2014), and is much more involved than the procedure for reachability, safety, and Büchi games. Their algorithm works as follows. During the fixpoint computation, for every vertex that is added in the winning region, they store a winning move together with the time stamp of the iteration count at which it was added to the winning region. After the fixed point is reached, they perform a post-analysis of all the stored information, and

construct a memoryless winning strategy. Their algorithm has been implemented in the tool PGSolver[1], along with some optimization heuristics.

2.4.6. Rabin Games

Let $G_1, R_1, \ldots, G_k, R_k \in 2^{\Sigma}$ be a set of atomic propositions over the alphabet Σ, and $Rabin(\mathcal{R})$ be the corresponding Rabin specification. The sure winning region for the Rabin game $\langle \mathcal{G}, Rabin(\mathcal{R}) \rangle$ can be computed using the following recursive procedure (Piterman and Pnueli, 2006):

```
   Func Rabin_main(Pairs)
1: Z^0 ← ∅
2: repeat for i = 0, 1, ... do
3:     p1 ← Cpre(Z^i)
4:     Z^{i+1} ← Rabin(Pairs, true, p1)
5: until Z^i = Z^{i+1}
6: W^sure ← Z^i
7: return W^sure
   Func Rabin(Pairs, seqnr, right)
1: U ← ∅
2: for p = 1, 2, ..., k do
3:     newPairs ← pairs \ ⟨G_p, R_p⟩
4:     Y^0 ← V
5:     repeat for i = 0, 1, ... do
6:         p2 ← right ∪ (seqnr ∩ ¬R_p ∩ G_p ∩ Cpre(Y^i))
7:         X^0 ← ∅
8:         repeat for j = 0, 1, ... do
9:             p3 ← p2 ∪ (seqnr ∩ ¬R_p ∩ Cpre(X^j))
10:            if newSet is empty then
11:                X^{j+1} ← p3
12:            else
13:                X^{j+1} ← Rabin(newPairs, seqnr ∩ ¬R_p, p3)
14:            end if
15:        until X^j = X^{j+1}
16:        Y^{i+1} ← X^j
17:    until Y^i = Y^{i+1}
18:    U ← U ∪ Y^i
19: end for
20: return U
```

The above algorithm can be equivalently expressed using the following μ-calculus formula:

[1] https://github.com/tcsprojects/pgsolver

$$\nu Y_{p_0}.\mu X_{p_0}.\bigcup_{p_1\in P}\nu Y_{p_1}.\mu X_{p_1}.\bigcup_{p_2\in P\setminus\{p_1\}}\nu Y_{p_2}.\mu X_{p_2}.\ldots.\bigcup_{p_k\in P\setminus\{p_1,\ldots,p_{k-1}\}}\nu Y_{p_k}.\mu X_{p_k}.\left[\bigcup_{j=0}^{k}\mathcal{C}_{p_j}\right], \tag{2.13a}$$

$$\text{where}\quad \mathcal{C}_{p_j} \coloneqq \left(\bigcap_{i=0}^{j}\overline{R}_{p_i}\right)\cap\left[\left(G_{p_j}\cap \mathit{Cpre}(Y_{p_j})\right)\cup\left(\mathit{Cpre}(X_{p_j})\right)\right], \tag{2.13b}$$

Piterman and Pnueli (2006) presented the strategy extraction procedure from the sure winning region in Rabin games. Their algorithm assigns a vector valued rank to every vertex in $\mathcal{W}^{sure}$, containing a priority ordering over the Rabin pairs and a vector of the worst possible numbers of steps until visits to the G_i sets (while avoiding the R_i sets that belong to Rabin pairs with lower priority according to the priority order). The winning strategy is constructed by assigning to every winning *Player* 0 vertex the successor with the minimum rank (according to lexicographic order).

2.5. Conclusion

We summarized several different fragments of LTL, which will be mainly used to specify linear time behavioral properties for various types of CPS models (continuous-time control systems, transition systems, etc.) in the rest of thesis. We defined the LTL formulas over some given finite alphabet. Later, when we use LTL formulas as properties of CPS models, the alphabet will be same as the set of "states" of the particular model.

We also discussed two types of finite graph games, namely $2\frac{1}{2}$-player games and 2-player games. For each type of game, we introduced the sure winning and the almost sure winning mode for the protagonist *Player* 0. (For 2-player games the two winning modes coincide.) The game is sure winning for *Player* 0 from a certain initial vertex if *Player* 0 has a strategy that is winning against every strategy of *Player* 1. The game is almost sure winning for *Player* 0 from a certain initial vertex if *Player* 1 has a strategy that is winning with probability 1 against every strategy of *Player* 1. Computation of sure winning vertices and strategies can be performed using known fixpoint algorithms that we summarized in this chapter. Computation of almost sure winning vertices and strategies can be performed using our novel fixpoint algorithms presented in Chap. 7. All these fixpoint algorithms will form the algorithmic basis for the core correct-by-construction synthesis questions discussed in the rest of the thesis.

Part I.

Continuous Systems and Finite Abstractions

3. Abstraction-Based Controller Design (ABCD) for Dynamical Systems

Part I of this thesis deals with building sound discrete abstractions of continuous dynamical systems for designing correct-by-construction-controllers. There are now a variety of such abstraction-based controller design (ABCD) methods in the literature (Girard and Pappas, 2007; Pola et al., 2008; Girard et al., 2010; Girard, 2012; Nilsson et al., 2017; Reissig et al., 2017). The general work-flow in all these ABCD methods is as follows: First, a time-sampled version of the continuous dynamics of the open-loop system (the *concrete system*) is abstracted by a symbolic finite state model (the *abstract system*). Second, automata-theoretic algorithms from finite-state reactive synthesis are used to synthesize a discrete controller on the abstract system for a given temporal logic specification. Finally, provided certain simulation relation holds between the concrete and abstract systems, the abstract controller can be refined back to a controller for the concrete system. The main differences between the existing approaches from the literature are in the type of concrete system that they can handle, the type of simulation relation they use between the concrete and the abstract systems, and the mechanism they use accordingly to build the abstract system; a survey of the existing approaches can be found at the end of this chapter.

In this chapter, we discuss a recent ABCD approach using feedback refinement relations (FRR) between the concrete and abstract systems (Reissig et al., 2017). One of the main advantages of FRR-based ABCD is that the refinement procedure becomes extremely simple and almost immediate. Moreover, it supports a very broad category of nonlinear systems without requiring any stability assumptions, unlike many other approaches in the literature.

For ensuring FRR between the concrete and abstract systems, the latter is computed by first fixing a parameter τ for the sample time and a parameter η for the state and input spaces, and then representing the abstract state space as a set of hypercubes, each of diameter η. The hypercubes partition the continuous concrete state space. The abstract transition relation adds a transition between two hypercubes iff there exists some state in the first hypercube which can reach some state of the second by following the original dynamics for time τ. This results in a transition system that over-approximates the effect of the original dynamics on the abstract state space.

The FRR-based ABCD approach was first implemented in the tool called `SCOTS` (Rungger and Zamani, 2016), and later was used in our tools `Mascot` and `Mascot-SDS`.

3.1. Continuous-Time Control Systems

Definition 3.1 (Continuous-Time Control Systems) A *control system* $\mathcal{S} = \langle \mathcal{X}, \mathcal{U}, \mathcal{W}, f \rangle$ consists of a state space $\mathcal{X} = \mathbb{R}^n$, a non-empty input space $\mathcal{U} \subseteq \mathbb{R}^m$, a disturbance space $\mathcal{W} = [\![-w, w]\!]$ such that $w \in \mathbb{R}^n_{\geq 0}$, and a nonlinear differential inclusion

$$\frac{d\xi(t)}{dt} \in f(\xi(t), u(t)) + \mathcal{W}, \tag{3.1}$$

where $f(\cdot, u)$ fulfills the usual conditions for existence and uniqueness of solution of the differential equation $\frac{d\xi(t)}{dt} = f(\xi(t), u(t))$. (For example, a sufficient condition for existence and uniqueness is that $f(\cdot, u)$ is locally Lipschitz continuous for all $u \in U$.)

A system $\mathcal{S}$ defines a perturbed continuous-time nonlinear system, and w is a component-wise bound on perturbations to its dynamics.

Given a continuous control input function $\mu \colon \mathbb{R}_{\geq 0} \to \mathcal{U}$ which maps every $t \in \mathbb{R}_{\geq 0}$ to a control input in $\mu(t) \in \mathcal{U}$, a solution of the inclusion in (3.1) is an absolutely continuous function $\xi \colon \mathbb{R}_{\geq 0} \to \mathcal{X}$ that fulfills (3.1) for almost every $t \in \mathbb{R}_{\geq 0}$. For such a μ and ξ, we call the pair $\langle \xi, \mu \rangle$ a *trajectory* and call ξ a *path* of $\mathcal{S}$. Observe that when $w = 0$, $\mathcal{W}$ is a singleton set, and we have a unique trajectory for a given μ. On the other hand when $w > 0$, we have many different trajectories for a given μ.

Definition 3.2 (Controller, Closed-Loop, and Closed-Loop Behavior) Let $\mathcal{S} = \langle \mathcal{X}, \mathcal{U}, \mathcal{W}, f \rangle$ be a continuous-time control system. A controller for $\mathcal{S}$ is a function $\mathcal{C} \colon \mathcal{X} \to \mathcal{U}$. The closed-loop formed by interconnecting $\mathcal{C}$ in feedback with $\mathcal{S}$ is itself a control system $\mathcal{S}^{\mathsf{cl}} = \langle \mathcal{X}, \{\alpha\}, \mathcal{W}, f^{\mathsf{cl}} \rangle$ where α is a dummy control input symbol and $f^{\mathsf{cl}}(x, \alpha) \equiv f(x, \mathcal{C}(x))$ for every $x \in \mathcal{X}$. The *closed-loop behavior* $\mathcal{B}(\mathcal{S}^{\mathsf{cl}})$ is the set of all paths of the closed-loop system $\mathcal{S}^{\mathsf{cl}}$.

3.2. The Control Problem

A control problem is the problem of computing the best-possible controller for a continuous-time control system such that a given $\mathrm{LTL}_{\setminus\bigcirc}$ specification is fulfilled. To state the problem precisely, we first need to generalize the semantics of an $\mathrm{LTL}_{\setminus\bigcirc}$ formula from the discrete sequences of alphabet symbols to continuous signals. Suppose φ is an $\mathrm{LTL}_{\setminus\bigcirc}$ formula (see Sec. 2.2.1 and Sec. 2.2.2 for the syntax) over a finite alphabet Σ and $w \colon \mathbb{R}_{\geq 0} \to \Sigma$ is a continuous function mapping every time point $t \in \mathbb{R}_{\geq 0}$ to a symbol in the alphabet Σ. Let us generalize the projection operator over w as follows: For any continuous time range $[t, t') \subseteq \mathbb{R}_{\geq 0}$, $w|_{[t,t')}$ is a continuous function $w' \colon [t, t') \to \Sigma$ such that $w'(s) \coloneqq w(s)$

for every $s \in [t, t')$. The continuous-time semantics are defined as follows:

$$
\begin{aligned}
&w \models \mathsf{true}, \\
&w \models p && \text{if } w(0) \in p, \\
&w \models \neg\varphi && \text{if } w \not\models \varphi, \\
&w \models \varphi_1 \wedge \varphi_2 && \text{if } w \models \varphi_1 \text{ and } w \models \varphi_2, \\
&w \models \varphi_1 \,\mathsf{U}\, \varphi_2 && \text{if } \exists t \in [0,\infty) \cdot w|_{[t,\infty)} \models \varphi_2 \text{ and } \forall t' \in [0,t) \cdot w|_{[t',\infty)} \models \varphi_1.
\end{aligned}
$$

The set of all the *continuous* functions satisfying φ is denoted as $\{\!\![\varphi]\!\!\}$ i.e. $\{\!\![\varphi]\!\!\} := \{w \colon \mathbb{R}_{\geq 0} \to \Sigma \mid w \models \varphi\}$.

Definition 3.3 (Control Problem and Optimal Controller) Let $\mathcal{S} = \langle \mathcal{X}, \mathcal{U}, \mathcal{W}, f \rangle$ be a continuous-time control system. For a given $\mathrm{LTL}_{\setminus\bigcirc}$ formula φ over a finite set of atomic propositions AP that is a set of set of states of $\mathcal{S}$ (i.e. $AP \subset 2^{\mathcal{X}}$), the pair $\langle \mathcal{S}, \varphi \rangle$ is called a *control problem*. The formula φ is called the *control specification* or simply *specification*. An *optimal controller* $\mathcal{C}$ for the control problem $\langle \mathcal{S}, \varphi \rangle$ is a controller such that the following hold:

(A) every closed-loop path satisfies φ, i.e. $\mathcal{B}(\mathcal{S}^{\mathsf{cl}}) \subseteq \{\!\![\varphi]\!\!\}_{\mathcal{X}}$, and

(B) for every other controller $\mathcal{C}'$ that satisfies (A), we have $\mathrm{dom}(\mathcal{C}') \subseteq \mathrm{dom}(\mathcal{C})$.

Optimal controller for a given control problem is not unique, and we use the notation $\mathit{OptCtrl}(\mathcal{S}, \varphi)$ to denote the set of all optimal controllers for the control problem $\langle \mathcal{S}, \varphi \rangle$.

Problem 1 (Controller Synthesis for Continuous-Time Control Systems.) *Let $\mathcal{S} = \langle \mathcal{X}, \mathcal{U}, \mathcal{W}, f \rangle$ be a continuous-time control system, $AP \subset 2^{\mathcal{X}}$ be a finite set of atomic propositions over the alphabet $\mathcal{X}$, and φ be a $\mathrm{LTL}_{\setminus\bigcirc}$ specification over AP. The controller synthesis problem asks to find an optimal controller $\mathcal{C} \in \mathit{OptCtrl}(\mathcal{S}, \varphi)$ for the control problem $\langle \mathcal{S}, \varphi \rangle$.*

3.2.1. A Sampled-Time Approximation of The Control Problem

In this thesis, we will approximately solve the above problem by abstracting away to a discrete time domain through sampling. We will restate our approximate problem statement after introducing the necessary concepts and the sampled-time abstraction of the system $\mathcal{S}$.

Definition 3.4 (Transition systems) A *transition system* $\mathcal{T} = \langle \mathcal{X}, \mathcal{U}, \mathcal{Y}, F, H \rangle$ consists of a state space $\mathcal{X}$, an input space $\mathcal{U}$, an output space $\mathcal{Y}$, and set-valued maps $F \colon \mathcal{X} \times \mathcal{U} \to 2^{\mathcal{X}}$ and $H \colon \mathcal{X} \times \mathcal{U} \to 2^{\mathcal{Y}}$ representing the transition function and the output labeling function, respectively. A transition system $\mathcal{T}$ is *finite* if $\mathcal{X}$, $\mathcal{U}$, and $\mathcal{Y}$ are finite. It is *simple* if $\mathcal{X} = \mathcal{Y}$ and $H(x, u) = x$ for all $x \in \mathcal{X}$ and $u \in \mathcal{U}$, and *static* if $\mathcal{X}$ is a singleton. If $\mathcal{T}$ is simple (resp. static) we use the triple $\mathcal{T} = \langle \mathcal{X}, \mathcal{U}, F \rangle$ (respectively $\mathcal{T} = \langle \mathcal{U}, \mathcal{Y}, H \rangle$

with $H : \mathcal{U} \to 2^{\mathcal{Y}}$) for notational convenience. The *behavior* $\mathcal{B}(\mathcal{T})$ of a transition system $\mathcal{T} = \langle \mathcal{X}, \mathcal{U}, \mathcal{Y}, F, H \rangle$ is given by the set

$$\{\xi \in \mathcal{X}^\infty \mid \forall k \in \mathrm{dom}^+(\xi) \,.\, \xi^k \in \textstyle\bigcup_{u \in \mathcal{U}} F(\xi^{k-1}, u)\}. \tag{3.2}$$

Every element in $\mathcal{B}(\mathcal{T})$ is a *path* of the transition system $\mathcal{T}$.

Like continuous-time control systems, we also define controllers and the associated concepts for transition systems. Given a simple transition system $\mathcal{T} = \langle \mathcal{X}, \mathcal{U}, F \rangle$, a *controller* (to be precise, a state-feedback controller) C of $\mathcal{T}$ is a function $C : \mathcal{X} \to \mathcal{U}$. Given a transition system $\mathcal{T} = \langle \mathcal{X}, \mathcal{U}, F \rangle$ and a controller C, the *closed-loop system* formed by interconnecting $\mathcal{T}$ and C in *feedback* is defined by the system $\mathcal{T}^{\mathsf{cl}} = \left\langle \mathcal{X}, \mathcal{U}, F^{\mathsf{cl}} \right\rangle$ where[1]

$$F^{\mathsf{cl}}(x, u) = \left\{ x' \;\middle|\; \begin{array}{l} u \in C(x) \wedge \\ x' \in F(x, u) \end{array} \right\}. \tag{3.3}$$

Given a transition system $\mathcal{T}$ and a specification φ over a set of atomic propositions $AP \subseteq 2^{\mathcal{X}}$, a control problem is the pair $\langle \mathcal{T}, \varphi \rangle$. An optimal controller $\mathcal{C}$ for the control problem $\langle \mathcal{T}, \varphi \rangle$ is a controller $\mathcal{C}$ such that the conditions A and B in Def. 3.3 is satisfied by $\mathcal{C}$. Just like continuous control systems, we use the notation $\mathit{OptCtrl}(\mathcal{T}, \varphi)$ denote the set of optimal controllers for the control problem $\langle \mathcal{T}, \varphi \rangle$.

Definition 3.5 (Sampled-time abstraction) Given a time sampling parameter $\tau > 0$, we define by $\vec{\mathcal{S}}(\mathcal{S}, \tau) \coloneqq \left\langle \mathcal{X}, \mathcal{U}, \vec{F} \right\rangle$ the simple transition system associated with $\mathcal{S}$ and τ, where

$$x' \in \vec{F}(x, u) \Leftrightarrow \exists \xi \in \mathrm{Sol}_f(\tau, u) \,.\, \xi(0) = x \wedge \xi(\tau) = x'. \tag{3.4}$$

The system $\vec{\mathcal{S}}(\mathcal{S}, \tau)$ is called the *sampled-time abstraction* of $\mathcal{S}$.

The state space of $\vec{\mathcal{S}}(\mathcal{S}, \tau)$ is still infinite; we next define a finite system associated with $\mathcal{S}$. If the control system $\mathcal{S}$ and parameter τ is clear from the context, we omit it in $\vec{\mathcal{S}}$.

Problem 2 (Sampled-Time Approximate Controller Synthesis Problem.) *Let $\mathcal{S} = \langle \mathcal{X}, \mathcal{U}, \mathcal{W}, f \rangle$ be a continuous-time control system, $\tau > 0$ be a sampling parameter, $\vec{\mathcal{S}}(\mathcal{S}, \tau)$ be the sampled-time abstraction, $AP \subset 2^{\mathcal{X}}$ be a finite set of atomic propositions over the alphabet $\mathcal{X}$, and φ be a $\mathrm{LTL}_{\setminus \bigcirc}$ specification over AP. The sampled-time approximation of Prob 1 asks to find an optimal controller $\mathcal{C} \in \mathit{OptCtrl}(\vec{\mathcal{S}}, \varphi)$ for the sampled-time control problem $\left\langle \vec{\mathcal{S}}, \varphi \right\rangle$.*

[1] In contrast to the definition used in Reissig et al. (2017), we keep the input used in F^{cl} explicit. This allows us to apply feedback refinement relations to closed loop systems.

3.3. Feedback Refinement Relations

We now recall the general procedure of abstraction-based controller synthesis (ABCD) using the framework of feedback refinement relations (FRR) as introduced by Reissig et al. (2017).

Definition 3.6 (Feedback refinement relation or FRR) Let $\mathcal{T}_i = \langle \mathcal{X}_i, \mathcal{U}_i, F_i \rangle$, $i \in \{1, 2\}$ be two simple transition systems, and suppose $\mathcal{U}_2 \subseteq \mathcal{U}_1$. A *feedback refinement relation* (FRR) from $\mathcal{T}_1$ to $\mathcal{T}_2$ is a total relation $Q \subseteq \mathcal{X}_1 \times \mathcal{X}_2$ such that for all $(x_1, x_2) \in Q$, we have (i) $\mathcal{U}_2(x_2) \subseteq \mathcal{U}_1(x_1)$, and (ii) $u \in \mathcal{U}_2(x_2) \Rightarrow Q(F_1(x_1, u)) \subseteq F_2(x_2, u)$. We write $\mathcal{T}_1 \preccurlyeq_Q \mathcal{T}_2$ if Q is an FRR from $\mathcal{T}_1$ to $\mathcal{T}_2$.

The specification is also lifted to the level of the abstraction, which is called the abstract specification (Reissig et al., 2017, Def. VI.2) .

Definition 3.7 (Abstract specification) Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two transition systems with Q being an FRR from $\mathcal{T}_1$ to $\mathcal{T}_2$, and let φ_1 be a specification for $\mathcal{T}_1$. Then a specification φ_2 for $\mathcal{T}_2$ will be called an *abstract specification* (with respect to Q) of φ_1 if for every $\xi_2 \in \langle\!\langle \varphi_2 \rangle\!\rangle_{\mathcal{X}_2}$ and every $\xi_1 \in \mathcal{X}_1^\infty$, (a) implies (b):

(a) For every $k \in \mathbb{N}$, $(\xi_1^k, \xi_2^k) \in Q$.

(b) $\xi_1 \in \langle\!\langle \varphi_1 \rangle\!\rangle_{\mathcal{X}_1}$.

If φ_2 is an abstract specification then we write $\langle \mathcal{T}_1, \varphi_1 \rangle \preccurlyeq_Q \langle \mathcal{T}_2, \varphi_2 \rangle$.

Definition 3.8 (Controller refinement) Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two transition systems with Q being an FRR from $\mathcal{T}_1$ to $\mathcal{T}_2$, and let $\mathcal{C}_2$ be a controller for $\mathcal{T}_2$. The refinement of the controller $\mathcal{C}_2$ for $\mathcal{T}_1$ is given by the function composition $\mathcal{C}_1 = \mathcal{C}_2 \circ Q$.

As shown by Reissig et al. (2017, Thm. VI.3), the refinement is sound.

Theorem 3.1 (Soundness) *Consider two transition systems $\mathcal{T}_1$ and $\mathcal{T}_2$ with their respective control specifications φ_1 and φ_2. Let $\langle \mathcal{T}_1, \varphi_1 \rangle \preccurlyeq_Q \langle \mathcal{T}_2, \varphi_2 \rangle$ for some FRR Q, and let the abstract controller $\mathcal{C}_2$ realizes ψ_2 on $\mathcal{T}_2$. Then the refined controller $\mathcal{C}_2 \circ Q$ realizes φ_1 on $\mathcal{T}_1$.*

3.4. Finite Abstraction

The computation of the finite abstraction from the given continuous-time system happens in two phases: First, we obtain a sampled-time abstraction of the given continuous-time system, by discretizing the continuous time set using a fixed sampling time. Second, we obtain a finite abstraction from the sampled-time abstraction, by discretizing the continuous state space into finitely many partition elements.

The abstraction of continuous-time systems uses a reachable set computation, formalized using a growth bound on the dynamics in (3.1). Given a positive parameter $\tau > 0$ and

a constant continuous input function $\mu_u\colon [0,\tau] \to \mathcal{U}$ which maps every $t \in [0,\tau]$ to a fixed input $u \in \mathcal{U}$, a solution of the inclusion in (3.1) on $[0,\tau]$ is an absolutely continuous function $\xi\colon [0,\tau] \to \mathcal{X}$ that fulfills (3.1) for almost every $t \in [0,\tau]$. We collect all such solutions in the set $\mathrm{Sol}_f(\tau,u)$. Given an initial condition $x_0 \in \mathcal{X}$, the solution to the unperturbed control system $\dot{\xi} = f(\xi(t),u(t))$ associated with (3.1) is unique, and its value at time $t \in [0,\tau]$ is denoted by $\zeta(t,x_0,\mu)$. Given a subset of states $X \subseteq \mathcal{X}$, and a subset of inputs $U \subseteq \mathcal{U}$ such that $[0,\tau] \times X \times U \subseteq \mathrm{dom}(\zeta)$, the map $\beta_\tau\colon \mathbb{R}^n_{\geq 0} \times U \to \mathbb{R}^n_{\geq 0}$ is a *growth bound* on X and U associated with τ and (3.1) if

$$\begin{aligned}&\forall r, r' \in \mathbb{R}^n_{\geq 0}, u \in U \,.\, r \geq r' \implies \beta_\tau(r,u) \geq \beta_\tau(r',u) \text{ and}\\ &\forall\, \xi \in \mathrm{Sol}_f(\tau,u), x_0 \in X \,.\\ &\quad \xi(0) \in X \implies |\xi(\tau) - \zeta(\tau,x_0,\mu)| \leq \beta_\tau(|\xi(0)-x_0|,u).\end{aligned}$$

A *cover* $\widehat{\mathcal{X}}$ of the state space $\mathcal{X}$ is a set of non-empty, closed hyper-intervals $[\![a,b]\!]$ with $a,b \in (\mathbb{R} \cup \{\pm\infty\})^n$ called *cells*, such that every $x \in \mathcal{X}$ belongs to some cell in $\widehat{\mathcal{X}}$. We adopt the convention that whenever we compare a set of *cells* $\widehat{\mathcal{X}}' \subseteq \widehat{\mathcal{X}}$ with a set of *states* $\mathcal{X}' \subseteq \mathcal{X}$, we actually refer to the set of states covered by the cells in $\mathcal{X}'$. For example, $\widehat{\mathcal{X}}' \subseteq \mathcal{X}'$ would actually mean $\cup_{\widehat{x} \in \widehat{\mathcal{X}}'} \widehat{x} \subseteq \mathcal{X}'$.

We assume that there exists a compact subset $\mathcal{X}' \subseteq \mathcal{X}$ of the state space, which is quantized by compact cells, whereas the (unbounded) region covered by $\widehat{\mathcal{X}} \setminus \widehat{\mathcal{X}}'$ is not of interest to the control problem and is covered by a finite number of large unbounded cells.

Given a *grid parameter* $\eta \in \mathbb{R}^n_{>0}$ and $\mathcal{X}' \subseteq \mathcal{X}$ with $\mathcal{X}' = [\![\alpha,\beta]\!]$ such that $\beta - \alpha = k\eta$ for some $k \in \mathbb{Z}^n$, the set

$$\eta\mathbb{Z}^n = \{c \in \mathcal{X}' \mid \exists k \in \mathbb{Z}^n.(\forall i \in [1;n].c_i = \alpha_i + k_i\eta_i - 0.5 * \eta_i)\} \tag{3.5}$$

defines the center points of cells in $\widehat{\mathcal{X}}'$ with diameter η, i.e.

$$\widehat{x} \in \widehat{\mathcal{X}}' \Rightarrow \exists c \in \eta\mathbb{Z}^n \,.\, \widehat{x} = c + [\![-\eta/2, \eta/2]\!]. \tag{3.6}$$

This results in congruent cells which are uniformly aligned on a grid.[1] We denote by $c_{\widehat{x}}$ the unique center point of $\widehat{x}$.

Definition 3.9 (Finite abstraction) The simple transition system $\widehat{\mathcal{S}}(\mathcal{S},\tau,\eta,\beta_\tau) \coloneqq \left\langle \widehat{\mathcal{X}}, \widehat{\mathcal{U}}, \widehat{F} \right\rangle$ is called a *finite abstraction* of $\mathcal{S}$ with the associated parameters τ, η, and β_τ if the following holds: (i) $\widehat{\mathcal{X}}$ is a finite cover of $\mathcal{X}$, there exists a non-empty subset $\widehat{\mathcal{X}}' \subseteq \widehat{\mathcal{X}}$ such that $\widehat{\mathcal{X}}'$ satisfies (3.6), and β_τ is a growth bound on $\widehat{\mathcal{X}}'$ and $\widehat{\mathcal{U}}$, (ii) $\widehat{\mathcal{U}}$ is a finite subset of $\mathcal{U}$, (iii) for all $\widehat{x} \in \widehat{\mathcal{X}} \setminus \widehat{\mathcal{X}}'$ and $u \in \widehat{\mathcal{U}}$, $\widehat{F}(\widehat{x},u) = \emptyset$, and (iv) for all $\widehat{x} \in \widehat{\mathcal{X}}'$, $\widehat{x}' \in \widehat{\mathcal{X}}$, and $u \in \widehat{\mathcal{U}}$, $\widehat{x}' \in \widehat{F}(\widehat{x},u)$ if and only if

$$\left(\zeta(\tau, c_{\widehat{x}}, u) + [\![-\beta_\tau(\frac{\eta}{2},u), \beta_\tau(\frac{\eta}{2},u)]\!]\right) \cap \widehat{x}' \neq \emptyset. \tag{3.7}$$

[1] In the original paper by Reissig et al. (2017) and in the standard distribution of their tool `SCOTS`, the grid is aligned such that (an extension of) $\eta\mathbb{Z}^n$ has a center point that coincides with the origin. Shifting this grid by $\eta/2$ yields our grid alignment.

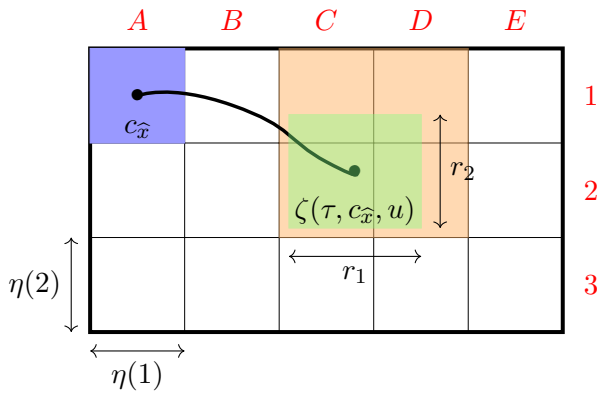


Figure 3.1.: Computation of abstract transitions based on growth bound: the blue box $(A1)$ is the abstract state $\widehat{x}$, the green box is the region $\big(\zeta(\tau, c_{\widehat{x}}, u) + [\![-\beta_\tau(\frac{\eta}{2}, u), \beta_\tau(\frac{\eta}{2}, u)]\!]\big)$, where we used the notation r_1, r_2 to denote the first and the second component of $2\beta_\tau(\frac{\eta}{2}, u)$, and the 4 orange cells $(C1, C2, D1, D2)$ constitute the set $\widehat{\theta}(\widehat{x}, u)$.

The computation of $\widehat{F}(\widehat{x}, u)$ has been illustrated in Fig. 3.1. If the control system $\mathcal{S}$ and parameters τ, η, and β_τ are clear from the context, we omit them in $\widehat{\mathcal{S}}$.

Remark 1 (Uniform growth bound) *We consider a uniform growth bound for notational simplicity. However, a uniform growth bound can be too restrictive for systems whose dynamics vary substantially in different parts of the states. To address this issue, the more general nonuniform growth bounds have been used in the literature (Weber et al., 2016). Another use of nonuniform growth bounds is in abstractions which smartly adapt to local changes in the environment (Bai and Mallik, 2020; Bai et al., 2019).*

It was shown by Reissig et al. (2017, Thm. VIII.4) that there is a simple FRR from $\vec{\mathcal{S}}$ to $\widehat{\mathcal{S}}$:

Theorem 3.2 (FRR from $\vec{\mathcal{S}}$ to $\widehat{\mathcal{S}}$) *The relation $\widehat{Q} \subseteq \mathcal{X} \times \widehat{\mathcal{X}}$ defined by $(x, \widehat{x}) \in \widehat{Q}$ if and only if $x \in \widehat{x}$ is an FRR between $\vec{\mathcal{S}}$ and $\widehat{\mathcal{S}}$, i.e., $\vec{\mathcal{S}} \preccurlyeq_{\widehat{Q}} \widehat{\mathcal{S}}$.*

In view of Thm. 3.2, we can apply ABCD by computing a controller for $\widehat{\mathcal{S}}$ which can then be refined to a controller for $\vec{\mathcal{S}}$ under the pre-conditions of Thm. 3.1.

3.5. The Synthesis Game

Let $\mathcal{S}$ be a continuous-time system, $\vec{\mathcal{S}}(\mathcal{S}, \tau)$ and $\widehat{\mathcal{S}}(\mathcal{S}, \tau, \eta, \beta_\tau)$ be respectively the sampled-time abstraction and finite abstraction of $\mathcal{S}$ such that $\vec{\mathcal{S}} \preccurlyeq_{\widehat{Q}} \widehat{\mathcal{S}}$. We consider control specifications for $\vec{\mathcal{S}}$ expressed using LTL formulas. Suppose φ is such an LTL formula defined using a set of atomic propositions $AP \subseteq 2^{\mathcal{X}}$, where $\mathcal{X}$ is the state space of $\vec{\mathcal{S}}$. We

first outline a procedure for computing an abstract LTL specification $\widehat{\varphi}$ on a set of atomic propositions $\widehat{AP} \subseteq 2^{\widehat{\mathcal{X}}}$, where $\widehat{\mathcal{X}}$ is the state space of $\widehat{\mathcal{S}}$. After that, we show how we can algorithmically compute an optimal controller for the control problem $\left\langle \widehat{\mathcal{S}}, \widehat{\varphi} \right\rangle$, which will then provide us a (possibly sub-optimal) controller for the control problem $\left\langle \vec{\mathcal{S}}, \varphi \right\rangle$ through refinement: $\mathcal{C} \circ \widehat{Q}$.

3.5.1. Abstract Specification

Let $\mathcal{S} = \langle \mathcal{X}, \mathcal{U}, \mathcal{W}, f \rangle$ be a continuous-time control system. We assume, without loss of generality, that the LTL formula φ over the alphabet $\mathcal{X}$ has been *provided in Positive Normal Form* (PNF). (Any non-PNF LTL formula can be transformed into an equivalent and canonical LTL formula in PNF, see Sec. 2.2.1.) The abstract LTL formula $\widehat{\varphi}$ will essentially represent the under-approximation of the set of valid words $[\![\varphi]\!]_{\mathcal{X}}$ using the relation $\widehat{Q}$.

First, let us consider specifications that are not parity properties; for parity properties, the abstraction process is more complicated and we come back to this later. Let φ be an LTL formula and $AP \subset 2^{\mathcal{X}}$ be a finite set of atomic propositions used in φ. Let $P \in AP$ be any arbitrary atomic proposition. Define the under and the over-approximation of P with the help of the abstract states as in the following:

$$\begin{aligned} \text{Under-approximation:} \quad & \underline{P} \coloneqq \{\widehat{x} \in \widehat{\mathcal{X}} \mid \widehat{x} \subseteq P\}, \\ \text{Over-approximation:} \quad & \overline{P} \coloneqq \{\widehat{x} \in \widehat{\mathcal{X}} \mid \widehat{x} \cap P \neq \emptyset\}. \end{aligned}$$

The abstract formula $\widehat{\varphi}$, that uses atomic propositions on $\widehat{\mathcal{X}}$, is obtained from φ as follows:

(a) Every positive literal "P" is replaced by the positive literal "$\underline{P}$", and

(b) every negated literal "$\neg P$" is replaced by the negated literal "$\neg\overline{P}$".

When it is irrelevant which of the above two cases it is, we will simply use $\widehat{P}$ to denote the new abstract proposition that was introduced in place of P. For instance, the well-known reach-avoid specification $\varphi = \neg O \,\mathsf{U}\, T$, for a set of obstacles $O \subseteq \mathcal{X}$ and a set of targets $T \subseteq \mathcal{X}$, will be transformed into $\widehat{\varphi} = \neg\widehat{O} \,\mathsf{U}\, \widehat{T} = \neg\overline{O} \,\mathsf{U}\, \underline{T}$; see Fig. 3.2 for a visual illustration.

The case of parity specification is a bit more involved, because approximating B_i-s may cause some abstract state to be included in no or multiple abstract $\widehat{B_i}$ sets, which is not permitted as per the definition of parity properties Def. 2.6. Let us take a closer look. First we convert the parity specification in (2.2) to the following formula in Positive Normal Form (PNF):

$$\mathit{Parity}(\mathcal{P}) \coloneqq \bigwedge_{i \in_{odd} [1;2k]} \left(\Diamond\Box\neg B_i \vee \bigvee_{j \in_{even} [i+1;2k]} \Box\Diamond B_j \right).$$

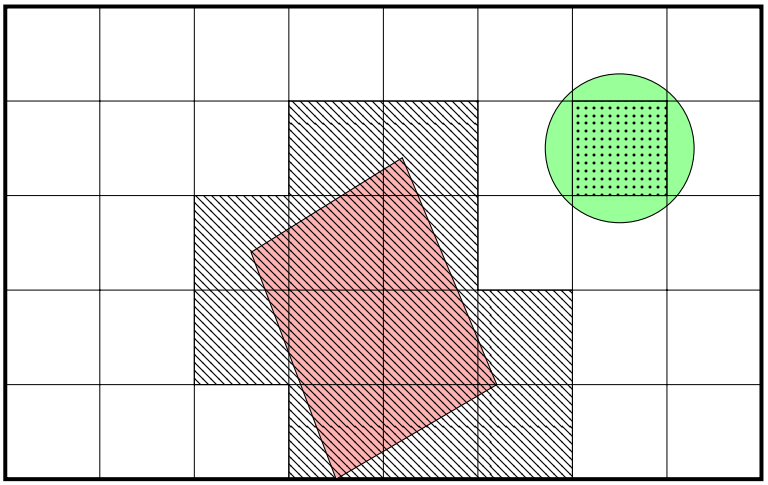


Figure 3.2.: An example reach-avoid specification $\neg O \,\mathsf{U}\, T$ with its abstract counterpart: The red quadrilateral represents the obstacle O and the green circle represents the target T. The abstract specification is given by $\neg\overline{O}\,\mathsf{U}\underline{T}$ where $\overline{O}$ is the hatched region and $\underline{T}$ is the dotted region.

If we naïvely replace the "$\neg B_i$" and "B_i" literals using "$\neg\overline{B_i}$" and "$\underline{B_i}$," we get:

$$\bigwedge_{i\in_{odd}[1;2k]} \left(\Diamond\Box\neg\overline{B_i} \vee \bigvee_{j\in_{even}[i+1;2k]} \Box\Diamond\underline{B_j} \right).$$

The above formula can be alternatively re-written as:

$$\bigwedge_{i\in_{odd}[1;2k]} \left(\Box\Diamond\overline{B_i} \rightarrow \bigvee_{j\in_{even}[i+1;2k]} \Box\Diamond\underline{B_j} \right). \tag{3.8}$$

We can express the above formula as a parity property with the set of atomic propositions $\widehat{\mathcal{P}} := \{\underline{B_0}, \overline{B_1}, \underline{B_2}, \overline{B_3}, \ldots, \overline{B_{2k-1}}, \underline{B_{2k}}\}$.

Unfortunately, this naïve approach will not always work: (i) When an abstract state $\widehat{x}$ intersects with multiple even priority sets and no odd priority set, then $\widehat{x}$ does not get any priority assigned in $\widehat{\mathcal{P}}$, and (ii) when $\widehat{x}$ intersects with multiple *odd* priority sets, then $\widehat{x}$ gets conflicting priority assignments (i.e. $\widehat{x}$ gets included in all the intersecting odd priority sets) in $\widehat{\mathcal{P}}$.

To circumvent this issue while maintaining soundness, we define the abstract priorities in a way that (i) and (ii) are resolved in a disadvantageous way to the controller. For case (i), we assign $\widehat{x}$ the *minimum even* priority that intersects with $\widehat{x}$. For case (ii), we assign $\widehat{x}$ the *maximum odd* priority that intersects with $\widehat{x}$. We illustrate this in Fig. 3.3.

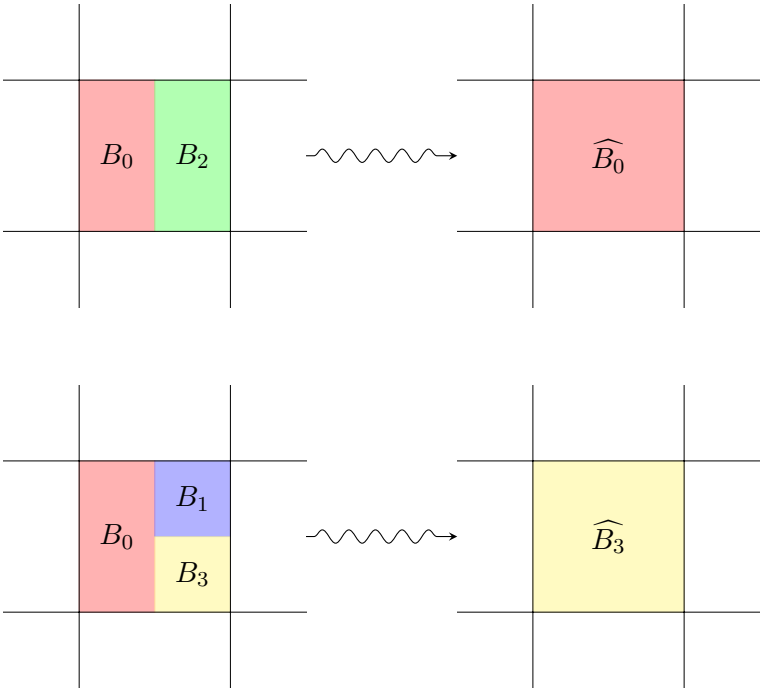


Figure 3.3.: Illustration of construction of abstract priorities for parity specifications. **Top row:** When an abstract state $\widehat{x}$ does not intersect with any odd priority set, then the priority of $\widehat{x}$ in the abstraction is the *minimum* of the intersecting even priority sets. **Bottom row:** When an abstract state $\widehat{x}$ intersects with odd priority set(s), then the priority of $\widehat{x}$ in the abstraction is the *maximum* of the intersecting odd priority sets.

Formally, the abstract priorities are defined as in the following:

$$i \text{ is even: } \widehat{B_i} \coloneqq \underline{B_i} \cup \left\{ \widehat{x} \in \widehat{\mathcal{X}} \middle| \forall j \in_{odd} [1; 2k] \,.\, \widehat{x} \cap B_j = \emptyset \wedge i = \min\{l \mid \widehat{x} \cap B_l \neq \emptyset\} \right\},$$
$$i \text{ is odd: } \widehat{B_i} \coloneqq \underline{B_i} \cup \left\{ \widehat{x} \in \widehat{\mathcal{X}} \middle| i = \max(\{l \in_{odd} [1; 2k] \mid \widehat{x} \cap B_l \neq \emptyset\}) \right\}.$$

It follows from a monotonicity argument (Kazemi and Soudjani, 2020, Thm. 5) of PNF non-parity formulas and a robustness argument for the parity formulas that $\widehat{\varphi}$ is indeed an abstract specification according to Def. 3.7:

Lemma 3.1 $\left\langle \vec{\mathcal{S}}, \varphi \right\rangle \preccurlyeq_{\widehat{Q}} \left\langle \widehat{\mathcal{S}}, \widehat{\varphi} \right\rangle$.

Proof When φ is a property other than parity, then the proof is by induction over the structure of the two LTL formulas φ and $\widehat{\varphi}$. From the construction of $\underline{P}$ and $\overline{P}$, it is clear that $\underline{P} \subseteq P$ and $\neg\overline{P} \subseteq \neg P$. Furthermore, for any pair of subformulas φ_1, φ_2 and their

respective abstractions $\widehat{\varphi_1}, \widehat{\varphi_2}$, the following are true:

$$\begin{aligned}
\left\langle \vec{\mathcal{S}}, \varphi_1 \wedge \varphi_2 \right\rangle &\preccurlyeq_{\widehat{Q}} \left\langle \widehat{\mathcal{S}}, \widehat{\varphi_1} \wedge \widehat{\varphi_2} \right\rangle, \\
\left\langle \vec{\mathcal{S}}, \varphi_1 \vee \varphi_2 \right\rangle &\preccurlyeq_{\widehat{Q}} \left\langle \widehat{\mathcal{S}}, \widehat{\varphi_1} \vee \widehat{\varphi_2} \right\rangle, \\
\left\langle \vec{\mathcal{S}}, \bigcirc \varphi_1 \right\rangle &\preccurlyeq_{\widehat{Q}} \left\langle \widehat{\mathcal{S}}, \bigcirc \widehat{\varphi_1} \right\rangle, \\
\left\langle \vec{\mathcal{S}}, \varphi_1 \,\mathsf{U}\, \varphi_2 \right\rangle &\preccurlyeq_{\widehat{Q}} \left\langle \widehat{\mathcal{S}}, \widehat{\varphi_1} \,\mathsf{U}\, \widehat{\varphi_2} \right\rangle, \\
\left\langle \vec{\mathcal{S}}, \varphi_1 \,\mathsf{R}\, \varphi_2 \right\rangle &\preccurlyeq_{\widehat{Q}} \left\langle \widehat{\mathcal{S}}, \widehat{\varphi_1} \,\mathsf{R}\, \widehat{\varphi_2} \right\rangle.
\end{aligned}$$

When φ is a parity property, then the proof follows by a robustness argument: Suppose $\widehat{\xi} \in \langle\!\langle \widehat{\varphi} \rangle\!\rangle_{\widehat{\mathcal{X}}}$, and let $\xi \in \mathcal{X}^\infty$ be a path of $\vec{\mathcal{S}}$ such that for every $k \in \mathbb{N}$, $(\xi^k, \widehat{\xi}^k) \in Q$. We argue that $\xi \in \langle\!\langle \varphi \rangle\!\rangle_{\mathcal{X}}$. Recall that as per our notation, $\mathrm{Rec}(\widehat{\xi})$ is the set of abstract states that appear infinitely many times on the run $\widehat{\xi}$. Suppose $\widehat{x} \in \mathrm{Rec}(\widehat{\xi})$ is the abstract state with the highest priority among all the states in $\mathrm{Rec}(\widehat{\xi})$, which is even according to our assumption (that $\widehat{\xi} \in \langle\!\langle \widehat{\varphi} \rangle\!\rangle_{\widehat{\mathcal{X}}}$); let the priority of $\widehat{x}$ be i for some $i \in_{even} [1; 2k]$. By our construction of $\widehat{\mathcal{P}}$, all the continuous states inside $\widehat{x}$ have either *same or higher even priority* than i, which shows that ξ, passing through $\widehat{x}$, has at least i even priority appearing infinitely often along it. Now let $\widehat{x}' \in \mathrm{Rec}(\widehat{\xi})$ be an arbitrary abstract state with odd priority j for some $j \in_{odd} [0; 2k]$, where $j < i$ according to our assumption (that $\widehat{\xi} \in \langle\!\langle \widehat{\varphi} \rangle\!\rangle_{\widehat{\mathcal{X}}}$). By construction of $\widehat{\mathcal{P}}$, all the continuous states inside $\widehat{x}'$ have either same or lower odd priority than j, which shows that ξ, passing through the states in $\mathrm{Rec}(\widehat{\xi})$ infinitely often, has all odd infinitely occurring priorities smaller than i. □

3.5.2. Abstract 2-Player Game

Suppose $\widehat{\mathcal{S}} = \left\langle \widehat{\mathcal{X}}, \widehat{\mathcal{U}}, \widehat{F} \right\rangle$ is the finite abstraction of a continuous-time system $\mathcal{S}$ (the abstraction parameters are unimportant). For simplicity, we assume that the control specification φ for $\mathcal{S}$ is definable using a regular or an ω-regular property from Sec. 2.2.3 and Sec. 2.2.4, specified using predicates over the state space $\mathcal{X}$. Naturally, this will imply the same for the abstract specification $\widehat{\varphi}$ for $\widehat{\mathcal{S}}$, specified using the predicates over $\widehat{\mathcal{X}}$. The consequence is that the controller is known to be memoryless in this case.

The more general case is when the specification is provided as an ω-*automaton* (such as a parity or a Rabin automaton) with the predicate $AP \subseteq \mathcal{X}$ serving as the automaton alphabet. By taking a synchronized product of this ω-automaton with the system $\mathcal{S}$, we can obtain a product control system and an ω-regular specification over the product state space such that every path satisfying the specification is also accepted by the original ω-automaton, and vice versa. Moreover, a stationary policy for the ω-regular specification gives a (possibly history-dependent) policy for the original automata-based specification. Thus, without loss of generality, we assume that an ω-regular specification is already given using a set of atomic propositions over the state space of the system.

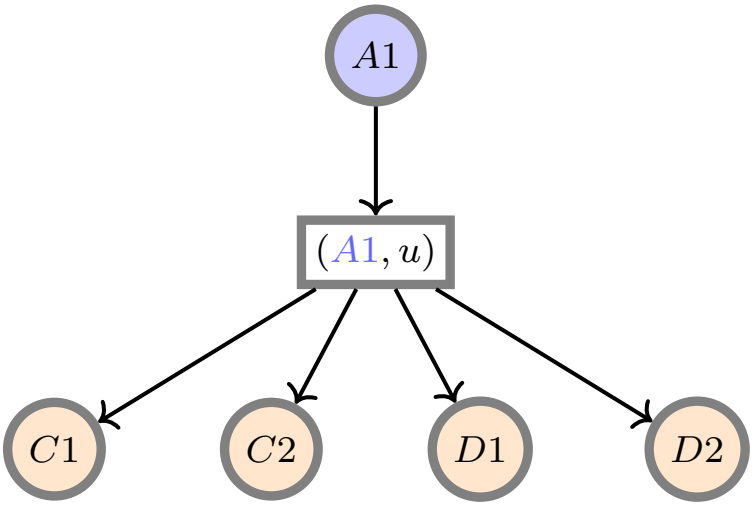


Figure 3.4.: Illustration of the construction of the abstract 2-player game graph $\mathcal{G}$ for the abstract transition shown in Fig. 3.1. The vertices in V_0 and V_1 are respectively indicated by circle and rectangular nodes.

The abstract 2-player game graph $\mathcal{G}$ is obtained by simply separating the control and the uncertainty aspect in $\widehat{\mathcal{S}}$, see Fig. 3.4. Whenever $\widehat{\mathcal{S}}$ reaches a state $\widehat{x}$ in $\widehat{\mathcal{S}}$, the following two-step game is played in $\mathcal{G}$: There are *Player* 0-vertices labeled as $\widehat{x}$, from which *Player* 0 chooses a control input $u \in \widehat{\mathcal{U}}$, causing a transition to a *Player* 1-vertex labeled $(\widehat{x}, u)$. From $(\widehat{x}, u)$ there are edges to every $\widehat{x}' \in \widehat{F}(\widehat{x}, u)$, and *Player* 1 chooses one of the successors. Formally, $\mathcal{G} = \langle V, V_0, V_1, E \rangle$ where

- $V = \widehat{\mathcal{X}} \cup \left(\widehat{\mathcal{X}} \times \widehat{\mathcal{U}}\right)$,
- $V_0 = \widehat{\mathcal{X}}$,
- $V_1 = \left(\widehat{\mathcal{X}} \times \widehat{\mathcal{U}}\right)$, and
- $E = (V_0 \times V_1) \cup \{(\widehat{x}, \widehat{u}, \widehat{x}') \in V_1 \times V_0 \mid \widehat{x}' \in \widehat{F}(\widehat{x}, \widehat{u})\}$.

Let the abstract specification for $\widehat{\mathcal{S}}$ be $\widehat{\varphi}$, defined using the propositions $AP \subseteq 2^{\widehat{\mathcal{X}}}$. For the 2-player game $\mathcal{G}$, we can use φ as the winning condition by interpreting the propositions AP over V_0 (V_0 is same as $\widehat{\mathcal{X}}$). Since the vertices in V_1 are excluded from AP, satisfaction of $\widehat{\varphi}$ is interpreted using only the projections of the plays over the vertices in V_0. Formally, consider the following play ρ over the game graph $\mathcal{G}$:

$$v^0(v^0, u^2)v^1(v^1, u^1)v^2(v^2, u^2)\ldots.$$

We say ρ is in $\langle\!\langle \widehat{\varphi} \rangle\!\rangle_V$ if the subsequence $v^0v^1v^2\ldots$ is in $\langle\!\langle \widehat{\varphi} \rangle\!\rangle_{V_0}$.

For simplicity and without loss of generality, let us assume that φ and $\widehat{\varphi}$ are given as parity specifications. Recall that memoryless strategies are optimal for sure winning in 2-player parity games (see Prop. 2.1). Suppose ρ_0 is a *memoryless Player* 0 strategy in

the 2-player game graph $\mathcal{G}$. A refined controller for the finite abstraction $\widehat{\mathcal{S}}$ is a controller $\mathcal{C}\colon \widehat{\mathcal{X}} \to \widehat{\mathcal{U}}$ where for every $\widehat{x} \in \widehat{\mathcal{X}}$, $\mathcal{C}(\widehat{x}) = \widehat{u}$ if $\rho_0(\widehat{x}) = (\widehat{x}, \widehat{u})$. The following theorem states that the refined controller is sound.

Theorem 3.3 (Soundness of the game construction) *Let $\widehat{\mathcal{S}}$ be the finite abstraction of a continuous-time system $\mathcal{S}$, $\widehat{\varphi}$ be an abstract specification for $\widehat{\mathcal{S}}$, and $\langle \mathcal{G}, \varphi \rangle$ be the abstract 2-player game. If π_0 is an optimal (sure) winning strategy for Player 0, then the refined controller for $\widehat{\mathcal{S}}$ is in OptCtrl$(\widehat{\mathcal{S}}, \widehat{\varphi})$.*

Proof Let $\mathcal{C}$ be the refined controller for $\widehat{\mathcal{S}}$, and $\widehat{\mathcal{S}}^{\mathsf{cl}}$ be the closed-loop obtained by interconnected $\widehat{\mathcal{S}}$ and $\mathcal{C}$ in feedback. We need to show that both (A) and (B) in Def. 3.3 are satisfied by $\mathcal{C}$.

(A) Let $\mathcal{W}^{sure}$ be the sure winning region of the game $\langle \mathcal{G}, \varphi \rangle$, and $v^0 \in \mathcal{W}^{sure}$ be any arbitrary vertex. By construction of $\mathcal{G}$ and $\mathcal{C}$, for every path ξ in the closed-loop $\widehat{\mathcal{S}}^{\mathsf{cl}}$ starting from v^0, there is a play $\rho = v^0(v^0, u^0)v^1(v^1, u^1)\ldots$ complying to π_0 (and some *Player* 1 strategy π_1) such that the subsequence $v^0v^1\ldots$ is same as ξ. Since π_0 is a winning strategy of *Player* 0 from v^0, hence for every *Player* 1 strategy π_1, every play complying to π_0 and π_1 and starting at v^0 is in $\langle\!\langle \widehat{\varphi} \rangle\!\rangle_V$. Hence, $\xi \in \langle\!\langle \widehat{\varphi} \rangle\!\rangle_{\widehat{\mathcal{X}}}$.

(B) Let $\mathcal{C}' \in$ OptCtrl$(\widehat{\mathcal{S}}, \widehat{\varphi})$ and $\widehat{x} \in \mathrm{dom}(\mathcal{C}')$. We show that $\widehat{x}$ is in the sure winning region $\mathcal{W}^{sure}$ of the game $\langle \mathcal{G}, \varphi \rangle$, i.e. $\widehat{x} \in \mathrm{dom}(\pi_0)$, which would imply that $\widehat{x} \in \mathrm{dom}(\mathcal{C})$ due to the sound refinement property proved in (A). If, for the sake of argument, $\widehat{x}$ were not in $\mathcal{W}^{sure}$, then *Player* 1 would have a strategy π_1 such that the play complying to π_0 and π_1 and starting from $\widehat{x}$ would *not* be in $\langle\!\langle \widehat{\varphi} \rangle\!\rangle_V$. By construction of $\mathcal{G}$, this would imply existence of a path in the closed loop of $\widehat{\mathcal{S}}$ and $\mathcal{C}'$ starting at $\widehat{x}$ that is *not* in $\langle\!\langle \widehat{\varphi} \rangle\!\rangle_{\widehat{\mathcal{X}}}$. This is a contradiction to the existence of such a $\mathcal{C}'$ and $\widehat{x}$ in the first place. □

The following corollary shows the ABCD approach for solving Prob. 2: Any optimal controller for the abstract control problem $\left\langle \widehat{\mathcal{S}}, \widehat{\varphi} \right\rangle$ can be refined to an optimal controller for the sampled-time control problem $\left\langle \vec{\mathcal{S}}, \varphi \right\rangle$.

Corollary 3.1 (Soundness of ABCD) *Let $\mathcal{S}$ be a continuous-time system, $\vec{\mathcal{S}}$ be the sampled-time abstraction, $\widehat{\mathcal{S}}$ be the finite abstraction, φ be an ω-regular control specification for $\vec{\mathcal{S}}$ defined using propositions over $\mathcal{X}$, and $\widehat{\varphi}$ be an ω-regular specification for $\widehat{\mathcal{S}}$ over $\widehat{\mathcal{X}}$ such that $\left\langle \vec{\mathcal{S}}, \varphi \right\rangle \preccurlyeq_{\widehat{Q}} \left\langle \widehat{\mathcal{S}}, \widehat{\varphi} \right\rangle$. Suppose $\langle \mathcal{G}, \varphi \rangle$ is the abstract 2-player parity game. If π_0 is a winning strategy of Player 0 in the game $\langle \mathcal{G}, \varphi \rangle$ and $\mathcal{C}$ is the corresponding refined controller for $\widehat{\mathcal{S}}$, then $\mathcal{C} \circ \widehat{Q} \in$ OptCtrl$(\vec{\mathcal{S}}, \varphi)$.*

Proof Follows from Thm. 3.1 and Thm. 3.3. □

3.6. Related Work

There are two broad categories of work, when it comes to correct-by-design controller synthesis techniques. The first one synthesizes the controller by simplifying the problem with the help of a simpler abstraction of the given continuous system; the ABCD technique presented in this thesis work falls in this category. We will discuss the work related to ABCD techniques in Sec. 3.6.2.

3.6.1. Correct-by-Design Controller Synthesis

The second category of correct-by-design techniques synthesizes the controller directly on the given continuous system. Most of them deal with safety and reachability specification, as compared to the more general LTL specifications that we consider here.

One class of techniques in this category searches for certain polynomial functions of the state of the system, whose existence guarantees the satisfaction of the given specification, as well as it gives the implementing control strategy (Prajna and Jadbabaie, 2004; Prajna and Rantzer, 2005; Prajna et al., 2007; Wieland and Allgöwer, 2007; Ravanbakhsh and Sankaranarayanan, 2017; Ames et al., 2019; Yang et al., 2020). For safety specification, this function is called control barrier function, and for reachability (infinite horizon) specifications, it is called control Lyapunov functions. Both the barrier function and the Lyapunov function can be searched using convex optimization, and for the special case for polynomial systems, more efficient sum-of-square optimization can be employed. Unfortunately, no method of this category is known that can handle the general class of LTL specifications. A deductive approach for combining Lyapunov functions and barrier functions to do synthesis for more general parity specifications was shown by Dimitrova and Majumdar (2014); however no means of synthesizing the certificates were provided.

Another class of techniques, which also uses convex programming to solve safety and reachability control synthesis problem, uses measure theoretic approaches to predict the evolution of the system trajectories (Korda et al., 2014; Henrion and Korda, 2014; Korda et al., 2016; Chen et al., 2020). This class also lacks methods for handling the general LTL specifications.

Finally, there are some techniques which use tools from optimal control to solve the controller synthesis problem. They reformulate the problem in terms of an equivalent "min-max" optimization of some appropriate value function. It turns out that the solution of this optimization problem coincides with the viscosity solution of a Hamilton-Jacobi Isaacs (HJI) partial differential equation (PDE). One can then solve the HJI PDE numerically to find an approximate solution of the controller synthesis problem. Unfortunately, the numerical solution requires discretization of the state space making it suffer from the curse of dimensionality, similar to the ABCD-based approaches; besides, no known method for handling LTL specifications exists. Recently, Bansal et al. (2017) presented an extensive survey of different variations of this method.

3.6.2. Abstraction-Based Controller Design

The original techniques for abstraction-based control relied on ε-alternating bisimulation relations (Girard and Pappas, 2007; Girard, 2012; Pola et al., 2008; Girard et al., 2010). These relations, when they exist, allow proving an "if and only if" result: a controller can be synthesized in the abstraction iff one exists in the original system. Nilsson et al. (2017) proposed an abstraction that captures progress properties in the continuous dynamics, which helps the synthesis algorithm to rule out spurious cycles introduced during the abstraction process. The notion of *feedback refinement relations* (FRR) was introduced by Reissig et al. (2017), and strengthened the notion of alternating simualtion relations (Alur et al., 1998) to the setting of continuous control. The same paper shows how to compute growth bounds for a non-linear system—this is the basis for the `SCOTS` (Rungger and Zamani, 2016) and `MASCOT` tools. Very recently, ABCD was extended for synthesis of output feedback controllers instead of state feedback controllers (Majumdar et al., 2020b).

Many academic tools were built (Rungger and Zamani, 2016; Jr. et al., 2010; Roy et al., 2011; Li and Liu, 2018; Mouelhi et al., 2013; Khaled and Zamani, 2019) and many practical controller synthesis problems were solved (Ames et al., 2015; Meyer et al., 2019; Zonetti et al., 2019; Nilsson et al., 2016; Yang et al., 2017; Coogan et al., 2016) using ABCD in recent years.

The automata-theoretic underpinnings of ABCD use algorithms for reactive synthesis for ω-regular specifications (Emerson and Jutla, 1991; Thomas, 1995; Maler et al., 1995).

3.7. Conclusion

We revisited the FRR-based ABCD algorithm in this chapter. We saw how a continuous-time dynamical system can be abstracted to a finite transition system using discretization of time, states, and inputs, and using a growth bound on the dynamics to over-approximate the sampled-time transitions. Later, the abstract finite transition system was used to construct an abstract 2-player game. A sure winning strategy for *Player* 0 in the constructed game can be refined into a sound sampled-time controller for the original system.

In the next two chapters, we extend this ABCD algorithm in two different directions. First, in Chap. 4, we address the poor scalability issue of FRR-based ABCD, caused by the brute-force discretization of the dynamics. We will see how we can use a multi-layered abstractions to make this approach more scalable, without sacrificing the quality of the synthesized controller.

Next, in Chap. 5, we will introduce our novel ABCD approach for a particular class of stochastic dynamical systems, called controlled Markov processes (CMP). CMPs can effectively model continuous-state sampled-time dynamical systems subjected to stochastic noise. We will show how we can synthesize controllers for CMPs to maximize the probability of satisfaction of a given parity specification.

4. Lazy Multi-Layered Controller Synthesis

In Chap. 3, we revisited the established Abstraction-Based Controller Design (ABCD) approach for synthesizing correct-by-construction controller for continuous-time dynamical systems. One of the bottlenecks of ABCD has been scalability, which stems from the exponential blowup caused by the discretization process in the abstraction stage. One way to improve the scalability is to use coarser discretization (i.e. larger abstract states), which reduces the size of the abstract systems. However, the coarser the discretization is, the higher is the chance of getting an empty controller domain as the outcome, due to the excessive inaccuracy of the abstraction.

In this chapter, we address this trade-off between scalability and controller domain size. We present our lazy multi-layered ABCD algorithm for continuous non-stochastic systems and reach-avoid and safety specifications. The idea is simple: Instead of having one uniformly coarse abstraction, we will maintain multiple partially explored abstractions of varying coarseness; say the abstractions are indexed with the numbers $[1; L]$ (1 is the finest), where L is the number of abstractions. The explorations of the abstractions happen in a purely need-based fashion, where we first try to perform synthesis using the coarser abstractions as much as possible, and lazily switch to the finer ones only when necessary. As a result, we obtain a more scalable ABCD algorithm, that returns a controller with the same domain as the one that we would obtain by performed a single-layered ABCD solely on layer 1 (i.e. the finest layer). Our exposition in this chapter follows our prior work (Hsu et al., 2018b,a).

4.1. Preliminaries of Multi-Layered ABCD

4.1.1. Multi-Layered Systems

Recall that for single-layered ABCD in Chap. 3, we used the abstraction parameters τ and η to denote the sampling time and the size of the abstract grid cells respectively. Now we use the parameters τ_l and η_l to denote the respective τ and η in abstraction layer l. Given a grid parameter $\underline{\eta}$, a time sampling parameter $\underline{\tau}$, and $L \in \mathbb{Z}_{>0}$, define $\eta_l = 2^{l-1}\underline{\eta}$ and $\tau_l = 2^{l-1}\underline{\tau}$. Our method is applicable to grid parameters defined by $\eta_1 = \underline{\eta}$ and $\eta_{l+1} = \gamma\eta_l$ for $l \in [1; L-1]$ and $\gamma \in \{2^k \mid k \in \mathbb{N}\}$, and sampling times $\tau_l = \alpha(l)\underline{\tau}$ where $\alpha : [1; L] \to \mathbb{R}$ is a monotonically increasing function. For notational simplicity, we restrict our attention to $\gamma = 2$ and $\alpha(l) = 2^{l-1}$.

Fix a continuous-time system $\mathcal{S} = \langle \mathcal{X}, \mathcal{U}, \mathcal{W}, f \rangle$ and a non-empty subset $\mathcal{X}' \subseteq \mathcal{X}$ with $\mathcal{X}' = [\![\alpha, \beta]\!]$, such that $\beta - \alpha = k\eta_L$ for some $k \in \mathbb{Z}^n$. For each $l \in [1; L]$, we define the grid $\eta_l \mathbb{Z}^n$ and the cover $\widehat{\mathcal{X}}'_l$ as in (3.5) and (3.6), respectively, by substituting η with η_l. For the single-layered ABCD, we defined the sampled-time abstraction and the finite abstraction of a continuous-time system in Def. 3.5 and Def. 3.9 respectively. In the case of multi-layered abstractions, we obtain a sequence of sampled-time abstractions $\{\vec{\mathcal{S}}_l(\mathcal{S}, \tau_l)\}_{l \in [1;L]}$ and a sequence of finite abstractions $\{\widehat{\mathcal{S}}_l(\mathcal{S}, \tau_l, \eta_l, \beta_{\tau_l})\}_{l \in [1;L]}$. For simplicity, we assume that all layers use the same continuous and abstract input spaces $\mathcal{U}$ and $\widehat{\mathcal{U}} \subseteq \mathcal{U}$, respectively. If $\mathcal{S}$, τ_l, η_l, and β_{τ_l} are clear from the context, we use $\vec{\mathcal{S}}_l$ and $\widehat{\mathcal{S}}_l$ as short forms of $\vec{\mathcal{S}}_l(\mathcal{S}, \tau_l)$ and $\widehat{\mathcal{S}}_l(\mathcal{S}, \tau_l, \eta_l, \beta_{\tau_l})$, respectively.

It trivially follows from our construction that, for all $l \in [1; L]$, we have $\vec{\mathcal{S}}_l \preccurlyeq_{\widehat{Q}_l} \widehat{\mathcal{S}}_l$, where $\widehat{Q}_l \subseteq \mathcal{X} \times \widehat{\mathcal{X}}_l$ is the FRR induced by $\widehat{\mathcal{X}}_l$. The set of relations $\{\widehat{Q}_l\}_{l \in [1;L]}$ induces transformers $\widehat{R}_{ll'} \subseteq \widehat{\mathcal{X}}_l \times \widehat{\mathcal{X}}_{l'}$ for $l, l' \in [1; L]$ between abstract states of different layers such that

$$\widehat{x} \in \widehat{R}_{ll'}(\widehat{x}') \iff \widehat{x} \in \widehat{Q}_l(\widehat{Q}_{l'}^{-1}(\widehat{x}')). \tag{4.1}$$

Given the sequence of finite abstractions $\{\widehat{\mathcal{S}}_l\}_{l \in [1;L]}$ with $\widehat{\mathcal{S}}_l = \left\langle \widehat{\mathcal{X}}_l, \widehat{\mathcal{U}}, \widehat{F}_l \right\rangle$, we can use $\widehat{R}_{ll'}$ to define the *multi-layered finite abstraction* $\widehat{\mathbf{S}} = \left\langle \widehat{\mathbf{X}}, \widehat{\mathcal{U}}, \widehat{\mathbf{F}} \right\rangle$, where $\widehat{\mathbf{X}} = \bigcup_{l \in [1;L]} \widehat{\mathcal{X}}_l$ and

$$\widehat{\mathbf{F}}(\widehat{x}, \widehat{u}) = \bigcup_{l \in [1;L]} \widehat{R}_{ll'}(\widehat{F}_{l'}(\widehat{x}, \widehat{u})) \tag{4.2}$$

for all $\widehat{x} \in \widehat{\mathcal{X}}_{l'}$, $l' \in [1; L]$, and $\widehat{u} \in \widehat{\mathcal{U}}$. Intuitively, $\widehat{\mathbf{S}}$ is a non-deterministic system; for every state $\widehat{x} \in \widehat{\mathcal{X}}_{l'}$ and input $\widehat{u} \in \widehat{\mathcal{U}}$, transitions to states of all layers are possible. That is, $\widehat{x}' \in \widehat{\mathbf{F}}(\widehat{x}, \widehat{u})$ if there exists an $\widehat{x}'' \in \widehat{\mathcal{X}}_{l'}$ such that $\widehat{x}'' \in \widehat{F}_{l'}(\widehat{x}, \widehat{u})$ and $\widehat{x}' \in \widehat{R}_{ll'}(\widehat{x}'')$ for some $l \in [1; L]$.

Similarly, given the sequence of sampled-time abstractions $\{\vec{\mathcal{S}}_l\}_{l \in [1;L]}$ with $\vec{\mathcal{S}}_l = (\mathcal{X}_l, \mathcal{U}, \vec{F}_l)$, we define a *multi-layered sampled-time abstraction* $\overrightarrow{\mathbf{S}} = \left\langle \mathcal{X}, \mathcal{U}, \vec{\mathbf{F}} \right\rangle$ such that

$$\vec{\mathbf{F}}(x, u) = \bigcup_{l \in [1;L]} \vec{F}_l(x, u) \tag{4.3}$$

for all $x \in \mathcal{X}$ and $u \in \mathcal{U}$. Again, $\overrightarrow{\mathbf{S}}$ is a non-deterministic system; in every state $x \in \mathcal{X}$ transitions of any duration τ_l, $l \in [1; L]$ can be chosen, which correspond to some $\vec{F}_l(x, u)$.

The behaviors $\mathcal{B}(\widehat{\mathbf{S}})$ and $\mathcal{B}(\overrightarrow{\mathbf{S}})$ are defined according to Def. 3.4.

Remark 2 *Even though we have $\vec{\mathcal{S}}_l \preccurlyeq_{\widehat{Q}_l} \widehat{\mathcal{S}}_l$ for all $l \in [1; L]$, each relation is evaluated for a different sampling time τ_l. Therefore, the relations $\widehat{R}_{ll'}$ cannot define an FRR between $\widehat{\mathcal{S}}_l$ and $\widehat{\mathcal{S}}_{l'}$ using Def. 3.6.*

Given that $\tau_l = 2^{l-1}\underline{\tau}$, a natural extension of FRR seems to be that, for any $(\widehat{x}_{l+1}, \widehat{x}_l) \in \widehat{R}_{(l+1)l}$, it holds that

$$\widehat{u} \in \widehat{\mathcal{U}} \implies \widehat{R}_{(l+1)l}(\widehat{F}_l(\widehat{F}_l(\widehat{x}_l, \widehat{u}), \widehat{u})) \subseteq \widehat{F}_{l+1}(\widehat{x}_{l+1}, \widehat{u}). \tag{4.4}$$

That is, we would expect that states $\widehat{x}_l \in \widehat{\mathcal{X}}_l$ and $\widehat{x}_{l+1} \in \widehat{\mathcal{X}}_{l+1}$ related via $\widehat{R}_{(l+1)l}$ remain related when the l-th layer transition function is applied to $\widehat{x}_l$ twice for τ_l (resulting in a duration $2\tau_l = \tau_{l+1}$) and when the $l+1$-th layer transition function is only applied once to $\widehat{x}_{l+1}$ (also resulting in a duration τ_{l+1}). While this seems intuitive, we can prove that (4.4) *does not hold in general. This is because applying the l-th layer transition function twice introduces extra approximation error due to the additional reachable set computation step (as in* (3.7)*).*

4.1.2. Multi-Layered Controllers

In this chapter, we only consider memoryless controllers. Given the multi-layered finite abstraction $\widehat{\mathbf{S}}$ as in (4.2) and some positive number of controllers $P \in \mathbb{Z}_{>0}$, we define a *multi-layered controller* $\mathbf{C}$ for $\widehat{\mathbf{S}}$ as a set of controllers $\mathbf{C} = \{\mathcal{C}_p\}_{p\in[1;P]}$ such that every controller $\mathcal{C}_p$ corresponds to a unique finite abstraction $\widehat{\mathcal{S}}_{l_p}$:

$$\forall p \in [1;P] \,.\, \exists! l_p \in [1;L] \,.\, \mathrm{dom}(\mathcal{C}_p) \subseteq \widehat{\mathcal{X}}_{l_p}, \tag{4.5}$$

where the symbol "$\exists!$" can be read as "there exists a unique ...." From now on, we will use the suffix l_p in the suffix to represent the associated layer index of the p-th controller $\mathcal{C}_p$. We do not require any connection between P and L. In particular, we allow for layers to have multiple controllers, i.e., $l_p = l_q$ for $p, q \in [1;P]$, $p \neq q$, and no controller at all, i.e. there might be $l \in [1;L]$ such that there exists no $p \in [1;P]$ with $l_p = l$.

Given a multi-layered controller $\mathbf{C}$ of $\widehat{\mathbf{S}}$, we define the *quantizer induced by* $\mathbf{C}$ as the total function $\mathbf{Q}\colon \mathcal{X} \to 2^{\widehat{\mathbf{X}}}$ such that for all $x \in \mathcal{X}$ we have $\widehat{x} \in \mathbf{Q}(x)$ if and only if *either* of the following holds:

(i) The abstract state $\widehat{x}$ corresponds to the coarsest abstraction of x with an available control: There exists a $p \in [1;P]$ such that

$$\widehat{x} \in \widehat{Q}_{l_p}(x) \wedge \widehat{x} \in \mathrm{dom}(\mathcal{C}_p)$$

and there exists no other $p' \in [1;P]$ with $l_{p'} > l_p$ such that

$$\widehat{R}_{l_{p'}l_p}(\widehat{x}) \in \mathrm{dom}(\mathcal{C}_{p'}).$$

(ii) The abstract state $\widehat{x}$ corresponds to the finest abstraction of x, and there is no control available for x in any of the abstractions: $\widehat{x} \in \widehat{Q}_1(x)$ and there exists no $p \in [1;P]$ such that

$$\widehat{R}_{l_p 1}(\widehat{x}) \in \mathrm{dom}(\mathcal{C}_p).$$

We define $\mathrm{img}(\mathbf{Q}) \coloneqq \{\widehat{x} \in \widehat{\mathbf{X}} \mid \exists x \in \mathcal{X} \,.\, \widehat{x} \in \mathbf{Q}(x)\}$. Intuitively, $\mathbf{Q}$ maps states $x \in \mathcal{X}$ to the coarsest abstract state $\widehat{x}$ that is both related to x and in the domain of the controller $\mathbf{C}$ (condition (i)). However, if such an abstract state does not exist, $\mathbf{Q}$ maps x to its related layer $l = 1$ states (condition (ii)). It should be noted that $\widehat{Q}_l$ is non-deterministic for states which lie at the boundary of two cells $\widehat{x}, \widehat{x}' \in \widehat{\mathcal{X}}_l$. If such an x happens to be located at the boundary of controller domains computed for different layers, $\mathbf{Q}$ maps x to two abstract cells within different layers.

4.1.3. Multi-Layered Closed Loops

Given a multi-layered controller $\mathbf{C}$ of a multi-layered abstract system $\widehat{\mathbf{S}}$, the usual construction of a closed-loop in Def. 3.2 results in a system which selects the layer l' of the next state non-deterministically (due to (4.2)). This will result in a blocking-behavior of the closed-loop whenever the selected layer does not correspond to a layer currently admitting a control input.

We therefore propose a different closed-loop definition for multi-layered systems which restricts available transitions to those connecting states in the image of $\mathbf{Q}$. Formally, the closed-loop formed by $\widehat{\mathbf{S}}$ and $\mathbf{C}$ is defined as the *multi-layered finite abstract closed-loop* $\widehat{\mathbf{S}}^{cl} = \left\langle \widehat{\mathbf{X}}, \widehat{\mathcal{U}}, \widehat{\mathbf{F}}^{cl} \right\rangle$ such that

$$\widehat{\mathbf{F}}^{cl}(\widehat{x}, \widehat{u}) = \{\widehat{x}' \mid \exists p \in [1; P] \,.\, \widehat{x} \in \mathrm{img}(\mathbf{Q}) \cap \widehat{\mathcal{X}}_{l_p} \wedge \widehat{u} \in \mathcal{C}_p(\widehat{x}) \wedge \widehat{x}' \in \Lambda_p(\widehat{x}, \widehat{u})\} \tag{4.6}$$

where $\widehat{x}' \in \Lambda_p(\widehat{x}, \widehat{u})$ if and only if

$$\exists \widehat{x}'' \in \widehat{F}_{l_p}(\widehat{x}, \widehat{u}) \,.\, \widehat{x}' \in \mathbf{Q}(\widehat{Q}_{l_p}^{-1}(\widehat{x}'')).$$

When refining $\mathbf{C}$ via $\mathbf{Q}$ to a controller for $\overrightarrow{\mathbf{S}}$, the resulting controller should select (i) the current input $u \in \widehat{\mathcal{U}} \subseteq \mathcal{U}$, and (ii) the duration τ_l for which this input should be applied to the underlying continuous system. As $\mathbf{Q}$ might map a particular state $x \in \mathcal{X}$ to multiple abstract states within different layers, we cannot define the refined controller as the serial composition $\mathbf{C} \circ \mathbf{Q}$ in analogy to Def. 3.8. Instead, we directly define the *multi-layered sampled-time abstract closed-loop* in analogy to $\widehat{\mathbf{S}}^{cl}$ as the discrete-time system $\overrightarrow{\mathbf{S}}^{cl} = \left\langle \mathcal{X}, \widehat{\mathcal{U}}, \vec{\mathbf{F}}^{cl} \right\rangle$ such that

$$\vec{\mathbf{F}}^{cl}(x, \widehat{u}) = \{x' \mid \exists p \in [1; P] \,.\, \widehat{x} \in \mathbf{Q}(x) \cap \widehat{\mathcal{X}}_{l_p} \wedge \widehat{u} \in \mathcal{C}_p(\widehat{x}) \wedge x' \in \vec{F}_{l_p}(x, \widehat{u})\}. \tag{4.7}$$

The respective closed-loop behaviors $\mathcal{B}(\widehat{\mathbf{S}}^{cl})$ and $\mathcal{B}(\overrightarrow{\mathbf{S}}^{cl})$ can now be defined as in Def. 3.2.

We extend the definition of control problem and optimal controller for both sampled-time and abstract multi-layered systems in the obvious way. We use the same notation $\left\langle \overrightarrow{\mathbf{S}}, \varphi \right\rangle$, $\left\langle \widehat{\mathbf{S}}, \varphi \right\rangle$, $\mathit{OptCtrl}\left(\overrightarrow{\mathbf{S}}, \varphi\right)$, and $\mathit{OptCtrl}\left(\widehat{\mathbf{S}}, \varphi\right)$ to denote respectively the sampled-time and the abstract multi-layered control problems and the set of sampled-time and the abstract multi-layered optimal controllers.

4.1.4. Multi-Layered Control Problem

Recall that in Prob. 1, we stated the optimal controller synthesis problem for continuous-time systems. Later, we considered a sampled-time approximation of this problem in Prob. 2, where we discretized the time domain using a *uniform* sampling interval. Now we are dealing with a family of sampled-time abstractions having *non-uniform* sampling parameters $\{\tau_l\}$. We accordingly restate Prob. 2 as in the following:

Problem 3 (Multi-layerd Sampled-Time Approximate Controller Synthesis.) *Let $\mathcal{S} = \langle \mathcal{X}, \mathcal{U}, \mathcal{W}, f \rangle$ be a continuous-time control system, $\overrightarrow{\mathbf{S}}$ be a multi-layered sampled-time abstraction, $AP \subset 2^{\mathcal{X}}$ be a finite set of atomic propositions over the alphabet $\mathcal{X}$, and φ be a $LTL_{\setminus\bigcirc}$ specification over AP. The multi-layered sampled-time approximation of Prob 1 asks to find an optimal controller $\mathcal{C} \in OptCtrl\left(\overrightarrow{\mathbf{S}}, \varphi\right)$ for the multi-layered sampled-time control problem $\left\langle \overrightarrow{\mathbf{S}}, \varphi \right\rangle$.*

4.1.5. Soundness

Intuitively, for any two systems $\mathcal{S}_1$ and $\mathcal{S}_2$, $\mathcal{S}_1 \preccurlyeq_Q \mathcal{S}_2$ ensures that given a state $x_1 \in \mathcal{X}_1$, no matter which related state $x_2 \in Q(x_1)$ and enabled control input $u \in \mathcal{U}_2(x_2)$ is used, all resulting states in $F_1(x_1, u)$ and $F_2(x_2, u)$ are related via Q. This intuition can be transferred to the closed-loop systems $\widehat{\mathbf{S}}^{cl}$ and $\overrightarrow{\mathbf{S}}^{cl}$ as follows. Intuitively, $\widehat{\mathbf{S}}^{cl}$ and $\overrightarrow{\mathbf{S}}^{cl}$ can generate all closed-loop trajectories without resolving the non-determinism induced by the set-valued maps $\mathbf{Q}$ and $\mathcal{C}_p$. If $\mathbf{Q}$ is an FRR from $\overrightarrow{\mathbf{S}}^{cl}$ to $\widehat{\mathbf{S}}^{cl}$, any implementation resolving this non-determinism in $\mathbf{Q}$ and $\mathcal{C}_p$ returns a sound closed-loop. This leads us to the following theorem.

Theorem 4.1 *Let $\mathbf{C}$ be a multi-layered controller for the abstract multi-layered system $\widehat{\mathbf{S}}$, $\mathbf{Q}$ be the quantizer induced by $\mathbf{C}$, and $\overrightarrow{\mathbf{S}}^{cl}$ and $\widehat{\mathbf{S}}^{cl}$ respectively be the multi-layered sampled-time and multi-layered finite abstract closed-loop. Then $\overrightarrow{\mathbf{S}}^{cl} \preccurlyeq_{\mathbf{Q}} \widehat{\mathbf{S}}^{cl}$.*

Proof We prove both conditions for FRR separately.
(i) Show $\widehat{\mathbf{F}}^{cl}(\widehat{x}, \widehat{u}) \neq \emptyset \Rightarrow \vec{\mathbf{F}}^{cl}(x, \widehat{u}) \neq \emptyset$: First observe that for any $l \in [1; L]$, it follows from the construction of $\vec{\mathcal{S}}_l$ and $\widehat{\mathcal{S}}_l$ from $\mathcal{S}$ that, for any $\widehat{u} \in \widehat{\mathcal{U}} \subseteq \mathcal{U}$, $\widehat{x} \in \widehat{\mathcal{X}}_l$, and $x \in \mathcal{X}$, it holds that $\widehat{F}_l(\widehat{x}, \widehat{u}) \neq \emptyset$ and $\vec{F}_l(x, \widehat{u}) \neq \emptyset$. With this, it follows from (4.6) and the fact that $\mathbf{Q}$ is strict that $\widehat{\mathbf{F}}^{cl}(\widehat{x}, \widehat{u}) \neq \emptyset$ iff $\widehat{x} \in \mathrm{img}(\mathbf{Q}) \cap \widehat{\mathcal{X}}_{l_p}$ and $\widehat{u} \in \mathcal{C}_p(\widehat{x})$, implying $\vec{\mathbf{F}}^{cl}(\widehat{x}, \widehat{u}) \neq \emptyset$ (from (4.7)).
(ii) Pick $(x, \widehat{x}) \in \mathbf{Q}$ and $\widehat{u} \in \mathcal{U}(\widehat{x})$ and show $\mathbf{Q}(\vec{\mathbf{F}}^{cl}(x, \widehat{u})) \subseteq \widehat{\mathbf{F}}^{cl}(\widehat{x}, \widehat{u})$: first, consider the case that $\widehat{x} \in \widehat{Q}_1(x)$ and case (ii) in the definition of $\mathbf{Q}$ holds. Then $\widehat{\mathbf{F}}^{cl}(\widehat{x}, \widehat{u}) = \emptyset$ and hence $\mathcal{U}(\widehat{x}) = \emptyset$, i.e., the statement trivially holds. Therefore, assume that case (i) of the definition of $\mathbf{Q}$ holds, implying that there exists some $p \in [1; P]$ such that $\widehat{x} \in \mathbf{Q}(x) \cap \widehat{\mathcal{X}}_{l_p}$ and $\widehat{x} \in \mathrm{dom}(\mathcal{C}_p)$. This implies $\widehat{u} \in \mathcal{C}_p(\widehat{x})$ (from part (i)) and therefore $\widehat{x}' \in \mathbf{Q}(\vec{\mathbf{F}}^{cl}(x, \widehat{u}))$ iff

$$\widehat{x}' \in \mathbf{Q}(\vec{F}_{l_p}(x, \widehat{u})). \tag{4.8}$$

Now recall that $\vec{\mathcal{S}}_{l_p} \preccurlyeq_{\widehat{Q}_{l_p}} \widehat{\mathcal{S}}_{l_p}$, which gives us $\widehat{Q}_{l_p}(\vec{F}_{l_p}(x, \widehat{u})) \subseteq \widehat{F}_{l_p}(\widehat{x}, \widehat{u})$. With this we see that (4.8) implies $\widehat{x}' \in \mathbf{Q}(\widehat{Q}_{l_p}^{-1}(\widehat{F}_{l_p}(\widehat{x}, \widehat{u})))$ and hence (4.8) implies $\widehat{x}' \in \Lambda_p(\widehat{x}, \widehat{u})$. With this, it immediately follows from (4.6) that $\widehat{x}' \in \widehat{\mathbf{F}}^{cl}(\widehat{x}, \widehat{u})$. □

Using the properties of feedback refinement relations, Thm. 4.1 implies that the usual soundness property of ABCD stated in Thm. 3.1 can be transferred to the multi-layered setting. This is summarized by the following corollary.

Corollary 4.1 *Given the preliminaries of Thm. 4.1, let* $\mathbf{C} \in \mathit{OptCtrl}(\widehat{\mathbf{S}}, \psi)$ *for a specification* ψ *with associated behavior* $\langle\!\langle \psi \rangle\!\rangle_{\widehat{\boldsymbol{X}}} \subseteq \mathcal{B}(\widehat{\mathbf{S}})$ *and* $\langle\!\langle \psi \rangle\!\rangle_{\mathcal{X}} \subseteq \mathcal{B}(\overrightarrow{\mathbf{S}})$*. Let* $\xi \in \mathcal{B}(\overrightarrow{\mathbf{S}})$ *and* $\widehat{\xi} \in \mathcal{B}(\widehat{\mathbf{S}})$ *be two trajectories such that (i)* $\mathrm{dom}(\xi) = \mathrm{dom}(\widehat{\xi})$*, (ii) for all* $k \in \mathrm{dom}(\xi_1)$*,* $(\xi(k), \widehat{\xi}(k)) \in \mathbf{Q}$*, and (iii)* $\widehat{\xi} \in \langle\!\langle \psi \rangle\!\rangle_{\widehat{\boldsymbol{X}}} \Rightarrow \xi \in \langle\!\langle \psi \rangle\!\rangle_{\mathcal{X}}$*. Then* $\mathcal{B}(\overrightarrow{\mathbf{S}}^{cl})|_{\mathcal{X}} \subseteq \langle\!\langle \psi \rangle\!\rangle_{\mathcal{X}}$*, i.e., the multi-layered sampled-time abstract closed-loop* $\overrightarrow{\mathbf{S}}^{cl}$ *defined in* (4.7) *fulfills specification* ψ*.*

Cor. 4.1 can be interpreted as follows. Consider the control system $\mathcal{S}$ at state x_0. This state is mapped by $\mathbf{Q}$ to $\widehat{x}_0 \in \mathrm{dom}(\mathcal{C}_p)$ for some p. Choosing any $u_0 \in \mathcal{C}_p(\widehat{x}_0)$ and applying this input for time τ_{l_p} to $\mathcal{S}$ results in a continuous trajectory ξ with $x_0 = \xi(0)$ and $x_1 = \xi(\tau_{l_p}) \in \vec{F}_{l_p}(x_0, u)$. Reapplying this procedure leads to an infinite trajectory ξ, with sampled version $\vec{\xi} = x_0 x_1 \ldots \in \mathcal{B}(\overrightarrow{\mathbf{S}})$ and abstract version $\widehat{\xi} = \widehat{x}_0 \widehat{x}_1 \ldots \in \mathcal{B}(\widehat{\mathbf{S}})$. As $\mathbf{C} \in \mathit{OptCtrl}(\widehat{\mathbf{S}}, \psi)$, condition (iii) in Cor. 4.1 ensures that $\vec{\xi} \in \langle\!\langle \psi \rangle\!\rangle_{\mathcal{X}}$. For reachability, this implies that $\vec{\xi}$ (and ξ) eventually reaches the target. For safety, this implies that $\vec{\xi}$ (i.e. sampling instances of ξ) forever remains inside the safe set.

4.2. Multi-Layered Synthesis Algorithms

Our synthesis algorithm of an abstract multi-layered controller $\mathbf{C} \in \mathit{OptCtrl}(\widehat{\mathbf{S}}, \psi)$ has three main ingredients. First, we use the usual 2-player game construction (see Sec. 3.5) from single-layered ABCD to compute the maximal sure winning region (i.e., states which can be controlled to fulfill the specification) and deduce an abstract controller. Second, we allow switching between abstraction layers while solving these single-layered synthesis problems by saving and reloading intermediate results of fixpoint computations from and to the lowest layer (to be discussed in this section). Third, through the use of *frontiers*, we compute abstractions lazily by only computing abstract transitions in parts of the state space currently explored by the fixpoint algorithm (Sec. 4.3). We prove that frontiers always over-approximate the set of states possibly added to the winning region in the corresponding synthesis step.

Let $\mathcal{S}$ be a continuous-time system, $B \subseteq \mathcal{X}$ be a set of safe states for the safety specification $\Box B$, and $O, T \subseteq \mathcal{X}$ be a set of obstacle and target states for the reach-avoid specification $\neg O \,\mathsf{U}\, T$. We present a controller synthesis algorithm which computes a multi-layered abstract controller $\mathbf{C}$ solving the safety and reach-avoid control problems $\left\langle \widehat{\mathbf{S}}, \Box B \right\rangle$ and $\left\langle \widehat{\mathbf{S}}, \neg O \,\mathsf{U}\, T \right\rangle$ over a sequence of L abstract systems $\widehat{\mathbf{S}} := \{\widehat{\mathcal{S}}_l\}_{l \in [1;L]}$; generalization to richer LTL specifications has been left as future work. As shown in Sec. 3.5.1, we use the over and the under-approximations of the atomic propositions B and T to obtain the abstract specifications in different abstraction layers. The abstract safety specification at layer l is given by $\Box \widehat{B}_l$ and the abstract reach-avoid specification at layer l is given by $\neg \widehat{O}_l \,\mathsf{U}\, \widehat{T}_l$.

Here, synthesis will perform the iterative computations for safety and bounded horizon reachability controllers from Sec. 3.5 at each layer, but also switch between abstraction layers during this computation.

The core idea that enables switching between layers during successive steps of the fixed-point iterations are the saving and re-loading of the computed winning states to and from the lowest layer $l = 1$ (indicated in green in the subsequently discussed algorithms). This projection is formalized by the operator

$$\Gamma^{\downarrow}_{ll'}(\Upsilon_{l'}) = \begin{cases} \widehat{R}_{ll'}(\Upsilon_{l'}), & l \leq l' \\ \{\widehat{x} \in \widehat{\mathcal{X}}_l \mid \widehat{R}_{l'l}(\widehat{x}) \subseteq \Upsilon_{l'}\}, & l > l' \end{cases} \tag{4.9}$$

where $l, l' \in [1; L]$ and $\Upsilon_{l'} \subseteq \widehat{\mathcal{X}}_{l'}$. The operation $\Gamma^{\downarrow}_{ll'}(\Upsilon_{l'}) \subseteq \widehat{\mathcal{X}}_l$ under-approximates a set $\Upsilon_{l'} \subseteq \widehat{\mathcal{X}}_{l'}$ with one in layer l.

As already mentioned, we will use the single-layered ABCD algorithm as a subroutine. But we cannot treat it as a complete black-box, since we need to perform abstraction and synthesis in an interleaved fashion. The central ingredient in the synthesis algorithms is the controllable predecessor operator *Cpre*, that we defined in (2.8). To achieve more flexibility and simpler notation, we reinterpret the operator *Cpre* directly over the state space of the abstraction, instead of the game graph. For any given layer l and any set of abstract states $\Psi \subseteq \widehat{\mathcal{X}}_l$ of $\widehat{\mathcal{S}}_l$, we define the controllable predecessor operator over the state space $\widehat{\mathcal{X}}_l$ as:

$$\mathit{Cpre}_{\widehat{\mathcal{S}}_l}(\Psi) \coloneqq \{\widehat{x} \in \widehat{\mathcal{X}}_l \mid \exists \widehat{u} \in \widehat{\mathcal{U}} \,.\, \widehat{F}_l(\widehat{x}, \widehat{u}) \subseteq \Psi\}. \tag{4.10}$$

It is easy to verify that $\mathit{Cpre}_{\widehat{\mathcal{S}}_l}(\Psi)$ is actually equivalent to $\mathrm{dom}(C)$ where $C \in \mathit{OptCtrl}\left(\widehat{\mathcal{S}}_l, \bigcirc\Psi\right)$.

In this section, we shall assume that each $\widehat{F}_l$ *is pre-computed* for all abstract states. (Actually, it suffices to compute $\widehat{F}_l$ for the safe states only, i.e. for the states $\widehat{B}_l$ in case of safety and $\widehat{\mathcal{X}}_l \setminus \widehat{O}_l$ in case of reach-avoid.) In Sec. 4.3, we shall compute $\widehat{F}_l$ lazily.

4.2.1. Safety Specifications

First we consider non-lazy synthesis in this section which assumes that $\widehat{\mathcal{S}}_l$ is pre-computed for all states within the safe set in every $l \in [1; L]$ before SafeIteration is called. This can be formalized by a wrapper function EagerSafe$(\widehat{B}_1, L)$ which first calls ComputeTransitions$(\Gamma^{\downarrow}_{l1}(\widehat{B}_1), l)$ = ComputeTransitions$(\widehat{B}_l, l)$ (see Alg. 2) for every $l \in [1; L]$ and then calls SafeIteration$(\widehat{B}_1, \emptyset, L, \emptyset)$.

Due to the monotonic nature of the iterative computation of safe sets, the set Ψ in Alg. 1 is always a subset of $\widehat{B}_1$ (see Lem. 4.1 for a formal proof). This implies that line 1 of Alg. 1 (indicated in gray) will never perform any exploration (as all needed transition relations are pre-computed) and can therefore be ignored in this section.

When initialized with SafeIteration$(\widehat{B}_1, \emptyset, L, \emptyset)$, Alg. 1 performs the following computations: it starts in layer $l = L$ with an outer recursion count $i = 1$ (not shown in Alg. 1) and reduces l, one step at the time, until $l = 1$ is reached, at which point it then starts over again from layer L with $i = i + 1$ and a new safe set Υ. In every such iteration i, one step of the safety fixed-point is performed for every layer (Line (2)) and

Algorithm 1 SafeIteration

Input: $\Psi \subseteq \widehat{\mathcal{X}}_1$, $\Upsilon \subseteq \widehat{\mathcal{X}}_1$, l, $\mathbf{W}$

1: ComputeTransitions$(\Gamma^{\downarrow}_{l1}(\Psi) \setminus \Gamma^{\downarrow}_{l1}(\Upsilon), l)$
2: $W \leftarrow Cpre_{\widehat{\mathcal{S}}_l}\left(\Gamma^{\downarrow}_{l1}(\Psi)\right) \cap \Gamma^{\downarrow}_{l1}(\Psi)$
3: $\mathbf{W} \leftarrow \mathbf{W} \cup W$ // store the controller domain, but not moves
4: $\Upsilon \leftarrow \Upsilon \cup \Gamma^{\downarrow}_{1l}(W)$
5: **if** $l \neq 1$ **then** // go finer
6: $\quad\langle \Psi, \mathbf{W} \rangle \leftarrow$ SafeIteration$(\Psi, \Upsilon, l-1, \mathbf{W})$
7: $\quad$**return** $\langle \Psi, \mathbf{W} \rangle$
8: **else**
9: $\quad$**if** $\Psi \neq \Upsilon$ **then**
10: $\quad\quad\langle \Psi, \mathbf{W} \rangle \leftarrow$ SafeIteration$(\Upsilon, \emptyset, L, \emptyset)$ // start new iteration
11: $\quad\quad$**return** $\langle \Psi, \mathbf{W} \rangle$
12: $\quad$**else**
13: $\quad\quad$**return** $\langle \Psi, \mathbf{W} \rangle$ // terminate
14: $\quad$**end if**
15: **end if**

the resulting set is stored in the layer 1 map $\Upsilon \subseteq \widehat{\mathcal{X}}_1$ (Line (4)), whereas $\Psi \subseteq \widehat{\mathcal{X}}_1$ keeps the knowledge of the previous iteration. If the finest layer is reached and we have $\Psi = \Upsilon$, the algorithm terminates. Let the maximum value of i upon termination be denoted by N. Otherwise Υ is copied to Ψ, Υ and $\mathbf{W}$ are reset to $\emptyset$ and SafeIteration starts a new iteration (see line 10).

After SafeIteration terminates, it returns a multi-layered controller domain $\mathbf{W} = \{W^l\}_{l\in[1;L]}$, with one controller domain per layer (see Line 3 in Alg. 1). The state-feedback control maps $\{\mathcal{C}^l\}_{l\in[1;L]}$ are computed afterward by choosing one input $\widehat{u} \in \widehat{\mathcal{U}}$ for every $\widehat{x} \in W^l$ such that

$$\widehat{u} = \mathcal{C}^l(\widehat{x}) \implies \widehat{F}_l(\widehat{x}, \widehat{u}) \subseteq \Gamma^{\downarrow}_{l1}(\Psi). \tag{4.11}$$

Note that states encountered for layer l in iteration i are saved to the lowest layer 1 (line 4 of Alg. 1) and "loaded" back to the respective layer l in iteration $i+1$ (line 2 of Alg. 1). Therefore, a state $\widehat{x} \in \widehat{\mathcal{X}}_l$ with $l > 1$, which was not contained in W as computed in layer l and iteration i via line 2 of Alg. 1, might still be included in $\Gamma^{\downarrow}_{l1}(\Psi)$ loaded in the next iteration $i+1$ when re-computing line 2 for l. This happens if all states $x \in \widehat{x}$ were added to Υ by some layer $l' < l$ in iteration i. This allows the algorithm to "bridge" regions that require a finer grid and to use layer L in all remaining regions of the state space. The latter is not true for the multi-layered safety algorithm given by Hsu et al. (2018b), as shown in Hsu et al. (2018a, Sec. I.A).

Algorithm 2 ComputeTransitions

Input: $\Upsilon \subseteq \widehat{\mathcal{X}}_l$, l
1: **for** $\widehat{x} \in \Upsilon, \widehat{u} \in \widehat{\mathcal{U}}$ **do**
2: **if** $\widehat{F}_l(\widehat{x}, \widehat{u})$ is undefined **then**
3: compute $\widehat{F}_l(\widehat{x}, \widehat{u})$
4: **end if**
5: **end for**

Algorithm 3 ExpandAbstraction$_m$

Input: $\Upsilon \subseteq \widehat{\mathcal{X}}_1$, l
1: $W' \leftarrow \mathrm{Pre}^m_{\widehat{A}^L_l}(\Gamma^{\uparrow}_{L1}(\Upsilon)) \setminus \Gamma^{\downarrow}_{L1}(\Upsilon)$
2: $W'' \leftarrow \Gamma^{\downarrow}_{lL}(W')$
3: ComputeTransitions$(W'' \cap \widehat{B}_l, l)$

Soundness and Relative Completeness[1] Due to the effect described above, the map W encountered in line 2 for a particular layer l throughout different iterations i might not be monotonically shrinking. However, the latter is true for layer 1, which is formalized by the following lemma.

Lemma 4.1 *Let* $\Psi^0 := \widehat{B}_1$ *and let* SafeIteration *be called by* EagerSafe$(\widehat{B}_1, L)$*, terminating after* N *iterations. Further, set* $\Psi^i := \Upsilon$ *whenever Alg. 1 reaches line 4 with* $l = 1$ *for the* i*-th time. Then it holds that* $\Psi^i \subseteq \Psi^{i-1}$*, hence* $\Psi^i \subseteq \widehat{B}_1$ *for all* $i \leq N$*.*

Proof Let W^i_l be the set computed in line 2 of Alg. 1 in the i-th iteration for l and observe that $W^i_l = \mathit{Cpre}_{\widehat{\mathcal{S}}_l}(\Gamma^{\downarrow}_{l1}(\Psi^{i-1})) \cap \Gamma^{\downarrow}_{l1}(\Psi^{i-1}) \subseteq \Gamma^{\downarrow}_{l1}(\Psi^{i-1})$. Using (4.9) this implies $\Gamma^{\downarrow}_{1l}(W^i_l) \subseteq \Psi^{i-1}$. As $\Psi^i = \bigcup_{l\in[1;L]} \Gamma^{\downarrow}_{1l}(W^i_l)$ we have $\Psi^i \subseteq \Psi^{i-1}$. □

This leads to our first main result, showing that EagerSafe$(\widehat{B}_1, L)$ is sound and relatively complete.

Theorem 4.2 *Let* $\left\langle \overrightarrow{\mathbf{S}}, \Box B \right\rangle$ *be a multi-layered safety control problem and* $\widehat{\mathbf{S}} = \{\widehat{\mathcal{S}}_l\}_{l\in[1;L]}$ *be a multi-layered finite-state abstraction. Let* $\langle \Psi^N, \mathbf{W} \rangle =$ EagerSafe$(\widehat{B}_1, L)$*, and* $\mathbf{C} = \{C^l\}_{l\in[1;L]}$ *be as defined in* (4.11) *for all* $l \in [1; L]$*. Further, let* W *be the domain of the single-layered optimal safety controller for* $l = 1$*, i.e.* $W = \mathrm{dom}(C)$ *where* $C \in \mathit{OptCtrl}(\widehat{\mathcal{S}}_1, \Box\widehat{B}_1)$*. Then* $\mathbf{C} \in \mathit{OptCtrl}\left(\overrightarrow{\mathbf{S}}, \Box B\right)$ *and* $W \subseteq \Psi^N$*, i.e.* $\mathbf{C}$ *is sound and relatively complete with respect to single-layered controller for layer* $l = 1$*.*

Proof To prove *soundness*, i.e., $\mathbf{C} \in \mathit{OptCtrl}\left(\overrightarrow{\mathbf{S}}, \mathbf{B}\right)$, we show that for every abstract trajectory $\widehat{\xi} \in \mathcal{B}(\widehat{\mathbf{S}})$ and for every $k \in \mathrm{dom}(\widehat{\xi})$, $\mathbf{Q}^{-1}(\widehat{\xi}(k)) \subseteq B$. Then from Cor. 4.1, the soundness claim will follow. The claim is true if for all $l \in [1; L]$ and $\widehat{x} \in \mathrm{dom}(C^l)$, it holds that (i) $\widehat{x} \in \widehat{B}_l$ and (ii) there exists $\widehat{u} \in C^l(\widehat{x})$ such that $\widehat{F}_l(\widehat{x}, \widehat{u}) \neq \emptyset$, and for all $\widehat{x}'' \in \widehat{F}_l(\widehat{x}, \widehat{u})$, $\mathbf{Q}(\widehat{Q}^{-1}_l(\widehat{x}'')) \neq \emptyset$. As in the proof of Lem. 4.1, define W^i_l to be the set computed in line 2 of Alg. 1 in the i-th iteration for l. Lem. 4.1 implies

[1] Absolute completeness of controller synthesis cannot be guaranteed by ABCS; we therefore provide completeness relative to the finest layer.

$W_l^i \subseteq \Gamma_{l1}^{\downarrow}(\Psi^{i-1})$, $\Psi^{i-1} \subseteq \widehat{B}_1$ and $\Gamma_{l1}^{\downarrow}(\widehat{B}_1) = \widehat{B}_l$ we have $W_l^i \subseteq \widehat{B}_l$. Further, line 3 of Alg. 1 implies $\mathrm{dom}(C^l) = W_l^N \subseteq \widehat{B}_l$, proving (i). As $\Psi^N = \Psi^{N-1}$, line 2 of Alg. 1 implies $\mathrm{dom}(C^l) = W_l^N \subseteq \mathit{Cpre}_{\widehat{\mathcal{S}}_l}(\Gamma_{l1}^{\downarrow}(\Psi^N))$. Hence there is $\widehat{u}$ such that (4.11) holds (from (4.10)), implying that $\widehat{F}_l(\widehat{x}, \widehat{u}) \neq \emptyset$. It follows from the definition of $\mathbf{Q}$ that $\mathbf{Q}(\widehat{Q}_l^{-1}(\widehat{x}'')) \neq \emptyset$.

For *completeness*, if $W \not\subseteq \Psi^N$, then there must be a state $\widehat{x} \in W$ and there must be a $j \in [0; N-1]$ such that $\widehat{x} \in \Psi^j$ but $\widehat{x} \notin \Psi^{j+1}$. We perform an induction over j to argue that this will imply: no matter what control action is chosen, there will be a path of length at most $(j+1)$ that will go outside the safe region $\widehat{B}_1$. When $j = 1$, the argument is trivial: If $\widehat{x}$ gets excluded from Ψ^1, then for every control action there must exist a direct transition to $\widehat{B}_1$. Induction hypothesis: Suppose the claim holds for j. Induction step: Let $\widehat{x} \in \Psi^{j+1}$ but $\widehat{x} \notin \Psi^{j+2}$. The safety fixpoint (see (2.10)) implies that for every control input $\widehat{u} \in \widehat{\mathcal{U}}$ from $\widehat{x}$, there is a transition that goes to some abstract state $\widehat{x}'$ outside Ψ^{j+1}. Since $\widehat{x}' \notin \Psi^{j+1}$, from the induction hypothesis it follows that (for every control choice) there is an at most $(j+1)$-long path that goes outside the safe region $\widehat{B}_1$. Thus, no matter what control action is chosen, there is an at most $(j+2)$-long path starting at $\widehat{x}$ and going to the unsafe region. Since $\widehat{x} \in W$ as per our initial assumption, hence we have reached a contradiction. □

It is important to mention that the algorithm EagerSafe is presented only to make a smoother transition to the lazy ABCD for safety (to be presented in Sec. 4.3.1). In practice, EagerSafe itself is of little algorithmic value as it is always slower than the single layered safety control problem $\left\langle \widehat{\mathcal{S}}_1, \Box B \right\rangle$ on the finest layer, but produces the same result. This is because in EagerSafe, the fixed-point computation in the finest layer does not use the coarser layers' winning domain in any meaningful way. So the computation in all the layers—except in $\widehat{\mathcal{S}}_1$—goes to waste.

4.2.2. Reach-Avoid Specifications

We consider the computation of an abstract multi-layered reach-avoid controller $\mathbf{C} \in \mathit{OptCtrl}(\widehat{\mathbf{S}}, \neg O \,\mathsf{U}\, T)$ by the iterative function ReachIteration in Alg. 4 assuming that $\widehat{\mathbf{S}}$ is pre-computed. We refer to this scenario by the wrapper function $\textsc{EagerReach}_m(\widehat{T}_1, \widehat{O}_1, L)$, calling $\textsc{ReachIteration}_m$ with parameters $(\widehat{T}_1, \widehat{O}_1, L, \emptyset)$, where $m \geq 1$ is a tuning parameter whose role will be explained shortly. Assume in this section that ComputeTransitions and $\textsc{ExpandAbstraction}_m$ do not modify anything (i.e., the gray lines of Alg. 4 are ignored in the execution).

The recursive procedure $\textsc{ReachIteration}_m$ in Alg. 4 implements the switching protocol informally discussed in Sec. 1.3.1. Lines 1–12 implement the fixed-point computation at the coarsest layer $\widehat{\mathcal{S}}_L$ by iterating the fixed-point over $\widehat{\mathcal{S}}_L$ until convergence (line 3). Afterward, $\textsc{ReachIteration}_m$ recursively calls itself (line 9) to see if the set of winning states (W) can be extended by a lower abstraction layer. Lines 12–28 implement the fixed-point computations in layers $l < L$ by iterating the fixed-point over $\widehat{\mathcal{S}}_l$ for m steps (line 14) for a given fixed parameter $m > 0$. If the analysis already reaches a fixed point, then, as in the first case, the algorithm $\textsc{ReachIteration}_m$ recursively calls itself (line

Algorithm 4 ReachIteration$_m$

Input: $\Upsilon \subseteq \widehat{\mathcal{X}}_1$,$\Psi \subseteq \widehat{\mathcal{X}}_1$, l, $\mathbf{C}$

1: **if** $l = L$ **then**
2: ComputeTransitions$(\Gamma^{\downarrow}_{l1}(\Psi), l)$
3: Synthesize $\mathcal{C} \in \mathit{OptCtrl}\left(\widehat{\mathcal{S}}_l, \left(\Gamma^{\downarrow}_{l1}(\Psi)\right) \cup \left(\Gamma^{\downarrow}_{l1}(\Upsilon)\right)\right)$
4: $\mathbf{C} \leftarrow \mathbf{C} \cup \{\mathcal{C}\}$
5: $\Upsilon \leftarrow \Upsilon \cup \Gamma^{\downarrow}_{1l}(\mathrm{dom}(\mathcal{C}))$ // save controller domain to Υ
6: **if** $L = 1$ **then** // single-layered reachability
7: **return** $\langle \Upsilon, \mathbf{C} \rangle$
8: **else** // go finer
9: $\langle \Upsilon, \mathbf{C} \rangle \leftarrow$ ReachIteration$_m$ $(\Upsilon, \Psi, l-1, \mathbf{C})$
10: **return** $\langle \Upsilon, \mathbf{C} \rangle$
11: **end if**
12: **else**
13: ExpandAbstraction$_m(\Upsilon, l)$
14: Synthesize $\mathcal{C} \in \mathit{OptCtrl}\left(\widehat{\mathcal{S}}_l, \left(\Gamma^{\downarrow}_{l1}(\Psi)\right) \cup^{\leq m} \left(\Gamma^{\downarrow}_{l1}(\Upsilon)\right)\right)$
15: $\mathbf{C} \leftarrow \mathbf{C} \cup \{\mathcal{C}\}$
16: $\Upsilon \leftarrow \Upsilon \cup \Gamma^{\downarrow}_{1l}(\mathrm{dom}(\mathcal{C}))$ // save controller domain to Υ
17: **if** Fixed-point is reached in line 14 **then**
18: **if** $l = 1$ **then** // finest layer reached
19: **return** $\langle \Upsilon, \mathbf{C} \rangle$
20: **else** // go finer
21: $\langle \Upsilon, \mathbf{C} \rangle \leftarrow$ ReachIteration$_m(\Upsilon, \Psi, l-1, \mathbf{C})$
22: **return** $\langle \Upsilon, \mathbf{C} \rangle$
23: **end if**
24: **else** // go coarser
25: $\langle \Upsilon, \mathbf{C} \rangle \leftarrow$ ReachIteration$_m(\Upsilon, \Psi, l+1, \mathbf{C})$
26: **return** $\langle \Upsilon, \mathbf{C} \rangle$
27: **end if**
28: **end if**

21) to check if further states can be added in a lower layer. If no fixed-point is reached in line 14, more states could be added in the current layer by running the reach-avoid synthesis procedure for more than m steps. However, this might not be efficient (see the example in Sec. 1.3.1). The algorithm therefore attempts to go coarser when recursively calling itself (line 25) to expand the fixed-point in a coarser layer instead. Intuitively, this is possible if states added by lower layer fixed-point computations have now "bridged" a region where precise control was needed and can now be used to enable control in coarser layers again. This also shows the intuition behind the parameter m. If we set it to $m = 1$, the algorithm might attempt to go coarser before this "bridging" is completed. The parameter m can therefore be used as a tuning parameter to adjust the frequency of such

attempts and is only needed in layers $l < L$. The algorithm terminates if a fixed-point is reached in the lowest layer (line 7 and line 19). In this case the layer 1 winning state set Υ and the multi-layered controller $\mathbf{C}$ is returned.

It was shown in (Hsu et al., 2018b) that this switching protocol ensures that EagerReach$_m$ is sound and complete with respect to layer 1.

Theorem 4.3 *Let* $\left\langle \overrightarrow{\mathbf{S}}, \neg O \,\mathsf{U}\, T \right\rangle$ *be a multi-layered reach-avoid control problem and* $\widehat{\mathbf{S}} = \{\widehat{\mathcal{S}}_l\}_{l\in[1;L]}$ *be a multi-layered finite-state abstraction. Let* $\langle \Upsilon, \mathbf{C}\rangle = \textsc{EagerReach}_m(\widehat{T}_1, \widehat{O}_1, L)$. *Further, let* W *be the domain of the single-layered optimal reach-avoid controller for* $l = 1$, *i.e.* $W = \mathrm{dom}(C)$ *where* $C \in \mathit{OptCtrl}(\widehat{\mathcal{S}}_1, \neg\widehat{O}_1 \,\mathsf{U}\, \widehat{T}_1)$. *Then* $\mathbf{C} \in \mathit{OptCtrl}\left(\overrightarrow{\mathbf{S}}, \neg O \,\mathsf{U}\, T\right)$ *and* $W \subseteq \Upsilon$, *i.e.* $\mathbf{C}$ *is sound and relatively complete with respect to single-layered controller for layer* $l = 1$.

Proof To see why EagerReach$_m$ is sound, consider the following induction on the depth of recursive calls d of ReachIteration$_m$. For depth 1 (base case), the single controller that we get in the set $\mathbf{C}$ is sound: this follows from the soundness of single-layered ABCD (in Alg. 4 Line. 3). Moreover, it is easy to observe that the controller obtained in depth $d+1$ can enforce a visit (in at most m steps) to the winning region of the controller obtained in depth d (from Line 9, 21, and 25), implying soundness by induction.

For completeness, we need to show $W \subseteq \Upsilon$. We can write Υ as the union of the single-layered controller domains, denoted as Υ^d, obtained in every recursion depth d of ReachIteration$_m$ in either line 3 or line 14. That is, if the maximum recursion depth is D, then $\Upsilon = \bigcup_{d\in[1;D]} \Upsilon^d$.

First, it should be noted that for any state $x \in \mathcal{X}$ for which $\mathbf{Q}(x) \cap \Upsilon^d \neq \emptyset$ holds, we have: If there exists $d' \in [1; D]$ and $\widehat{x} \in \mathbf{Q}(x) \cap \Upsilon^d$ such that $\widehat{R}_{l_{d'} l_d}(\widehat{x}) \cap \Upsilon^{d'} \neq \emptyset$ then $d' \leq d$. Hence, the quantizer $\mathbf{Q}$ formally defined via a ranking over layers l_d in Sec. 4.1.2 is equivalent to a quantizer defined via the ranking induced by the induction depth d.

To see why $W \subseteq \Upsilon$ holds, recall that, for any $x \in \widehat{Q}_1^{-1}(W)$, every trajectory $\xi \in \mathcal{B}(\vec{\mathcal{S}}_1^{cl})$ with $\xi(0) = x$ will reach (the projection of) the target T, and that the ReachIteration$_m$ algorithm will eventually reach $l = 1$. Now assume that ReachIteration$_m$ was run until depth d where $l = 1$ and let $k' \in \mathrm{dom}(\xi)$ be such that for all $k > k'$ with $k \in \mathrm{dom}(\xi)$, $\xi(k) \in \bigcup_{d'<d} \mathbf{Q}^{-1}(\Upsilon^{d'})$ while this is not true for $\xi(k')$. Given that $l_d = 1$, we execute the single-layered synthesis for $l = 1$ implying that $\xi(k')$ up to $\xi(k' - m)$ will be added to the domain of controller Υ^d and saved in Υ. As $x \in \widehat{Q}_1^{-1}(W)$, ReachIteration$_m$ can only terminate if $\widehat{Q}_1(\xi(0))$ is eventually reached. Therefore, we can apply the above argument iteratively to prove the statement. □

4.3. Lazy Exploration Algorithm within Multi-Layered ABCD

We now consider the case where the multi-layered abstractions $\widehat{\mathbf{S}}$ are computed lazily. Given the multi-resolution fixed-points discussed in the previous section, this requires

tightly over-approximating the region of the state space which might be explored by the reach-avoid or the safety fixpoint in the current layer, called the *frontier*. Then abstract transitions are only constructed for *frontier states* and the currently considered layer l via Alg. 2. As already discussed in Sec. 1.3.1, the computation of *frontier states* differs for safety and reachability objectives.

4.3.1. Safety Specifications

As our main contribution, we now consider the case where the multi-layered abstraction $\widehat{\mathbf{S}}$ is not pre-computed. This is implemented by LazySafe$(\widehat{B}_1, L)$ which simply calls SafeIteration$_1(\widehat{B}_1, \emptyset, L, \emptyset)$. With this, line 1 of Alg. 1 is used to explore transitions in all states in layer l which are (i) not marked unsafe by all layers in the previous iteration, i.e., are in $\Gamma^{\downarrow}_{l1}(\Psi)$, but (ii) cannot stay safe for i times-steps in any layer $l' > l$, i.e., are not in $\Gamma^{\downarrow}_{l1}(\Upsilon)$. In the first iteration of SafeIteration$(\widehat{B}_1, \emptyset, L, \emptyset)$ the set $\Gamma^{\downarrow}_{l1}(\Psi) \setminus \Gamma^{\downarrow}_{l1}(\Upsilon)$ is same as $\Gamma^{\downarrow}_{1L}(\widehat{B}_1) = \widehat{B}_L$. Hence, for layer L all transitions for states inside the safe set are pre-computed in the first iteration of Alg. 1. This is in general not true for lower layers $l < L$.

To ensure that the lazy exploration of the state space is still sound and relatively complete, we show in the following lemma that all states which need to be checked for safety in layer l of iteration i are indeed explored.

Lemma 4.2 *Let W^i_l and Υ^i_l ($\widetilde{W}^i_l$ and $\widetilde{\Upsilon}^i_l$) denote the set computed in line 2 and 4 of Alg. 1 called by* EagerSafe$(\widehat{B}_1, L)$ *(*LazySafe$(\widehat{B}_1, L)$*) the i-th time for layer l. Then for all $i \in [1; N]$ and $l \in [1; L]$, it holds that*

$$\widetilde{W}^i_l \subseteq W^i_l \quad \textit{and} \quad \Upsilon^i_l = \widetilde{\Upsilon}^i_l. \tag{4.12}$$

Proof First observe that the algorithm Alg. 1 (called by EagerSafe$(\widehat{B}_1, L)$ and LazySafe$(\widehat{B}_1, L)$, resp.) starts with $i = 1$ and $l = L$. It first decrements l (while keeping i constant) until $l = 1$ is reached, and then increments i to $i + 1$ and resets $l = 1$ to $l = L$. We prove invariance of W and Υ to both steps separately, to show that (4.12) holds.

First consider the incrementation of i in line 10 of Alg. 1. This implies that Υ and $\widetilde{\Upsilon}$ are copied to Ψ and $\widetilde{\Psi}$. Hence, given the notation of this lemma, we have $\Psi^{i+1}_L = \Upsilon^i_1$ and $\widetilde{\Psi}^{i+1}_L = \widetilde{\Upsilon}^i_1$. Given this observation we do an induction over i to show that $\widetilde{W}^{i+1}_L = W^{i+1}_L$ and $\Upsilon^{i+1}_L = \widetilde{\Upsilon}^{i+1}_L$ are true for all i. For $i = 1$ (base case), $\Upsilon^1_1 = \widetilde{\Upsilon}^1_1 = \widehat{B}_1$, which in turn implies $\widetilde{W}^1_L = W^1_L$. For the induction step, observe that the induction hypothesis implies $\Upsilon = \widetilde{\Upsilon}$ and $\Psi = \widetilde{\Psi}$ in the right side of line 1-4 of Alg. 1, no matter if Alg. 1 is called by EagerSafe$(\widehat{B}_1, L)$ or LazySafe$(\widehat{B}_1, L)$. This implies $\widetilde{W}^{i+1}_L = W^{i+1}_L$ (computed in line 2) and $\Upsilon^{i+1}_L = \widetilde{\Upsilon}^{i+1}_L$ (updated in line 4) whenever the claim holds for i.

Second, we consider decrementing l while keeping i constant. We do an induction over l by assuming $\Upsilon^i_{l+1} = \widetilde{\Upsilon}^i_{l+1}$ and show that this implies $\widetilde{W}^i_l \subseteq W^i_l$ and $\Upsilon^i_l = \widetilde{\Upsilon}^i_l$. Observe that the base case is $l + 1 = L$, which was established by the induction

over i. For the induction step over l, observe that the induction assumption implies $\Gamma^{\downarrow}_{l1}(\widetilde{\Psi}^{i-1}) \setminus \Gamma^{\downarrow}_{l1}(\widetilde{\Upsilon}^{i}_{l+1}) = \Gamma^{\downarrow}_{l1}(\Psi^{i-1}) \setminus \Gamma^{\downarrow}_{l1}(\Upsilon^{i}_{l+1})$ and, as $\Psi^i \subseteq \widehat{B}_1$ (from Lem. 4.1), we have $\Gamma^{\downarrow}_{l1}(\Psi^{i-1}) \setminus \Gamma^{\downarrow}_{l1}(\Upsilon^{i}_{l+1}) \subseteq \Gamma^{\downarrow}_{l1}(\widehat{B}_1)$. Further, observe that EagerSafe$(\widehat{B}_1, L)$ explores $\Gamma^{\downarrow}_{l1}(\widehat{B}_1)$ once, while LazySafe$(\widehat{B}_1, L)$ explores $\Gamma^{\downarrow}_{l1}(\Psi^{i-1}) \setminus \Gamma^{\downarrow}_{l1}(\Upsilon^{i}_{l+1})$ via line 1 of Alg. 1 in every iteration i. This implies that for the computation of $\widetilde{W}^i_l$ in line 10 of Alg. 1 via (4.10) the transition function $\widehat{F}_l(\widehat{x}, \widehat{u})$ is computed for a subset of states compared to the computation of W^i_l. With this it immediately follows from (4.10) that $\widetilde{W}^i_l \subseteq W^i_l$. To show the right side of (4.12), recall that $\widehat{F}_l(\widehat{x}, \widehat{u})$ is at least computed for the set $\Gamma^{\downarrow}_{l1}(\Psi^{i-1}) \setminus \Gamma^{\downarrow}_{l1}(\Upsilon^{i}_{l+1})$ via line 1 of Alg. 1 (and possibly for some more states which were explored in previous iterations) when $\widetilde{W}^i_l$ is computed. Using (4.10) this implies that $\widetilde{W}^i_l \supseteq \left(\mathit{Cpre}_{\widehat{\mathcal{S}}_l}(\Gamma^{\downarrow}_{l1}(\Psi^{i-1})) \setminus \Gamma^{\downarrow}_{l1}(\Upsilon^{i}_{l+1}) \right) \cap \Gamma^{\downarrow}_{l1}(\Psi^{i-1}) = W^i_l \setminus \Gamma^{\downarrow}_{1l}(\Upsilon^{i}_{l+1})$. With this we have $\Upsilon^i_l = \Upsilon^i_{l+1} \cup \Gamma^{\downarrow}_{1l}(W^i_l) = \Upsilon^i_{l+1} \cup \left(\Gamma^{\downarrow}_{1l}(W^i_l) \setminus \Psi'^i_{l+1} \right) = \Upsilon^i_{l+1} \cup \Gamma^{\downarrow}_{1l}\left(W^i_l \setminus \Gamma^{\downarrow}_{l1}(\Upsilon^i_{l+1}) \right) \subseteq \Psi'^i_{l+1} \cup \Gamma^{\downarrow}_{1l}(\widetilde{W}^i_l) = \widetilde{\Upsilon}^i_{l+1} \cup \Gamma^{\downarrow}_{1l}(\widetilde{W}^i_l) = \widetilde{\Upsilon}^i_l$ and $\widetilde{\Upsilon}^i_l = \widetilde{\Upsilon}^i_{l+1} \cup \Gamma^{\downarrow}_{1l}(\widetilde{W}^i_l) = \Upsilon^i_{l+1} \cup \Gamma^{\downarrow}_{1l}(\widetilde{W}^i_l) \subseteq \Psi'^i_{l+1} \cup \Gamma^{\downarrow}_{1l}(W^i_l) = \Upsilon^i_l$, which completes the induction step over l. □

Now as direct consequence of Thm. 4.2 and Lem. 4.2, we present our main result on the lazy multi-layered safety control:

Theorem 4.4 *Let $\left\langle \overrightarrow{\mathbf{S}}, \Box B \right\rangle$ be a multi-layered safety control problem and $\widehat{\mathbf{S}} = \{\widehat{\mathcal{S}}_l\}_{l \in [1;L]}$ be a multi-layered finite-state abstraction. Let $\langle \Psi^N, \mathbf{C} \rangle =$ LazySafe$(\widehat{B}_1, L)$ such that $\mathbf{C} = \{C^l\}_{l \in [1;L]}$ be defined as in (4.11) for all $l \in [1;L]$. Further, let W be the domain of the single-layered safety controller for $l = 1$, i.e. $W = \mathrm{dom}(C)$ where $C \in \mathit{OptCtrl}(\widehat{\mathcal{S}}_1, \Box B)$. Then $\mathbf{C} \in \mathit{OptCtrl}(\mathcal{S}, \mathbf{B})$ and $W \subseteq \Psi^N$, i.e., $\mathbf{C}$ is sound and relatively complete with respect to the single-layered controller for layer $l = 1$.*

Proof First recall that Lem. 4.2 implies $\Psi^i = \widetilde{\Psi}^i$ for all $i \in [1; N]$. Therefore Lem. 4.1 equivalently holds for $\widetilde{\Psi}^i$ and the completeness proof of Thm. 4.2 is equivalent to the one of Thm. 4.4. For the soundness proof, observe that (4.12) implies $B^l = \widetilde{W}^i_l \subseteq W^i_l \subseteq \widehat{B}_l$ and $B^l = \widetilde{W}^i_l \subseteq W^i_l \subseteq \mathit{Cpre}_{\widehat{\mathcal{S}}_l}(\Gamma^{\downarrow}_{l1}(\Psi^N))$, from which (i) and (ii) follow. □

4.3.2. Reach-Avoid Specifications

We now consider the lazy computation of a multi-layered reach-avoid controller $\mathbf{C} \in \mathit{OptCtrl}(\widehat{\mathbf{S}}, \neg O \,\mathsf{U}\, T)$. We refer to this scenario by the wrapper function LazyReach$_m(\widehat{T}_1, \widehat{O}_1, L)$ which calls ReachIteration$_m(\widehat{T}_1, \widehat{O}_1, L, \emptyset)$.

In the first iteration of ReachIteration$_m$ we have the same situation as in LazySafe; given that $\Psi = \widehat{\mathcal{X}}_1 \setminus \widehat{O}_1$, line 2 in Alg. 4 pre-computes all transitions for states inside the safe set and ComputeTransitions does not perform any computations for layer L in further iterations. For $l < L$ however, the situation is different. As the reach-avoid synthesis algorithm is a smallest fixed-point, it iteratively enlarges the set $\widehat{T}_1$ (given when ReachIteration is initialized). Computing transitions for all not yet explored states in

every iteration would therefore be very wasteful (see the example in Sec. 1.3.1). Therefore, ExpandAbstraction$_m$ determines an over-approximation of the frontier states instead in the following manner: it computes the predecessors (not controllable predecessors!) of the already-obtained set Υ optimistically by (i) using (coarse) auxiliary abstractions for this computation and (ii) applying a *cooperative predecessor* operator.

This requires a set of auxiliary systems, given by

$$\widehat{\mathbf{A}} = \{\widehat{A}_l^L\}_{l=1}^L, \qquad \widehat{A}_l^L \coloneqq \widehat{\mathcal{S}}(\mathcal{S}, \tau_l, \eta_L) = \left\langle \widehat{\mathcal{X}}_L, \widehat{\mathcal{U}}, \widehat{F}_l^L \right\rangle. \tag{4.13}$$

The abstract transition function $\widehat{F}_l^L$ induced by $\mathcal{S}$ captures the τ_l-duration transitions in the coarsest layer state space $\widehat{\mathcal{X}}_L$. Using τ_l instead of τ_L is important, as τ_L might cause "holes" between the computed frontier and the current target Υ which cannot be bridged by a shorter duration control actions in layer l. This would render LazyReach$_m$ unsound. Also note that in ExpandAbstraction$_m$, we do not restrict attention to the safe set. This is because $\widehat{R}_l \supseteq \widehat{R}_L$, and when the inequality is strict then the safe states in layer l which are possibly winning but are covered by an obstacle in layer L (see Fig. 1.1 in Chap. 1) can also be explored.

For $\Upsilon \subseteq \widehat{\mathcal{X}}_L$ and $l \in [1; L]$, we define the *cooperative predecessor* operator

$$\mathrm{Pre}_{\widehat{A}_l^L}(\Upsilon) = \{\widehat{x} \in \widehat{\mathcal{X}}_L \mid \exists \widehat{u} \in \widehat{\mathcal{U}} \,.\, \widehat{F}_l^L(\widehat{x}, \widehat{u}) \cap \Upsilon \neq \emptyset\}. \tag{4.14}$$

in analogy to the controllable predecessor operator in (4.10). We apply the cooperative predecessor operator m times in ExpandAbstraction$_m$, i.e.,

$$\begin{aligned} &\mathrm{Pre}^1_{\widehat{A}_l^L}(\Upsilon) = \mathrm{Pre}_{\widehat{A}_l^L}(\Upsilon) \text{ and} \\ &\mathrm{Pre}^{j+1}_{\widehat{A}_l^L}(\Upsilon) = \mathrm{Pre}^j_{\widehat{A}_l^L}(\Upsilon) \cup \mathrm{Pre}_{\widehat{A}_l^L}(\mathrm{Pre}^j_{\widehat{A}_l^L}(\Upsilon)). \end{aligned} \tag{4.15}$$

Calling ExpandAbstraction$_m$ with parameters $\Upsilon \subseteq \widehat{\mathcal{X}}_1$ and $l < L$ applies $\mathrm{Pre}^m_{\widehat{A}_l^L}$ to the over-approximation of Υ by abstract states in layer L. This over-approximation is defined as the dual operator of the under-approximation operator $\Gamma^{\downarrow}_{ll'}$:

$$\Gamma^{\uparrow}_{ll'}(\Upsilon_{l'}) \coloneqq \begin{cases} \widehat{R}_{ll'}(\Upsilon_{l'}), & l \leq l' \\ \{\widehat{x} \in \widehat{\mathcal{X}}_l \mid \widehat{R}_{l'l}(\widehat{x}) \cap \Upsilon_{l'} \neq \emptyset\}, & l > l' \end{cases} \tag{4.16}$$

where $l, l' \in [1; L]$ and $\Upsilon_{l'} \subseteq \widehat{\mathcal{X}}_{l'}$. Finally, m controls the size of the frontier set and determines the maximum progress that can be made in a single backwards synthesis run in a layer $l < L$.

It can be shown that all states which might be added to the winning state set in the current iteration are indeed explored by this frontier construction, implying that LazyReach$_m(\widehat{T}_1, \widehat{O}_1, L)$ is sound and complete with respect to layer 1. In other words, Thm. 4.3 can be transfered from EagerReach$_m$ to LazyReach$_m$.

Theorem 4.5 *Let* $\left\langle \overrightarrow{\mathbf{S}}, \neg O \,\mathsf{U}\, T \right\rangle$ *be a multi-layered reach-avoid control problem and* $\widehat{\mathbf{S}} = \{\widehat{\mathcal{S}}_l\}_{l\in[1;L]}$ *be a multi-layered finite-state abstraction. Let* $\langle \Upsilon, \mathbf{C}\rangle = \textsc{LazyReach}_m(\widehat{T}_1, \widehat{O}_1, L)$. *Further, let* W *be the domain of the single-layered optimal reach-avoid controller for* $l = 1$, *i.e.* $W = \mathrm{dom}(C)$ *where* $C \in \mathit{OptCtrl}(\widehat{\mathcal{S}}_1, \neg\widehat{O}_1 \,\mathsf{U}\, \widehat{T}_1)$. *Then* $\mathbf{C} \in \mathit{OptCtrl}\left(\overrightarrow{\mathbf{S}}, \neg O \,\mathsf{U}\, T\right)$ *and* $W \subseteq \Upsilon$, *i.e.* $\mathbf{C}$ *is sound and relatively complete with respect to single-layered controller for layer* $l = 1$.

We prove Thm. 4.5 using a chain of intermediate results. First we state some properties of $\Gamma^{\downarrow}_{ll'}(\cdot)$ and $\Gamma^{\uparrow}_{ll'}(\cdot)$, which follow from their definitions in a straightforward manner. Let $A_{l'}, B_{l'} \subseteq \widehat{\mathcal{X}}_{l'}$ be any two sets. Then
(a) $\Gamma^{\uparrow}_{ll'}(\cdot) = \Gamma^{\uparrow}_{lk}(\Gamma^{\uparrow}_{kl'}(\cdot))$ and $\Gamma^{\downarrow}_{ll'}(\cdot) = \Gamma^{\downarrow}_{lk}(\Gamma^{\downarrow}_{kl'}(\cdot))$ for all k such that $\min(\{l, l'\}) \leq k \leq \max(\{l, l'\})$.
(b) $\Gamma^{\downarrow}_{ll'}(\cdot)$ and $\Gamma^{\uparrow}_{ll'}(\cdot)$ are monotonic, i.e. $A_{l'} \subseteq B_{l'} \Rightarrow \Gamma^{\downarrow}_{ll'}(A_{l'}) \subseteq \Gamma^{\downarrow}_{ll'}(B_{l'})$ and $A_{l'} \subseteq B_{l'} \Rightarrow \Gamma^{\uparrow}_{ll'}(A_{l'}) \subseteq \Gamma^{\uparrow}_{ll'}(B_{l'})$.
(c) For $l \leq l'$, both $\Gamma^{\downarrow}_{ll'}(\cdot)$ and $\Gamma^{\uparrow}_{ll'}(\cdot)$ are closed under union and intersection.
(d) $l \leq l' \Rightarrow \Gamma^{\downarrow}_{ll'}(\cdot) \equiv \Gamma^{\uparrow}_{ll'}(\cdot)$
(e) $l \leq l' \Rightarrow \Gamma^{\downarrow}_{l'l}(\Gamma^{\downarrow}_{ll'}(A_{l'})) = \Gamma^{\uparrow}_{l'l}(\Gamma^{\downarrow}_{ll'}(A_{l'})) = A_{l'}$. Using (d), we additionally have $l \leq l' \Rightarrow \Gamma^{\downarrow}_{l'l}(\Gamma^{\uparrow}_{ll'}(A_{l'})) = \Gamma^{\uparrow}_{l'l}(\Gamma^{\uparrow}_{ll'}(A_{l'})) = A_{l'}$, i.e., when $l \leq l'$, the composition $\Gamma^{*}_{l'l} \circ \Gamma^{*}_{ll'}$ for $* \in \{\uparrow, \downarrow\}$ is the identity function.
(f) For all $x \in X$, $\widehat{Q}_{l'}(x) \in A_{l'} \Rightarrow \widehat{Q}_l(x) \in \Gamma^{\uparrow}_{ll'}(A_{l'})$. Equivalently, for all $\widehat{x}' \in \widehat{\mathcal{X}}_{l'}$, $\widehat{x}' \in A_{l'} \Rightarrow \widehat{R}_{ll'}(\widehat{x}') \in \Gamma^{\uparrow}_{ll'}(A_{l'})$.
(g) For all $x \in X$, $\widehat{Q}_l(x) \in \Gamma^{\downarrow}_{ll'}(A_{l'}) \Rightarrow \widehat{Q}_{l'}(x) \in A_{l'}$. Equivalently, for all $\widehat{x} \in \widehat{\mathcal{X}}_l$, $\widehat{x} \in \Gamma^{\downarrow}_{ll'}(A_{l'}) \Rightarrow \widehat{R}_{l'l}(\widehat{x}) \in A_{l'}$.

Using (a)-(f), it immediately follows that

$$l < l': \quad \Gamma^{\downarrow}_{ll'}(A_{l'}) \subseteq A_l \Rightarrow A_{l'} \subseteq \Gamma^{\uparrow}_{l'l}(A_l) \tag{4.17a}$$

$$l > l': \quad \Gamma^{\downarrow}_{ll'}(A_{l'}) \subseteq A_l \Leftarrow A_{l'} \subseteq \Gamma^{\uparrow}_{l'l}(A_l) \tag{4.17b}$$

$$\text{Always}: \quad \Gamma^{\downarrow}_{ll'}(A_{l'}) \supseteq A_l \Leftrightarrow A_{l'} \supseteq \Gamma^{\uparrow}_{l'l}(A_l) \tag{4.17c}$$

where $A_l \subseteq \widehat{\mathcal{X}}_l$. The implications in (4.17a) and (4.17b) are strict.

The soundness and relative completeness of $\textsc{LazyReach}_m$ follows from Thm. 4.3, if we can ensure that in every iteration of $\textsc{LazyReach}_m$ the set of states returned by $\textsc{ExpandAbstraction}_m$, for which the abstract transition relation is computed, is *not smaller* than the set of states subsequently added to Ψ by the m-step reach-avoid fixpoint in the next iteration (line 14 of Alg. 4). A crucial monotonicity assumption, fulfilled by our growth bound-based abstractions, is that for any given layer $l \in [1; L]$ and any fixed control input $\widehat{u} \in \widehat{\mathcal{U}}$, the auxiliary transition relation $\widehat{F}^L_l(\cdot, \widehat{u})$ over-approximates the abstract transition relation $\widehat{F}_l(\cdot, \widehat{u})$:

Assumption 1 *Consider any* $l \in [1; L]$ *and any* $\widehat{u} \in \widehat{\mathcal{U}}$. *Let* $\Upsilon_l \subseteq \widehat{\mathcal{X}}_l$ *and* $\Upsilon_L \subseteq \widehat{\mathcal{X}}_L$ *be two sets of abstract states such that* $\Upsilon_l \subseteq \Upsilon_L$.[1] *Then* $\widehat{F}_l(\Upsilon_l, \widehat{u}) \subseteq \widehat{F}_L(\Upsilon_L, \widehat{u})$.

[1] We write $\Upsilon_l \subseteq \Upsilon_L$ with $\Upsilon_l \subseteq \widehat{\mathcal{X}}_l$, $\Upsilon_m \subseteq \widehat{\mathcal{X}}_L$ as short for $\bigcup_{\widehat{x}\in\Upsilon_l} \widehat{x} \subseteq \bigcup_{\widehat{x}\in\Upsilon_L} \widehat{x}$.

We first observe that computing the under-approximation of the m-step cooperative predecessor with respect to the auxiliary system $\widehat{A}_l^L$ of a set Ψ_L (as used in ExpandAbstraction$_m$) over-approximates the set obtained by computing the m-step cooperative predecessor with respect to the abstract system $\widehat{\mathcal{S}}_l$ for a set Ψ_l if Ψ_L over-approximates Ψ_l.

Lemma 4.3 *Let $\widehat{\mathbf{S}}$ be a multi-layered abstract system and $\widehat{\mathbf{A}}$ be a multi-layered auxiliary system satisfying Assumption 1, and let $\Psi_l \subseteq \widehat{\mathcal{X}}_l$ and $\Psi_L \subseteq \widehat{\mathcal{X}}_L$ for some $l < L$ such that $\Psi_L \supseteq \Gamma^{\uparrow}_{Ll}(\Psi_l)$. Then $\Gamma^{\downarrow}_{lL}(\mathrm{Pre}_{\widehat{A}_l^L}(\Psi_L)) \supseteq \mathrm{Pre}_{\widehat{\mathcal{S}}_l}(\Psi_l)$. Furthermore, for all $m > 0$, it holds that $\Gamma^{\downarrow}_{lL}(\mathrm{Pre}^m_{\widehat{A}_l^L}(\Psi_L)) \supseteq \mathrm{Pre}^m_{\widehat{\mathcal{S}}_l}(\Psi_l)$, where $\mathrm{Pre}_{\widehat{\mathcal{S}}_l}$ and $\mathrm{Pre}^m_{\widehat{\mathcal{S}}_l}$ for $\widehat{\mathcal{S}}_l$ are defined analogously to* (4.14) *and* (4.15)*, respectively.*

Proof (Proof of Lem. 4.3) Fix an abstract control input $\widehat{u} \in \widehat{\mathcal{U}}$. Let $\widehat{x} \in \mathrm{Pre}_{\widehat{\mathcal{S}}_l}(\Psi_l)$, which by definition (4.14) implies that $\widehat{F}_l(\widehat{x}, \widehat{u}) \cap \Psi_l \neq \emptyset$. Let $\widehat{y} = \Gamma^{\uparrow}_{Ll}(\{\widehat{x}\})$. Then by observing that $\widehat{x} \subseteq \widehat{y}$, and using Assump. 1, we have $\widehat{F}_l(\widehat{x}) \subseteq \widehat{F}_l^L(\widehat{y}, \widehat{u})$, which implies $\widehat{F}_l^L(\widehat{y}, \widehat{u}) \cap \Psi_L \neq \emptyset$ (since $\Psi_l \subseteq \Psi_L$). Hence, $\widehat{y} \in \mathrm{Pre}_{\widehat{A}_l^L}(\Psi_L)$. Moreover, using (4.17c) we have $\widehat{x} \in \Gamma^{\downarrow}_{lL}(\{\widehat{y}\})$ which leads to $\widehat{x} \in \Gamma^{\downarrow}_{lL}(\mathrm{Pre}^1_{\widehat{A}_l^L}(\Psi_L))$.

The second claim is proven by induction on m. The base case for $m = 1$ is given by the first claim proven above. Now assume that $\Gamma^{\downarrow}_{lL}(\mathrm{Pre}^m_{\widehat{A}_l^L}(\Psi_L)) \supseteq \mathrm{Pre}^m_{\widehat{\mathcal{S}}_l}(\Psi_l)$ holds for some $m > 0$. This together with (4.17c) implies:

$$\mathrm{Pre}^m_{\widehat{A}_l^L}(\Psi_L) \supseteq \Gamma^{\uparrow}_{Ll}(\mathrm{Pre}^m_{\widehat{\mathcal{S}}_l}(\Psi_l)). \tag{4.18}$$

Now note that by (4.15), we have $\mathrm{Pre}^{m+1}_{\widehat{A}_l^L}(\cdot) = \mathrm{Pre}_{\widehat{A}_l^L}(\mathrm{Pre}^m_{\widehat{A}_l^L}(\cdot)) \cup \mathrm{Pre}^m_{\widehat{A}_l^L}(\cdot)$ and it holds that

$$\begin{aligned}
&\Gamma^{\downarrow}_{lL}(\mathrm{Pre}^{m+1}_{\widehat{A}_l^L}(\Psi_L)) \\
=&\Gamma^{\downarrow}_{lL}\left(\mathrm{Pre}_{\widehat{A}_l^L}\left(\mathrm{Pre}^m_{\widehat{A}_l^L}(\Psi_L)\right) \cup \mathrm{Pre}^m_{\widehat{A}_l^L}(\Psi_L)\right) \\
=&\Gamma^{\downarrow}_{lL}\left(\mathrm{Pre}_{\widehat{A}_l^L}\left(\mathrm{Pre}^m_{\widehat{A}_l^L}(\Psi_L)\right)\right) \cup \Gamma^{\downarrow}_{lL}\left(\mathrm{Pre}^m_{\widehat{A}_l^L}(\Psi_L)\right) && (4.19)\\
\supseteq&\mathrm{Pre}_{\widehat{\mathcal{S}}_l}\left(\mathrm{Pre}^m_{\widehat{\mathcal{S}}_l}(\Psi_l)\right) \cup \mathrm{Pre}^m_{\widehat{\mathcal{S}}_l}(\Psi_l) = \mathrm{Pre}^{m+1}_{\widehat{\mathcal{S}}_l}(\Psi_l), && (4.20)
\end{aligned}$$

□

where (4.19) follows from (c) and (4.20) follows by applying the first claim twice: (i) for the left side of the "∪", by replacing Ψ_l and Ψ_L in the first claim of Lem. 4.3 by $\mathrm{Pre}^m_{\widehat{\mathcal{S}}_l}(\Psi_l)$ and $\mathrm{Pre}^m_{\widehat{A}_l^L}(\Psi_L)$ respectively, while noting that (4.18) gives the necessary pre-condition, and (ii) for the right side of the "∪".

Lem. 4.3 can be used to show that ExpandAbstraction$_m$ constructs the transition function $\widehat{F}_l(\widehat{x}, \widehat{u})$ for all $\widehat{x}$ (and all $\widehat{u}$) which are in the winning state set computed by the m-step reach-avoid fixpoint $\mathit{OptCtrl}\left(\widehat{\mathcal{S}}_l, \left(\Gamma^{\downarrow}_{l1}(\Psi)\right) \mathsf{U}^{\leq m} \left(\Gamma^{\downarrow}_{l1}(\Upsilon)\right)\right)$ in line 14 of Alg. 4.

Lemma 4.4 *For all $l < L$, $\Psi \subseteq \widehat{\mathcal{X}}_1$, and $C \in \mathit{OptCtrl}\left(\widehat{\mathcal{S}}_l, \left(\Gamma^{\downarrow}_{l1}(\Psi)\right) \cup^{\leq m} \left(\Gamma^{\downarrow}_{l1}(\Upsilon)\right)\right)$, it holds that $\widehat{x} \in \mathrm{dom}(C) \setminus \Gamma^{\downarrow}_{l1}(\Psi)$ implies $\widehat{x} \in W''$, where W'' is returned by the second line of* ExpandAbstraction$_m(\Psi, l)$.

Proof Let (i) $\widehat{x} \in \mathrm{dom}(C)$ and (ii) $\widehat{x} \notin \Gamma^{\downarrow}_{l1}(\Psi)$. Then it follows from (i) that

$$\widehat{x} \in \mathrm{dom}(C) \Rightarrow\ \widehat{x} \in \mathit{Cpre}^m_{\widehat{\mathcal{S}}_l}(\Gamma^{\downarrow}_{l1}(\Psi)) \Rightarrow\ \widehat{x} \in \mathrm{Pre}^m_{\widehat{\mathcal{S}}_l}(\Gamma^{\downarrow}_{l1}(\Psi)).$$

Consider the inequality $\Gamma^{\uparrow}_{L1}(\Psi) \supseteq \Gamma^{\uparrow}_{Ll}(\Gamma^{\downarrow}_{l1}(\Psi))$ which can be verified from the properties (a)-(f). Then Lem. 4.3 and (g) give

$$\widehat{x} \in \Gamma^{\downarrow}_{lL}(\mathrm{Pre}^m_{\widehat{A}^L_l}(\Gamma^{\uparrow}_{L1}(\Psi))) \Rightarrow \widehat{R}_{Ll}(\widehat{x}) \in \mathrm{Pre}^m_{\widehat{A}^L_l}(\Gamma^{\uparrow}_{L1}(\Psi)).$$

Now (ii) and (g) gives

$$\widehat{R}_{Ll}(\widehat{x}) \notin \Gamma^{\downarrow}_{Ll}(\Gamma^{\downarrow}_{l1}(\Psi)) \Rightarrow\ \widehat{R}_{Ll}(\widehat{x}) \notin \Gamma^{\downarrow}_{L1}(\Psi).$$

Combining the last two observations with (f) and (b) we get

$$\begin{aligned} &\widehat{R}_{Ll}(\widehat{x}) \in \mathrm{Pre}^m_{\widehat{A}^L_l}(\Gamma^{\uparrow}_{L1}(\Psi)) \setminus \Gamma^{\downarrow}_{L1}(\Psi) = W' \\ &\Rightarrow\ \widehat{x} \in \Gamma^{\uparrow}_{lL}(W') \Rightarrow\ \widehat{x} \in \Gamma^{\downarrow}_{lL}(W') = W''. \end{aligned}$$ □

Proof (Proof of Thm. 4.5) With Lem. 4.4, soundness and relative completeness of LazyReach$_m$ directly follows from Thm. 4.3, as shown in the following. We build the proof on top of Thm. 4.3. We prove two things: that both algorithms terminate after the same depth of recursion D, and that the overall controller domain that we get from EagerReach$_m$ is same as the one that we get from LazyReach$_m$, i.e. $\cup_{d\in[1;D]}\mathrm{dom}(\underline{C}^d) = \cup_{d\in[1;D]}\mathrm{dom}(C^d)$, where $\underline{C}^d$ and C^d are the controllers obtained in depth d of the algorithms EagerReach$_m$ and LazyReach$_m$ respectively. (We actually prove a stronger statement: for all $d \in [1; D]$, $\mathrm{dom}(\underline{C}^d) = \mathrm{dom}(C^d)$.) Then, since EagerReach$_m$ is sound and complete with respect to the single-layered reach-avoid ABCD, hence so will be LazyReach$_m$.

The "$\supseteq$" direction of the second proof is trivial and is based on two simple observations: (a) the amount of information of the abstract transition systems $\widehat{\mathbf{S}}$ which is available to LazyReach$_m$ is never greater than the same available to EagerReach$_m$; (b) whenever LazyReach$_m$ invokes ExpandAbstraction$_m$ for computing transitions for some set of abstract states, ExpandAbstraction$_m$ returns the *full information* of the outgoing transitions for those states to LazyReach$_m$. The second part is crucial, as partial information of outgoing transitions might possibly lead to false positive states in the controller domain. Combining these two arguments, we have that for all $d \in [1; D]$ $\mathrm{dom}(\underline{C}^d) \supseteq \mathrm{dom}(C^d)$. (We are yet to show that the maximum recursion depth is D for both the algorithms EagerReach$_m$ and LazyReach$_m$.)

The other direction will be proven by induction on the depth of the recursive calls of the two algorithms. Let $\underline{l}_d$ and l_d be the corresponding layers in depth d of algorithm LazyReach$_m$ and EagerReach$_m$ respectively. It is clear that $\mathrm{dom}(\underline{C}^1) = \mathrm{dom}(C^1)$ and $\underline{l}_1 = l_1 = L$ (induction base) since we start with full abstract transition system for layer L in both cases. Let us assume that for some depth d, $\mathrm{dom}(\underline{C}^{d'}) = \mathrm{dom}(C^{d'})$ and $\underline{l}_{d'} = l_{d'}$ holds true for all $d' \leq d$ (induction hypothesis). Now in LazyReach$_m$, the check in Line 17 of LazyReach$_m$ is fulfilled iff the corresponding check in Line 15 of EagerReach$_m$ (i.e. (Hsu et al., 2018b, Alg. 1)) is fulfilled. This means that $\underline{l}_{d+1} = l_{d+1}$. This shows by induction that (a) the maximum depth of recursion in LazyReach$_m$ and EagerReach$_m$ are the same (call it D), and (b) the concerned layer in each recursive call is the same for both algorithms.

Now in the beginning of depth $d+1$, we have that $\Psi = \cup_{d' \leq d} \Gamma^{\downarrow}_{1 \underline{l}_{d'}} \mathrm{dom}(\underline{C}^{d'}) = \cup_{d' \leq d} \Gamma^{\downarrow}_{1 l_{d'}} \mathrm{dom}(C^{d'})$. From now on, let's call $l_{d+1} = l$ for simpler notation. Let $\widehat{x} \in \widehat{\mathcal{X}}_l$ be a state which was added in depth $d+1$ in the controller domain $\mathrm{dom}(\underline{C}^{d+1})$ for the first time, i.e. (a) $\widehat{x} \in \mathrm{dom}(\underline{C}^{d+1})$, and (b) $\widehat{x} \in \Gamma^{\downarrow}_{l1}(\Psi)$. Then by Lem. 4.4 we have that $\widehat{x} \in W''$.

Since ExpandAbstraction$_m$ also computes all the outgoing transitions from the states in W'' (Line 3 in Alg. 3), hence full information of the outgoing transitions of all the states which are added in $\mathrm{dom}(\underline{C}^{d+1})$ will be available to the LazyReach$_m$ algorithm in depth $d+1$. In other words given $\widehat{x} \in \widehat{\mathcal{X}}_l$, if there is an m-step controllable path from $\widehat{x}$ to Ψ in EagerReach$_m$, there will be an m-step controllable path in LazyReach$_m$ as well. Hence $\widehat{x}$ will be added in $\mathrm{dom}(C^{d+1})$ as well. This proves that for all $d \in [1; D]$ $\mathrm{dom}(\underline{C}^d) \subseteq \mathrm{dom}(C^d)$. □

4.4. Implementation and case study

We have implemented the presented multi-layered controller synthesis algorithms in C++ as an extension to `SCOTS` (Rungger and Zamani, 2016). `SCOTS` is a tool for ABCD using FRR. It natively supports the fixed-point computations for reachability, safety, and reach-avoid specifications. Like `SCOTS`, our implementation uses *binary decision diagrams* (BDDs); a set of abstract states $\widehat{\mathcal{X}}'$ is represented as a BDD over a set of boolean variables. An assignment of the boolean variables uniquely represents one state $\widehat{x} \in \widehat{\mathcal{X}}'$.

4.4.1. An Efficient Under- and Over-Approximation using BDDs

Key to our lazy multi-layered algorithms is the efficient implementation of the operators $\Gamma^{\downarrow}_{ll'}$ and $\Gamma^{\uparrow}_{ll'}$ defined in (4.9) and (4.16) respectively. We observe: since the ratio of η-s of two adjacent layers is 2 (see Sec. 4.1.1), the BDD variables across different layers can be allocated to follow a particular pattern (shown in Fig. 4.1 for the simple case $|\widehat{\mathcal{X}}_L| = 2$). With this observation, we now describe an efficient BDD implementation for $\Gamma^{\downarrow}_{ll'}(\cdot)$ and $\Gamma^{\uparrow}_{ll'}\cdot$ assuming an 1D state space; our implementation generalizes to any state space dimension.

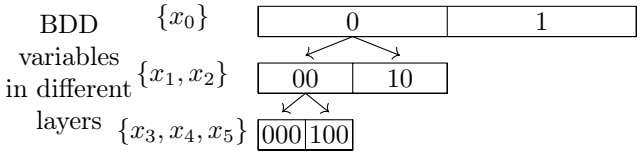


Figure 4.1.: Arrangement of BDD variables across different layers for one state dimension and $|\widehat{\mathcal{X}}_L| = 2$.

Suppose $\mathcal{B}$ is a given BDD defined over the variables $\{x_1, \ldots, x_p\}$, the set $\{y_1, \ldots, y_p\}$ represents a set of new variables, and $[\![\mathcal{B}]\!][\{x_1, \ldots, x_p\}/\{y_1, \ldots, y_p\}]$ represents the BDD obtained from $\mathcal{B}$ by renaming the variables from $\{x_1, \ldots, x_p\}$ to $\{y_1, \ldots, y_p\}$. Recall that BDD renaming is linear in the size of the BDD.

Let $W_l \subseteq \widehat{\mathcal{X}}_l$ be a set of states in layer l represented by the BDD $\mathcal{W}_l$ over a set of boolean variables $\{b_j, b_{j+1}, \ldots, b_k\}$. We compute the BDD representations $\underline{\mathcal{W}}_{l+1} = \Gamma^{\downarrow}_{(l+1)l}(W_l)$ and $\underline{\mathcal{W}}_{l-1} = \Gamma^{\downarrow}_{(l-1)l}(W_l)$ by simple transformations

$$\underline{\mathcal{W}}_{l+1} = [\![\forall b_j.\mathcal{W}_l]\!][\{b_{j+1}, \ldots, b_k\}/\{b_{2j-k}, \ldots, b_{j-1}\}] \text{ and}$$
$$\underline{\mathcal{W}}_{l-1} = [\![\mathcal{W}_l]\!][\{b_j, \ldots, b_k\}/\{b_{k+2}, \ldots, b_{2k-j+2}\}],$$

and the BDD representations $\overline{\mathcal{W}}_{l+1} = \Gamma^{\uparrow}_{(l+1)l}(W_l)$ and $\overline{\mathcal{W}}_{l-1} = \Gamma^{\uparrow}_{(l-1)l}(W_l)$ by simple transformations

$$\overline{\mathcal{W}}_{l+1} = [\![\exists b_j.\mathcal{W}_l]\!][\{b_{j+1}, \ldots, b_k\}/\{b_{2j-k}, \ldots, b_{j-1}\}] \text{ and}$$
$$\overline{\mathcal{W}}_{l-1} = [\![\mathcal{W}_l]\!][\{b_j, \ldots, b_k\}/\{b_{k+2}, \ldots, b_{2k-j+2}\}].$$

4.4.2. Performance Evaluation

We have implemented our algorithms in the tool called `Mascot` and we present some brief evaluation.[1]

Safety Control Problem for a DC-DC Boost Converter (Hsu et al., 2018a)

We evaluate our safety algorithm on a benchmark DC-DC boost converter example from the literature (Girard et al., 2010; Mouelhi et al., 2013; Rungger and Zamani, 2016). The system $\mathcal{S}$ is a second order differential inclusion $\dot{\xi}(t) \in A_p\xi(t) + b + W$ with two switching modes $p \in \{1, 2\}$, where

$$b = \begin{bmatrix} \frac{v_s}{x_l} \\ 0 \end{bmatrix}, A_1 = \begin{bmatrix} -\frac{r_l}{x_l} & 0 \\ 0 & -\frac{1}{x_c}\frac{r_0}{r_0+r_c} \end{bmatrix}, A_2 = \begin{bmatrix} -\frac{1}{x_l}(r_l + \frac{r_0 r_c}{r_0+r_c}) & \frac{1}{5}(-\frac{1}{x_l}\frac{r_0}{r_0+r_c}) \\ 5\frac{r_0}{r_0+r_c}\frac{1}{x_c} & -\frac{1}{x_c}\frac{1}{r_0+r_c} \end{bmatrix},$$

with $r_0 = 1$, $v_s = 1$, $r_l = 0.05$, $r_c = 0.5r_l$, $x_l = 3$, $x_c = 70$ and $W = [-0.001, 0.001] \times [-0.001, 0.001]$. A physical and more detailed description of the model can be found in (Girard et al., 2010). The safety specification that we consider is given by $\Box B$, where

[1] Available at `http://mascot.mpi-sws.org/`.

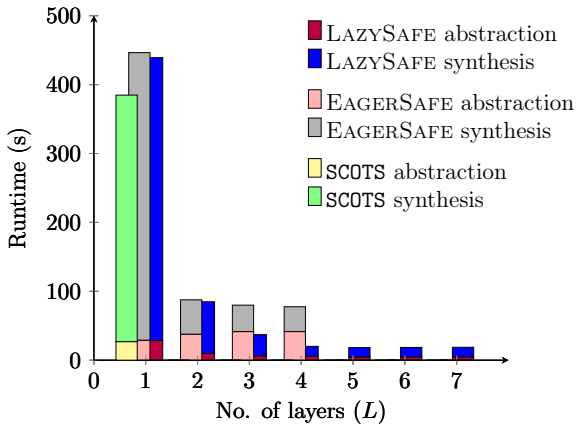


Figure 4.2.: Run-time comparison of LazySafe, EagerSafe, and SCOTS (the single-layered synthesis using the finest layer) on the DC-DC boost converter example. $L > 4$ is not used for EagerSafe since coarser layers fail to produce a non-empty winning set. The same is true for $L > 7$ in LazySafe.

$B = [1.15, 1.55] \times [5.45, 5.85] \subseteq \mathcal{X}$. We evaluate the performance of our LazySafe algorithm on this benchmark and compare it to EagerSafe and SCOTS (the single-layered baseline using the finest layer). For LazySafe and EagerSafe, we vary the number of layers used. The results are presented in Fig. 4.2. In the experiments, the finest layer is common, and is parameterized by $\eta_1 = [0.0005, 0.0005]$ and $\tau_1 = 0.0625$. The ratio between the grid parameters and the sampling times of the successive layers is 2.

From Fig. 4.2, we see that LazySafe is significantly faster than both EagerSafe (and SCOTS) as L increases. The single layered case ($L = 1$) takes slightly more time in both LazySafe and EagerSafe due to the extra bookkeeping in the multi-layered algorithms. In Fig. 4.3, we visualize the domain of the constructed transitions and the synthesized controllers in each layer for LazySafe$(\cdot, 6)$. The safe set is mostly covered by cells in the two coarsest layers. This phenomenon is responsible for the computational savings over LazySafe$(\cdot, 1)$.

Reach-Avoid Control Problem for a Unicycle

We use a nonlinear kinematic system model commonly known as the *unicycle model*, specified as

$$\dot{\xi}_1 \in u_1 \cos(\xi_3) + W_1 \quad \dot{\xi}_2 \in u_1 \sin(\xi_3) + W_2 \quad \dot{\xi}_3 = u_2$$

where ξ_1 and ξ_2 are the state variables representing 2D Cartesian coordinates, ξ_3 is a state variable representing the angular displacement, u_1, u_2 are control input variables that influence the linear and angular velocities respectively, and W_1, W_2 are the perturbation bounds in the respective dimensions given by $W_1 = W_2 = [-0.05, 0.05]$. The perturbations render this deceptively simple problem computationally intensive. We run controller

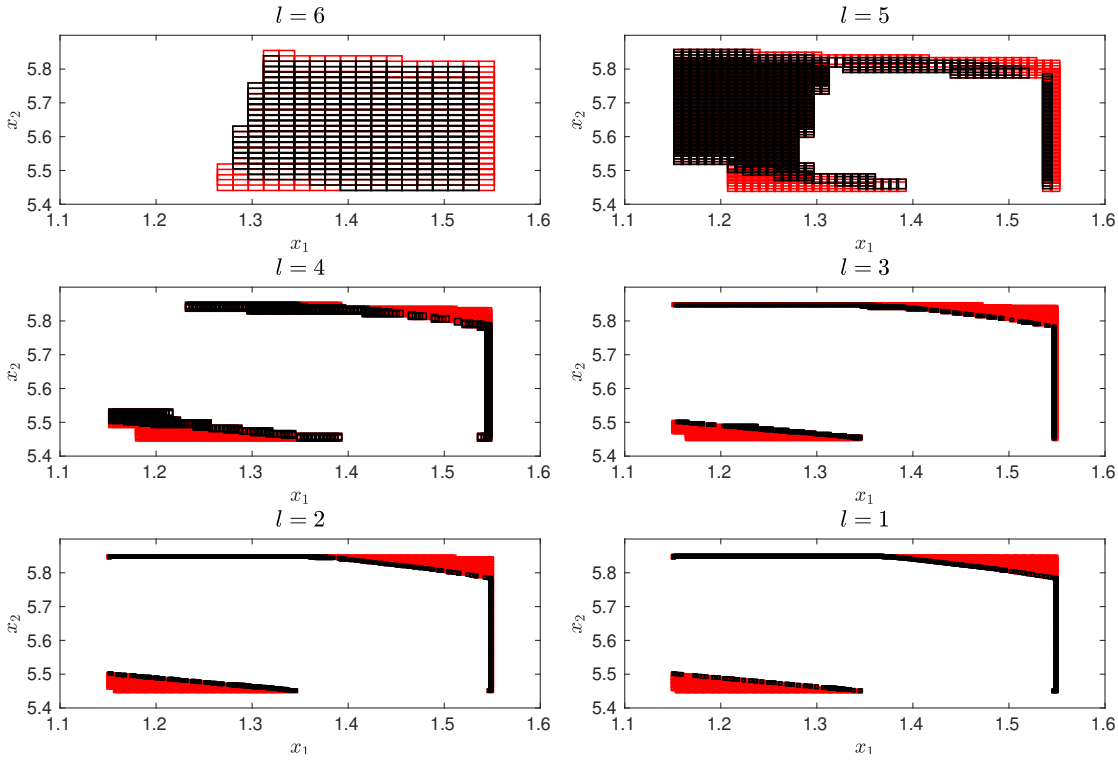


Figure 4.3.: Domain of the computed transitions (union of red and black region) and the synthesized controllers (black region) for the DC-DC boost converter example, computed by LazySafe$(\cdot, 6)$.

synthesis experiments for the unicycle inside a two dimensional space with obstacles and a designated target area, as shown in Fig. 4.4. We use three layers for the multi-layered algorithms EagerReach and LazyReach. All experiments presented in this subsection were performed on a Intel Core i5 3.40 GHz processor.

Algorithm Comparison. Table 4.1 shows a comparison between `SCOTS` (single-layered algorithm using the finest layer), EagerReach$_2$, and LazyReach$_2$ algorithms. The projection to the state space of the transitions constructed by LazyReach$_2$ for the finest abstraction is shown in Fig. 4.4b. The corresponding visualization for EagerReach$_2$ would show all of the uncolored space being covered by red. The savings of LazyReach$_2$ over EagerReach$_2$ can be mostly attributed to this difference.

Varying State Space Complexity. We investigate how the lazy algorithm and the multi-layered baseline perform with respect to the structure of the state space, achieved by varying the number of identical obstacles, o, placed in the open area of the state space. The runtimes for EagerReach$_2$ and LazyReach$_2$ are plotted in Fig. 4.5. We observe that LazyReach$_2$ runs fast when there are few obstacles by only constructing the abstraction in the finest layer for the immediate surroundings of those obstacles. By $o = 20$,

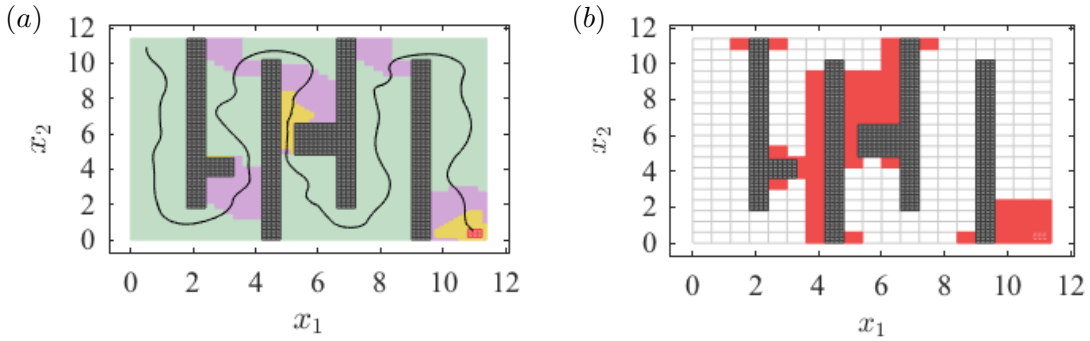


Figure 4.4.: (a) Solution of the unicycle reach-avoid problem by LazyReach$_2$. (b) Cells of the finest layer ($l = 1$) for which transitions were computed during LazyReach$_2$ are marked in red. For EagerReach$_2$, all uncolored cells would also be red.

	SCOTS	EagerReach$_2$	LazyReach$_2$
Abstraction	2590	2628	588
Synthesis	818	73	21
Total	3408 (126%)	2701 (100%)	609 (22.5%)

Table 4.1.: Comparison of running times (in seconds) of reachability algorithms on the perturbed unicycle system.

LazyReach$_2$ explores the entire state space in the finest layer, and its performance is slightly worse than that of EagerReach$_2$ (due to additional bookkeeping). The general decreasing trend in the abstraction construction runtime for EagerReach$_2$ is because transitions outgoing from obstacle states are not computed.

4.5. Related Work (for Scalable ABCD)

Most ABCD techniques in the literature relies on a state-space discretization to be able to handle the general LTL specifications. Notable exceptions are the discretization-free approaches which use control input sequences as abstract states (Corronc et al., 2013; Zamani et al., 2015; Girard and Gößler, 2020); unfortunately, their method only works for stable systems.

There are two broad categories of techniques for addressing the scalability issues in discretization-based ABCD approaches: The first type employs faster numerical methods for computing the abstraction (Coogan and Arcak, 2015), and the second one tries to limit the size of the abstraction. Our lazy approach (Hsu et al., 2019, 2018a) falls in the second category and is orthogonal to the first category.

Among the ones attempting to get a smaller abstraction, there are two subcategories: The first one uses some known structural property of the system such as sparse dependence

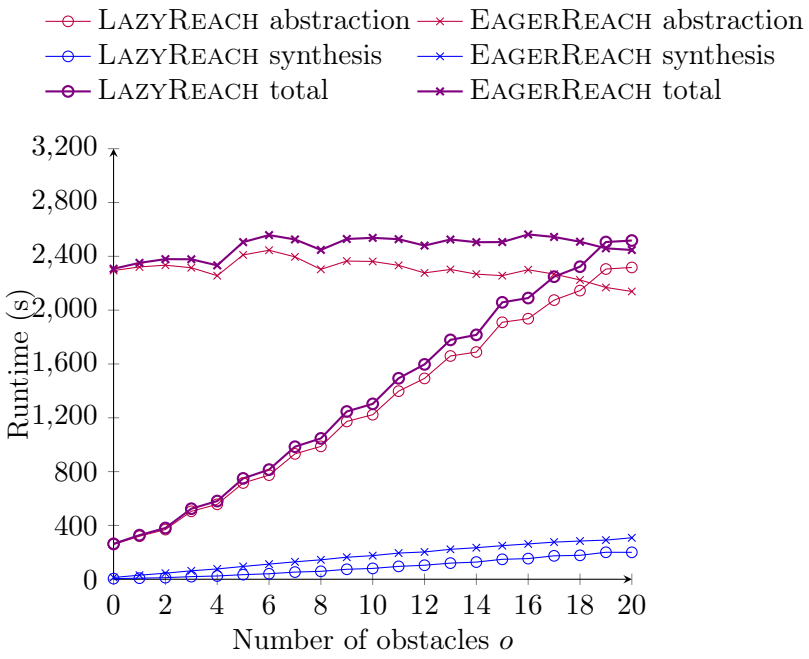


Figure 4.5.: Runtime with increasing number of obstacles

of variables (Gruber et al., 2017), compositionality (Mallik et al., 2018, 2017; Saoud et al., 2020, 2018b), monotonicity (Kim et al., 2017), etc. The second one uses the specification as a guide for the abstraction process, and only computes those parts of the abstraction which are necessary for a successful synthesis (Hussien and Tabuada, 2018; Cámara et al., 2011b; Girard and Gößler, 2020; Nilsson et al., 2017; Dutreix et al., 2020). Our lazy multilayered (Hsu et al., 2019, 2018a) approach falls in this second subcategory and is orthogonal to the first subcategory. Compared to our technique, the other ones either assume stability (Cámara et al., 2011b; Dutreix et al., 2020), performs lazy exploration of inputs instead of states (and thus orthogonal) (Hussien and Tabuada, 2018), or provides a less optimal solution for the safety specifications (Nilsson et al., 2017).

Our exposition in this chapter follows our work on lazy multi-layered ABCD (Hsu et al., 2018b,a).

ABCD continues to be an active research direction, with new results targeting the expressivity and scalability of the method. Over the years, the curse of dimensionality has proven to be the main computational bottleneck for application of ABCD in real-world problems. As the number of system variables increases, the discretization step produces exponentially many states or inputs. There have been many heuristic approaches in the past which have dealt with this scalability issue for ABCD. Broadly, they can be classified into two categories: the first uses the system specification to limit the computational effort, and the second uses the system dynamic model to do the same. Our multi-layered ABCD technique falls in the first category.

Multi-layered algorithms were proposed by Girard et al. (Cámara et al., 2011b,a)

(recently revisited in (Girard et al., 2016)) in the special case of safety and reachability control of *unperturbed* switched systems (our treatment, in contrast, has a disturbance controlled by an adversary).

Subsequently, multi-layered ABCD was implemented in a *lazy* way, by selectively computing the abstraction of finer layers of abstraction as needed, rather than up front (Hsu et al., 2018a,c). The idea there is that we start with a fully computed coarsest layer of abstraction, and locally switch to the finer layers only when the synthesis cannot progress anymore in the coarser layer. Similar techniques also appeared independently in the context of multi-resolution abstractions (Nilsson and Ozay, 2014; Bulancea et al., 2018). An orthogonal approach is to lazily compute the abstraction by only partially and incrementally reavealing the inputs while computing the transitions (Hussien and Tabuada, 2018).

The second approach for improving scalability of ABCD is to exploit the structure of the underlying system model. One notable technique in this category uses the decomposed structure of a given system to compute local abstractions and controllers (Tazaki and Imura, 2008; Mallik et al., 2017, 2018). While the decomposition-based technique addresses systems with many loosely coupled components, there are recent approaches which can even use the loose coupling between different variables of a monolithic system component (Gruber et al., 2017). Another example of such a technique is the fast but relatively imprecise abstraction technique for monotone systems (Kim et al., 2017).

4.6. Conclusion

The known FRR-ABCD algorithm usually performs an uniform state-space discretization at the time of building the finite abstraction (see Chap. 3). This process leads to an exponential blow-up in the size of the abstract state space for high-dimensional systems, causing a huge computational bottleneck. To deal with this issue, we presented our lazy multi-layered ABCD algorithm for reach-avoid and safety specifications. We create multiple layers of abstractions of different granularities. Our synthesis algorithm tries to use the coarsest abstractions as much as possible, and switches to the finer abstractions only when necessary for that particular synthesis problem. We implemented our algorithms in the tool `Mascot-SDS`, using which we demonstrated significant performance improvement over the baseline single-layered version. We leave the more involved ω-regular specifications as part of future work.

5. Abstraction-Based Controller Design for Controlled Markov Processes

In Chap. 3, we saw how (discrete-time) controllers can be synthesized for continuous dynamical systems via Abstraction-Based Controller Design (ABCD). There we used Feedback Refinement Relations (FRR) to first construct a finite abstraction of the system. Then we solved an abstract 2-player game that was constructed by separating the control and the disturbance aspects in the finite abstraction. Finally, the *Player* 0 winning strategy from the abstract game was refined to a controller for the sampled-time abstraction of the given system. It was shown that the obtained controller is sound.

In this chapter, we turn our attention to controller synthesis for continuous-state discrete-time stochastic dynamical systems, formalized as *controlled Markov processes* (CMP). CMPs are control systems with a probabilistic transition function, called the transition kernel. As a result, for a given controller, we obtain a probability distribution over the set of resulting paths. The goal of this chapter is to present a novel ABCD framework for CMPs where we want to satisfy the specification with maximum probability.

5.1. Controlled Markov Processes

Definition 5.1 (Controlled Markov Process (CMP)) A *controlled Markov process (CMP)* is a tuple $\mathcal{S} = \langle \mathcal{X}, \mathcal{U}, T_{\mathfrak{s}} \rangle$, where $\mathcal{X}$ is a Borel space called the *state space*, $\mathcal{U}$ is a finite set called the *input space*, and $T_{\mathfrak{s}}$ is a conditional stochastic kernel $T_{\mathfrak{s}} \colon \mathcal{B}(\mathcal{X}) \times \mathcal{X} \times \mathcal{U} \to [0,1]$ with $\mathcal{B}(\mathcal{X})$ being the Borel sigma-algebra on the state space and $(\mathcal{X}, \mathcal{B}(\mathcal{X}))$ being the corresponding measurable space. The kernel $T_{\mathfrak{s}}$ assigns to any $x \in \mathcal{X}$ and $u \in \mathcal{U}$ a probability measure $T_{\mathfrak{s}}(\cdot|x,u)$ on the measurable space $(\mathcal{X}, \mathcal{B}(\mathcal{X}))$ so that for any set $A \in \mathcal{B}(\mathcal{X})$, $P_{x,u}(A) = \int_A T_{\mathfrak{s}}(ds|x,u)$, where $P_{x,u}$ denotes the conditional probability $P(\cdot|x,u)$.

In general, the input space $\mathcal{U}$ can be any Borel space and the set of valid inputs can be state dependent. While our results can be extended to this setting, for ease of exposition, we consider the special case where $\mathcal{U}$ is a finite set and any input can be taken at any state. This choice is motivated by the digital implementation of controllers with a finite number of possible actuations.

The evolution of a CMP is as follows. For $k \in \mathbb{N}$, let X^k denote the state at the kth time step and A^k the input chosen at that time. If $X^k = x \in \mathcal{X}$ and $A^k = u \in \mathcal{U}$, then

the system moves to the next state X^{k+1}, according to the probability distribution $P_{x,u}$. Once the transition into the next state has occurred, a new input is chosen, and the process is repeated.

Given a CMP $\mathcal{S}$, a *finite path* of length $n+1$ is a sequence

$$w = (x^0, x^1, \ldots, x^n), \quad n \in \mathbb{N},$$

where $x^i \in \mathcal{X}$ are state coordinates of the path. The space of all paths of length $n+1$ is denoted $\mathcal{X}^{n+1}$. An *infinite path* of the CMP $\mathcal{S}$ is the sequence $w = (x^0, x^1, x^2, \ldots)$, where $x^i \in \mathcal{X}$ for all $i \in \mathbb{N}$. The space of all infinite paths is denoted by $\mathcal{X}^\omega$. The spaces $\mathcal{X}^{n+1}$ and $\mathcal{X}^\omega$ are endowed with their respective product topologies and are Borel spaces.

Definition 5.2 (Controller and Probability Measure on Paths) Let $\mathcal{S}$ be a CMP. A *controller* of $\mathcal{S}$ is a universally measurable function $\mathcal{C}\colon \mathcal{X} \to \mathcal{U}$ such that at any time step $n \in \mathbb{N}$, the input u^n is taken to be $\mathcal{C}(x^n) \in \mathcal{U}$. We denote the class of all such controllers by Π. For a given initial probability measure $\alpha\colon \mathcal{B}(\mathcal{X}) \to [0,1]$ of the CMP $\mathcal{S}$, $\mathcal{C}$ induces a unique probability measure on the canonical sample space of paths (Hernández-Lerma and Lasserre, 1996), denoted by $P_\alpha^{\mathcal{C}}$ with the expectation $\mathbb{E}_\alpha^{\mathcal{C}}$.

In this thesis, we only deal with deterministic memoryless controllers (also known as stationary policies), but more general forms of controllers exist in the literature.

In the case when the initial probability measure is supported on a single state $x \in \mathcal{S}$, i.e., $\alpha(x) = 1$, we write $P_x^{\mathcal{C}}$ and $\mathbb{E}_x^{\mathcal{C}}$ in place of $P_\alpha^{\mathcal{C}}$ and $\mathbb{E}_\alpha^{\mathcal{C}}$, respectively. We denote the set of probability measures on $(\mathcal{X}, \mathcal{B}(\mathcal{X}))$ by $\mathfrak{D}$.

Given any ω-regular specification φ defined using a set of predicates over the state space $\mathcal{X}$ of $\mathcal{S}$, we use the notation $\mathcal{S} \models \varphi$ to denote the set of all paths of $\mathcal{S}$ which satisfy φ. Thus, $P_\alpha^{\mathcal{C}}(\mathcal{S} \models \varphi)$ denotes the probability of satisfaction of φ by $\mathcal{S}$ under the effect of the controller $\mathcal{C}$, when the initial probability measure is given by α. Often we will use Linear Temporal Logic (LTL) notation to express ω-regular properties. The syntax and semantics of LTL can be found in standard literature (Baier and Katoen, 2008).

5.2. The Control Problem

Let $\mathcal{S} = \langle \mathcal{X}, \mathcal{U}, T_{\mathfrak{s}} \rangle$ be a CMP and $\mathcal{P} = \langle B_0, B_1, \ldots, B_\ell \rangle$ be a partition of the alphabet $\mathcal{X}$ where the partition elements $B_0, \ldots, B_\ell$ are all measurable sets. We are interested in the maximal probability that the parity specification $\mathit{Parity}(\mathcal{P})$ (see Def. 2.6 for the definition) can be satisfied by paths of a CMP $\mathcal{S}$ starting from a particular state $x \in \mathcal{X}$ under stationary controllers.

The set of all infinite paths $w \in \mathcal{X}^\omega$ of a CMP $\mathcal{S}$ that satisfy the property $\mathit{Parity}(\mathcal{P})$ is represented as the event $\mathcal{S} \models \mathit{Parity}(\mathcal{P})$. The proof of measurability of the event $\mathcal{S} \models \mathit{Parity}(\mathcal{P})$ goes back to the work by Vardi (1985) that provides the proof for probabilistic finite state programs. The proof for a CMP follows similar principles, using the observation that $\mathcal{S} \models \mathit{Parity}(\mathcal{P})$ can be written as a Boolean combination of events $\mathcal{S} \models \Box\Diamond A$, where A is a measurable set, and $\Box\Diamond A$ is a canonical G_δ set in the Borel hierarchy.

It is well-known that every ω-regular specification whose propositions range over measurable subsets of the state space of a CMP can be modeled as a deterministic parity automaton (Gradel and Thomas, 2002, Thm. 1.19). By taking a synchronized product of this parity automaton with the CMP, we can obtain a product CMP and a parity specification over the product state space such that every path satisfying the parity specification also satisfies the original ω-regular specification and vice versa. Moreover, a stationary controller for the parity specification gives a (possibly history-dependent) controller for the original specification. Thus, without loss of generality, we assume that an ω-regular objective is already given as a parity condition using a partition of the state space of the system.

Given a controller $\mathcal{C} \in \Pi$ and an initial state $x \in \mathcal{X}$, we define the satisfaction probability and the supremum satisfaction probability as

$$f(x, \mathcal{C}) \coloneqq P_x^{\mathcal{C}}(\mathcal{S} \models \mathit{Parity}(\mathcal{P})) \text{ and} \tag{5.1}$$

$$f^*(x) \coloneqq \sup_{\mathcal{C} \in \Pi} P_x^{\mathcal{C}}(\mathcal{S} \models \mathit{Parity}(\mathcal{P})), \tag{5.2}$$

respectively. An *optimal controller* for the parity condition is a controller $\mathcal{C}^*$ such that $f^*(x) = f(x, \mathcal{C}^*)$ for all $x \in \mathcal{X}$. Note that an optimal controller need not exist, since the supremum may not be achieved by any controller. Our goal is to study the following *optimal controller synthesis* problem.

Problem 4 (Optimal controller Synthesis) *Given $\mathcal{S}$ and a parity specification $\mathit{Parity}(\mathcal{P})$, find an optimal controller $\mathcal{C}^*$, if it exists, together with $f^*(x)$ such that $P_x^{\mathcal{C}^*}(\mathcal{S} \models \mathit{Parity}(\mathcal{P})) = f^*(x)$.*

While the satisfaction probability (5.1) and the supremum satisfaction probability (5.2) are both well-defined, we are not aware of any work characterizing necessary or sufficient conditions for existence of optimal controllers on continuous-space CMPs for parity specifications. Additionally, we restrict attention to *stationary* controllers. While it is possible to define more general classes of controllers, that depend on the entire history and use randomization over $\mathcal{U}$, we are again unaware of any work that characterizes the class of controllers that are sufficient for optimal control of CMPs for parity specifications. For finite-state systems, stationary controllers are sufficient and we restrict attention to them.

Since we do not yet know whether optimal controllers exist or whether they can be computed, we focus on providing an approximation procedure to compute a possibly sub-optimal controller and guaranteed lower bounds on the optimal satisfaction probability. Our procedure relies on first approximating almost sure winning regions (i.e., where the specification can be satisfied with probability one), and then solving a reachability problem as formalized next.

5.2.1. A Two-Stage Approximate Solution

Definition 5.3 (Almost sure winning region) Given a CMP $\mathcal{S}$, a controller $\mathcal{C}$, and a parity specification $\mathit{Parity}(\mathcal{P})$, the state $x \in \mathcal{X}$ satisfies the specification *almost surely*

if $f(x, \mathcal{C}) = 1$. The almost sure *winning region* of the controller $\mathcal{C}$ is defined as

$$\mathsf{WinDom}(\mathcal{S}, \mathcal{C}) \coloneqq \{x \in \mathcal{X} \mid f(x, \mathcal{C}) = 1\}. \tag{5.3}$$

We also define the *maximal almost sure winning region* as

$$\mathsf{WinDom}^*(\mathcal{S}) \coloneqq \{x \in \mathcal{X} \mid f^*(x) = 1\}. \tag{5.4}$$

Note that $\mathsf{WinDom}(\mathcal{S}, \mathcal{C}) \subseteq \mathsf{WinDom}^*(\mathcal{S})$ for any controller $\mathcal{C} \in \Pi$. It is clear by definition of the winning region that for any controller $\mathcal{C}$, the satisfaction probability $P_x^{\mathcal{C}}(\mathcal{S} \models \mathit{Parity}(\mathcal{P}))$ is equal to 1 for any x in the winning region $W \coloneqq \mathsf{WinDom}(\mathcal{S}, \mathcal{C})$. The next theorem states that this satisfaction probability is lower bounded by the probability of reaching the winning region W for any $x \notin W$. We denote such a reachability by $(\mathcal{S} \models \Diamond W) \coloneqq \{w = (x^0, x^1, x^2, \ldots) \mid \exists n \in \mathbb{N} \,.\, x^n \in W\}$.

Theorem 5.1 *For any controller $\mathcal{C} \in \Pi$ on CMP $\mathcal{S}$, and $W \coloneqq \mathsf{WinDom}(\mathcal{S}, \mathcal{C})$, we have*

$$\begin{array}{ll} P_x^{\mathcal{C}}(\mathcal{S} \models \mathit{Parity}(\mathcal{P})) = 1 & \text{if } x \in W \text{ and} \\ P_x^{\mathcal{C}}(\mathcal{S} \models \mathit{Parity}(\mathcal{P})) \geq P_x^{\mathcal{C}}(\mathcal{S} \models \Diamond W) & \text{if } x \notin W. \end{array} \tag{5.5}$$

Intuitively, the inequality in the second part of (5.5) is due to the $\mathit{Parity}(\mathcal{P})$ specification may be satisfied with positive probability even though the path always stays outside of W. When the state space is finite (i.e., for finite Markov decision processes), equality holds (Baier and Katoen, 2008). However, equality need not hold for general CMPs: Majumdar et al. (2020a) shows an example where the maximal almost sure winning region is empty even though a Büchi specification is satisfied with positive probability.

Proof (Proof of Thm. 5.1) By the definition of the winning set, we already know that $P_s^{\rho}(\mathfrak{S} \models \mathit{Parity}(\mathcal{P})) = 1$ for all $s \in \mathsf{WinDom}(\mathcal{S}, \rho)$. Take any $s \notin W \coloneqq \mathsf{WinDom}(\mathcal{S}, \rho)$. Define τ to be the first time step when the path visits the set W. Note that τ is a random variable taking values in $\mathbb{N} \cup \{\infty\}$. We use the law of total probability by making the event $(\mathfrak{S} \models \mathit{Parity}(\mathcal{P}))$ conditional on τ. Then we have

$$\begin{aligned} & P_s^{\rho}(\mathfrak{S} \models \mathit{Parity}(\mathcal{P})) \\ & = \sum_{n=0}^{\infty} P_s^{\rho}(\mathfrak{S} \models \mathit{Parity}(\mathcal{P}) \,|\, \tau = n) P_s^{\rho}(\tau = n) \\ & + P_s^{\rho}(\mathfrak{S} \models \mathit{Parity}(\mathcal{P}) \,|\, \tau = \infty) P_s^{\rho}(\tau = \infty) \\ & = \mathbb{E}_s^{\rho}\left[P_{s^n}^{\rho}(\mathfrak{S} \models \mathit{Parity}(\mathcal{P}) \,|\, s^1, s^2, \ldots, s^n, \tau = n)\right] \\ & + P_s^{\rho}(\mathfrak{S} \models \mathit{Parity}(\mathcal{P}) \wedge \tau = \infty) \\ & =^* \sum_{n=0}^{\infty} P_s^{\rho}(s^1, s^2, \ldots, s^{n-1} \in \mathcal{S} \backslash W, s^n \in W) \\ & + P_s^{\rho}(\mathfrak{S} \models \mathit{Parity}(\mathcal{P}) \wedge \mathfrak{S} \models \Box \mathcal{S} \backslash W) \\ & \geq P_s^{\rho}(\mathfrak{S} \models \Diamond W). \end{aligned}$$

The equality (*) holds due to $s^n \in W$ and $P_{s^n}^{\rho}(\mathfrak{S} \models \mathit{Parity}(\mathcal{P})) = 1$. □

The next theorem establishes that for any controller $\mathcal{C}$, the winning region is an absorbing set, i.e., paths starting from this set will stay in the set almost surely.

Theorem 5.2 *For any controller $\mathcal{C}$, The set $W = \mathsf{WinDom}(\mathcal{S}, \mathcal{C})$ is an absorbing set, i.e., $T_{\mathfrak{s}}(W|x, \mathcal{C}(x)) = 1$ for all $x \in W$. This implies $P_x^{\mathcal{C}}(\mathcal{S} \models \Diamond \mathcal{X} \backslash W) = 0$ for all $x \in W$.*

The proof of this theorem utilizes the fact that $Parity(\mathcal{P})$ is a long-run property and its satisfaction is independent of the prefix of a path.

Proof (Proof of Thm. 5.2) For any $s \in W$, we have

$$\begin{aligned}
&P_s^{\rho}(\mathfrak{S} \models Parity(\mathcal{P})) \\
&= \int_{\mathcal{S}} P_{s_1}^{\rho}(\mathfrak{S} \models Parity(\mathcal{P})) T_{\mathfrak{s}}(ds_1 | s, \rho(s)) \\
&= \int_{W} T_{\mathfrak{s}}(ds_1 | s, \rho(s)) + \int_{\mathcal{S}\backslash W} P_{s_1}^{\rho}(\mathfrak{S} \models Parity(\mathcal{P})) T_{\mathfrak{s}}(ds_1 | s, \rho(s)).
\end{aligned}$$

This means

$$\begin{aligned}
&\int_{\mathcal{S}\backslash W} (1 - P_{s_1}^{\rho}(\mathfrak{S} \models Parity(\mathcal{P}))) T_{\mathfrak{s}}(ds_1 | s, \rho(s)) = 0 \Rightarrow \\
&\forall \epsilon > 0, P_s^{\rho}\left[(1 - P_{s_1}^{\rho}(\mathfrak{S} \models Parity(\mathcal{P}))) \mathbf{1}_{\mathcal{S}\backslash W}(s_1) \geq \epsilon\right] \leq \frac{0}{\epsilon} = 0,
\end{aligned}$$

where the last inequality is a consequence of Markov's inequality for non-negative random variables. By taking the union over a monotone positive sequence $\{\epsilon_n \to 0\}$, we get

$$\begin{aligned}
&P_s^{\rho}\left[(1 - P_{s_1}^{\rho}(\mathfrak{S} \models Parity(\mathcal{P}))) \mathbf{1}_{\mathcal{S}\backslash W}(s_1) > 0\right] = 0, \\
&P_s^{\rho}\left[s_1 \in \mathcal{S}\backslash W \text{ and } P_{s_1}^{\rho}(\mathfrak{S} \models Parity(\mathcal{P})) < 1\right] = 0, \\
&P_s^{\rho}\left[s_1 \in \mathcal{S}\backslash W\right] = 0.
\end{aligned}$$

□

Thm. 5.1 and Thm. 5.2 enable us to decompose the maximization of $P_x^{\mathcal{C}}(\mathcal{S} \models Parity(\mathcal{P}))$ with respect to controllers $\mathcal{C}$ into two sub-problems. First, find a controller that gives the largest winning region W and employ that controller when the state $x \in W$. Then, find a controller that maximizes the reachability probability $P_x^{\mathcal{C}}(\mathcal{S} \models \Diamond W)$ and employ that controller as long as $x \notin W$.

Computation of the reachability probability has been studied extensively in the literature for both infinite horizon (Tkachev and Abate, 2012, 2014; Tkachev et al., 2017; Haesaert and Soudjani, 2018) and finite horizon (Soudjani and Abate, 2013, 2012; Lahijanian et al., 2015; Kariotoglou et al., 2017; Lesser and Oishi, 2017; Soudjani et al., 2017; Vinod and Oishi, 2018; Lavaei et al., 2019; Jagtap et al., 2019) using different abstract models and computational methods. Together with an algorithm that underapproximates the region of almost sure satisfaction, these approaches can be used to provide a lower bound on the probability of satisfaction of the parity condition. In the rest of the chapter, we focus on the following problem (the first half of (5.5)).

Problem 5 (Approximate Maximal Winning Region) *Given $\mathcal{S}$ and a parity specification Parity($\mathcal{P}$), find a (sub-optimal) controller $\mathcal{C} \in \Pi$, its winning region $\mathsf{WinDom}(\mathcal{S}, \mathcal{C}) \neq \emptyset$, and a bound on the volume of the set difference $\mathsf{WinDom}^*(\mathcal{S}) \backslash \mathsf{WinDom}(\mathcal{S}, \mathcal{C})$.*

In Sec. 5.3-5.5, we provide a solution for Prob. 5 via the paradigm of abstraction-based controller design. Not surprisingly, we get a tighter (i.e., larger) approximation of $\mathsf{WinDom}^*(\mathcal{S})$ if we use a finer discretization of the state space during the abstraction step. We also provide an over-approximation of $\mathsf{WinDom}^*(\mathcal{S})$, and show closeness of the under- and over-approximation of $\mathsf{WinDom}^*(\mathcal{S})$ in the numerical example provided in Sec. 5.6.

5.3. Abstraction-Based Controller Design

The main result of this chapter is a solution to Prob. 5 via a *symbolic algorithm* over abstract $2\frac{1}{2}$*-player games*. This is in the same spirit of abstraction-based controller design (ABCD) for non-stochastic systems as outlined in Chap. 3. The difference is that in standard ABCD techniques for non-stochastic systems, the abstract game is a 2-player game, where the adversarial *Player* 1 controls *both* external disturbances and the discretization-related nondeterminism. The key insight in our abstraction step is that the stochastic nature of the underlying CMP allows choosing disturbances in a *fair* random way instead of purely adversarially. We model this by introducing an additional *random* player (also called the $\frac{1}{2}$-player) resulting in a so called $2\frac{1}{2}$-player game (Condon, 1992; Chatterjee et al., 2003; Chatterjee and Henzinger, 2012). In the resulting abstract game, only the effect of the discretization is handled by *Player* 1 in an adversarial manner. The random player picks the external disturbance uniformly at random.

In Sec. 5.3.1, we show how a CMP can be abstracted into a $2\frac{1}{2}$-player game. We defer the computation of the optimal almost sure winning strategy to Chap. 7. In Sec. 5.3.3, we show how an almost sure winning strategy in the abstract $2\frac{1}{2}$-player game is refined, and that the resulting controller is almost sure winning for the original CMP and its associated parity specification. This establishes soundness of our ABCD technique to solve Problem 5.

5.3.1. Abstraction: CMPs to $2\frac{1}{2}$-Player Games

Given a CMP $\mathcal{S} = \langle \mathcal{X}, \mathcal{U}, T_s \rangle$ and a parity specification *Parity*($\mathcal{P}$) for a partition $\mathcal{P} = \langle B_0, B_1, \ldots, B_\ell \rangle$ of the state space $\mathcal{X}$ we construct an abstract $2\frac{1}{2}$-player game $\mathcal{G} = \langle V, V_0, V_1, V_r, E \rangle$ with an abstract parity winning condition *Parity*($\widehat{\mathcal{P}}$). In the abstract game, we relax the restriction that $\widehat{\mathcal{P}}$ needs to be a partition of the set of vertices V. In order to ensure that *Parity*($\widehat{\mathcal{P}}$) is well defined, we impose the restriction that every infinite play must have infinitely many occurrences of vertices from at least one of the sets in $\mathcal{P}$. In other words, we require that every set of vertices $U \subseteq V$ for which there is no $i \in [1; \ell]$ with $U \cap B_i \neq \emptyset$ must be "transient" vertices.

State-space abstraction. Like abstraction for non-stochastic systems (Sec. 3.4), we construct a finite cover over the continuous state space in order to form the abstract state

space. A cover $\widehat{\mathcal{X}}$ of the state space $\mathcal{X}$ is a set of non-empty, closed subsets of $\mathcal{X}$—called the *abstract states*—such that every $x \in \mathcal{X}$ belongs to at least one abstract state. The cover $\widehat{\mathcal{X}}$ is called the *abstract state space*. For the moment, when we present the ABCD theory for CMPs, we assume that the shape of the abstract states can be anything. Later, in Sec. 5.4, we will present an abstraction algorithm for which we will require the abstract states to be hyper-rectangular.

We introduce the *abstraction relation* $Q \subseteq \mathcal{X} \times \widehat{\mathcal{X}}$ as a relation between the continuous and the abstract states; formally $Q \coloneqq \{(x, \widehat{x}) \in \mathcal{X} \times \widehat{\mathcal{X}} \mid x \in \widehat{x}\}$. The inverse of the abstraction relation is called the *concretization relation*, and is written as $Q^{-1} \subseteq \widehat{\mathcal{X}} \times \mathcal{X}$, $Q^{-1} \coloneqq \{(\widehat{x}, x) \in \widehat{\mathcal{X}} \times \mathcal{X} \mid (x, \widehat{x}) \in Q\}$. Syntactically, the abstraction relation Q is same as the Feedback Refinement Relation (FRR) considered in Chap. 3. However, we will not use the properties of the FRR in this chapter, and hence we use this different name to avoid confusion.

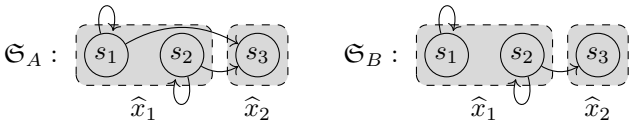
Transition abstraction. We also introduce an over- and an under-approximation of the probabilistic transitions of the CMP $\mathcal{S}$ using the non-deterministic abstract transition functions $\overline{F}\colon \widehat{\mathcal{X}} \times \mathcal{U} \to 2^{\widehat{\mathcal{X}}}$ and $\underline{F}\colon \widehat{\mathcal{X}} \times \mathcal{U} \to 2^{\widehat{\mathcal{X}}}$ with the following properties:

$$\overline{F}(\widehat{x}, u) \supseteq \{\widehat{x}' \in \widehat{\mathcal{X}} \mid \exists x \in \widehat{x} \,.\, T_{\mathfrak{s}}(\widehat{x}' \mid x, u) > 0\}, \tag{5.6a}$$

$$\underline{F}(\widehat{x}, u) \subseteq \{\widehat{x}' \in \widehat{\mathcal{X}} \mid \exists \varepsilon > 0 \,.\, \forall x \in \widehat{x} \,.\, T_{\mathfrak{s}}(\widehat{x}' \mid x, u) \geq \varepsilon\}. \tag{5.6b}$$

To understand the need for both $\overline{F}$ and $\underline{F}$ and the way they are constructed, consider the following examples. Intuitively, given an abstract state $\widehat{x}$ and an input u, the set $\overline{F}$ over-approximates the set of abstract states reachable by probabilistic transitions from $\widehat{x}$ on input u. On the other hand, $\underline{F}$ under-approximates those abstract states which can be reached by *every* state in $\widehat{x}$ with probability bounded away from zero.

Example 5.1 Consider the two CMPs, $\mathcal{S}_A$ and $\mathcal{S}_B$:



The circles are concrete states x_i, the dashed boxes denote abstract states $\widehat{x}_i$, and the edges denote transitions with positive probability between concrete states x_i. Consider the left abstract state $\widehat{x}_1$. Here, the adversary decides which concrete state (i.e., x_1 or x_2) the game is in. In both $\mathcal{S}_A$ and $\mathcal{S}_B$, $\overline{F}$ says that both $\widehat{x}_1$ and $\widehat{x}_2$ are reachable from $\widehat{x}_1$. In $\mathcal{S}_A$, $\underline{F}$ contains both $\widehat{x}_1$ and $\widehat{x}_2$, in $\mathcal{S}_B$, $\underline{F}$ is empty. An adversary that plays according to $\overline{F}$ is too strong: it can keep playing the self loop in x_2, while the stochastic nature of the CMP ensures that eventually x_2 will transition to x_3. In order to follow the probabilistic semantics, we must ensure the adversary picks a distribution whose support contains both abstract states.

In $\mathcal{S}_A$, the probabilistic behavior of the two concrete states x_1 and x_2 are very different: x_1 stays in $\widehat{x}_1$ with probability one and x_2 stays in $\widehat{x}_1$ or moves probabilistically to $\widehat{x}_2$.

To ensure correct behavior, we look at possible supports of distributions induced by the dynamics: these are the possible subsets of abstract states between $\underline{F}$ and $\overline{F}$. Here, the game either stays in $\widehat{x}_1$ or (eventually) moves to $\widehat{x}_2$ and, in our reduction, we force the adversary to commit to one of the two options. □

The parameter ε states that there is a uniform lower bound on transition probabilities for all states in an abstract state. This ensures that, provided $\widehat{x}$ is visited infinitely often and u is applied infinitely often from $\widehat{x}$, then $\widehat{x}'$ will be reached almost surely from $\widehat{x}$. In the absence of a uniform lower bound, this property need not hold for infinite state systems; for example, if the probability goes to zero, the probability of escaping $\widehat{x}$ can be strictly less than one.

While it is difficult to compute $\overline{F}$ and $\underline{F}$ in general, they can be approximated for the important subclass of stochastic dynamical systems with *affine* disturbances; we outline this approximation procedure later in Sec. 5.4.

Abstract $2\frac{1}{2}$-player game graph. Given the abstract state space $\widehat{\mathcal{X}}$ and the over and under-approximations of the transition functions $\underline{F}$ and $\overline{F}$, we are ready to construct the abstract $2\frac{1}{2}$-player game graph induced by a CMP.

Definition 5.4 Let $\mathcal{S} = \langle \mathcal{X}, \mathcal{U}, T_{\mathfrak{s}} \rangle$ be a given CMP, $\widehat{\mathcal{X}}$ be the abstract state space, and $\overline{F}$ and $\underline{F}$ be the abstract transition functions as defined in (5.6). Then the induced abstract $2\frac{1}{2}$-player game graph $\mathcal{G} = \langle V, V_0, V_1, V_r, E \rangle$ is defined as:

- $V_0 = \widehat{\mathcal{X}}$ and $V_1 = \widehat{\mathcal{X}} \times \mathcal{U}$;
- $V_r = \bigcup_{v_1 \in V_1} V_r(v_1)$, where
  $V_r(v_1) := \{v_r \subseteq \widehat{\mathcal{X}} \mid \underline{F}(v_1) \subseteq v_r \subseteq \overline{F}(v_1), 1 \leq |v_r| \leq |\underline{F}(v_1)| + 1\}$;
- and it holds that
  - for all $v_0 \in V_0$, $E(v_0) = \{(v_0, u) \mid u \in \mathcal{U}\}$
  - for all $v_1 \in V_1$, $E(v_1) = V_r(v_1)$, and
  - for all $v_r \in V_r$, $E(v_r) = \{v_0 \in V_0 \mid v_0 \in v_r\}$.

Note that $V_r(v_1)$ contains non-empty subsets of $\widehat{\mathcal{X}}$ that includes all the abstract states in $\underline{F}(v_1)$ and possibly include only one additional element from $\overline{F}(v_1)$. The construction is illustrated in Fig. 5.1.

In the reduced game, *Player* 0 models the controller, *Player* 1 models the effect of discretization of the state space of $\mathcal{S}$, and the random edges from the states in V_r model the stochastic nature of the transitions of $\mathcal{S}$. Intuitively, the game graph in Def. 5.4 captures the following interplay which is illustrated in Fig. 5.1: At every time step, the controller for $\mathcal{S}$ has to choose a control input $u \in \mathcal{U}$ based on the current vertex $\widehat{x}$ of $\mathcal{G}$. Since the controller is oblivious to the precise continuous state $x \in \mathcal{X}$ of $\mathcal{S}$, hence u is required to be an optimal choice for *every* continuous state $x \in \widehat{x}$. This requirement is materialized by introducing a fictitious adversary (i.e. *Player* 1) who, given $\widehat{x}$ and u, picks a continuous state $x \in \widehat{x}$ from which the control input u is to be applied. Now, we know that no matter what continuous state x is chosen by *Player* 1, $T_{\mathfrak{s}}(\underline{F}(\widehat{x}, u) \mid x, u) > \varepsilon$

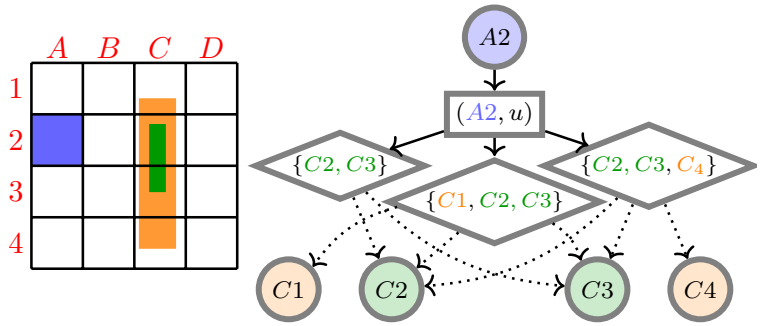


Figure 5.1.: Illustration of the construction of the abstract $2^1/_2$-player game (right) from a continuous-state CMP (left). The state space of the CMP is discretized into rectangular abstract states $A1, \ldots, D3$; $\underline{F}(A2, u) = \{C2, C3\}$ (intersecting the green region), and $\overline{F}(A2, u) = \{C1, C2, C3, C4\}$ (intersecting orange region). V_0, V_1 and V_r are indicated by circle, rectangular and diamond-shaped vertices. Random vertices are dashed.

holds for some $\varepsilon > 0$. This explains why every successor of the $(\widehat{x}, u) \in V_1$ states contains the set of vertices $\underline{F}(\widehat{x}, u)$. Moreover, depending on which exact $x \in \widehat{x}$ *Player* 1 chooses, with positive probability the system might go to some state in $\overline{F}(\widehat{x}, u) \setminus \underline{F}(\widehat{x}, u)$. This is materialized by adding every state in $\overline{F}(\widehat{x}, u) \setminus \underline{F}(\widehat{x}, u)$ at a time to the successors of the states in V_1 (see Def. 5.4). Finally, we assume that the successor from every state in V_r is chosen uniformly at random (indicated by dotted edges in Def. 5.4). Later, it will be evident that the exact probability values are never used for obtaining the almost sure winning region, and so we could have chosen any other probability distribution.

Abstract parity winning condition. To conclude the abstraction of a given CMP $\mathcal{S}$ and its parity specification *Parity*$(\mathcal{P})$ with the set of priorities $\mathcal{P} = \langle B_0, \ldots, B_\ell \rangle$, we have to formally translate the priority sets B_i over subsets of states of the CMP into a set of priorities $\widehat{\mathcal{P}} = \left\langle \widehat{B}_0, \ldots, \widehat{B}_\ell \right\rangle$ partitioning the vertices of the abstract $2^1/_2$-player game graph $\mathcal{G}$. To this end, we use the same method from Sec. 3.5.1 to assign priorities to the set of *Player* 0 vertices V_0 (i.e. the abstract states $\widehat{x}$). Let $\widehat{\mathcal{P}} = \left\langle \widehat{B_0}, \ldots, \widehat{B_\ell} \right\rangle$ be the set of abstract priorities. Clearly, the set $\widehat{\mathcal{P}}$ does not partition the set of vertices V, unlike our formalization of parity winning condition in Def. 2.6. Indeed, we implicitly assign an "undefined" color "$-$" to all vertices $V_1 \cup V_r$. Thereby, we only interpret the given parity winning condition over a projection of a play to its *Player* 0 vertices. Formally, a play ρ over the abstract game graph $\mathcal{G}$ starting from a vertex $x^0 \in V_0$ is of the form:

$$\rho = x^0, (x^0, u^0), (\{x^{0,0}, \ldots, x^{0,i_0}\}), x^1, (x^1, u^1), (\{x^{1,0}, \ldots, x^{1,i_1}\}), \ldots$$

where $x^k \in \{x^{k,0}, \ldots, x^{k,i_k}\}$ for all $k \in \mathbb{N}$. The projection of the play ρ to the player 0 states is defined as $\mathrm{Proj}_{V_0}(\rho) = x^0, x^1, \ldots$. Let φ be an ω-regular winning condition defined using a set of predicates over V_0. We use the convention that $(\mathcal{G} \models \varphi)$ will denote the set of every infinite play ρ of $\mathcal{G}$, for any arbitrary pair of strategies of *Player* 0 and

Player 1, such that $\mathrm{Proj}_{V_0}(\rho)$ satisfies φ. This convention is well-defined because every infinite play of $\mathcal{G}$ will have infinitely many occurrences of vertices from V_0 in it: This follows from the strict alternation of the vertices in V_0, V_1, and V_r, as per Def. 5.4.

5.3.2. Abstract Controller Synthesis

So far we have seen how the $2\frac{1}{2}$-player parity game $\left\langle \mathcal{G}, \mathit{Parity}(\widehat{\mathcal{P}}) \right\rangle$ is constructed from the CMP $\mathcal{S}$ and the specification $\mathit{Parity}(\mathcal{P})$ according to Def. 5.4 and Sec. 3.5.1. Next, we need to solve the game $\left\langle \mathcal{G}, \mathit{Parity}(\widehat{\mathcal{P}}) \right\rangle$ to compute the almost sure winning states and the associated almost sure winning strategy of *Player* 0. This can be performed using the efficient symbolic fixpoint algorithm presented in Sec. 7. Our symbolic fixpoint exploits the intrinsic fairness properties of the random edges: If a random vertex is visited infinitely often then *every* random edge will be taken infinitely often. We take the existing fixpoints for 2-player games, and transform them by embedding this fairness property. The resulting novel fixpoints compute the almost sure winning region and the almost sure winning strategy for the 2.5-player games, while retaining the computational aspects of the original 2-player fixpoint algorithms. The outcome of the synthesis step is a deterministic memoryless strategy for *Player* 0.

5.3.3. Controller Refinement

Suppose that the abstract $2\frac{1}{2}$-player parity game $\left\langle \mathcal{G}, \mathit{Parity}(\widehat{\mathcal{P}}) \right\rangle$ constructed from the CMP $\mathcal{S}$ via Def. 5.4 and Sec. 3.5.1 has been solved as discussed in Sec. 5.3.2. Hence, we know the almost sure winning region and the associated deterministic memoryless strategy $\pi_0 \in \Pi^{\mathrm{DM}}$ for *Player* 0. We refine π_0 to a controller $\mathcal{C} \in \Pi$ for the CMP $\mathcal{S}$ as follows.

Definition 5.5 (Controller refinement) Let $\mathcal{S}$ be a CMP with parity specification $\mathit{Parity}(\mathcal{P})$ and $\left\langle \mathcal{G}, \widehat{\mathcal{P}} \right\rangle$ its induced finite $2\frac{1}{2}$-player parity game with deterministic memoryless almost sure *Player* 0 winning strategy $\pi_0 \in \Pi^{\mathrm{DM}}$. Then a controller $\mathcal{C} \in \Pi$ is called *the refinement of* π_0 iff for every $x \in \mathcal{X}$, if $x \in \widehat{x}$ for some $\widehat{x} \in \widehat{\mathcal{X}}$, and if $\pi_0(\widehat{x}) = (\widehat{x}, u) \in V_1$ for some $u \in \mathcal{U}$, then $\mathcal{C}(x) \coloneqq u$.

With the completion of this last step of our ABCD method for stochastic nonlinear systems we can finally state our main theorem providing a solution to Problem 5, which we prove in Sec. 5.5.

Theorem 5.3 (Solution of Problem 5) *Let $\mathcal{S}$ be a CMP and $\mathit{Parity}(\mathcal{P})$ be a given parity specification. Let $\left\langle \mathcal{G}, \mathit{Parity}(\widehat{\mathcal{P}}) \right\rangle$ be the abstract $2\frac{1}{2}$-player game defined in Def. 5.4. Suppose, a vertex $\widehat{x} \in V_0$ is almost sure winning for Player 0 in the game $\left\langle \mathcal{G}, \mathit{Parity}(\widehat{\mathcal{P}}) \right\rangle$, and $\pi_0 \in \Pi^{\mathrm{DM}}$ is the corresponding Player 0 winning strategy. Then the refinement $\mathcal{C}$ of π_0 ensures that $\widehat{x} \subseteq \mathsf{WinDom}(\mathcal{S}, \mathcal{C})$.*

Remark 3 (Quality of the Approximation) *An* over-*approximation of the optimal almost sure winning domain* $\mathsf{WinDom}^*(\mathcal{S})$ *of* $\mathcal{S}$ *with respect to Parity*$(\mathcal{P})$ *can be computed via* $\left\langle \mathcal{G}, \mathit{Parity}(\widehat{\mathcal{P}}) \right\rangle$ *as well. To obtain an* over-*approximation, we solve this abstract game* cooperatively*. That is, we let player Player* 0 *choose both its own moves and the moves of Player* 1 *to win almost surely with respect to Parity*$(\widehat{\mathcal{P}})$*. We use this over-approximation to check the quality of the under-approximation in Sec. 5.6.*

5.4. Computation of the Abstract Transition Functions

We have outlined the ABCD method for synthesis of controllers for CMPs with ω-regular specifications. The introduced abstraction uses the abstract transition functions $\overline{F}$ and $\underline{F}$ (defined in (5.6)), which we so far assumed to be magically known. In the following, we present an algorithm for computing $\overline{F}$ and $\underline{F}$ for the special class of nonlinear stochastic dynamical systems with affine stochastic noise.

The stochastic dynamical system. A stochastic dynamical system is expressed using a tuple $\Delta = \langle \mathcal{X}, \mathcal{U}, f, t_w \rangle$, where $\mathcal{X} \subset \mathbb{R}^n$ is the state space, $\mathcal{U}$ is a finite input space, $f \colon \mathcal{X} \times \mathcal{U} \to \mathcal{X}$ is the nominal dynamics, and $t_w \colon \mathbb{R}^n \to \mathbb{R}_{\geq 0}$ is the density function of the stochastic noise. The state evolution of Δ is described as

$$x^{k+1} = f(x^k, u^k) + \varsigma^k, \quad k \in \mathbb{N}, \tag{5.7}$$

where $x^k \in \mathcal{X}$ and $u^k \in \mathcal{U}$ are states and inputs for each $k \in \mathbb{N}$, and $(\varsigma^0, \varsigma^1, \ldots)$ is assumed to be a sequence of independent and identically distributed (i.i.d.) random variables with density function t_w representing a stochastic disturbance.

One can construct a CMP $\mathcal{S} = \langle \mathcal{X}, \mathcal{U}, T_{\mathfrak{s}} \rangle$ over states $\mathcal{X}$ and inputs $\mathcal{U}$ from (5.7) by noticing that for any given state x^k and input u^k at time k, the next state is a random variable defined as a function of ς^k. In particular, the stochastic transition kernel $T_{\mathfrak{s}}(\cdot | x^k, u^k)$ can be computed as $T_{\mathfrak{s}}(A \mid x, u) = \int_A t_w(x' - f(x, u))dx'$ for all $A \in \mathcal{B}(\mathcal{X})$ (Kallenberg, 2002).

For the construction of the abstraction we assume that $t_w(\cdot)$ is piecewise continuous and has a bounded support $D \subset \mathbb{R}^n$ (the set of points in $\mathbb{R}^n$ on which t_w is strictly positive), and $f(\cdot, u)$ is continuous for all $u \in \mathcal{U}$.

The abstraction. We assume that $\mathcal{X} = \mathcal{X}' \cup \{\phi\}$, where $\mathcal{X}'$ is a compact hyper-rectangular working region of the system and ϕ is a sink state representing the complement of $\mathcal{X}'$. The disturbance has a compact support $D \subset \mathbb{R}^n$. Let $\widehat{\mathcal{X}}'$ be a hyper-rectangular partition of $\mathcal{X}'$. The overall abstract state space is $\widehat{\mathcal{X}} = \widehat{\mathcal{X}}' \cup \{\phi\}$. Given an abstract state $\widehat{x} = [\![a, b]\!] \in \widehat{\mathcal{X}}'$ and a control input $u \in \mathcal{U}$, we denote the approximate *nominal* reachable set of $\mathcal{S}$ by $\Phi(\widehat{x}, u)$ such that

$$\Phi(\widehat{x}, u) \supseteq \bigcup_{s \in \mathsf{cl}(\widehat{x})} f(s, u), \tag{5.8}$$

where $\mathsf{cl}(\widehat{x})$ is the closure of the set $\widehat{x}$. Note that $\Phi(\widehat{x}, u)$ can be computed using any reachability analysis method for deterministic dynamical systems (Coogan and Arcak,

2015; Dang and Testylier, 2012); further details on this follow in Sec. 5.4.1. Define the functions $S_1, S_2 : \widehat{\mathcal{X}} \times \mathcal{U} \to 2^{\mathbb{R}^n}$ such that

$$S_1 : (\widehat{x}, u) \mapsto \overline{D} \oplus \Phi(\widehat{x}, u) \quad \text{and} \tag{5.9}$$

$$S_2 : (\widehat{x}, u) \mapsto \underline{D} \ominus (-\Phi(\widehat{x}, u)), \tag{5.10}$$

where $\overline{D} \supseteq D$ is any over-approximation of the support of disturbance D and $\underline{D} \subseteq D$ is any compact under-approximation of D. The operators $\oplus$ and $\ominus$ are Minkowski sum and Minkowski difference of two sets, respectively, and the minus sign in $(-\Phi(\widehat{x}, u))$ is applied to all elements.

Theorem 5.4 *Let $\Delta = \langle \mathcal{X}, \mathcal{U}, f, t_w \rangle$ be a dynamical system and $\mathcal{S} = \langle \mathcal{S}, \mathcal{U}, T_{\mathfrak{s}} \rangle$ be the CMP induced by $\mathcal{S}$. Define $\overline{F}$ and $\underline{F}$ as below:*

$$\begin{aligned}\overline{F}(\widehat{x}, u) := \{\widehat{x}' \in \widehat{\mathcal{X}} \mid (\widehat{x}' \neq \phi) \Rightarrow (\widehat{x}' \cap S_1(\widehat{x}, u) \neq \emptyset) \wedge \\ (\widehat{x}' = \phi) \Rightarrow (S_1(\widehat{x}, u) \not\subseteq \mathcal{X}')\},\end{aligned} \tag{5.11}$$

$$\begin{aligned}\underline{F}(\widehat{x}, u) := \{\widehat{x}' \in \widehat{\mathcal{X}} \mid (\widehat{x}' \neq \phi) \Rightarrow (\lambda(\widehat{x}' \cap S_2(\widehat{x}, u)) > 0) \wedge \\ (\widehat{x}' = \phi) \Rightarrow \lambda(S_2(\widehat{x}, u) \backslash \mathcal{S}') > 0\},\end{aligned} \tag{5.12}$$

where $\lambda(\cdot)$ gives the Lebesgue measure (volume) of a set. Then $\overline{F}$ and $\underline{F}$ satisfy the conditions in (5.6)*, which are restated here for convenience:*

$$\begin{aligned}\mathbf{A}: \quad & \overline{F}(\widehat{x}, u) \supseteq \{\widehat{x}' \in \widehat{\mathcal{X}} \mid \exists x \in \widehat{x} \,.\, T_{\mathfrak{s}}(\widehat{x}' \mid x, u) > 0\}, \\ \mathbf{B}: \quad & \underline{F}(\widehat{x}, u) \subseteq \{\widehat{x}' \in \widehat{\mathcal{X}} \mid \exists \varepsilon > 0 \,.\, \forall x \in \widehat{x} \,.\, T_{\mathfrak{s}}(\widehat{x}' \mid x, u) \geq \varepsilon\}.\end{aligned}$$

Proof For inequality **A**, consider any pair of abstract states $\widehat{x}, \widehat{x}' \in \widehat{\mathcal{X}}'$ and input $u \in U$ and there exists $x \in \widehat{x}$, $T_{\mathfrak{s}}(\widehat{x}' \mid x, u) > 0$. We show that $\widehat{x}' \in \overline{F}(\widehat{x}, u)$:

$$\begin{aligned}&\int_{\widehat{x}'} t_w(x' - f(x, u))dx' > 0 \Rightarrow \int_{\widehat{x}' \ominus \{f(x,u)\}} t_w(w)dw > 0 \\ &\Rightarrow (\widehat{x}' \ominus \{f(x, u)\}) \cap D \neq \emptyset \\ &\Rightarrow \exists w \in D, \exists x' \in \widehat{x}' \text{ such that } \quad w = x' - f(x, u) \\ &\Rightarrow \exists w \in D, \exists x' \in \widehat{x}' \text{ such that } \quad x' = f(x, u) + w.\end{aligned}$$

At the same time we know that $f(x, u) \in \Phi(\widehat{x}, u)$ since $x \in \widehat{x}$. Then,

$$x' \in S_1(\widehat{x}, u) \Rightarrow \widehat{x}' \cap S_1(\widehat{x}, u) \neq \emptyset \Rightarrow \widehat{x}' \in \overline{F}(\widehat{x}, u).$$

A similar reasoning holds for the case of $\widehat{x}' = \phi$.

For inequality **B**, take $\widehat{x} \in \widehat{\mathcal{X}}'$, input $u \in U$, and $\widehat{x}' \in \underline{F}(\widehat{x}, u)$ such that $\widehat{x}' \neq \emptyset$. Then $\lambda(\widehat{x}' \cap S_2(\widehat{x}, u)) > 0$ according to (5.12). For any $x' \in \widehat{x}' \cap S_2(\widehat{x}, u)$, we have

$$\begin{aligned}&x' \in S_2(\widehat{x}, u) \Rightarrow \{x'\} \oplus (-\Phi(\widehat{x}, u)) \subseteq \underline{D} \\ &\Rightarrow x' - f(x, u) \subseteq \underline{D} \quad \forall x \in \mathsf{cl}(\widehat{x}) \\ &\Rightarrow T_{\mathfrak{s}}(\widehat{x}' \mid x, u) \geq \int_{\widehat{x}' \cap S_2(\widehat{x}, u)} t_w(x' - f(x, u))dx' > 0\end{aligned}$$ □

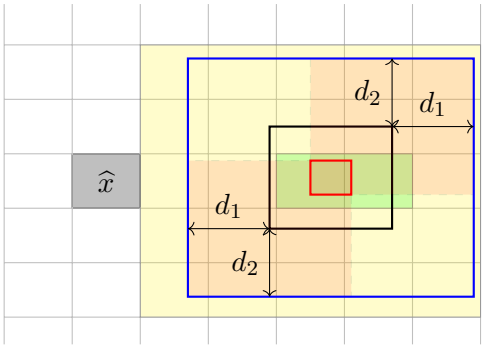


Figure 5.2.: Illustration of abstraction computation: given the abstract state $\widehat{x}$ (filled with grey) and some control input u, first the nominal reachable set is over-approximated (black rectangle). Next, the sets S_1 (blue rectangle) and S_2 (red rectangle) are computed. Finally, the images of the transition functions $\overline{F}$ (filled with yellow) and $\underline{F}$ (filled with green) are the abstract states intersecting with S_1 and S_2 respectively.

The right-hand side is strictly positive since the integrand is strictly positive and the domain of integration has a positive measure. It is also assumed that f is continuous and t_w piecewise continuous. Therefore, we have a positive function over the compact domain $\mathsf{cl}(\widehat{x})$, which will have a positive minimum:

$$\exists \varepsilon > 0 \,.\, \forall x \in \mathsf{cl}(\widehat{x}) \,.\, T_{\mathfrak{s}}(\widehat{x}' \mid x, u) \geq \varepsilon \;\; \Rightarrow \;\; \widehat{x}' \in \underline{F}(\widehat{x}, u).$$

The abstraction procedure can be summarized as follows: first compute the approximate nominal reachable set $\Phi(\widehat{x}, u)$ in (5.8), then take the Minkowski sum and difference for S_1, S_2 in (5.9)-(5.10), and finally compute the transition relations (5.12)-(5.11). Fig. 5.2 illustrates the abstraction procedure for a 2-d system and when D is of the form $[-d_1, d_1] \times [-d_2, d_2]$.

5.4.1. Computation for Mixed-Monotone Systems

If the function $f(\cdot, u)$ is mixed-monotone for every $u \in \mathcal{U}$ and the partition sets are hyper-rectangles, then the nominal reachable set $\Phi(\widehat{x}, u)$ can be computed particularly efficiently. We recall the definition of mixed-monotonicity (Coogan and Arcak, 2015).

Definition 5.6 Let $g : \mathcal{X} \to \mathcal{X}$ be a function, and $\leq_{\mathcal{X}}$ be an order relation on $\mathcal{X}$ induced by positive cones. The function g is called mixed-monotone with respect to $\leq_{\mathcal{X}}$ (or simply mixed-monotone if $\leq_{\mathcal{X}}$ is obvious from the context) if there exists a function $h : \mathcal{X} \times \mathcal{X} \to \mathcal{X}$—called the decomposition function—with the following properties:

1. $\forall x \in \mathcal{X} \,.\, h(x, x) = g(x)$,
2. $\forall x_1, x_2, y \in \mathcal{X} \,.\, (x_1 \leq_{\mathcal{X}} x_2) \Rightarrow (h(x_1, y) \leq_{\mathcal{X}} h(x_2, y))$, and

3. $\forall x, y_1, y_2 \in \mathcal{X} \,.\, (y_1 \leq_{\mathcal{X}} y_2) \Rightarrow (h(x, y_2) \leq_{\mathcal{X}} h(x, y_1))$.

Intuitively, a mixed-monotone function can be decomposed into an increasing and a decreasing component. This phenomenon can be seen from the definition of the decomposition function. The following proposition (Coogan and Arcak, 2015, Thm. 1) shows a fast over-approximation method of the image of a rectangular set under a mixed-monotone function.

Proposition 5.1 *Let g be a mixed-monotone function with the decomposition function h, and $[\![a, b]\!] \subseteq \mathcal{X}$ be any hyper-rectangle. The image of $[\![a, b]\!]$ under g can be over-approximated as $[\![h(a, b), h(b, a)]\!]$.*

For mixed-monotone $f(\cdot, u)$ with decomposition function h_u, the function $\Phi(\widehat{x}, u)$ can be computed using Prop. 5.1 as $\Phi(\widehat{x}, u) = [\![h_u(a, b), h_u(b, a)]\!]$ for $\widehat{x} = [\![a, b]\!]$.

5.5. Proof of Theorem 5.3

Proof outline. To prove Thm. 5.3, we first decompose both the original and the abstract parity specifications $\mathit{Parity}(\mathcal{P})$ and $\mathit{Parity}(\widehat{\mathcal{P}})$ into a combination of more manageable safety and reachability sub-parts. That is, for every state reachable by a finite play in $\mathcal{S}$ and for every odd priority i, we consider a local safety specification ψ_S and a local reachability specification ψ_R defined by

$$\psi_S := \Box\neg\left(\bigcup_{j\in_{odd}[i,\ell]} B_j\right) \qquad \text{and} \qquad \psi_R := \Diamond\left(\psi_S \vee \bigvee\nolimits_{j\in_{even}[i+1;\ell]} B_j\right). \tag{5.13}$$

Intuitively, ψ_R requires that every time an odd priority—say B_i—is visited in $\mathcal{S}$, eventually either B_i and higher odd priorities should never occur or an even priority B_j with $j > i$ should occur, almost surely. Similarly, for the abstract $2\frac{1}{2}$-player parity game $\left\langle \mathcal{G}, \widehat{\mathcal{P}} \right\rangle$ we consider the local safety winning condition $\widehat{\psi}_S$ and a local reachability winning condition $\widehat{\psi}_R$ defined by

$$\widehat{\psi}_S := \Box\neg\left(\bigcup_{j\in_{odd}[i,\ell]} \widehat{B}_j\right) \qquad \text{and} \qquad \widehat{\psi}_R := \Diamond\left(\widehat{\psi}_S \vee \bigvee\nolimits_{j\in_{even}[i+1;\ell]} \widehat{B}_j\right). \tag{5.14}$$

While the above decomposition needs to be established both for $\mathcal{G}$ and for $\mathcal{S}$, the directions of the respective proof differ. For $\mathcal{S}$ we show that if ψ_R holds for a state reachable by a finite path over $\mathcal{S}$, then the original specification $\mathit{Parity}(\mathcal{P})$ is satisfied by a continuation of the path using the refined controller $\mathcal{C}$ (**Step 1**). For $\mathcal{G}$ we show that if $\mathit{Parity}(\widehat{\mathcal{P}})$ is satisfied, then $\widehat{\psi}_R$ holds for every state visited by a play compatible with the almost sure winning strategy π_0 in $\left\langle \mathcal{G}, \widehat{\mathcal{P}} \right\rangle$ (**Step 2**). Further, we show that satisfaction of $\widehat{\psi}_S$ (resp. $\widehat{\psi}_R$) in $\mathcal{G}$ implies satisfaction of ψ_S (resp. ψ_R) in $\mathcal{S}$ (**Step 3-5**). With this, we have all ingredients to prove Thm. 5.3 (**Step 6**).

Step 1: Decomposition of *Parity*($\mathcal{P}$). We prove a *sufficient* condition for satisfaction of *Parity*($\mathcal{P}$) in $\mathcal{S}$ if ψ_R holds.

Lemma 5.1 *Let $\mathcal{S}$ be a CMP, Parity($\mathcal{P}$) be a parity specification, $x^0 \in \mathcal{X}$ be a given initial state, and $\mathcal{C}$ be a controller. Suppose the following holds for every finite path $(x^0, \dots, x^n) \in \mathcal{X}^{n+1}$ of $\mathcal{S}$ and every $i \in_{odd} [1;\ell]$:*

$$x^n \in B_i \Rightarrow P^{\mathcal{C}}_{x^n}(\mathcal{S} \models \psi_R) = 1. \tag{5.15}$$

Then $P^{\mathcal{C}}_{x^0}(\mathcal{S} \models Parity(\mathcal{P})) = 1$.

Proof (Proof of Lem. 5.1) Define for any arbitrary $i \in_{odd} [1;\ell]$ the event $E_i \coloneqq (\mathcal{S} \models \psi_i)$ with the specification $\psi_i \coloneqq \left(\bigvee_{j\in_{odd}[i;\ell]} \square\lozenge B_j \rightarrow \bigvee_{j\in_{even}[i+1;\ell]} \square\lozenge B_j\right)$. We want to show that $P^{\mathcal{C}}_{x^0}(\mathcal{S} \models Parity(\mathcal{P})) = P^{\mathcal{C}}_{x^0}\left(\bigcap_i E_i\right) = 1$. We prove this by showing $P^{\mathcal{C}}_{x^0}(\overline{E_i}) = 0$ for every $i \in_{odd} [1;\ell]$. Once we show this, the result follows according to the standard inequalities:

$$P^{\mathcal{C}}_{x^0}\left(\bigcap_i E_i\right) = 1 - P^{\mathcal{C}}_{x^0}\left(\bigcup_i \overline{E_i}\right) \geq 1 - \sum_i P^{\mathcal{C}}_{x^0}\left(\overline{E_i}\right) = 1$$
$$\text{where } P^{\mathcal{C}}_{x^0}(\overline{E_i}) = P^{\mathcal{C}}_{x^0}\left((\mathcal{S} \models \vee_j \square\lozenge B_j) \cap (\mathcal{S} \models \wedge_k \lozenge\square\neg B_k)\right)$$

with $i \in_{odd} [1;\ell]$, $j \in_{odd} [i;\ell]$, and $k \in_{even} [i+1;\ell]$. Define the random variable τ to be the largest time instance when the trajectory visits one of the sets B_k. Also define $\tau' > \tau$ to be the first time instance after τ when the trajectory visits one of the sets B_j again. Note that for any trajectory satisfying $\vee_j \square\lozenge B_j$ and $\wedge_k \lozenge\square\neg B_k$, both τ and τ' are well-defined and bounded. According to the assumption (5.15), we have

$$P^{\mathcal{C}}_{x^0}(\overline{E_i} \,|\, \tau' = n, x^0, x^1, \cdots, x^n)$$
$$= P^{\mathcal{C}}_{x^n}\left((\mathcal{S} \models \vee_j \square\lozenge B_j) \cap (\mathcal{S} \models \wedge_k \lozenge\square\neg B_j)\right) = 0.$$

By taking the expectation with respect to the condition $(\tau', x^0, \cdots, x^n)$, we conclude that $P^{\mathcal{C}}_{x^0}(\overline{E_i}) = \mathbb{E}^{\mathcal{C}}_{x^0}\left[P^{\mathcal{C}}_{x^0}(\overline{E_i} \,|\, \tau' = n, x^0, x^1, \cdots, x^n)\right] = \mathbb{E}^{\mathcal{C}}_{x^0}[0] = 0$, i.e., $\overline{E_i}$ has a zero probability. □

Step 2: Decomposition of *Parity*($\widehat{\mathcal{P}}$). We present a *necessary* condition for satisfaction of *Parity*($\widehat{\mathcal{P}}$) in $\mathcal{G}$ if $\widehat{\psi}_R$ holds.

Lemma 5.2 *Let $\left\langle \mathcal{G}, \widehat{\mathcal{P}} \right\rangle$ be a 2½-player parity game, and v^0 be a given vertex of $\mathcal{G}$. Suppose $\pi_0^* \in \Pi_0^{\mathrm{DM}}$ is a Player 0 strategy such that $\inf_{\pi_1\in\Pi_1} P^{\pi_0^*,\pi_1}_{v^0}(\mathcal{G} \models Parity(\widehat{\mathcal{P}})) = 1$. Then given every finite play $v^0 \dots v^n \in V^*$ such that there exists a Player 1 strategy $\pi_1 \in \Pi_1$ with $P^{\pi_0^*,\pi_1}_{v^0}(\mathcal{G} \models v^0 \dots v^n) > 0$, the following holds for every $i \in_{odd} [1;\ell]$:*

$$v^n \in \widehat{B}_i \Rightarrow \inf_{\pi_1\in\Pi_1} P^{\pi_0^*,\pi_1}_{v^n}\left(\mathcal{G} \models \lozenge\widehat{\psi}_R\right) = 1. \tag{5.16}$$

The only new factor in Eq. (5.16) is the presence of the adversarial effect of the *Player* 1 strategies.

PROOF It follows from the definition of the parity winning condition in (2.2) that a vertex $v^0 \in V$ is almost sure winning using the strategy π_0^* if the following condition is fulfilled:

$$\inf_{\pi_1 \in \Pi_1} P_{v^0}^{\pi_0^*, \pi_1} \left(\mathcal{G} \models \bigwedge_{i \in_{odd} [1;\ell]} \Box \left(\widehat{B}_i \Rightarrow \Diamond \widehat{\psi}_R \right) \right) = 1. \tag{5.17}$$

From the semantics of LTL, (5.17) implies:

$$\inf_{\pi_1 \in \Pi_1} P_{v^0}^{\pi_0^*, \pi_1} \left(\mathcal{G} \models \bigwedge_i \forall m \in \mathbb{N} \,.\, (v^m \in \widehat{B}_i) \Rightarrow \Diamond \widehat{\psi}_R \right) = 1, \tag{5.18}$$

with $i \in_{odd} [1;\ell]$. We show that (5.18) implies for every finite play $v^0 \dots v^n \in V^*$, occurring with a positive probability $p_1 > 0$ for some strategy of *Player* 1, (5.16) holds. Suppose, for contradiction's sake, there exists some $i \in_{odd} [1;\ell]$ such that $v^n \in \widehat{B}_i$ and (5.16) does not hold, i.e., $\inf_{\pi_1 \in \Pi_1} P_{v^n}^{\pi_0^*, \pi_1} \left(\mathcal{G} \models \Diamond \widehat{\psi}_R \right) < 1$, implying existence of some $0 < p_2 \leq 1$ with $\sup_{\pi_1 \in \Pi_1} P_{v^n}^{\pi_0^*, \pi_1} \left(\mathcal{G} \not\models \Diamond \widehat{\psi}_R \right) = p_2$. This results in satisfaction of the parity winning condition with a probability of *at most* $(1 - p_1 \cdot p_2) < 1$, contradicting (5.18). □

Step 3: Refinement of $\widehat{\psi}_S$ to ψ_S. Let $U \subseteq \mathcal{X}$ be a given set of states of $\mathcal{S}$, and $\underline{U} \subseteq \widehat{\mathcal{X}}$ be the under-approximation of U in the abstract state space $\widehat{\mathcal{X}}$, defined as:

$$\underline{U} \coloneqq \{ \widehat{x} \in \widehat{\mathcal{X}} \mid \widehat{x} \subseteq U \}. \tag{5.19}$$

We show that almost sure safety with respect to a given set $\underline{U}$ in $\mathcal{G}$ implies the same with respect to the set U in $\mathcal{S}$; this will later be used to infer $\widehat{\psi}_S \Rightarrow \psi_S$.

Proposition 5.2 *Let $\mathcal{S}$ be a CMP and $\mathcal{G}$ be a finite $2^1/_2$-player game graph as defined in Def. 5.4. Suppose $U \subseteq \mathcal{X}$ is a given set of states of $\mathcal{S}$, $\underline{U} \subseteq V_0$ is the under-approximation of U using the set of Player 0 vertices of $\mathcal{G}$, and assume that there is a Player 0 vertex $v \in \underline{U}$ for which there is a strategy $\pi_0 \in \Pi_0^{\mathrm{DM}}$ of Player 0 such that $\inf_{\pi_1 \in \Pi_1} P_v^{\pi_0, \pi_1}(\mathcal{G} \models \Box \underline{U}) = 1$. Then the refinement $\mathcal{C}$ of π_0 ensures that for every state $x \in v$, $P_x^{\mathcal{C}}(\mathcal{S} \models \Box U) = 1$.*

PROOF It is known that for safety properties, almost sure satisfaction coincides with sure satisfaction, i.e., $\inf_{\pi_1 \in \Pi_1} P_v^{\pi_0, \pi_1}(\mathcal{G} \models \Box \underline{U}) = 1$ if and only if for every strategy $\pi_1 \in \Pi_1$, every infinite play of $\mathcal{G}$ stays inside U at all time (de Alfaro and Henzinger, 2000). In other words, there must be a controlled invariant set W inside $\underline{U}$ for the strategy π_0, and $v \in W$. This controlled invariant set can be obtained by considering the 2-player game, obtained from $\mathcal{G}$ by removing all the random vertices, and redirecting the outgoing transitions of a given *Player* 1 vertex $v' \in V_1$ to the *Player* 0 vertices within the set $\overline{F}(v', \pi_0(v')) \subseteq V_0$. Since $\overline{F}(v', \pi_0(v'))$ over-approximates the set of all the continuous states reachable from v' using the input $\pi_0(v')$, hence if *Player* 0 can fulfill $\Box \underline{U}$ using the strategy $\pi_0(v')$, then $\mathcal{C}$ can fulfill $\Box Q^{-1}(\underline{U})$ from every state $x \in v'$ in $\mathcal{S}$. (This follows from the standard arguments in abstraction-based control using over-approximation based abstractions (Reissig et al., 2017).) Since $\underline{U}$ is an under-approximation of U, hence $Q^{-1}(\underline{U}) \subseteq U$, which implies that *Player* 0 can also fulfill $\Box U$ using the refined controller $\mathcal{C}$. □

Step 4: Refinement for $\widehat{\psi}_R$ to ψ_R. Let $U \subseteq \mathcal{X}$ be a set of states of $\mathcal{S}$, and $\underline{U} \subseteq \widehat{\mathcal{X}}$ be the under-approximation of U using the abstract state space $\widehat{\mathcal{X}}$, as per the definition in (5.19). We show that almost sure reachability with respect to the *Player* 0 vertices $\underline{U}$ in $\mathcal{G}$ implies the same with respect to the set U in $\mathcal{S}$; this will be used to infer $\widehat{\psi}_R \Rightarrow \psi_R$.

Suppose $\pi_0 \in \Pi_0^{\mathrm{DM}}$ is some strategy of *Player* 0 in the game $\langle \mathcal{G}, \Diamond \underline{U} \rangle$. We introduce a ranking function $r \colon V_0 \to \mathbb{N} \cup \{\infty\}$ as a certificate for almost sure satisfaction of the winning condition $\Diamond \underline{U}$. The ranking function r is defined inductively as follows:

$$r(v) = \begin{cases} 0 & v \in \underline{U}, \\ \infty & \inf_{\pi_1 \in \Pi_1} P_v^{\pi_0,\pi_1}(\mathcal{G} \models \Diamond \underline{U}) < 1, \\ i+1 & \min\{n \in \mathbb{N} \mid \inf_{\pi_1 \in \Pi_1} P_v^{\pi_0,\pi_1}(\mathcal{G} \models \bigcirc r^{-1}(n)) > 0\} = i \\ & \qquad \wedge \inf_{\pi_1 \in \Pi_1} P_v^{\pi_0,\pi_1}(\mathcal{G} \not\models \bigcirc r^{-1}(\infty)) = 1. \end{cases} \tag{5.20}$$

Note that every vertex $v \in V$ gets a rank: If $r(v) \neq \infty$, then $\inf_{\pi_1 \in \Pi_1} P_v^{\pi_0,\pi_1}(\mathcal{G} \models \Diamond \underline{U}) = 1$ by definition of r. In this case, there must exist some path to $\underline{U}$, i.e., $\inf_{\pi_1 \in \Pi_1} P_v^{\pi_0,\pi_1}(\mathcal{G} \not\models \bigcirc r^{-1}(\infty)) = 1$ must be true, and moreover $\inf_{\pi_1 \in \Pi_1} P_v^{\pi_0,\pi_1}(\mathcal{G} \models \bigcirc r^{-1}(n)) > 0$ will be true for some n. Thus, $r(v) = n + 1$.

From the ranking function $r(\cdot)$ defined in (5.20), it is clear that $\inf_{\pi_1 \in \Pi_1} P_v^{\pi_0,\pi_1}(\mathcal{G} \models \Diamond \underline{U}) = 1$ implies $r(v) \neq \infty$. We first identify some local structural properties of the abstract transition functions $\overline{F}$ and $\underline{F}$ evaluated on some abstract states with finite ranking.

Lemma 5.3 *Suppose $\pi_0 \in \Pi_0^{\mathrm{DM}}$ is some strategy of Player 0. For every $v \in V_0$ with $r(v) = i \neq \infty$, $i > 0$, both $\overline{F}(v, \pi_0(v)) \cap r^{-1}(\infty) = \emptyset$ and either of the following holds:*

1. *$\underline{F}(v, \pi_0(v)) \cap r^{-1}(i-1) \neq \emptyset$, or*
2. *$\underline{F}(v, \pi_0(v)) = \emptyset$ and $\overline{F}(v, \pi_0(v)) \subseteq r^{-1}(i-1)$.*

Proof Firstly, $\overline{F}(v, \pi_0(v)) \cap r^{-1}(\infty) = \emptyset$ should always hold as otherwise *Player* 1 would have a strategy to reach a state in $r^{-1}(\infty)$ with nonzero probability in the next step.

Suppose (2) does not hold, implying either (a) $\underline{F}(v, \pi_0(v)) \neq \emptyset$, or (b) the existence of a vertex $v' \in \overline{F}(v, \pi_0(v))$ with $r(v') \neq i - 1$. Then $\underline{F}(v, \pi_0(v)) \cap r^{-1}(i-1) \neq \emptyset$ must hold, as otherwise, for case (a) and (b) *Player* 1 would have strategies π_1 with $\pi_1(v, \pi_0(v)) = (\underline{F}(v, \pi_0(v)))$ and $\pi_1(v, \pi_0(v)) = (\underline{F}(v, \pi_0(v)) \cup \{v'\})$ respectively, such that $P_v^{\pi_0,\pi_1}(\mathcal{G} \models \bigcirc r^{-1}(i-1)) = 0$.

On the other hand, suppose (1) does not hold. Then $\underline{F}(v, \pi_0(v)) = \emptyset$ must be true, as otherwise *Player* 1 would have a strategy π_1 with $\pi_1(v, \pi_0(v)) = (\underline{F}(v, \pi_0(v)))$ such that $P_v^{\pi_0,\pi_1}(\mathcal{G} \models \bigcirc r^{-1}(i-1)) = 0$. Moreover, $\overline{F}(v, \pi_0(v)) \subseteq r^{-1}(i-1)$ must also be true, as otherwise there would exist a vertex $v' \in \overline{F}(v, \pi_0(v))$ with $r(v') \neq i - 1$, and *Player* 1 would have a strategy π_1 with $\pi_1(v, \pi_0(v)) = (\underline{F}(v, \pi_0(v)) \cup \{v'\}) = (\{v'\})$ such that $P_v^{\pi_0,\pi_1}(\mathcal{G} \models \bigcirc r^{-1}(i-1)) = 0$. □

The following lemma establishes soundness of the reduction with respect to reachability specifications.

Proposition 5.3 *Let $\mathcal{S}$ be a CMP and $\mathcal{G}$ be a finite $2\frac{1}{2}$-player game graph as defined in Def. 5.4. Suppose there is a Player 0 vertex $v \in V_0$ in $\mathcal{G}$ and a set of vertices $\underline{U} \subseteq V_0$, which is a an under-approximation of the set of states $U \subseteq \mathcal{X}$ of $\mathcal{S}$. If $\pi_0 \in \Pi_0^{\mathrm{DM}}$ is a Player 0 strategy with $\inf_{\pi_1 \in \Pi_1} P_v^{\pi_0,\pi_1}(\mathcal{G} \models \Diamond \underline{U}) = 1$, then the refinement $\mathcal{C} \in \Pi$ of π_0 ensures that for every state $x \in v$, $P_x^{\mathcal{C}}(\mathcal{S} \models \Diamond U) = 1$.*

Proof It follows from the definition of the ranking function in (5.20) that the set of almost sure winning vertices for the winning condition $\Diamond U$ is given by all the vertices with finite rank. We show that for every vertex v with a finite rank, the refinement $\mathcal{C} \in \Pi$ of π_0 ensures that from every state $x \in v$, $P_x^{\mathcal{C}}(\mathcal{S} \models \Diamond U) = 1$.

First, trajectories starting from any state $x \in v$ with $r(v) \neq \infty$ never go to the region $Q^{-1}(r^{-1}(\infty))$. This follows from the identity $\overline{F}(v, \pi_0(v)) \cap r^{-1}(\infty) = \emptyset$ in Lem. 5.3 and because $\overline{F}(v, \pi_0(v))$ is an overapproximation of the one step reachable set from the states within vertex v. Hence, every infinite trajectory of $\mathcal{S}$ starting at x will visit the states in $\mathcal{X} \setminus Q^{-1}(r^{-1}(\infty))$ infinitely often.

The rest of the proof shows that if a trajectory visits the states $\mathcal{X} \setminus Q^{-1}(r^{-1}(\infty) \cup r^{-1}(0))$ infinitely often, then the trajectory will almost surely satisfy $\Diamond Q^{-1}(r^{-1}(0)) = \Diamond Q^{-1}(\underline{U})$, implying that it will almost surely satisfy $\Diamond U$ as well (since $Q^{-1}(\underline{U}) \subseteq U$). The proof is by induction over the largest rank assigned by r. For the base case, let the largest rank assigned by r be 2. We show that every state $x \in \mathcal{X}$ starting from inside a vertex v with $r(v) = 1$ or $r(v) = 2$ will almost surely reach $Q^{-1}(\underline{U})$, i.e., $P_x^{\mathcal{C}}(\mathcal{S} \models \Box Q^{-1}(r^{-1}(1) \cup r^{-1}(2))) = 0$. Note that the events $\{\Diamond \Box Q^{-1}(r^{-1}(2))\}$ and $\{\Box \Diamond Q^{-1}(r^{-1}(1))\}$ form a partition of the event of $\{\Box Q^{-1}(r^{-1}(1) \cup r^{-1}(2))\}$. Therefore,

$$\begin{aligned} &P_x^{\mathcal{C}}(\mathcal{S} \models \Box Q^{-1}(r^{-1}(1) \cup r^{-1}(2))) \\ &\qquad = P_x^{\mathcal{C}}(\mathcal{S} \models \Box Q^{-1}(r^{-1}(1) \cup r^{-1}(2)) \wedge \Diamond \Box Q^{-1}(r^{-1}(2))) \\ &\qquad + P_x^{\mathcal{C}}(\mathcal{S} \models \Box Q^{-1}(r^{-1}(1) \cup r^{-1}(2)) \wedge \Box \Diamond Q^{-1}(r^{-1}(1))). \end{aligned}$$

The first term is upper bounded by $P_x^{\mathcal{C}}(\mathcal{S} \models \Diamond \Box Q^{-1}(r^{-1}(2)))$ which is zero, because $P_x^{\mathcal{C}}(\mathcal{S} \models \Box Q^{-1}(r^{-1}(2))) = \prod_{n=1}^{\infty}(1-\varepsilon)^n = 0$, where ε is the lower bound probability of transitions used in the definition of $\underline{F}$. The second term is also zero because the event requires the number of transitions from $Q^{-1}(r^{-1}(1))$ to be infinite. To see this, let $\boldsymbol{i}_n = (i_0, i_1, \ldots, i_n)$ be the first $(n+1)$ time instances that a trajectory visits $Q^{-1}(r^{-1}(1))$. Then,

$$\begin{aligned} &P_x^{\mathcal{C}}(\mathcal{S} \models \Box Q^{-1}(r^{-1}(1) \cup r^{-1}(2)) \wedge \Box \Diamond Q^{-1}(r^{-1}(1))) = \\ &\sum_{\boldsymbol{i}_n} P_x^{\mathcal{C}}(\mathcal{S} \models \Box Q^{-1}(r^{-1}(1) \cup r^{-1}(2)) \wedge \Box \Diamond Q^{-1}(r^{-1}(1)) \mid \boldsymbol{i}_n) P_x^{\mathcal{C}}(\boldsymbol{i}_n) \\ &\leq \sum_{\boldsymbol{i}_n} (1-\varepsilon)^n P_x^{\mathcal{C}}(\boldsymbol{i}_n) = (1-\varepsilon)^n. \end{aligned}$$

The last inequality is due to either Cond. (1) or Cond. (2) of Lem. 5.3 applied to the vertices in $v \in r^{-1}(1)$. Note that this inequality holds for any n. By taking the limit

when n goes to infinity, we have that this second term is also zero. Hence the base case is established.

For the induction hypothesis, assume that the claim holds when the maximum rank assigned by the function r is i. Then for the induction step, i.e., when the maximum rank is $i+1$, we can follow same argument, as we did for the states with rank 2 in the base case, to show that every infinite trajectory inside $\mathcal{X} \setminus Q^{-1}(r^{-1}(\infty) \cup r^{-1}(0))$ will never get trapped inside $Q^{-1}(r^{-1}(i+1))$, which will mean that the trajectory will visit the states in $\mathcal{X} \setminus Q^{-1}(r^{-1}(\infty) \cup r^{-1}(0) \cup r^{-1}(i+1))$ infinitely often. Then it follows from the induction hypothesis that the trajectories will reach $Q^{-1}(\underline{U})$ almost surely. □

Step 5: Refinement of runs. We show that every finite path in $\mathcal{S}$ can be mapped to a positive probability finite play in $\mathcal{G}$; this will be used to establish a bridge from the universal quantification over finite paths in $\mathcal{S}$ to the existential quantification over finite plays in $\mathcal{G}$.

Lemma 5.4 *Let $\mathcal{S}$ be a CMP, $\mathcal{G}$ be the abstract game graph as defined in Def. 5.4, $\pi_0 \in \Pi_0^{\mathrm{DM}}$ be an arbitrary Player 0 strategy in the game $\mathcal{G}$, and $x \in \mathcal{X}$ be a state of $\mathcal{S}$. Suppose $\mathcal{C} \in \Pi$ is the refinement of π_0. Then for every finite trajectory $x^0 \dots x^n \in \mathcal{X}^*$ of $\mathcal{S}$ in the support of the distribution $P^{\mathcal{C}}_{x^0}$, there exists a Player 1 strategy $\pi_1 \in \Pi_1$ such that $P^{\pi_0,\pi_1}_{\widehat{x}^0}(\mathcal{G} \models \widehat{x}^0 \dots \widehat{x}^n) > 0$, where $\widehat{x}^i = Q(x^i)$ for every $i \in [0;n]$.*

Proof The initial state $x^0 \in \widehat{x}^0$, and for every $0 \leq i < n$, from the definition of $\overline{F}$ it follows that $\widehat{x}^{i+1} \in \overline{F}(\widehat{x}^i, \pi_0(\widehat{x}^i))$. Thus, from every *Player* 1 vertex $(\widehat{x}^i, \pi_0(\widehat{x}^i))$, there is a successor vertex in V_r whose successor is $\widehat{x}^{i+1}$. Hence, for every $0 \leq i < n$, there is some move of *Player* 1 which causes a transition to $\widehat{x}^{i+1}$ with some positive probability p^i. Then $P^{\pi_0,\pi_1}_{\widehat{x}^0}(\mathcal{G} \models \widehat{x}^0 \dots \widehat{x}^n) = \prod_{i=0}^{n-1} p^i > 0$. □

Step 6: The final assembly of the proof. Finally, we finish the proof of Thm. 5.3 by stitching everything together. It is known that memoryless strategies suffice for winning almost surely in $2^1/2$-player parity games (Zielonka, 2004). Let $\pi_0^* \in \Pi_0^{\mathrm{DM}}$ be the witness strategy of *Player* 0 to almost surely win from the vertex $\widehat{x}^*$ in the game $\left\langle \mathcal{G}, \mathit{Parity}(\widehat{\mathcal{P}}) \right\rangle$, and $\mathcal{C}^*$ be the refinement of π_0^*. We claim that $\widehat{x}^* \subseteq \mathsf{WinDom}(\mathcal{S}, \mathcal{C}^*)$.

We will show that for *every* finite path of $\mathcal{S}$ starting within $\widehat{x}^*$ and ending in some odd priority state B_i, at one point either B_i and any higher odd priority states will not be visited any more, or a state of higher even priority will be visited eventually. Then the claim will follow from Lem. 5.1. We know from Lem. 5.4 that existence of a finite path $x^0 \dots x^n$ of $\mathcal{S}$ implies existence of an abstract play $\widehat{x}^0 \dots \widehat{x}^n$ such that $\sup_{\pi_1 \in \Pi_1} P^{\pi_0,\pi_1}_{\widehat{x}^0}(\mathcal{G} \models \widehat{x}^0 \dots \widehat{x}^n) > 0$. Moreover, by construction of $\widehat{\mathcal{P}}$, if $x^n \in B_i$ with an odd i then $\widehat{x}^n \in \widehat{B}_j$ with odd j and $j \geq i$. Since π_0^* is an almost sure winning strategy,

hence, by using Lem. 5.2, we know that the following holds:

$$\inf_{\pi_1 \in \Pi_1} P^{\pi_0^*, \pi_1}_{\widehat{x}^n} \left(\mathcal{G} \models \Diamond \left(\Box \neg \underbrace{\left(\bigcup_{j \in_{odd}[i;\ell]} \widehat{B}_j \right)}_{\mathbf{E}} \vee \overbrace{\cup_{j \in_{even}[i+1;\ell]} \widehat{B}_j}^{\mathbf{F}} \right) \right) = 1. \tag{5.21}$$

By construction of $\widehat{\mathcal{P}}$, the set of vertices $\mathbf{E}$ is an over-approximation of the set $\left(\bigcup_{j \in_{odd}[i;\ell]} B_j\right)$, and so the negation of $\mathbf{E}$ is an under-approximation of the latter. From Prop. 5.2, we can infer that the set of abstract states from which the winning condition $\Box\neg\mathbf{E}$ is satisfied (almost) surely using the strategy π_0^* is an under-approximation of the set of continuous states from which the winning condition $\Box\neg\left(\bigcup_{j \in_{odd}[i;\ell]} B_j\right)$ is satisfied almost surely using the controller $\mathcal{C}^*$. Furthermore, by construction of $\widehat{\mathcal{P}}$, the set of vertices $\mathbf{F}$ is an under-approximation of the set $\cup_{j \in_{even}[i+1;\ell]} B_j$. From all of these observations, together with Prop. 5.3 and Lem. 5.1, we can infer Thm. 5.3.

5.6. Numerical Examples

5.6.1. 2-Dimensional Bistable Switch

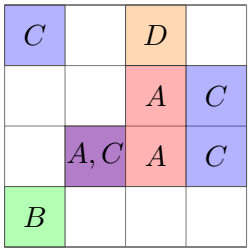
We consider the controller synthesis problems for a two-dimensional stochastic bistable switch with a couple of different parity specifications; the examples have been adopted, mutatis mutandis, from the work of Dutreix et al. (2020). The dynamics of the system is modeled by the following difference equations:

$$x_1^{k+1} = x_1^k + \left(-a \cdot x_1^k + x_2^k\right) \cdot \tau + u_1^k + \varsigma_1^k \tag{5.22}$$

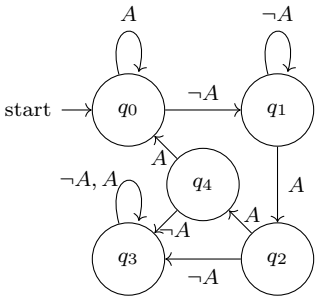
$$x_2^{k+1} = x_2^k + \left(\frac{\left(x_1^k\right)^2}{\left(x_1^k\right)^2 + 1} - b \cdot x_2^k\right) \cdot \tau + u_2^k + \varsigma_2^k, \tag{5.23}$$

where x_1, x_2 are the state variables, u_1, u_2 are the control inputs, a, b are constant parameters, τ is the sampling time, and ς_1, ς_2 are stochastic noises. We assume that the domain of the state variables is $[0.0, 4.0] \times [0.0, 4.0]$, and we saturate the state trajectories at the boundary of the domain. We consider a finite set of values for both of the control inputs: for every k, $u_1^k, u_2^k \in \{-0.05, 0.0, 0.05\}$. The values of the constants are given by: $a = 1.3$, $b = 0.25$, and $\tau = 0.05$. Finally, we assume that the stochastic noise samples $(\varsigma_1^k, \varsigma_2^k)$ are drawn from a piecewise continuous density function with the support $D = [-0.4, -0.2] \times [-0.4, -0.2]$.[1]

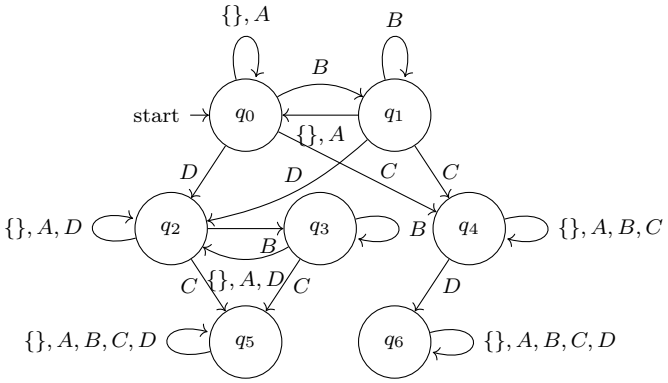
[1] Dutreix et al. (2020) considered the density function to be given by truncated Gaussian distribution with support D. In our work, we disregard the shape of the distribution because we restrict the focus to only the qualitative satisfaction of the specification.



(a) The state predicates.



(b) Parity automaton for φ_1; the prioritized partition is $\mathcal{P} = \langle \emptyset, \{q_2, q_3, q_4\}, \{q_0, q_1\} \rangle$. $\neg A$ stands for any element in $\{\{\}, B, C, D\}$.



(c) Parity automaton for φ_2; the prioritized partition is $\mathcal{P} = \langle \{q_0, q_2, q_4\}, \{q_1, q_3, q_5, q_6\} \rangle$.

Figure 5.3.: The state predicates and the parity specifications.

Let A, B, C, D be sets of states, as shown in Fig. 5.3a. We are interested in synthesizing the almost sure winning controllers for the above system for the following two LTL specifications:

$$\varphi_1 \coloneqq \square\left((\neg A \wedge \bigcirc A) \rightarrow (\bigcirc \bigcirc A \wedge \bigcirc \bigcirc \bigcirc A)\right),$$
$$\varphi_2 \coloneqq (\square\lozenge B \rightarrow \lozenge C) \wedge (\lozenge D \rightarrow \square\neg C).$$

The specifications φ_1 and φ_2 can be represented using the parity automata [1] shown in Fig. 5.3b and 5.3c. Firstly, for each of the two specifications, we compute a product with the system model. Secondly, we apply the algorithm from Chap. 7 on the product system to solve the synthesis problem. Both of these steps have been implemented on the open-source tool Mascot-SDS (Majumdar et al., 2020a). By symbolically encoding the abstract $2\frac{1}{2}$-player game using BDD-s, and by using sophisticated acceleration techniques for solving symbolic fixpoint algorithms from the literature (Long et al., 1994; Piterman and Pnueli, 2006), we achieve significant improvement in performance, in comparison

[1] Dutreix et al. (2020) modeled φ_1 and φ_2 using Rabin automata, and we transformed them into (language-) equivalent parity automata to match our setup.

with the implementation of the enumerative algorithm of Dutreix et al. (2020) in the tool called StochasticSynthesis.[1]

The tool StochasticSynthesis has a couple of additional features compared to our implementation in Mascot-SDS. Firstly, it performs an adaptive abstraction refinement procedure for achieving better computational efficiency over uniform abstractions. This is an orthogonal optimization tool that is known to be effective in discretization-based approaches for controller synthesis (Soudjani and Abate, 2013; Cámara et al., 2011a,b; Nilsson et al., 2017; Hsu et al., 2018b), and we expect that our uniform abstraction-based synthesis procedure will benefit further from this in the future. Secondly, StochasticSynthesis also addresses the quantitative aspect of the synthesis problem, which is to maximize the probability of satisfying the given specification. In all our experiments, we disabled this second feature of StochasticSynthesis, since the quantitative part is not our main algorithmic contribution.

We performed all the experiments on a Macbook Pro (2015) laptop equipped with a 2.7 GHz Intel Core i5 processor and a 16 GB RAM.

We performed two sets of experiments to compare the performance of Mascot-SDS and StochasticSynthesis, one with the adaptive refinement feature of StochasticSynthesis disabled and one with the same enabled. The results have been summarized in Tab. 5.1, Fig. 5.4, and Fig. 5.5. All the experiments empirically show that the approximation error reduces with finer discretization. We highlight the other main findings in the following: (A) When the abstraction-refinement feature of StochasticSynthesis was disabled, Mascot-SDS outperformed StochasticSynthesis by a large margin. In fact, for some levels of discretization, StochasticSynthesis crashed due to memory limitation, whereas Mascot-SDS consumed quite manageable amount of memory and synthesized controllers within reasonable amount of time. (B) Even when the abstraction-refinement feature of StochasticSynthesis was enabled, for achieving the same level of approximation error, Mascot-SDS was significantly faster for φ_1 and was competitive for φ_2, and consumed much less memory for both φ_1 and φ_2: For φ_1, at one point Mascot-SDS was more than 150 times faster and consumed around 150 times lesser memory. These findings demonstrate the superior capabilities of our symbolic solution approach, which can be potentially further improved by using adaptive refinement techniques.

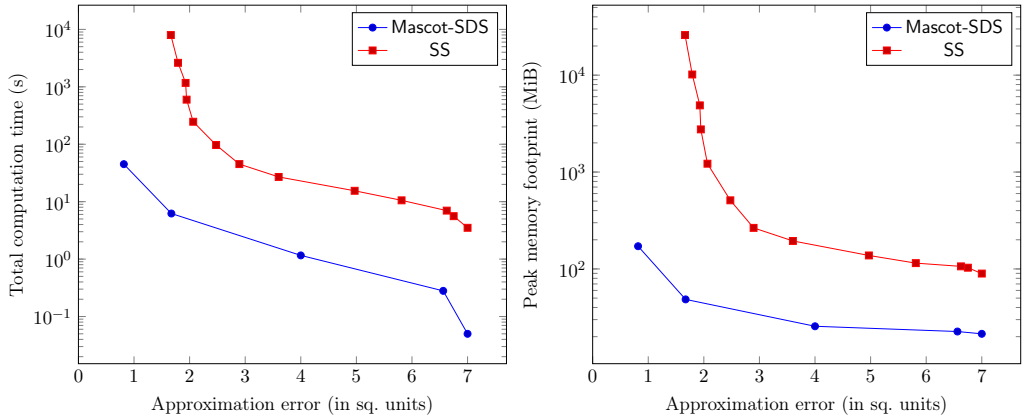
5.6.2. 3-Dimensional Vehicle

We consider the controller synthesis problem for a mobile robot, modeled using the sampled-time version of perturbed Dubins vehicle (Majumdar et al., 2020a). The system has three state variables, denoted as x_1, x_2, and x_3, and representing respectively the position along the X-coordinate, the position along the Y-coordinate, and the steering angle. The vehicle moves with a constant forward velocity V (maintained by, e.g., a low level cruise control system), which is set to 1 unit in this example. The single control input u is responsible for moving the steering wheel, and thus changing the direction of

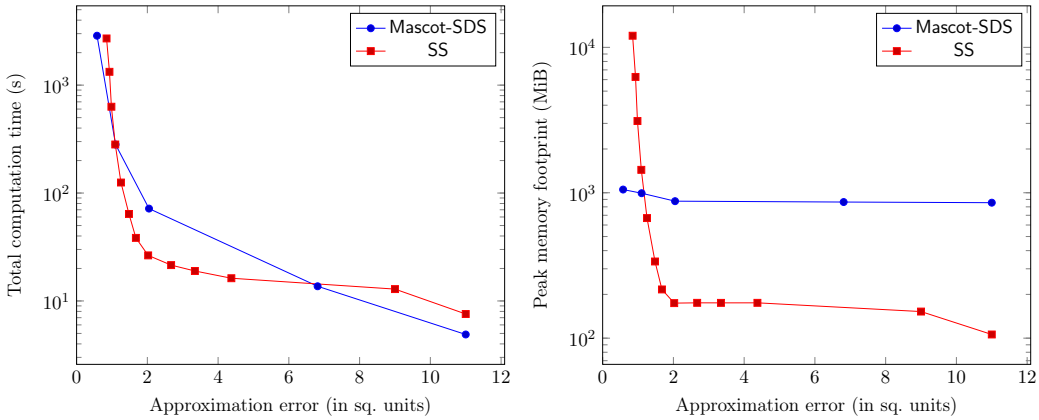
[1] Repository: `https://github.com/gtfactslab/StochasticSynthesis`,
commit nr.: `888b9dcf67369a732b8c225d790bd3343e4442e5`

Specification	Size of abstract states	Approximation error		Abstraction time		Total synthesis time		Peak memory footprint	
		Mascot-SDS	SS	Mascot-SDS	SS	Mascot-SDS	SS	Mascot-SDS	SS
φ_1	$1/2 \times 1/2$	7.0	9.0	0.003 s	0.4 s	0.02 s	8 s	21 MiB	125 MiB
	$1/4 \times 1/4$	6.6	5.9	0.04 s	13 s	0.2 s	18 s	23 MiB	1 GiB
	$1/8 \times 1/8$	4.0	3.0	0.2 s	35 min 5 s	0.7 s	9 min 18 s	26 MiB	81 GiB
	$1/16 \times 1/16$	1.7	OoM	0.9 s	OoM	5 s	OoM	50 MiB	127 GiB
	$1/32 \times 1/32$	0.8	OoM	5 s	OoM	37 s	OoM	171 MiB	127 GiB
φ_2	$1/2 \times 1/2$	11.0	11.8	0.004 s	0.6 s	5 s	30 s	864 MiB	156 MiB
	$1/4 \times 1/4$	6.8	3.4	0.02 s	15 s	13 s	55 s	866 MiB	1 GiB
	$1/8 \times 1/8$	2.0	1.8	0.2 s	34 min 20 s	1 min 10 s	16 min 1 s	890 MiB	81 GiB
	$1/16 \times 1/16$	1.1	OoM	0.9 s	OoM	4 min 37 s	OoM	994 MiB	126 GiB
	$1/32 \times 1/32$	0.6	OoM	5 s	OoM	45 min 39 s	OoM	1 GiB	127 GiB

Table 5.1.: Performance comparison between Mascot-SDS and StochasticSynthesis (abbreviated as SS) (Dutreix et al., 2020) when both tools are restricted to use *uniform grid-based abstraction* only. Col. 1 shows the specification considered, Col. 2 shows the size of each individual abstract state, Col. 3 and 4 compare the approximation errors, Col. 5 and 6 compare the abstraction computation times, Col. 7 and 8 compare the total synthesis times (combined time for computing the over- and the under-approximations of the almost sure winning regions), and Col. 9 and 10 compare the peak memory footprint (as measured using the "time" command) for both tools. "OoM" stands for out-of-memory. The length of the sides of the abstract states is measured in units, and the approximation error is measured in sq. units (the area covered by the abstract states which are in the over-approximation but not in the under-approximation).

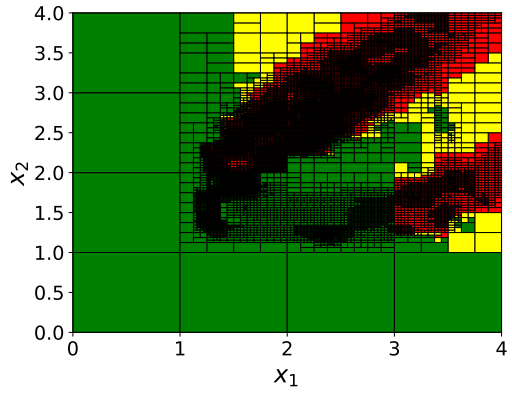


(a) Specification φ_1

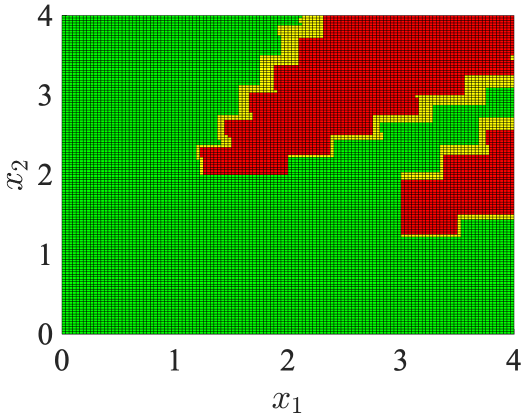


(b) Specification φ_2

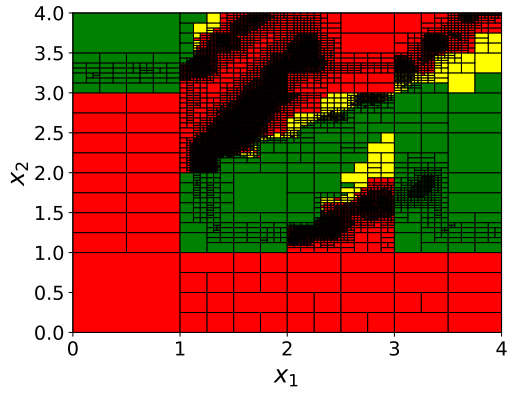
Figure 5.4.: Performance comparison between Mascot-SDS and StochasticSynthesis (abbreviated as SS) (Coogan and Arcak, 2015) when the latter *was allowed to use its inbuilt abstraction refinement process for better performance.* The different points on the plots were obtained by running Mascot-SDS using different sizes of abstract states, and running SS using different numbers of refinement stages.



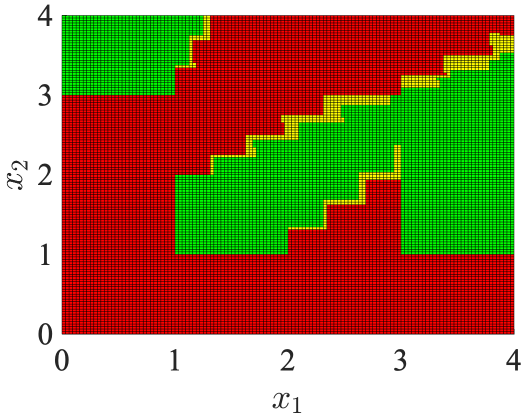
(a) φ_1 using StochasticSynthesis



(b) φ_1 using Mascot-SDS.



(c) φ_2 using StochasticSynthesis.



(d) φ_2 using Mascot-SDS.

Figure 5.5.: Visualization of the under-approximations—in green—and the over-approximations—in yellow and green, combined—of the almost sure winning regions for the specifications φ_1 and φ_2 as computed using the tools Mascot-SDS and StochasticSynthesis; in red are the complements of the over-approximations. For Mascot-SDS, the abstract states were chosen of the size $1/32 \times 1/32$. For StochasticSynthesis, initially the abstracts states were chosen of the size 1×1, and then the tool was made to execute 14 refinement steps to improve the approximation of the solution.

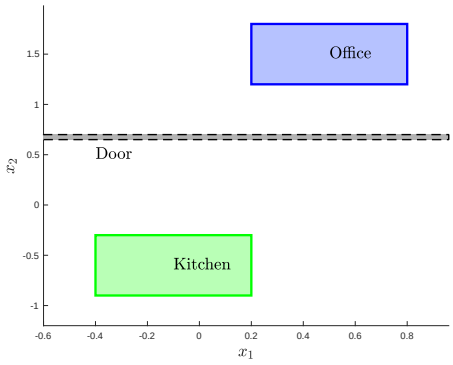


Figure 5.6.: Vehicle state space annotated with the location of the office, the kitchen, and the door.

the movement. The sampled-time dynamics for all $k \in \mathbb{N}$ and for all inputs $u^k \neq 0$ is given as follows:

$$\begin{aligned} x_1^{k+1} &= x_1^k + \frac{V}{u^k}\left[\sin(x_3^k + u^k\tau) - \sin(x_3^k)\right] + \varsigma_1^k \\ x_2^{k+1} &= x_2^k - \frac{V}{u^k}\left[\cos(x_3^k + u^k\tau) - \cos(x_3^k)\right] + \varsigma_2^k \\ x_3^{k+1} &= x_3^k + u^k\tau + \varsigma_3^k; \end{aligned}$$

when $u^k = 0$ then the dynamics can be obtained by taking limit $u^k \to 0$ in the right hand side of the above equations. for all $k \in \mathbb{N}$ with $u^k = 0$. The sampling time is $\tau = 0.1\,sec$ and $(\varsigma_1^k, \varsigma_2^k, \varsigma_3^k)$ is a collection of stochastic noise samples drawn from a piecewise continuous density function with the support $D = [-0.06, 0.06] \times [-0.06, 0.06] \times [-0.06, 0.06]$. We assume that the states of the vehicle moves inside the domain $[-0.6, 0.96] \times [-1.2, 1.98] \times [-\pi, \pi]$.

Fig. 5.6 shows the state space of the robot with various annotations for certain sets of states. The specification is provided using the following atomic propositions: 1. $A_0 \leftrightarrow$ Door is open,
2. $G_0 \leftrightarrow$ Robot inside office,
3. $G_1 \leftrightarrow$ Robot inside kitchen, and
4. Crash $\leftrightarrow$ Robot hits the door when it is closed. There is a safety requirement that the robot should never hit the closed door, i.e., $\square\neg$Crash. The rest of the specification is provided in an implication form. We assume that the following property is satisfied by the environment:

(a) the door opens infinitely often, i.e., $\square\lozenge A_0$,

(b) whenever the door is open, it remains open until the robot reaches the kitchen, i.e., $\square\,(A_0 \to (A_0 \mathcal{U} G_1))$.

If the environment satisfies the above, then the robot has to fulfill the following:

(a) The robot serves the request infinitely often, i.e., $\square\lozenge G_0$, and

Size of abstract states	Volume of the gap	Computation time		
		Abstraction	Over-approximation	Under-approximation
$0.1 \times 0.1 \times 0.1$	6.6	$< 1m$	$9m$	$31m$
$0.08 \times 0.08 \times 0.08$	4.8	$2m$	$84m$	$4h$
$0.06 \times 0.06 \times 0.06$	4.5	$7m$	$102m$	$9h$

Table 5.2.: Performance evaluation of our method on the Dubins vehicle: Col. 1 shows the size of abstract states, Col. 2 shows the volume of the difference between the over and the under-approximation, and Col. 3, 4, and 5 respectively show the computation time for the $2^1/_2$-player game, computation time for the over-approximation, and computation time for the under-approximation of the winning region.

(b) the robot goes to the kitchen infinitely often, i.e., $\square\lozenge G_1$.

The overall specification for the robot can be summarized as:

$$\begin{aligned} &\square\neg\text{Crash} \\ \wedge\ &(\square\lozenge A_0 \wedge \square\,(A_0 \rightarrow (A_0 \mathcal{U} G_1)) \rightarrow \square\lozenge G_0 \wedge \square\lozenge G_1)\,. \end{aligned} \tag{5.24}$$

The specification in (5.24) can be modeled as a 3-color parity automaton.We computed the synchronous product of the parity automaton and the vehicle's dynamics model.

We used the infrastructure of Mascot-SDS (Majumdar et al., 2020a) to compute a $2^1/_2$-player game and to synthesize an almost sure winning controller for the product system. We performed the experiments on a computer with 3.3GHz Intel Xeon E5 v2 processor and 256 GB RAM. We used three different levels of discretization for the abstract state space for computing the $2^1/_2$-player game. The results are summarized in Tab. 5.2. We would like to highlight two key facts which came out of the experiments: (a) In all three cases, when we treated the noise in the worst case fashion, the synthesis process failed to provide us any controller, and (b) as we decreased the size of the abstract states (i.e., finer abstraction), the gap between the over and the under-approximation of the controller domain got monotonically smaller, which empirically confirms the intuition that the quality of the controller improves with finer abstraction.

We also visualize a couple of different simulations with the obtained controller in Figs. 5.7a–5.7b. We empirically show that whenever the assumption A_0 continues to hold recurrently, the G_0 and G_1 also hold recurrently. In contrary, when A_0 does not hold persistently, G_0 and G_1 also does not hold persistently. This empirically validates our claim that the synthesized controller is sound.

5.7. Related Work

There are many extensions of ABCD to stochastic systems (Zamani et al., 2013, 2014, 2015; Haesaert et al., 2017; Lahijanian et al., 2015; Svorenová et al., 2015). Several tools

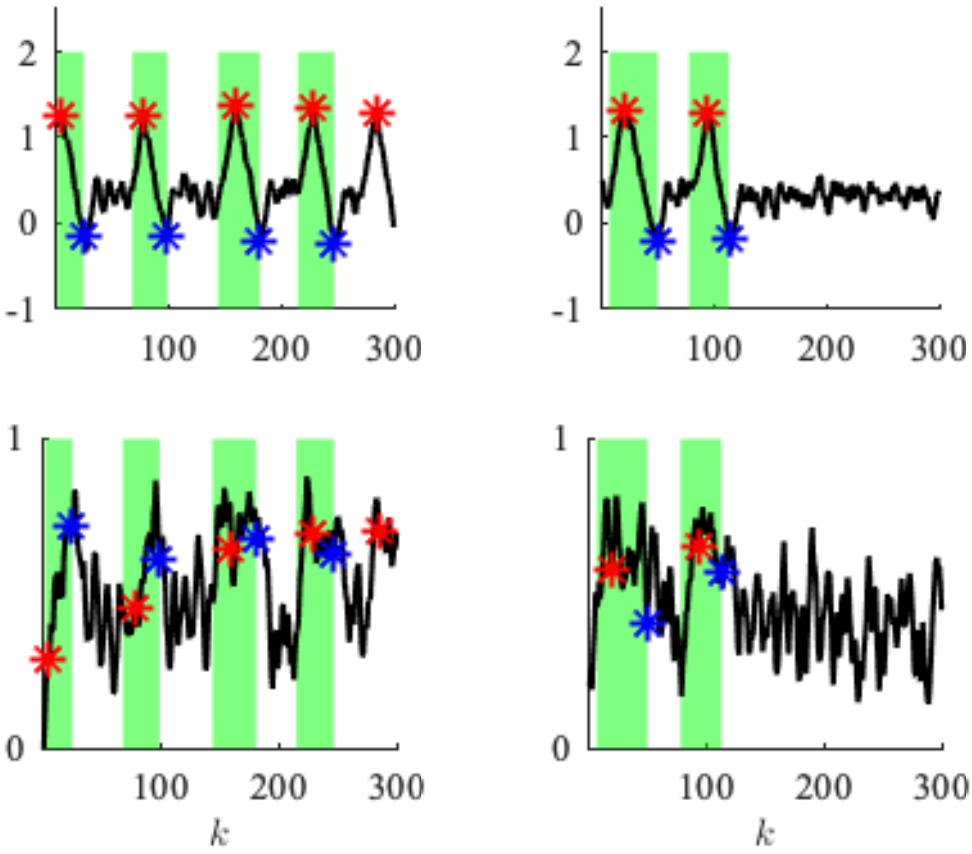


Figure 5.7.: The figures in the top and the bottom row respectively show the trajectories of the states x_1 and x_2 with respect to time. The green regions show when the assumption A_0 was satisfied, and the red and blue plot markers show when the guarantees G_0 and G_1 were satisfied respectively.

were also developed for such systems (Soudjani et al., 2015; Lavaei et al., 2020a).

Unfortunately, most of the existing methods can only properly handle finite horizon specifications, such as bounded time reachability or bounded time safety. Notable exceptions are the papers by Svorenová et al. (2015) and by Dutreix et al. (2020): The paper by Svorenová et al. (2015) only considered stochastic linear systems (as opposed to nonlinear systems in our setting) and GR(1) specifications, a subclass of the LTL specifications. The paper by Dutreix et al. (2020), which came after our paper (Majumdar et al., 2020a), considers nonlinear systems and the more general class of ω-regular specifications as us. An important advantage of our method as compared to theirs is the symbolic nature of our algorithm for a substantial part of the synthesis process. (Recall that we compute the almost sure winning region using symbolic algorithm and the rest using the existing non-symbolic algorithms.)

5.8. Conclusion

In this chapter, we considered the problem of synthesizing an optimal controller for a CMP so that a given parity specification is satisfied with maximum probability. We showed that ABCD can be used to approximately solve the problem with a guaranteed lower-bound on the satisfaction probability. Our solution approach is to break the problem into two parts, a qualitative part followed by a quantitative part. The qualitative part requires computing an under-approximation of the maximal almost sure winning region W, along with the respective controller. The quantitative part requires solving an optimal reachability problem to the set W, along with the respective controller. While there are known algorithms for solving the quantitative part, we propose a novel symbolic ABCD algorithm for solving the qualitative part. For this, we discretize the state space of the CMP to build an abstract $2^1/_2$-player game, and we show that an almost sure winning strategy for the abstract game can be refined back to an almost sure winning controller for the original CMP. We implement the ABCD algorithm for the qualitative part in the tool called `Mascot-SDS`, where for the abstract synthesis of the $2^1/_2$-player game, we use an efficient algorithm that will be presented in Chap. 7. We compared our tool with an independently developed contemporary tool solving the same problem, and showed that on a comparable set-up, our tool significantly outperforms the other one. Integration of the quantitative part in `Mascot-SDS` has been left out for future research.

Part II.

Synthesis using Discrete Abstractions

6. Assume-Guarantee Distributed Synthesis

Part II of this thesis considers synthesis of controllers using discrete system models.

In this chapter, we consider the well-known distributed reactive synthesis problem (Pnueli and Rosner, 1990). Suppose there are two finite transition systems which are connected in feedback, are subjected to external disturbances, and have to satisfy a pair of local temporal specifications. The distributed reactive synthesis problem asks to compute local controllers for each system such that both the local specifications are satisfied simultaneously. As we already discussed in the motivating example in Sec. 1.3.3, the main challenge of distributed synthesis is how to account for the feedback from the other system in a way that is permissive enough to give us a controller, yet restrictive enough to be able to provide soundness guarantees against every possible behavior of the other system.

The advantage of distributed synthesis over monolithic synthesis is that it is modular, meaning the controller of each system can be synthesized (partly) in isolation and in parallel. Modularity is a crucial aspect of designing large scale systems, where designing a monolithic centralized controller for the entire network may be infeasible. Moreover, modularity offers easy maintenance of such systems, in the sense that replacement of one system may be possible without readjusting the controllers of the other systems.

In this chapter, we show how two systems connected in feedback can negotiate a pair of assume-guarantee contracts, such that satisfaction of the contracts will ensure satisfaction of their local specifications. The negotiation procedure is a sound but incomplete solution to the otherwise undecidable distributed synthesis problem.

6.1. Assume-Guarantee Decompositions

6.1.1. Systems

In this chapter, we model systems as finite transition systems (see Def. 3.4), although we use a slightly different formalism having explicitly modeled disturbances and designated initial states. A system $S = \langle \mathcal{X}, x_{\mathsf{in}}, \mathcal{U}, \mathcal{W}, F, \mathcal{Y}, H \rangle$ consists of a finite *state space* $\mathcal{X}$, an initial state $x_{\mathsf{in}} \in \mathcal{X}$, a finite *control input space* $\mathcal{U}$, a finite *disturbance input space* $\mathcal{W}$ that can be expressed as the cartesian product of a finite *internal* disturbance input space $\mathcal{W}^{int}$ and a finite *external* disturbance input space $\mathcal{W}^{ext}$ (i.e. $\mathcal{W} = \mathcal{W}^{int} \times \mathcal{W}^{ext}$), a total *transition function* $F \colon \mathcal{X} \times \mathcal{W} \times \mathcal{U} \to \mathcal{X}$, a finite *output space* $\mathcal{Y}$, and an *output labeling function* $H \colon \mathcal{X} \to \mathcal{Y}$.

We make the control and the disturbance inputs explicit in the runs of a system. A *run* of a system is an infinite sequence $\xi = \{(x^i, w^i, u^i)\}_{i\in\mathbb{N}} \in (\mathcal{X} \times \mathcal{U} \times \mathcal{W})^\omega$ such that $F(x^i, w^i, u^i) = x^{i+1}$ for each $i \in \mathbb{N}$. Sometime, for convenience, we express a run using the notation $x^0 \xrightarrow{w^0,u^0} x^1 \xrightarrow{w^1,u^1} x^2 \xrightarrow{w^2,u^2} \ldots$. Unless otherwise mentioned, we will always assume that a run starts at the initial state, i.e., $x^0 = x_{\mathsf{in}}$. The output of the run is the sequence $H(x^0)H(x^1)\ldots$, which maps states to their output labels. Intuitively, first, the set of state variables are set to the initial value x^0. In each subsequent step, first an arbitrary environment input from $\mathcal{W}$ is picked, and then a control input from $\mathcal{U}$ is picked. The transition relation determines the new state, and the output labeling function determines the new output based on the new state.

For the run ξ, recall that we write $\mathbf{proj}_{\mathcal{X}}(\xi)$ for the sequence $x^0x^1\ldots \in \mathcal{X}^\omega$, $\mathbf{proj}_{\mathcal{W}}(\xi)$ for $w^0w^1\ldots \in \mathcal{W}^\omega$, and for the decomposition $\mathcal{W}^{int} \times \mathcal{W}^{ext} = \mathcal{W}$, we write $\mathbf{proj}_{\mathcal{W}^{int}}(\xi)$ and $\mathbf{proj}_{\mathcal{W}^{ext}}(\xi)$ for $\mathbf{proj}_{\mathcal{W}^{int}}(w^0w^1\ldots) \in \mathcal{W}^{int^\omega}$ and $\mathbf{proj}_{\mathcal{W}^{ext}}(w^0w^1\ldots) \in \mathcal{W}^{ext^\omega}$ respectively. We slightly abuse the notation and additionally write $\mathbf{proj}_{\mathcal{Y}}(\xi)$ for $H(x^0)H(x^1)\ldots \in \mathcal{Y}^\omega$.

Let S_0 and S_1 be two systems; assume $\mathcal{X}_0$, $\mathcal{X}_1$, $\mathcal{Y}_0$, and $\mathcal{Y}_1$ are all disjoint. We require that $\mathcal{Y}_0 \subseteq \mathcal{W}_1^{int}$ and $\mathcal{Y}_1 \subseteq \mathcal{W}_0^{int}$, which enables the two systems to interact among themselves using their outputs. We define the parallel composition $S_0 \parallel S_1$ (see Fig. 6.1) as the system $\langle \mathcal{X}, x_{\mathsf{in}}, \mathcal{U}, \mathcal{W}, F, \mathcal{Y}, H\rangle$, where $\mathcal{X} = \mathcal{X}_0 \times \mathcal{X}_1$, $x_{\mathsf{in}} = (x_{\mathsf{in}0}, x_{\mathsf{in}1})$, $\mathcal{U} = \mathcal{U}_0 \times \mathcal{U}_1$, $\mathcal{W} = \mathcal{W}_0^{ext} \times \mathcal{W}_1^{ext}$, and $\mathcal{Y} = \mathcal{Y}_0 \times \mathcal{Y}_1$ such that $H((x_0, x_1)) = (H_0(x_0), H_1(x_1))$ and $F((x_0, x_1), (w_0, w_1), (u_0, u_1)) = (x_0', x_1')$ iff $F_0(x_0, (H_1(x_1), w_0), u_0) = x_0'$ and $F_1(x_1, (H_0(x_0), w_1), u_1) = x_1'$.

A run ξ of $S_0 \parallel S_1$ is a sequence

$$x^0 \xrightarrow{w^0,(u_0^0,u_1^0)} x^1 \to \ldots$$

where for each $i \geq 0$, we have $x^i \in \mathcal{X}$, $w^i \in \mathcal{W}$, $u_0^i \in \mathcal{U}_0$, $u_1^i \in \mathcal{U}_1$, such that $x^0 = x_{\mathsf{in}}$ and for each $i \geq 0$, we have $\delta(x^i, w^i, (u_0^i, u_1^i)) = x^{i+1}$. Intuitively, the two systems S_0 and S_1 run synchronously in parallel. The state of the composed system is a pair of states of each system. In each step, an environment selects a disturbance input from the set $\mathcal{W}$. Each system sees its disturbance input, which consists of the external disturbance input from the environment as well as the internal disturbance input that is part of the output of the other system, picks a control input, and updates its state based on its transition function.

Since all the disturbances in $\mathcal{W}$ of $S_0 \parallel S_1$ are from the environment, hence $\mathbf{proj}_{\mathcal{W}}(\xi) = \mathbf{proj}_{\mathcal{W}^{ext}}(\xi)$ for every run ξ of $S_0 \parallel S_1$. On the other hand, for $i \in \{0, 1\}$, we use the operator $\mathbf{proj}_{\mathcal{W}_i^{int}}(\xi)$ to denote the sequence of internal disturbance inputs fed to system S_i, i.e. $\mathbf{proj}_{\mathcal{W}_i^{int}}(\xi) = \mathbf{proj}_{\mathcal{Y}_{1-i}}(\xi)$. We also use the operator $\mathbf{proj}_{\mathcal{W}_i^{int}}(\cdot)$ on sequences of states $x^0x^1\ldots \in \mathcal{X}^\omega$: $\mathbf{proj}_{\mathcal{W}_i^{int}}(x^0x^1\ldots) = \mathbf{proj}_{\mathcal{Y}_{1-i}}(x^0x^1\ldots)$.

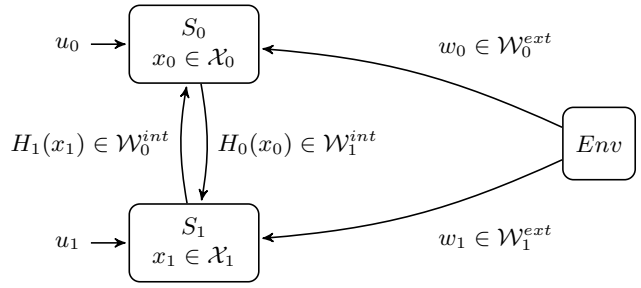


Figure 6.1.: A composition $S_0 \parallel S_1$

6.1.2. Distributed Realizability

Specifications and Realizability. A *specification* for a system is a language $\psi \subseteq (\mathcal{X} \times \mathcal{W})^\omega$ that describes the correct runs; a run $x^0 \xrightarrow{w^0,u^0} x^1 \to \dots$ *satisfies* a specification ψ if $(x^0, w^0)(x^1, w^1) \dots \in \psi$. A *local* specification is a language $\psi \subseteq \mathcal{X}^\omega$. A run satisfies a local specification ψ if it satisfies the specification $\{(x^0, w^0)(x^1, w^1) \dots \in (\mathcal{X} \times \mathcal{W})^\omega \mid x^0 x^1 \dots \in \psi\}$.

A *strategy* for a system is a function $\pi \colon (\mathcal{X} \times \mathcal{W})^+ \to \mathcal{U}$. Likewise, an environment strategy is a function $\pi' \colon (\mathcal{X} \times \mathcal{W})^* \times \mathcal{X} \to \mathcal{W}$. A run $x^0 \xrightarrow{w^0,u^0} x^1 \to \dots$ is compliant with π and/or π' if for each $i \geq 0$, we have $u^i = \pi(x^0, w^0, \dots, x^i, w^i)$ and/or $w^i = \pi'(x^0, w^0, \dots, x^{i-1}, w^{i-1}, x^i)$. We denote by $\xi(\pi, \pi')$ the unique run compliant with π and π'. Unless stated otherwise, a compliant run must start from $x^0 = x_{\mathsf{in}}$.

A system S *can realize* a specification ψ (or ψ is realizable by S) if there is a system strategy π, called the *realization strategy for* ψ, such that for all runs ξ compliant with π it holds that $\mathbf{proj}_{\mathcal{X} \times \mathcal{W}}(\xi) \in \psi$. Intuitively, we model the realizability of a specification as a two-player game between the system and the environment; the system can realize a specification if it has a realization strategy such that no matter how the environment plays, the resulting run belongs to the language of the specification.

We sometime use Linear Temporal Logic (see Sec. 2.2.1) notation to express local specifications over some finite alphabet $AP \in 2^{\mathcal{X}}$. Recall that, given an atomic proposition $B \subseteq \mathcal{X}$, we write $\Box B$ (safety: "always B"), $\Diamond B$ (reachability: "eventually B"), and $\Box\Diamond B$ (Büchi "eventually always B") to denote respectively the sets $\{x^0 x^1 \dots \mid \forall i \geq 0 \,.\, x^i \in B\} \subseteq \mathcal{X}^\omega$, $\{x^0 x^1 \dots \mid \exists i \geq 0 \,.\, x^i \in B\} \subseteq \mathcal{X}^\omega$, and $\{x^0 x^1 \dots \mid \forall i \geq 0 \,.\, \exists j \geq i \,.\, x^j \in B\} \subseteq \mathcal{X}^\omega$. See Sec. 2.2.1 for more detailed description of syntax and semantics of LTL.

Distributed Realizability. Now consider a composition $S_0 \parallel S_1$. Suppose $\psi_0 \subseteq \mathcal{X}_0^\omega$ and $\psi_1 \subseteq \mathcal{X}_1^\omega$ are local specifications defined on the state variables of each system. One can define realization for the composition $S_0 \parallel S_1$ by considering a game between the composed system and the environment. However, such a centralized realization strategy may require coordination, e.g., to know the states of the two systems at some point. Instead, we want the realization strategies to be *distributed*: each component S_0 and S_1 should be able to pick their control inputs based solely on the local history of valuations to state variables and their own disturbance inputs.

We model the distributed synthesis problem as a game (of incomplete information) between three players: system S_0, system S_1, and the environment. The game starts from the initial state x_{in} of $S_0 \parallel S_1$. In each step of the game, first, the environment picks external disturbance in $\mathcal{W}$, and then the systems S_0 and S_1 independently and simultaneously pick control inputs and the game proceeds to the next state.

We require that the strategies of system S_0 and system S_1 only depend on the history visible to them. Thus, we define a strategy of system S_i, $i \in \{0, 1\}$, to be a function of the form $(\mathcal{X}_i \times \mathcal{W}_i)^+ \to \mathcal{U}_i$. Fixing strategies π_0, π_1, and π' of systems S_0, S_1, and the environment, respectively, yields a unique run $\xi(\pi_0, \pi_1, \pi')$ of the system $S_0 \parallel S_1$.

The *distributed synthesis* problem $\langle S_0, \psi_0, S_1, \psi_1 \rangle$ for the composition $S_0 \parallel S_1$ with local specifications $\psi_0 \subseteq \mathcal{X}_0^\omega$ and $\psi_1 \subseteq \mathcal{X}_1^\omega$ asks if there exist strategies $\pi_0 \colon (\mathcal{X}_0 \times \mathcal{W}_0)^+ \to \mathcal{U}_0$ and $\pi_1 \colon (\mathcal{X}_1 \times \mathcal{W}_1)^+ \to \mathcal{U}_1$ such that for all strategies $\pi' \colon (\mathcal{X} \times \mathcal{W})^* \times \mathcal{X} \to \mathcal{W}$, we have that $\mathbf{proj}_{\mathcal{X}_0}(\xi(\pi_0, \pi_1, \pi')) \in \psi_0$ and $\mathbf{proj}_{\mathcal{X}_1}(\xi(\pi_0, \pi_1, \pi')) \in \psi_1$. In that case, we say $S_0 \parallel S_1$ can *realize* the distributed synthesis problem.

Clearly, if S_0 and S_1 can each realize the local specifications ψ_0 and ψ_1 respectively, then $S_0 \parallel S_1$ can also realize the distributed synthesis problem. This is because the strategies do not make any assumptions on the behavior of the other system. However, it is possible that S_0 and S_1 do not each realize their specifications but they realize the distributed synthesis problem; for example, one system can use an assumption about the behavior of the other.

Unfortunately, the distributed synthesis problem is undecidable in general (Pnueli and Rosner, 1990). We summarize the discussion in the following proposition.

Proposition 6.1 *(1) If for $i \in \{0, 1\}$, the system S_i realizes ψ_i then the composition $S_0 \parallel S_1$ realizes the distributed synthesis problem $\langle S_0, \psi_0, S_1, \psi_1 \rangle$. (2) (Pnueli and Rosner, 1990) The distributed synthesis problem is undecidable.*

We introduce some notation. For $i \in \{0, 1\}$, we define the *realizable region*, denoted as $\langle\langle S_i \rangle\rangle \psi_i$, as the largest subset of $\mathcal{X}_i$ such that for all states $x \in \langle\langle S_i \rangle\rangle \psi_i$ there exists a run ξ compliant with a realization strategy π of ψ_i that visits x. Now consider a composition $S_0 \parallel S_1$. We say system S_i *can maybe-realize* ψ_i (or ψ_i is *maybe-realizable* by S_i) if there is a pair of (possibly co-ordinated) strategies π_0, π_1, called the *joint realization strategy*, such that for all strategies π' it holds that $\mathbf{proj}_{\mathcal{X}_i}(\xi(\pi_0, \pi_1, \pi')) \in \psi_i$. We define the *maybe-realizable region*, denoted as $\langle\langle S_0, S_1 \rangle\rangle \psi_i$, as the largest subset of $\mathcal{X}_i$ such that for all states in $x \in \langle\langle S_i \rangle\rangle \psi_i$ there exists a run ξ compliant with a *joint realization strategy* π_0, π_1 of ψ_i that visits x. We also define the *surely unrealizable region* as the complement of the maybe-realizable region.

6.1.3. Assume-Guarantee Contracts

Given a system S, an *assume-guarantee contract*—a *contract* in short—is a pair $\langle A, G \rangle$ of *safety* languages called the *assumption* $A \subseteq \mathcal{W}^{int^\omega}$ and the *guarantee* $G \subseteq \mathcal{X}^\omega$.

Definition 6.1 Let S be a system, and $\langle A, G\rangle$ be a contract. Then S can realize $\langle A, G\rangle$ if and only if there exists a system strategy π, such that for all $k > 0$ and for all finite runs $r \equiv x^0 \xrightarrow{w^0,u^0} x^1 \to \dots x^k$ compliant with π, *either* of the following holds:
(a) $\mathbf{proj}_{\mathcal{X}}(r) \in \mathrm{pref}(G)$,
(b) there exists $0 \leq l < k$ such that $\mathbf{proj}_{\mathcal{W}^{int}}(r)|_{[0;l]} \notin \mathrm{pref}(A)$.
That is, a violation of G is preceded by a violation of A. The respective strategy π is called a *realization strategy for* $\langle A, G\rangle$.

For a contract $\langle A, G\rangle$ and specification ψ, we say S *can realize* ψ *under* contract $\langle A, G\rangle$, written S can realize the specification $\langle A \rhd \psi \rhd G\rangle$, if there exists a strategy of S that is both a realization strategy for the contract $\langle A, G\rangle$ and a realization strategy for the specification $(A \Rightarrow \psi)$. The maybe-realizability and sure unrealizability of a contract $\langle A, G\rangle$ and the specification $\langle A \rhd \psi \rhd G\rangle$ are defined analogously.

Definition 6.2 Consider a system composition $S_0 \parallel S_1$. Let $\langle A_0, G_0\rangle$ and $\langle A_1, G_1\rangle$ be a pair of contracts for respectively S_0 and S_1 such that $\emptyset \subsetneq A_i \subseteq \mathcal{W}_i^{int^\omega} = \mathcal{Y}_{1-i}^{\omega}$ and $\emptyset \subsetneq G_i \subseteq \mathcal{X}_i^{\omega}$. Then the contracts $\langle A_0, G_0\rangle$ and $\langle A_1, G_1\rangle$ are *compatible* if the following conditions are met for both $i \in \{0, 1\}$:
(a) $G_i \subseteq H_i^{-1}(A_{1-i})$, and (b) S_i realizes $\langle A_i, G_i\rangle$.

The composition of compatible contracts satisfies the following claim, motivated by Chandy and Misra (1988); Alur and Henzinger (1999); McMillan (1999); Namjoshi and Trefler (2010); Saoud et al. (2018a); Ozay et al. (2011):

Theorem 6.1 ***[Assume Guarantee Decomposition]*** *Let $\langle S_0, \psi_0, S_1, \psi_1\rangle$ be the input to a distributed synthesis problem. Let $\langle A_0, G_0\rangle$ and $\langle A_1, G_1\rangle$ be a pair of compatible contracts of S_0 and S_1 respectively. If S_0 can realize ψ_0 under $\langle A_0, G_0\rangle$ and S_1 can realize ψ_1 under $\langle A_1, G_1\rangle$, then $S_0 \parallel S_1$ can realize the distributed synthesis problem.*

Proof We use the following notation. For a given run $\xi \equiv x^0 \xrightarrow{w^0,u^0} x^1 \to \dots x^k \xrightarrow{w^k,u^k} x^{k+1} \to \dots$ and for a given $k \geq 0$, we write $\mathrm{pref}^k(\xi)$ for the prefix of the run $x^0 \xrightarrow{w^0,u^0} x^1 \xrightarrow{w^1,u^1} \dots \xrightarrow{w^{k-1},u^{k-1}} x_k$ of length k.
▶ First, observe that a compatible contract implies that there exist two strategies π_0 and π_1 which fulfill the conditions in Def. 6.1 for the individual systems S_0 and S_1. Let, for some strategy of the external environment, $\xi \equiv x^0 \xrightarrow{w^0,(u_0^0,u_1^0)} x^1 \to \dots$ be a run of $S_0 \parallel S_1$ that is compliant with both π_0 and π_1. First, for both $i \in \{0, 1\}$, we prove by induction that for every k, $\mathbf{proj}_{\mathcal{X}_i}(\mathrm{pref}^k(\xi)) \in \mathrm{pref}(G_i)$; then because G_i is a safety language, it will be established that $\mathbf{proj}_{\mathcal{X}_i}(\xi) \in G_i$.
▷ The base case: For $k = 0$, $\mathbf{proj}_{\mathcal{X}_i}(\mathrm{pref}^k(\xi)) = x_{\mathsf{in}i}$, and we see that Cond. (a) in Def. 6.1 must hold. (Cond. (b) can not be true since l cannot be negative.)
▷ Induction step: Fix a $k \geq 0$ such that $\mathbf{proj}_{\mathcal{X}_i}(\mathrm{pref}^k(\xi)) \in \mathrm{pref}(G_i)$ for both $i \in \{0, 1\}$. That is, Cond. (a) in Def. 6.1 holds. We show that the same is true for $k + 1$. We obtain the following chain of implications: From the assumption we have $\mathbf{proj}_{\mathcal{X}_i}(\mathrm{pref}^k(\xi)) \in$

$\text{pref}(G_i)$. With this, it follows from Def. 6.2 (a) that $\mathbf{proj}_{\mathcal{Y}_i}(\text{pref}^k(\xi)) \in \text{pref}(A_{1-i})$. This implies that for all $0 \leq l < k+1$, $\mathbf{proj}_{\mathcal{Y}_i}(\text{pref}^l(\xi)) \in \text{pref}(A_{1-i})$. This in turn implies that Cond. (b) in Def. 6.1 does not hold for $\text{pref}^{k+1}(\xi)$ for both $i \in \{0,1\}$. Therefore, Cond. (a) must hold, which proves the induction step. ▶ With this we get $\mathbf{proj}_{\mathcal{X}_i}(\text{pref}^k(\xi)) \in \text{pref}(G_i)$ for any $k \in N$. As G_i is a safety language, this implies $\mathbf{proj}_{\mathcal{X}_i}(\xi) \in G_i$. Then it follows from Def. 6.2 (a) that $\mathbf{proj}_{\mathcal{Y}_i}(\xi) \in A_{1-i}$.
▶ Both systems S_i can additionally realize their specification Φ_i under the given contract if there exist strategies π_0 and π_1 which renders both contracts compatible (implying $\mathbf{proj}_{\mathcal{Y}_i}(\xi) \in A_{1-i}$ over all its compliant traces from above) and additionally ensuring that Φ_i holds on all traces on which A_i holds. As the latter is always true, we see that $\mathbf{proj}_{\mathcal{X}_i}(\xi) \in \Phi_i$ for both $i \in \{0,1\}$. □

6.2. The Negotiation Process

Our goal is to iteratively compute a pair of compatible contracts. Our procedure will be *sound*: if it returns contracts $\langle A_0, G_0\rangle$ and $\langle A_1, G_1\rangle$, we shall be certain that the premises of Thm. 6.1 hold. However, since the distributed synthesis problem is undecidable, we may not find compatible contracts.

The iterative computation progresses in rounds. Initially (round 0), the assumptions A_0, A_1 and the guarantees G_0, G_1 allow all behaviors. In each round, S_0 and S_1 check if each can realize the specification $\langle A_i \rhd \psi_i \rhd G_i\rangle$. If so, the iteration ends, and we return the current assumptions and guarantees. On the other hand, if either system surely cannot realize its respective contract, then there is no point continuing and the process stops with failure. If none of the above happens, the negotiation process continues and the systems take turns to refine their assumptions and guarantees.

The key step in refining the assumptions and guarantees requires finding a *sufficient assumption on the other system* that enables realization of the current specification. In principle, this assumption should also be *maximally permissive* to offer maximal freedom to the other system. We first define maximally permissive sufficient assumptions, and then use this definition to formalize the negotiation procedure. At the same time, we also demonstrate that maximal permissiveness comes with its own technical challenges and lack of practicality. So in the end, we resort to a more practical and non-maximally permissive assumption, and show how to compute them.

6.2.1. Maximally Permissive Sufficient Assumption

We fix an input $\langle S_0, \psi_0, S_1, \psi_1\rangle$. A language $L \subseteq \mathcal{Y}_{1-i}^{\omega}$ is a *maximally permissive sufficient assumption*—simply *assumption* in short—for ψ_i if (1) L is *sufficient*, i.e., S_i can realize $(L \Rightarrow \psi_i)$ and (2) L is *maximally permissive*, i.e., S_i cannot realize $(L' \Rightarrow \psi_i)$ for any proper superset $L' \supsetneq L$.

Intuitively, an assumption is a sufficient restriction on the other system: S_i can realize its specification provided the other system always produces outputs belonging to the assumption. Moreover, the assumption is maximal in that no proper superset is sufficient

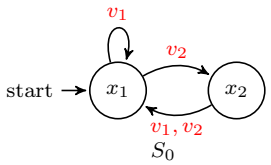


Figure 6.2.: A simple game

for S_i to realize its specification: this ensures we restrict the other system in the least pervasive way.

If S_i can already realize ψ_i on its own, then the assumption is all of $\mathcal{Y}^{\omega}_{1-i}$. If S_i surely cannot realize ψ_i, then $\emptyset$ is a maximal assumption. The maximality constraint rules out "trivial" solutions such as $L = \emptyset$ in other cases.

Example 6.1 Consider system S_0 in Fig. 6.2. We assume $\mathcal{U}_0$ is a singleton, and so we have omitted it from the figure. There is a single system strategy π that picks the singleton action at each state. The internal disturbance input has values $\{v_1, v_2\}$. Consider the specification $\square\diamond x_2$, that requires the state x_2 to be visited infinitely often. The system cannot realize this property if from some point on, the environment keeps playing v_1 when the state is x_1. Thus, a maximally permissive sufficient assumption is $L_1 = ((v_1^*(v_2(v_1 + v_2))^*)^* v_2)^\omega$. A sufficient assumption is $L_2 = (v_1 v_2)^\omega$, which is *not* maximally permissive because $L_1 \supsetneq L_2$.

The following example suggests that maximally permissive sufficient assumptions are not unique.

Example 6.2 Consider a system with two control inputs H and T and two disturbance inputs, also called H and T. From the initial state x_{in}, the control inputs H and T take the system to two different states h and t, respectively. Once at h or t, the state returns to x_{in} only if the environment plays H or T, respectively, but otherwise the state goes to *bad* and subsequently stays there no matter what input or action is chosen. Consider the specification $\square\neg bad$. Since the system does not know what the environment will play next, the specification is not realizable. However, *any* singleton ω-language is a maximally permissive sufficient assumption. Thus, there are infinitely many incomparable assumptions, and they even may not be ω-regular.

Maximality is useful to ensure non-trivial assumptions but, as Ex. 6.2 shows, this may lead to unbounded iterations through strategies and counter-strategies. Moreover, a maximal assumption might leave the system with only a single available realization strategy (Chatterjee et al., 2008). To mitigate these two issues, we consider an under-approximation of maximal assumptions. Note that strengthening an assumption retains realizability. A sufficient assumption L for ψ is called *universally* maximally permissive, if (1) L is a sufficient assumption, and (2) L is the intersection of every maximally permissive assumption. Universally maximal assumptions give up completeness—$\emptyset$ is the only universally maximal assumption in Ex. 6.2—but bound the search space of

assumptions because it is unique. Moreover, it gives maximal freedom to the system in choosing from all possible realization strategies.

Procedure *FindAssumptions*. For the moment, we assume the following: there is a procedure *FindAssumptions* that given a system S, a specification ψ, and safe ω-languages $\langle A, G\rangle$, returns a safe sufficient assumption for the specification $\langle A \rhd \psi \rhd G\rangle$. We expect that the assumption gives as much freedom to the other system as possible. We shall discuss implementations of *FindAssumptions* subsequently.

6.2.2. Negotiation

As mentioned earlier, the iterative computation of contracts proceeds in rounds, called *negotiation*. We call the overall iterative algorithm *Negotiate*, and the steps are summarized in Alg. 5. Initially (round 0), the existing assumptions A_0, A_1 and the guarantees G_0, G_1 allow all behaviors:

$$A_0^{(0)} = \mathcal{Y}_1^\omega, \;\; G_0^{(0)} = \mathcal{X}_0^\omega, \;\; A_1^{(0)} = \mathcal{Y}_0^\omega, \;\; G_1^{(0)} = \mathcal{X}_1^\omega.$$

Suppose we have constructed $A_i^{(r)}, G_i^{(r)}$ for $i \in \{0,1\}$ in round r. Let us look at round $r+1$.

In round $r+1$, if either system S_i surely cannot realize the specification $G_i^{(r)} \cap \psi_i$, we can give up with failure—certainly, we shall not find a refinement that works. In this case Alg. 5 returns **DoesNotExist**. On the other hand, if both systems can realize their specification under the current contracts $\left\langle A_i^{(r)}, G_i^{(r)} \right\rangle$, we have converged and we can stop and return the current contracts.

If the above conditions do not hold, we first pick the assumption L from *FindAssumptions* $\left(S_0, \left\langle A_0^{(r)}, G_0^{(r)} \right\rangle, \psi_0\right)$, and if L is non-trivial then update the assumptions and guarantees as follows:

$$A_0^{(r+1)} := A_0^{(r)} \cap L, \quad G_1^{(r+1)} := G_1^{(r)} \cap H_1^{-1}(L). \tag{6.1}$$

Then, we pick the additional sufficient assumption L' from *FindAssumptions* $\left(S_1, \left\langle A_1^{(r)}, G_1^{(r+1)} \right\rangle, \psi_1\right)$, and if L' is non-trivial then update the assumptions and guarantees as follows:

$$A_1^{(r+1)} := A_1^{(r)} \cap L', \quad G_0^{(r+1)} := G_0^{(r)} \cap H_0^{-1}(L'). \tag{6.2}$$

We move to the next round with these new contracts. Note that, at the end of every round, Alg. 5 either fails to find a contract and returns **DoesNotExist**, or else it obtains the sets $A_0^{(r)}$, $A_1^{(r)}$, $G_0^{(r)}$, and $G_1^{(r)}$ which are *all nonempty* as required by Def. 6.2.

We are ready to state our main theorem on *Negotiate*.

Theorem 6.2 *If Negotiate*$(S_0, \psi_0, S_1, \psi_1)$ *returns contracts* (A_0, G_0), (A_1, G_1), *then the contracts* (A_0, G_0) *and* (A_1, G_1) *are compatible, and moreover each* S_i *can realize* $\langle A_i \rhd \psi_i \rhd G_i\rangle$ *for* $i \in \{0,1\}$.

Algorithm 5 *Negotiate*

Input: $\langle S_0, \psi_0, S_1, \psi_1 \rangle$
Output: A_0, G_0, A_1, G_1 *or* **DoesNotExist**
1: $A_0^{(0)} \leftarrow \mathcal{Y}_1^\omega$, $G_0^{(0)} \leftarrow \mathcal{X}_0^\omega$, $A_1^{(0)} \leftarrow \mathcal{Y}_0^\omega$, $G_1^{(0)} \leftarrow \mathcal{X}_1^\omega$
2: **for** $r = 0, 1, 2, \ldots$ **do**
3: **if** S_i can realize $\langle A_i^{(r)} \rhd \psi_i \rhd G_i^{(r)} \rangle$ for both $i \in \{0, 1\}$ **then**
4: **return** $A_0^{(r)}, G_0^{(r)}, A_1^{(r)}, G_1^{(r)}$
5: **end if**
6: **if** S_i surely cannot realize $\langle A_i^{(r)} \rhd \psi_i \rhd G_i^{(r)} \rangle$ for either of $i \in \{0, 1\}$ **then**
7: **return DoesNotExist**
8: **end if**
9: $L \leftarrow \mathit{FindAssumptions}\left(S_0, \left\langle A_0^{(r)}, G_0^{(r)} \right\rangle, \psi_0\right)$
10: $A_0^{(r+1)} := A_0^{(r)} \cap L$, $G_1^{(r+1)} := G_1^{(r)} \cap H_1^{-1}(L)$
11: $L' \leftarrow \mathit{FindAssumptions}\left(S_1, \left\langle A_1^{(r)}, G_1^{(r+1)} \right\rangle, \psi_1\right)$
12: $A_1^{(r+1)} := A_1^{(r)} \cap L'$, $G_0^{(r+1)} := G_0^{(r)} \cap H_0^{-1}(L')$
13: **end for**

Proof (Proof of Thm. 6.2) We show that both the conditions of Def. 6.2 are met: Cond. a follows by induction over the round indices r and by the construction of the assumptions and guarantees in each round. (Actually, we maintain the invariant $G_i^{(r)} = H_i^{-1}(A_{1-i}^{(r)})$ at each round r, which is stronger than Cond. a.) Cond. b follows from the condition on successful termination. □

6.2.3. Implementing *FindAssumptions*

We now describe the algorithm *FindAssumptions* to compute a safe under-approximation of the universally maximally permissive sufficient assumption. For algorithmic effectiveness, we only find assumptions and guarantees which are safe *ω-regular languages* (compared to safe ω-languages as per Def. 6.1). This restriction allows us to implement *FindAssumptions* by using operations on *finite* structures. Our algorithm uses as subroutines both the (non-distributed) realizability algorithm from Pnueli and Rosner (1989); Maler et al. (1995); Thomas (1995) and the algorithm to compute environment assumptions from Chatterjee et al. (2008).

Subroutine I: Centralized Reactive Synthesis. The following theorem (summarizing Pnueli and Rosner (1989); Thomas (1995)) outlines a method that can be used for solving a reactive synthesis problem in the presence of assume-guarantee contracts. Given a system S, a natural number $d > 0$, and a mapping $c : \mathcal{X} \to \{0, \ldots, d\}$ that maps each state to a priority, recall that a *parity objective* $\Psi(c)$ states that the minimum priority visited infinitely often is even.

Theorem 6.3 *Let $S = \langle \mathcal{X}, x_{\mathsf{in}}, \mathcal{U}, \mathcal{W}, F, \mathcal{Y}, H \rangle$ be a system, and $\mathcal{W} = \mathcal{W}^{int} \times \mathcal{W}^{ext}$ where $\mathcal{W}^{int}$ is a set of internal disturbance inputs under the control of another system S'. Given*

a pair of ω-regular languages $A \subseteq \mathcal{W}^{int\omega}$ and $G \subseteq \mathcal{X}^\omega$, and an ω-regular specification $\Phi \subseteq \mathcal{X}^\omega$, there is an effectively constructible system $\widetilde{S} = (\widetilde{X}, \widetilde{x_{\mathsf{in}}}, U, W, \widetilde{\delta}, Y, \widetilde{h})$, a number $d \geq 0$, and a parity specification $\Psi(c)$ for a mapping $c\colon \widetilde{X} \to \{0, \ldots, d\}$, such that the following hold:

(i) System $\widetilde{S}$ can realize $\Psi(c)$ if and only if system S can realize $\langle A \rhd \Phi \rhd G \rangle$.

(ii) System $\widetilde{S}$ can maybe-realize $\Psi(c)$ in the composition $\widetilde{S} \parallel S'$ if and only if system S can maybe-realize $\langle A \rhd \Phi \rhd G \rangle$ in the composition $S \parallel S'$.

(iii) There is a mapping from the set of memoryless realization strategies of the form $\widetilde{\pi}\colon \widetilde{X} \to U$ of $\widetilde{S}$ to the set of realization strategies of S.

(iv) There is a mapping from the set of memoryless joint realization strategies of the form $\widetilde{\pi}\colon \widetilde{X} \to U \times V$ of $\widetilde{S}$ and S' to the set of joint realization strategies of S and S'.

The system $\widetilde{S}$ is essentially the product of the original system with a deterministic parity automaton for the specification $\langle A \rhd \Phi \rhd G \rangle$ (see, e.g., Chatterjee et al. (2008)). Recall that parity specifications have memoryless realization strategies, and each strategy of $\widetilde{S}$ can be converted to a strategy (possibly using memory) for the original system.

Subroutine II: Finding Environment Assumptions. The second subroutine is the method of Chatterjee et al. (2008) to compute a minimal set of sufficient environment assumptions for satisfaction of a given omega-regular specification. At a high-level, their algorithm imposes safety and liveness restriction on sets of environment behaviors which help the system to realize its specification. The safety restrictions, here called *safe-sufficient restrictions*, require that certain environment actions be never applied at certain system states. The liveness restrictions, here called *live-sufficient restrictions*, require that certain environment actions be repeatedly taken if certain system states are repeatedly visited (strong liveness property). We formalize these in the following.

Fix a system composition $S_0 \parallel S_1$. First, we formalize safe-sufficient and live-sufficient restrictions with respect to system S_0, with the understanding that similar results can be obtained for S_1 just by changing the indices. Recall that we assumed $\mathcal{W}_0^{int} = \mathcal{Y}_1$, and so in the remaining portion of this section we use them interchangeably. Let $\Phi \subseteq (\mathcal{X}_0)^\omega$ be a local parity specification for S_0.

A set of pairs $E_s \subseteq \mathcal{X}_0 \times \mathcal{Y}_1$ is a *safe-sufficient restriction* on S_1 for Φ if there exists a strategy $\pi\colon \mathcal{X}_0 \times \mathcal{W}_0 \to \mathcal{U}$ of S_0 such that for every joint counter-strategy $\pi'\colon \mathcal{X}_0 \to \mathcal{W}_0$ of S_1 and the environment, the resulting run $\xi(\pi, \pi') \equiv x^0 \xrightarrow{w^0,u^0} x^1 \xrightarrow{w^1,u^1} x^2 \xrightarrow{w^2,u^2} \ldots$ satisfies the following: either (a) there exists a $i \geq 0$ with $(x^i, \mathbf{proj}_{\mathcal{Y}_1}(w^i)) \in E_s$, or (b) for all $i \geq 0$, $x^i \in \langle\!\langle S_0, S_1 \rangle\!\rangle \Phi$. A safe-sufficient restriction E_s is *unfair* if there is a run ξ and there is a pair $(x, y) \in E_s$ such that at some time instant j, $\mathbf{proj}_{\mathcal{X}_0}(\xi)^j = x$ and $\mathbf{proj}_{\mathcal{Y}_1}(\xi)^j = y$, and yet $\xi \in \psi$. A safe-sufficient restriction is *fair* if it is not unfair, and moreover it is *minimal* if no other safe-sufficient restriction of smaller size exists.

A set of pairs $E_l \subseteq \mathcal{X}_0 \times \mathcal{Y}_1$ is a *live-sufficient restriction* on S_1 for Φ if there exists a strategy $\pi\colon \mathcal{X}_0 \to U$ of S_0 such that for every joint counter-strategy $\pi'\colon \mathcal{X}_0 \to \mathcal{W}_0$

of S_1 and the environment, the resulting run $\xi(\pi, \pi')$ satisfies the following: either (a) there exists a pair $(x, y) \in E_l$ such that in the run $\xi(\pi, \pi')$, x appears infinitely often but after some finite time step, y never immediately follows x, or (b) $\mathbf{proj}_{\mathcal{X}_0}(\xi(\pi, \pi')) \in \Phi$. A live-sufficient restriction E_l is called *minimal* if no other live-sufficient restriction of smaller size exists. A live-sufficient restriction E_l is called *locally minimal* if no strict subset of E_l is itself a live-sufficient restriction.

The restrictions E_s and E_l for Φ induce a safety assumption $\Psi_{E_s} \subseteq \mathcal{Y}_1^\omega$ and a liveness assumption $\Psi_{E_l} \subseteq \mathcal{Y}_1^\omega$ on the output behavior of the system S_1 respectively. The set Ψ_{E_s} is the set of all infinite output words $y \in \mathcal{Y}_1^\omega$ of S_1 such that there exists a run ξ with $y = \mathbf{proj}_{\mathcal{Y}_1}(\xi)$, and for all $i \geq 0$, $(x^i, y^i) \notin E_s$. The set Ψ_{E_l} is the set of all infinite output words $y \in \mathcal{Y}_1^\omega$ of S_1 such that there exists a run ξ with $y = \mathbf{proj}_{\mathcal{Y}_1}(\xi)$, and for all $(x^i, y^i) \in E_l$, either x^i appears finitely often in ξ or (x^i, w^i) appears infinitely often in ξ.

The important observations about Ψ_{E_s} and Ψ_{E_l} are summarized in the following theorem.

Theorem 6.4 *(Chatterjee et al., 2008) For a system composition $S_0 \parallel S_1$, the following assertions hold:*

1. *If S_0 can maybe-realize Φ, then there exists a unique minimal fair safe-sufficient restriction E_s on S_1.*

2. *If (a) Φ is a reachability, safety, or Büchi specification and (b) S_0 can realize the specification $\Box\langle\langle S_0, S_1\rangle\rangle\Phi$, then if there exists a sufficient assumption $\Psi \neq \emptyset$ for Φ, then there exists a live-sufficient restriction E_l on S_1.*

3. *Let $\Phi_0 \subseteq \mathcal{X}_0^\omega$ be a parity specification, and E_s, E_l be respectively the safe-sufficient and the live-sufficient restrictions on S_1 for Φ. If $\Psi = \Psi_{E_s} \cap \Psi_{E_l} \neq \emptyset$, then Ψ is a sufficient assumption for Φ.*

4. *The unique minimal fair safe-sufficient restriction E_s can be computed in polynomial time for parity objective Φ, whereas computation of a minimal live-sufficient restriction E_l is NP-hard already for Büchi specifications. There is a polynomial time algorithm for finding a locally minimal live-sufficient restriction E_l for parity specifications.*

Implementation. Consider a call to *FindAssumptions* with input S_i, A_i, G_i, and Φ_i. The procedure first checks if S_i can realize the specification $\langle A_i \rhd \Phi_i \rhd G_i\rangle$. If so, it returns the set of all output strings of C_{1-i}: all environment behaviors are allowed.

Otherwise, using Theorem 6.4, *FindAssumptions* computes *the* minimal fair safe-sufficient restriction E_s and *a* locally minimal live-sufficient restriction E_l for the winning condition $\langle \mathsf{true} \rhd \Phi_i \rhd G_i\rangle$. The reason we replaced A_i with true in this case is to avoid the trivial case when S_i realizes $\langle A_i \rhd \Phi_i \rhd G_i\rangle$ with the contradictory assumption that demands A_i be violated.

Next, *FindAssumptions* under-approximates the liveness assumption Ψ_{E_l} by a safety language $\Psi_{E_{l\to s}}$ as outlined in the following. First we introduce some notation. For the

live-sufficient restriction E_l and for a given state $x \in \mathcal{X}_i$, we use the notation $E_l(x)$ to denote the set $\{y \in \mathcal{Y}_i \mid (x, y) \in E_l\}$. We assume that the elements of the set E_l has been assigned some index, and we use the notation $E_l(x)_i$ to denote the i-th element of the set $E_l(x)$. We use the notation $|E_l(x)|$ to denote the cardinality of the set $E_l(x)$. The set $\Psi_{E_{l\to s}}$ is the set of all output words $w \in \mathcal{Y}_{1-i}^{\omega}$ produced by C_{1-i} which satisfy the following: (a) there exists a run ξ (for some set of strategies) of S_i with $w = \mathbf{proj}_{\mathcal{Y}_{1-i}}(\xi)$, and (b) for all $x \in \mathrm{dom}(E_l)$, every j-th occurrence of x in the projection $\mathbf{proj}_{\mathcal{X}_i}(\xi)$ should be immediately followed by $E_l(x)_k$ where $k = j \mod |E_l(x)|$. The procedure *FindAssumptions* returns the safety language $\Psi_{E_s} \cap \Psi_{E_{l\to s}}$.

In the following theorem, we formally state the properties of the procedure *FindAssumptions*.

Theorem 6.5 *Let the language returned by the procedure FindAssumptions*(S, A, G, Φ) *be* Ψ*. The following assertions hold:*

1. *The language* Ψ *is a sufficient assumption for the specification* $\langle A \rhd \Phi \rhd G\rangle$*.*
2. *When* $\langle \mathsf{true} \rhd \Phi \rhd G\rangle$ *is a safety language,* Ψ *is a universally maximally permissive and sufficient assumption for the specification* $\langle \mathsf{true} \rhd \Phi \rhd G\rangle$*.*

Proof (1) Suppose *FindAssumptions* returns the language $\Psi := \Psi_{E_s} \cap \Psi_{E_{l\to s}}$. Let $\Psi' := \Psi_{E_s} \cap \Psi_{E_l}$. First we show that Ψ is a sufficient assumption for the specification $\langle \mathsf{true} \rhd \Phi \rhd G\rangle$. By Thm. 6.4.3, we have that Ψ' is a sufficient assumption for realizing the specification $\langle \mathsf{true} \rhd \Phi \rhd G\rangle$. We show that $\Psi_{E_{l\to s}} \subseteq \Psi_{E_l}$, which would imply that $\Psi \subseteq \Psi'$, which in turn would establish that Ψ is also a sufficient assumption. For every valid run ξ of the underlying game graph belonging to the set $\Psi_{E_{l\to s}}$, for *every* visit of a state $s \in \mathrm{dom}(E_l)$, the environment chooses edges from E_l in a round-robin fashion. This trivially implies that the strong fairness condition in Ψ_{E_l} is satisfied by ξ, and hence $\xi \in \Psi_{E_l}$. This proves the claim $\Psi_{E_{l\to s}} \subseteq \Psi_{E_l}$, and it is established that Ψ is a sufficient assumption for $\langle \mathsf{true} \rhd \Phi \rhd G\rangle$.

Now we show that Ψ is also a sufficient assumption for the specification $\langle A \rhd \Phi \rhd G\rangle$. There are either of the following two ways that $\langle A \rhd \Phi \rhd G\rangle$ can be satisfied by the system:
(i) The specification $\Phi \wedge G \equiv \langle \mathsf{true} \rhd \Phi \rhd G\rangle$ always holds.
(ii) There exists a finite run $r \equiv x^0 \xrightarrow{w^0,u^0} x^1 \to \ldots x^k$ compliant with π, such that $\mathbf{proj}_{\mathcal{X}}(r) \in \mathrm{pref}(G)$, and $\mathbf{proj}_{\mathcal{W}^{int}}(r) \notin \mathrm{pref}(A)$.
This shows that $\langle A \rhd \Phi \rhd G\rangle$ is a weakening of the specification $\langle \mathsf{true} \rhd \Phi \rhd G\rangle$ for any $\emptyset \subsetneq A \subseteq \mathcal{W}_i^{int^{\omega}}$. Since the component can realize $\Psi \Rightarrow \langle \mathsf{true} \rhd \Phi \rhd G\rangle$, hence it can also realize $\Psi \Rightarrow \langle A \rhd \Phi \rhd G\rangle$. Thus, Ψ is indeed a sufficient assumption for the specification $\langle A \rhd \Phi \rhd G\rangle$.
(2) When $\langle \mathsf{true} \rhd \Phi \rhd G\rangle$ is a safety language, then $E_l = \emptyset$ and as a result $\Psi = \Psi_{E_s}$. The rest follows from the fact that E_s is the minimal fair safe-sufficient restriction . □

Note that, due to the non-uniqueness of the locally minimal live-sufficient restriction and the non-uniqueness of the ordering that we impose on the elements of the set $E_l(x)$ for every x, the assumption computed by *FindAssumptions* is in general not unique.

Nevertheless, owing to the finiteness of the systems there are only *finitely* many possible ways to choose both of these, and as a result the number of assumptions is also *finite*. This creates the possibility of extending our basic negotiation procedure to backtrack and retry with a different set of assumptions upon failure. Naturally, this *finitely branching negotiation* procedure will be relatively more "complete" than the non-branching procedure *Negotiate*. However, for a cleaner exposition of the main theory and as a first step, in this paper we only focus on the non-branching version with a focus on "soundness", and plan to work in future on the branching version with a focus on "relative completeness".

6.3. Implementation and Approximations

We have built a prototype C++-based tool called Agnes that implements the negotiation algorithm. Agnes is freely available at `https://github.com/kmallik/Agnes`. The name Agnes stands for **A**ssume-**G**uarantee **NE**gotiation for distributed **S**ynthesis. Agnes accepts descriptions of the systems and the local specifications in a list representation (list of states, list of transitions, etc.) given as text files. At the moment, the tool only supports safety and deterministic Büchi conditions as local specifications. If the negotiation is successful, Agnes outputs contracts where the guarantees (same as assumptions of the other systems) are modeled as finite automata, also stored using a list representation or in DOT format for visualization.

We have implemented a number of heuristics on top the general algorithm for better performance. We describe the main ones below.

6.3.1. Pattern-based Under-approximation of Assumptions

Because the construction of Thm. 6.3 depends on the sizes of these automata, we now discuss a heuristic that tries to find small automata. Observe that if C realizes $\langle A \rhd \psi \rhd G \rangle$, then it also realizes $\langle A' \rhd \psi \rhd G \rangle$ for any $A' \subseteq A$. Thus, we heuristically find stronger assumptions that can be implemented by smaller automata. The intuition behind our heuristic is that we assume bad behaviors of the environment (those that do not satisfy the safety assumption) can be identified by a set of short unsafe suffixes: as long as the environment does not produce these short suffixes, the assumption continues to hold. It is inspired by the notion of l-completeness in control (Moor et al., 2002), which abstracts a system by only tracking the states visited in the last l steps.

Our heuristic works as follows. First, we note that, because the assumption is a safety language, the *FindAssumptions* procedure returns a *universal* Büchi automaton over words for a prefix-closed ω-language. Since the language is prefix-closed, the automaton has a single rejecting sink state, and all other states are accepting. We can dualize this automaton to get a non-deterministic Büchi automaton for the negation of the language. In the negation, the sink state is the only accepting state. Let us call this automaton $\mathcal{A}$.

Given $\mathcal{A}$ and a (user-supplied) parameter k, we now construct an automaton $\mathcal{B}$ that accepts a superset of the language of $\mathcal{A}$. The automaton $\mathcal{B}$ keeps all states of $\mathcal{A}$ that have some path of length at most k to the unique accepting state and merges all

states for which the shortest path to the accepting state is greater than k. Formally, $Q_{\mathcal{B}} = \{q \in Q_{\mathcal{A}} \mid \exists \text{ path from } q \text{ to the accepting state of length} \leq k\} \cup \{r\}$, for a new state r. Consider a mapping $\lambda : Q_{\mathcal{A}} \to Q_{\mathcal{B}}$ that maps the subset $Q_{\mathcal{B}} \setminus \{r\}$ identically to itself and maps every other state to r. The transitions of $\mathcal{B}$ consist of transitions $(\lambda(q), a, \lambda(q'))$ for each transition (q, a, q') in $\mathcal{A}$. The initial and final states of $\mathcal{B}$ are the map of the initial and final states of $\mathcal{A}$ under λ.

Clearly, the number of states in $\mathcal{B}$ is less than or equal to that of $\mathcal{A}$ and $L(\mathcal{A}) \subseteq L(\mathcal{B})$. Finally, we dualize $\mathcal{B}$ to get back a universal Büchi automaton that accepts the new assumption which is contained in the original assumption.

While this heuristic that the bad environment behaviors can be identified by short suffixes might not work in general, it worked well in our examples.

6.3.2. Büchi Specifications

We now present an optimization that is orthogonal to the one presented in Sec. 6.3.1. Recall that the procedure *FindAssumptions* requires computation of locally minimal live-sufficient restriction, for which we used the algorithm proposed by Chatterjee et al. (2008). Here we present a greedy algorithm for the same purpose, but for the special case when the specification of the system $\widetilde{C}$ in Thm. 6.3 can be represented as a Büchi condition $\square\lozenge B$ for some $B \subseteq \widetilde{\mathcal{X}}$. Already for this case, computing the minimal live-sufficient restriction is NP-hard (Chatterjee et al., 2008, Thm. 11). The algorithm presented by Chatterjee et al. (2008) to compute a *locally* minimal live-sufficient restriction takes $\mathcal{O}(n^6)$ time, where n is the size of the state space. Our greedy algorithm runs in time $\mathcal{O}(n^3)$.

We introduce some notation before presenting our algorithm. Fix a composition $S_0 \parallel S_1$ and a specification $\langle A_0 \rhd \psi_0 \rhd G_0 \rangle$ for the system S_0. Consider a Büchi specification $\square\lozenge B$ for some $B \subseteq \widetilde{\mathcal{X}_0}$, where $\widetilde{\mathcal{X}_0}$ is the state space of the product system $\widetilde{C_0}$ as defined in Thm. 6.3. Let E_l be a live-sufficient restriction, and $\mathit{AssumeFair}(E_l, \square\lozenge B)$ be the set of infinite sequences of states $x_0 x_1 \ldots$ such that there exists a strategy π of S_0 and a joint strategy π' of S_1 and the environment such that $\mathbf{proj}_{\widetilde{\mathcal{X}_0}}(\xi(\pi, \pi')) = x_0 x_1 \ldots$ and $x_0 = \widetilde{x_{\mathsf{in}0}}$, and moreover either (a) there exists a pair $(x, w) \in E_l$ such that in the run $\xi(\pi, \pi')$, x appears infinitely often but after some finite time step, w never immediately follows x, or (b) $\mathbf{proj}_{\widetilde{\mathcal{X}_0}}(\xi(\pi, \pi')) \in \Phi$.

Alg. 6 uses a greedy algorithm to compute a live-sufficient restriction E_l: the algorithm progressively expands the realizable region for $\lozenge B$ by greedily adding all the favorable restrictions on S_1 to E_l whenever needed.

The heuristic does not generalize to other ω-regular specifications. Recall that in μ-calculus notation, the Büchi fixpoint is written as follows (see Sec. 2.4.4):

$$Z^* = \nu Z \,.\, \mu Y \,.\, (B \cap \mathrm{Cpre}(Z)) \cup \mathrm{Cpre}(Y),$$

where $\mathrm{Cpre}\colon 2^{\mathcal{X}} \to 2^{\mathcal{X}}$ is the controllable predecessor operator, and Z^* is the final Büchi winning region. In Alg. 6, we exploited the fact that when S_1 co-operates with S_0, Z^* is a priori known to be the set $\langle\!\langle S_0, S_1 \rangle\!\rangle \square\lozenge B$. Then the problem gets simplified to finding a locally minimal live-sufficient restriction E_l such that S_0 can (independently) realize

Algorithm 6 Compute a live-sufficient restriction E_l

Input: System S_0 in the composition $S_0 \parallel S_1$, Büchi states $B \subseteq \mathcal{X}_0$
Output: A live-sufficient restriction E_l on S_1, or **DoesNotExist**

1: **if** $x_{\mathsf{in}0} \notin \langle\langle S_0, S_1 \rangle\rangle \Box\Diamond B$ **then**
2: **return DoesNotExist**
3: **end if**
4: $\mathit{Target} \leftarrow B \cap \langle\langle S_0, S_1 \rangle\rangle \Box\Diamond B$
5: $\mathit{WinDom} \leftarrow \langle\langle S_0 \rangle\rangle \Diamond \mathit{Target}$
6: $E_l \leftarrow \emptyset$
7: **while** $\mathit{AssumeFair}(E_l, \Box\Diamond B)$ is not realizable **do**
8: $$E_l \leftarrow E_l \cup \left\{ (x,y) \in (\mathcal{X}_0 \times \mathcal{Y}_1) \;\middle|\; \begin{array}{ll} & x \notin \mathit{WinDom} \wedge \exists u \in \mathcal{U}_0 \,.\, \forall w \in \mathcal{W}_0^{ext} \,. \\ & F_0(x,(y,w),u) \in \mathit{WinDom} \\ \wedge & \exists y' \in \mathcal{Y}_1 \,.\, \forall u \in \mathcal{U}_0 \,.\, \exists w \in \mathcal{W}_0^{ext} \,. \\ & F_0(x,(y',w),u) \notin \mathit{WinDom} \end{array} \right\}$$
9: $\mathit{Target} \leftarrow \mathit{WinDom} \cup \mathrm{dom}(E_l)$
10: $\mathit{WinDom} \leftarrow \langle\langle S_0 \rangle\rangle \Diamond \mathit{Target}$
11: **end while**
12: **return** E_l

the conditional *reachability* problem $\mathit{AssumeFair}(E_l, \Diamond(\mathrm{Cpre}(Z^*) \cap B))$. We solve this problem by iterating through a growing sequence of E_l, where in each iteration we add those environment behaviors to E_l which would immediately help to grow the winning region for $\Diamond(\mathrm{Cpre}(Z^*) \cap B)$. While this heuristic works well for the Büchi specification, it will not work so well for every other ω-regular specification. Already for co-Büchi specifications, where the order of the "μ" and "ν" iterations gets reversed, this heuristic is not suitable.

6.4. Experimental Evaluation

6.4.1. Notation

Before summarizing the experimental results, let us introduce some notation. Given a system S_i and a contract $\langle A_i, G_i \rangle$, we use $|G_i|$ to represent the size of the state space of the universal Büchi automaton which accepts the language G_i. We use the parameter k to represent the user-supplied parameter used to determine the level of minimization used for minimizing the size of the contracts (see Sec. 6.3.1): we start with $k = 1$ and then keep increasing the value of k until a pair of compatible contracts is found, or until a point when we realize that increasing k any more would not change the outcome, whichever happens earlier. We use the status flag **S** and **F** to represent these two situations respectively. Indeed, it was found while inspecting the examples that the cases with status **F** do not have a contract that can be represented using a safe under-approximation of the universally maximally permissive assumptions. We put a cross mark ("×") for the entries

Table 6.1.: Experimental evaluation of the distributed packet sending problem.

l_0, p_0, t_0	l_1, p_1, t_1	$\lvert\mathcal{X}_0\rvert$	$\lvert\mathcal{X}_1\rvert$	No pattern-based opt		Pattern-based optimization (Sec. 6.3.1)									
						$k=1$		$k=2$		$k=3$		$k=4$		$k=5$	
				time(s)	status	time(s)	status	time(s)	status	time(s)	status	time(s)	status	time(s)	status
				$\lvert G_0\rvert$	$\lvert G_1\rvert$	$\lvert G_0\rvert$	$\lvert G_1\rvert$	$\lvert G_0\rvert$	$\lvert G_1\rvert$	$\lvert G_0\rvert$	$\lvert G_1\rvert$	$\lvert G_0\rvert$	$\lvert G_1\rvert$	$\lvert G_0\rvert$	$\lvert G_1\rvert$
$(1,1,1)$	$(0,1,1)$	5	3	<0.001	**F**	0.001	**F**								
				×	×	×	×								
$(1,1,2)$	$(0,1,1)$	7	3	0.001	**S**	0.001	**S**								
				2	4	2	2								
$(1,1,2)$	$(1,1,3)$	7	9	0.004	**S**	0.001	**F**	<0.001	**F**	0.003	**S**				
				5	4	2	2	2	3	4	4				
$(2,2,4)$	$(1,1,3)$	35	9	0.018	**S**	<0.001	**F**	0.001	**F**	0.010	**S**				
				5	8	2	2	2	5	4	6				
$(2,2,5)$	$(1,1,3)$	43	9	0.112	**S**	0.002	**F**	0.002	**F**	0.027	**S**				
				8	11	2	2	2	5	4	6				
$(2,2,5)$	$(2,2,5)$	43	43	0.085	**S**	0.001	**F**	0.002	**F**	0.003	**F**	0.066	**S**		
				6	11	2	2	2	5	2	6	5	11		
$(3,3,14)$	$(2,2,8)$	255	67	18.996	**S**	0.006	**F**	0.007	**F**	0.011	**F**	0.312	**S**		
				40	99	2	2	2	6	2	16	5	17		
$(4,3,14)$	$(3,2,8)$	339	99	42.254	**S**	0.007	**F**	0.010	**F**	0.023	**F**	0.027	**F**	14.967	**S**
				47	129	2	2	2	6	2	22	2	23	18	129

$|G_0|$ and $|G_1|$ in all the failed cases as they are irrelevant.

The experimental results are going to be summarized in Table 6.1 and Table 6.2. The key highlight in the tables is that the pattern-based minimization of the assumptions (see Sec. 6.3.1) turns out to be extremely beneficial while performing the negotiation. In the table, the red cells show the computation time when this optimization was disabled, whereas the blue cells show the computation time for the smallest value of k for which the negotiation was successful. It can be observed that as the systems' state spaces get larger, the saving gets higher. Also, observe the difference in the sizes of the contracts: when this optimization is disabled, the contract sizes (given by $|G_0|$ and $|G_1|$) tend to be much higher.

6.4.2. A Distributed Packet Sending Problem

Our first example is a parameterized and scaled up version of the distributed packet sending problem introduced in Sec. 1.3.3. The parameters for the system S_i with $i \in \{0, 1\}$ are given by: 1. Number of packets to be sent l_i, 2. maximum delay between two consecutive packet transmissions p_i, and 3. the overall time limit to send all the packets t_i. Essentially the only difference with the additional parameters l_i and p_i is that there are two additional counters in the state space to keep track of the respective constraints. In addition to the state *done*, there is one more special *sink* state called *excessive-delay* to mark the event that the elapsed time between two consecutive transmissions exceeded the allowed bound p_i. Then the local specification ψ_i for each system can be formalized as $\Diamond(s = \mathit{done})$. Table 6.1 summarizes the experimental results.

Table 6.2.: Experimental evaluation of the tandem queuing network example.

t_0	p, t_1	$\lvert\mathcal{X}_0\rvert$	$\lvert\mathcal{X}_1\rvert$	No pattern-based opt		Pattern-based optimization (Sec. 6.3.1)													
						$k=1$		$k=2$		$k=3$		$k=4$		$k=5$		$k=6$		$k=7$	
				time(s)	status	time(s)	status	time(s)	status	time(s)	status	time(s)	status	time(s)	status	time(s)	status	time(s)	status
				$\lvert G_0\rvert$	$\lvert G_1\rvert$	$\lvert G_0\rvert$	$\lvert G_1\rvert$	$\lvert G_0\rvert$	$\lvert G_1\rvert$	$\lvert G_0\rvert$	$\lvert G_1\rvert$	$\lvert G_0\rvert$	$\lvert G_1\rvert$	$\lvert G_0\rvert$	$\lvert G_1\rvert$	$\lvert G_0\rvert$	$\lvert G_1\rvert$	$\lvert G_0\rvert$	$\lvert G_1\rvert$
3	(1, 1)	8	4	0.055	**S**	0.002	**F**	0.002	**F**	0.006	**F**	0.055	**S**						
				15	6	2	2	2	3	5	4	15	6						
3	(2, 1)	8	6	0.073	**S**	0.002	**F**	0.002	**F**	0.008	**F**	0.076	**S**						
				17	6	2	2	2	3	5	4	17	6						
3	(2, 2)	8	8	0.008	**F**	0.027	**F**												
				×	×	×	×												
4	(2, 2)	10	8	0.262	**S**	0.003	**F**	0.003	**F**	0.006	**F**	0.031	**S**						
				32	9	2	2	2	3	2	4	7	5						
4	(3, 2)	10	10	0.019	**F**	0.079	**F**												
				×	×	×	×												
5	(3, 2)	12	8	1.076	**S**	0.003	**F**	0.003	**F**	0.007	**F**	0.013	**F**	0.065	**S**				
				62	13	2	2	2	3	2	4	2	5	9	6				
5	(3, 3)	12	12	0.030	**F**	0.109	**F**												
				×	×	×	×												
7	(3, 3)	16	12	1.560	**S**	0.004	**F**	0.004	**F**	0.008	**F**	0.010	**F**	0.013	**F**	1.456	**F**	0.150	**S**
				13	8	2	2	2	3	2	4	2	5	2	6	48	26	13	8

6.4.3. A Distributed Tandem Queuing Network Problem

Our second example is a tandem queuing network similar to the one by Hermanns et al. (2003). Suppose there is a shared queue of *bounded size*. There is a system S_0 that pushes objects to the queue from one end, and there is a system S_1 that pops an object at a time from the other end for processing. System S_0 can only sense if the queue is full or not, has two control actions *push* and $wait_0$, and produces two outputs *busy* and $idle_0$ which represent whether S_0 pushed or waited in the last cycle respectively. If the queue is full then S_0 is forced to wait until S_1 draws the next object. On the other hand, system S_1 can only sense if the queue is empty or not, has three control actions *draw*, $wait_1$, and *process*, and produces three outputs *loaded*, $idle_1$, and *processing* which represent whether S_1 drew an object, waited, or processed an object in the last clock cycle respectively. If the queue is empty, then S_1 is forced to stay *idle* until an object appears in the queue. Let p be some given positive parameter. If the queue is not empty, then S_1 can draw an object, in which case it has to perform p number of actions on the object before being able to draw another one.

Note that none of the processes has the exact information of the size of the queue. As a result they cannot certainly predict if the queue is going to be full/empty in the next step just by observing their own and the other component's output sequences. We model this uncertainty by introducing an imaginary *environment* player, who can control when the queue is full and when it is empty in an unpredictable manner.

Let t_0 and t_1 be two positive parameters. We introduce two special events for formalizing the specification: (a) System S_0 goes to the *shut-down* state if it remains idle for t_0 consecutive time steps and (b) system S_1 goes to the *cool-down* state if it remains idle for t_1 consecutive time steps. The local specifications ψ_0 and ψ_1 of S_0 and S_1 are respectively: $\psi_0 = \square\neg$*shut-down* and $\psi_1 = \square\lozenge$*cool-down*. Table 6.2 summarizes the experimental

results.

6.5. Related Work

The distributed reactive synthesis problem is known to be undecidable, thanks to the seminal paper of Pnueli and Rosner (1990). They showed that the problem is decidable with non-elementary time complexity for certain restricted architecture of the underlying interconnection topology of the systems. Later, syntactic characterizations of every decidable architecture was presented in several papers (Finkbeiner and Schewe, 2005; Kupferman and Vardi, 2001). The architecture that we consider in our work is known to be undecidable for LTL specification.

Even with a solid theoretical understanding of the problem, unfortunately, the amount of literature on distributed synthesis is quite sparse. Among notable distributed synthesis techniques are bounded synthesis (Finkbeiner and Schewe, 2013), in which the existence of distributed controllers up to a certain size is reduced to a constraint satisfaction problem in (quantified) Boolean logic, and the work by Alur et al. (2018), where distributed controllers are synthesized by solving local cooperative synthesis problems followed by conflict resolution at a global level. In contrast, our approach is modular, thanks to the modular nature of assume-guarantee proof systems.

6.6. Conclusion

We presented a sound but incomplete procedure for solving the distributed reactive synthesis problem with two systems connected in feedback. If each system can satisfy its local specification without any assumption on the other system, then we already have a solution. Otherwise, our negotiation procedure iteratively searches for a pair of assume-guarantee contracts, where each system makes progressively stricter assumption on the other system in order to fulfill the assumption imposed by the other system together with its own specification. We implemented the negotiation algorithm in our tool called `Agnes`, and show its effectiveness on a couple of examples. We only considered distributed synthesis with two systems, and leave the generalization to more number of systems for future research.

7. Symbolic Algorithms for ω-Regular Games under Strong Transition Fairness

We present an efficient symbolic algorithm for computing the sure winning region (and sure winning strategy) in 2-player fair adversarial games with Rabin winning conditions. Recall that (see Sec. 1.3.4 for an intuitive explanation) fair adversarial games are 2-player games with a set of *Player* 1 edges being designated as live edges. The transition fairness condition assumes that for every live edge, if the source vertex is visited infinitely often, then *Player* 1 will choose that edge infinitely many times. For any given ω-regular specification, the winning condition of *Player* 0 then gets modified to

$$\text{Transition Fairness Assumption} \;\Rightarrow\; \omega\text{-Regular Specification.}$$

Our algorithm for solving fair adversarial Rabin games consists of a surprisingly simple syntactic transformation of the usual symbolic fixpoint algorithms for solving regular 2-player Rabin games. Using reductions, we also obtain algorithms for various other ω-regular winning conditions. Furthermore, we also get a direct symbolic algorithm for computing almost sure winning region in $2^1/_2$-player Rabin games, that is provably faster than the state-of-the-art. We implemented our algorithm in the prototype tool called `Fairsyn`, and show that our approach is significantly faster than the existing approaches.

All the proofs of this chapter are included in App. A.

7.1. Fair Adversarial Games

Definition 7.1 (Fair Adversarial Games) A *fair adversarial game* is a pair $\langle \mathcal{G}, E^\ell \rangle$ where $\mathcal{G}$ is a 2-player game and E^ℓ is a subset of *Player* 1 edges (i.e. $E^\ell \subseteq (V_1 \times V) \cap E$) called the *live* edges.

We use the notation $V^\ell \coloneqq \mathrm{dom}(E^\ell)$ to denote the set of *Player* 1 vertices in the domain of E^ℓ.

Intuitively, the edges in E^ℓ represent *fairness assumptions* on *Player* 1: for *every* edge $(v, v') \in E^\ell$, if v is visited infinitely often along a play, we expect that the edge (v, v') is picked infinitely often by *Player* 1. I.e., if a vertex v is visited infinitely often, *every* outgoing live edge of v is expected to be taken infinitely often.

We write $\mathcal{G}^\ell = \langle \mathcal{G}, E^\ell \rangle$ to denote a game graph with live edges, and extend notions such as plays, strategies, winning conditions, winning region, etc., from game graphs to

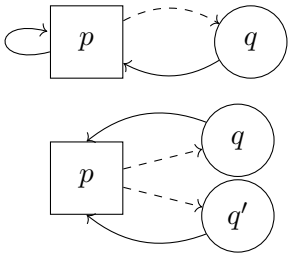


Figure 7.1.: Two fair adversarial games.

those with live edges. A play π over $\mathcal{G}^\ell$ is *strongly transition fair* if it satisfies the LTL formula:

$$\alpha \coloneqq \bigwedge_{(v,v')\in E^\ell} (\Box\Diamond v \rightarrow \Box\Diamond(v \wedge \bigcirc v')) \,. \tag{7.1}$$

Given $\mathcal{G}^\ell$ and a winning condition φ, *Player* 0 wins the *fair adversarial game* over $\mathcal{G}^\ell$ for the winning condition φ from a vertex $v^0 \in V$ if *Player* 0 wins the game over $\mathcal{G}^\ell$ for the winning condition $\alpha \rightarrow \varphi$ from v^0.

We have two interesting observations about fair adversarial games. First, live edges allow to rule out particular strategies of *Player* 1, making it easier for *Player* 0 to win in certain situations. Consider for example a game graph (Fig. 7.1 (top)) with two vertices p and q. Vertex p (square) is a *Player* 1 vertex and vertex q is a *Player* 0 vertex (circle). The edge (p, q) is a live edge (dashed). Suppose the specification for *Player* 0 is $\varphi = \Box\Diamond q$. In the absence of the live edge, *Player* 0 does not win for this specification from p, because *Player* 1 can trap the game in p by always choosing p itself as the successor. In contrast, *Player* 0 wins from p in the fair adversarial game, because the assumption on the live edge (p, q) forces *Player* 1 to infinitely often choose the transition to q.

Second, fairness assumptions modeled by live edges restrict the strategy choices of *Player* 1 less than assuming that *Player* 1 chooses probabilistically between these edges. Consider for example a fair adversarial game with one *Player* 1 vertex p (square) which has two outgoing live edges to states q and q' (see Fig. 7.1 (bottom). If *Player* 1 chooses randomly between edges (p, q) and (p, q'), *every* finite sequence of visits to states q and q' will happen infinitely often with probability one. This is not true in the fair adversarial game. Here *Player* 1 is allowed to choose a particular sequence of visits to states q and q' (e.g., only $qq'qq'qq'qq'\ldots$), as long as both are visited infinitely often.

7.2. Fair Adversarial Rabin Games

This section presents our main result on fair adversarial games, which is a symbolic fixpoint algorithm that computes the winning region of *Player* 0 in the fair adversarial game over $\mathcal{G}^\ell$ with respect to any ω-regular property formalized as a Rabin winning condition.

Our new fixpoint algorithm has multiple unique features.
(I) It works directly over $\mathcal{G}^\ell$, without requiring any pre-processing step to reduce $\mathcal{G}^\ell$ to a

"normal" two-player game. This feature allows us to obtain a direct symbolic algorithm for stochastic games as a by-product (see Sec. 7.4).
(II) Conceptually, our symbolic algorithm is *not* more complex than the known algorithm solving Rabin games over "normal" 2-player game graphs by Piterman and Pnueli (2006) (see Sec. 7.2.3).
(III) Our new fixpoint algorithm is obtained from the known algorithm of Piterman and Pnueli (2006) by a simple syntactic change (as previewed in (1.6)). We simply replace all controllable predecessor operators over least fixpoint variables by the almost sure predecessor operator invoking the preceding maximal fixpoint variable. This makes the proof of our new fixpoint algorithm conceptually simple (see Sec. 7.2.2).

At a higher level, our syntactic change is a very simple yet efficient transformation to incorporate environment assumptions expressible by live edges into reactive synthesis while retaining computational efficiency. Most remarkably, this transformation also works *directly* for fixpoint algorithms solving reachability, safety, Büchi, (generalized) co-Büchi, Rabin-chain and parity games, as these can be formalized as particular instances of a Rabin game (see Sec. 7.2.4). Moreover, it also works for generalized Büchi and GR(1) games. However, as these games are particular instances of a *generalized* Rabin game, we prove these special cases separately in Sec. 7.3 after formally introducing generalized Rabin games.

7.2.1. The Symbolic Algorithm

Monotone Set Transformers for Fair Adversarial Games: Our goal is to develop symbolic fixpoint algorithms to characterize the winning region of a fair adversarial game over a game graph with live edges. We use similar techniques as the symbolic fixpoint algorithms used for computing sure winning regions in 2-player games (see Sec. 2.4).

Recall that in Sec. 2.4, we introduced the set transformer $\mathrm{Pre}_0^\exists$ and $\mathrm{Pre}_1^\forall$ as the existential and a type of universal predecessor operator. (Since we are dealing with 2-player games, so we ignore the $\mathrm{Pre}_r^\forall$ operator for now.) Further, recall that the controllable predecessor operator *Cpre* for the 2-player games can be written as $\mathit{Cpre}(S) = \mathrm{Pre}_0^\exists(S) \cup \mathrm{Pre}_1^\forall(S)$. Now we define two additional operators which take advantage of the fairness assumption on the live edges of $\mathcal{G}^\ell$. Given two sets $S, T \subseteq V$, we define the *live-existential* and the *almost sure predecessor* operators:

$$\mathrm{Lpre}^\exists(S) \coloneqq \{v \in V^\ell \mid E^\ell(v) \cap S \neq \emptyset\}, \quad \text{and} \tag{7.2a}$$

$$\mathit{Apre}(S,T) \coloneqq \mathit{Cpre}(T) \cup \Big(\mathrm{Lpre}^\exists(T) \cap \mathrm{Pre}_1^\forall(S)\Big). \tag{7.2b}$$

Intuitively, the almost sure predecessor operator[1] $\mathit{Apre}(S,T)$ computes the set of all states that can be controlled by *Player* 0 to stay in T (via $\mathit{Cpre}(T)$) *as well as* all *Player* 1 states in V^ℓ that (a) will *eventually* make progress towards T if *Player* 1 obeys its fairness-assumptions encoded in α (via $\mathrm{Lpre}^\exists(T)$) and (b) will never leave S in the "meantime" (via $\mathrm{Pre}_1^\forall(S)$). We see that all set transformers are monotonic with respect

[1] We will justify the naming of this operator later in Rem. 4.

to set inclusion. Further, $Cpre(T) \subseteq Apre(S,T)$ always holds, $Cpre(T) = Apre(S,T)$ if $V^\ell = \emptyset$, and $Apre(S,T) \subseteq Cpre(S)$ if $T \subseteq S$; see Lem. A.1 for a proof.

Fair adversarial Rabin Games: We consider Rabin specification over the alphabet V. Recall that a Rabin winning condition is defined by the set $\mathcal{R} = \{\langle G_1, R_1\rangle, \ldots, \langle G_k, R_k\rangle\}$, where $G_i, R_i \subseteq V$ for all $i \in [1;k]$. We say that $\mathcal{R}$ has index set $P = [1;k]$. A play π satisfies the *Rabin condition* $\mathcal{R}$ if π satisfies the LTL formula

$$\varphi \coloneqq \bigvee_{i\in P} \left(\Diamond\Box \overline{R}_i \wedge \Box\Diamond G_i\right). \tag{7.3}$$

We now present our new symbolic fixpoint algorithm to compute the winning region of *Player* 0 in the fair adversarial game over $\mathcal{G}^\ell$ with respect to a Rabin winning condition $\mathcal{R}$.

Theorem 7.1 *Let $\mathcal{G}^\ell = \langle \mathcal{G}, E^\ell\rangle$ be a game graph with live edges and $\mathcal{R}$ be a Rabin condition over $\mathcal{G}$ with index set $P = [1;k]$. Further, let Z^* denote the fixed point*

$$\nu Y_{p_0}.\mu X_{p_0}. \bigcup_{p_1\in P} \nu Y_{p_1}.\mu X_{p_1}. \bigcup_{p_2\in P\setminus\{p_1\}} \nu Y_{p_2}.\mu X_{p_2}. \ldots \bigcup_{p_k\in P\setminus\{p_1,\ldots,p_{k-1}\}} \nu Y_{p_k}.\mu X_{p_k}. \left[\bigcup_{j=0}^{k} \mathcal{C}_{p_j}\right], \tag{7.4a}$$

$$\text{where} \quad \mathcal{C}_{p_j} \coloneqq \left(\bigcap\nolimits_{i=0}^{j} \overline{R}_{p_i}\right) \cap \left[\left(G_{p_j} \cap Cpre(Y_{p_j})\right) \cup \left(Apre(Y_{p_j}, X_{p_j})\right)\right], \tag{7.4b}$$

with[1] *$p_0 = 0$, $G_{p_0} \coloneqq \emptyset$ and $R_{p_0} \coloneqq \emptyset$. Then Z^* is equivalent to the winning region $\mathcal{W}$ of Player 0 in the fair adversarial game over $\mathcal{G}^\ell$ for the winning condition φ in* (7.3)*. Moreover, the fixpoint algorithm runs in $O(n^{k+2}k!)$ symbolic steps, and a memoryless winning strategy for Player 0 can be extracted from it.*

7.2.2. Proof Outline

Given a Rabin winning condition over a "normal" 2-player game, Piterman and Pnueli (2006) provided a symbolic fixpoint algorithm which computes the winning region for *Player* 0. The fixpoint algorithm in their paper is almost identical to our fixpoint algorithm in (7.4): it only differs in the last term of the constructed $\mathcal{C}$-terms in (7.4b). Piterman and Pnueli (2006) define the term $\mathcal{C}_{p_j}$ as

$$\left(\bigcap\nolimits_{i=0}^{j} \overline{R}_{p_i}\right) \cap \left[\left(G_{p_j} \cap Cpre(Y_{p_j})\right) \cup \left(Cpre(X_{p_j})\right)\right].$$

Intuitively, a single term $\mathcal{C}_{p_j}$ computes the set of states that always remain within $Q_{p_j} \coloneqq \bigcap_{i=0}^{j} \overline{R}_{p_i}$ while always re-visiting G_{p_j}. I.e, given the simpler (local) winning condition

$$\psi \coloneqq \Box Q \wedge \Box\Diamond G \tag{7.5}$$

[1] The Rabin pair $\langle G_{p_0}, R_{p_0}\rangle = \langle\emptyset,\emptyset\rangle$ in (7.4) is artificially introduced to make the fixpoint representation more compact. It is not part of $\mathcal{R}$.

for two sets $Q, G \subseteq V$, the set

$$\nu Y.\ \mu X.\ Q \cap [(G \cap \mathit{Cpre}(Y)) \cup (\mathit{Cpre}(X))] \tag{7.6}$$

is known to define exactly the states of a "normal" 2-player game $\mathcal{G}$ from which *Player* 0 has a strategy to win the game with winning condition ψ Maler et al. (1995). Such games are typically called *Safe Büchi Games*. The key insight in the proof of Thm. 7.1 is to show that the new definition of $\mathcal{C}$-terms in (7.4b) via the new *almost sure predecessor operator Apre* actually computes the winning state sets of *fair adversarial* safe Büchi games. Subsequently, we generalize this intuition to the fixpoint for the Rabin games.

Fair Adversarial Safe Büchi Games: A fair adversarial safe Büchi game is formalized in the following theorem.

Theorem 7.2 *Let $\mathcal{G}^\ell = \langle \mathcal{G}, E^\ell \rangle$ be a game graph with live edges and $Q, G \subseteq V$ be two state sets over $\mathcal{G}$. Further, let*

$$Z^* \coloneqq \nu Y.\ \mu X.\ Q \cap [(G \cap \mathit{Cpre}(Y)) \cup (\mathit{Apre}(Y, X))]\,. \tag{7.7}$$

Then Z^* is equivalent to the winning region of Player 0 in the fair adversarial game over $\mathcal{G}^\ell$ for the winning condition ψ in (7.5). Moreover, the fixpoint algorithm runs in $O(n^2)$ symbolic steps, and a memoryless winning strategy for Player 0 can be extracted from it.

Intuitively, the fixed points in (7.6) and (7.7) consist of two parts: (a) A minimal fixed point over X which computes (for any fixed value of Y) the set of states that can *reach* the "target state set" $T \coloneqq Q \cap G \cap \mathit{Cpre}(Y)$ while staying inside the safe set Q, and (b) a maximal fixed point over Y which ensures that the only states considered in the target T are those that allow to re-visit a state in T while staying in Q.

By comparing (7.6) and (7.7) we see that our syntactic transformation only changes part (a). Hence, in order to prove Thm. 7.2 it essentially remains to show that this transformation works for the even simpler *safe reachability games*.

Fair Adversarial Safe Reachability Games: A safe reachability condition is a tuple $\langle T, Q \rangle$ with $T, Q \subseteq V$ and a play π satisfies the *safe reachability condition* $\langle T, Q \rangle$ if π satisfies the LTL formula

$$\psi \coloneqq Q\,\mathcal{U}\,T. \tag{7.8}$$

A safe reachability game is often called a *reach-while-avoid* game, where the safe sets are specified by an unsafe set $R \coloneqq \overline{Q}$ that needs to be avoided. Their fair adversarial version is formalized in the following theorem, proved in App. A.2.2.

Theorem 7.3 *Let $\mathcal{G}^\ell = \langle \mathcal{G}, E^\ell \rangle$ be a game graph with live edges and $\langle T, Q \rangle$ be a safe reachability winning condition. Further, let*

$$Z^* \coloneqq \nu Y.\ \mu X.\ T \cup (Q \cap \mathit{Apre}(Y, X)). \tag{7.9}$$

Then Z^* is equivalent to the winning region of Player 0 in the fair adversarial game over $\mathcal{G}^\ell$ for the winning condition ψ in (7.8). Moreover, the fixpoint algorithm runs in $O(n^2)$ symbolic steps, and a memoryless winning strategy for Player 0 can be extracted from it.

To gain some intuition on the correctness of Thm. 7.3, let us recall that the fixed-point for safe reachability games *without* live edges is given by:

$$\mu X.\ T \cup (Q \cap \mathit{Cpre}(X)). \tag{7.10}$$

Intuitively, the fixed point in (7.10) is initialized with $X^0 = \emptyset$ and computes a sequence $X^0, X^1, \ldots, X^k$ of increasing sets until $X^k = X^{k+1}$. We say that v has rank r if $v \in X^r \setminus X^{r-1}$. All states contained in X^r allow *Player* 0 to force the play to reach T in at most $r-1$ steps while staying in Q. The corresponding *Player* 0 strategy ρ_0 is known to be winning w.r.t. (7.8) and along every play π compliant with ρ_0, the path π remains in Q and the rank is always decreasing.

To see why the same strategy is also *sound* in the *fair adversarial* safe reachability game $\mathcal{G}^\ell$, first recall that for vertices $v \notin V^\ell$ of $\mathcal{G}^\ell$, the almost sure pre-operator $\mathit{Apre}(X, Y)$ simplifies to $\mathit{Cpre}(X)$. With this, we see that for every $v \notin V^\ell$ a *Player* 0 winning strategy $\widetilde{\rho}_0$ in $\mathcal{G}^\ell$ can always force plays to stay in Q and to decrease their rank, similar to ρ_0. With this, we see that plays π which are compliant with such a strategy $\widetilde{\rho}_0$ and visit a vertex in V^ℓ only finitely often satisfy (7.8).

The only interesting case for soundness of Thm. 7.3 are therefore plays π that visits states in V^ℓ infinitely often. However, as the number of vertices is finite, we only have a finite number of ranks and hence a certain vertex $v \in V^\ell$ with a finite rank r needs to get visited by π infinitely often. Due to the definition of *Apre* we however know that only states $v \in V^\ell$ are contained in X^r if v has an outgoing *live* edge reaching X^k with $k < r$. With this, reaching v infinitely often implies that also a state with rank k s.t. $k < r$ will get visited infinitely often. As $X^1 = T$ we can show by induction that T is eventually visited along π while π always remains in Q until then.

In order to prove *completeness* of Thm. 7.3 we need to show that all states in $V \setminus Z^*$ are loosing for Player 0. Here, again the reasoning is equivalent to the "normal" safe reachability game for $v \notin V^\ell$. For vertices $v \in V^\ell$, we see that v is not added to Z^* via *Apre* if $v \notin T$ and either (i) all its outgoing live transitions do not make progress towards T or, (ii) it has some outgoing edge (not necessarily a live one) that makes it leave Z^*). One can therefore construct a *Player* 1 strategy that for (i)-vertices always chooses a live transition and thereby never makes progress towards T (also if v is visited infinitely often), and for (ii)-vertices ensures that they are only visited once on plays which remain in Q. This ensures that (ii)-vertices never make progress towards T via their possibly existing rank-decreasing live edges.

A detailed soundness and completeness proof of Thm. 7.3 along with the respective *Player* 0 and *Player* 1 strategy construction is provided in App. A.2.2. In addition, Thm. 7.2 is proven in App. A.2.2 by a reduction to Thm. 7.3 for every iteration over Y.

Example 7.1 (Fair adversarial safe reachability game) We consider a fair adversarial safe reachability game over the game graph depicted in Fig. 7.2 with target vertex set $T = G = \{6, 9\}$ and safe vertex set $Q = V \setminus \{1\}$.

We denote by Y^m the m-th iteration over the fixpoint variable Y in (7.9), where $Y^0 = V$. Further, we denote by X^{mi} the set computed in the i-th iteration over the

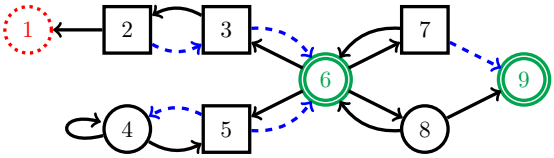


Figure 7.2.: Fair adversarial game graph discussed in Examples 7.1 and 7.2 with vertex sets $G = \{6, 9\}$ (double cycled, green), $\overline{Q} = \{1\}$ (red,dotted) and live edges $E^{\ell} = \{(2,3),(3,6),(5,4),(5,6),(7,9))\}$ (dashed, blue). *Player* 0 and *Player* 1 vertices are indicated by cycles and boxes, respectively.

fixpoint variable X in (7.9) during the computation of Y^m where $X^{m0} = \emptyset$. We further have $X^{m1} = T = \{6, 9\}$ as $Apre(\cdot, \emptyset) = \emptyset$. Now we compute

$$\begin{aligned} X^{12} &= T \cup (Q \cap Apre(Y^0, X^{11})) \\ &= \{6,9\} \cup (V \setminus \{1\} \cap [\underbrace{Cpre(X^{11})}_{\{8\}} \cup \underbrace{(\mathrm{Lpre}^{\exists}(X^{11}) \cap \mathrm{Pre}_1^{\forall}(V))}_{\{3,5,7\}}]) = \{5,6,7,8,9\} \end{aligned} \tag{7.11}$$

We observe that the only vertex added to X via the *Cpre* term is vertex 8. States $\{3, 5, 7\}$ are added due to the existing live edge leading to a target vertex. Here, we note that vertex 7 is added due to its live edge to vertex 9. The additional requirement $\mathrm{Pre}_1^{\forall}(V)$ in $Apre(Y^0, X^{11})$ is trivially satisfied for all vertices at this point as $Y^0 = V$ and can therefore be ignored. Doing one more iteration over X we see that now vertex 4 gets added via the *Cpre* term (as it is a *Player* 0 vertex that allows progress towards 5) and vertex 2 is added via the *Apre* term (as it allows progress to 3 via a live edge). The iteration over X terminates with $Y^1 = X^{1*} = V \setminus \{1\}$.

Re-iterating over X for Y^1 gives $X^{22} = X^{12} = \{5, 6, 7, 8, 9\}$ as before. However, now vertex 2 does not get added to X^{23} because vertex 2 has an edge leading to $V \setminus Y^1 = \{1\}$. Therefore the iteration over X terminates with $Y^2 = X^{2*} = V \setminus \{1, 2\}$. When we now re-iterate over X for Y^2 we see that vertex 3 is not added to X^{32} any more, as vertex 3 has a transition to $V \setminus Y^2 = \{1, 2\}$. Therefore the iteration over X now terminates with $Y^3 = X^{3*} = V \setminus \{1, 2, 3\}$. Now re-iterating over X does not change the vertex set anymore and the fixed-point terminates with $Y^* = Y^3 = V \setminus \{1, 2, 3\}$.

We note that the fixed-point formula (7.10) for "normal" safe reachability games terminates after two iterations over X with $X^* = \{6, 8, 9\}$, as vertex 8 is the only vertex added via the *Cpre* operator in (7.11). Due to the stricter notion of *Cpre* requiring that *all* outgoing edges of *Player* 0 vertices make process towards the target, (7.10) does not require an outer largest fixed-point over Y to "trap" the play in a set of vertices which allow progress when "waiting long enough". This "trapping" required in (7.9) via the outer fixed-point over Y actually fails for vertices 2 and 3 (as they are excluded form the winning set of (7.9)). Here, *Player* 1 can enforce to "escape" to the unsafe vertex 1 in two steps before 2 and 3 are visited infinitely often (which would imply progress towards 6 via the existing live edges).

We see that the winning region in the "normal" game is significantly smaller than

the winning region for the fair adversarial game, as adding live transitions restricts the strategy choices of *Player* 1, making it easier for *Player* 0 to win the game.

Example 7.2 (Fair adversarial safe Büchi game) We now consider a fair adversarial safe Büchi game over the game graph depicted in Fig. 7.2 with sets $G = \{6, 9\}$ and $Q = V \setminus \{1\}$.

We first observe that we can rewrite the fixed-point in (7.7) as

$$\nu Y.\ \mu X.\ [Q \cap G \cap \mathit{Cpre}(Y)] \cup [Q \cap (\mathit{Apre}(Y, X))]\,. \tag{7.12}$$

Using (7.12) we see that for $Y^0 = V$ we can define $T^0 := Q \cap G \cap \mathit{Cpre}(V) = G = \{6, 9\}$. Therefore the first iteration over X is equivalent to (7.11) and terminates with $Y^1 = X^{1*} = V \setminus \{1\}$.

Now, however, we need to re-compute T for the next iteration over X and obtain $T^1 = Q \cap G \cap \mathit{Cpre}(Y^1) = V \setminus \{1\} \cap \{6, 9\} \cap V \setminus \{1, 2, 9\} = \{6\}$. This re-computation of T^1 checks which target vertices are re-reachable, as required by the Büchi condition. As vertex 9 has no outgoing edge it is trivially not re-reachable.

With this, we see that for the next iteration over X we only have one target vertex $T^1 = \{6\}$. If we recall that vertex 7 is added to X^{22} due to its live edge to 9, we see that it is now not added anymore. Intuitively, we have to exclude 7 as *Player* 1 can always decide to take the live edge towards 9 from 7 (also if 7 only gets visited once), and therefore prevents to re-visit a target state.

Now, vertices 2 and 3 get eliminated for the same reason as in the safe reachability game within the second and third iteration over Y. The overall fixed-point computation therefore terminates with $Y^* = Y^3 = \{4, 5, 6, 8\}$.

Proof of Thm. 7.1: With Thm. 7.3 and Thm. 7.2 in place, the proof of Thm. 7.1 is essentially equivalent to the proof of Piterman and Pnueli (2006) while utilizing Thm. 7.3 and Thm. 7.2 at all suitable places. For completeness, we give the full proof of Thm. 7.1, including the memoryless strategy construction, in App. A.2.3. In addition, we illustrate the steps of the fixed-point algorithm in (7.4) with a simple fair adversarial Rabin game (depicted in Fig. A.1 in App. A.1) which has two acceptance pairs in App. A.1.

Remark 4 *We remark that the fixpoint* (7.9)*, as well as the Apre operator, are similar in structure to the solution of almost surely winning states in concurrent reachability games de Alfaro et al. (1998). In concurrent games, the fixed point captures the largest set of states in which the game can be trapped while maintaining a positive probability of reaching the target. In our case, the fixed point captures the largest set of states in which Player* 0 *can keep the game while ensuring a visit to the target either directly or through the live edges. The commonality justifies our notation and terminology for Apre.*

7.2.3. Complexity

Complexity Analysis of (7.4)**:** For Rabin games with k Rabin pairs, Piterman and Pnueli (2006) show a fixpoint formula with alternation depth $2k+1$. Using the accelerated

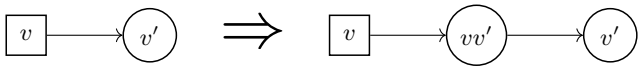


Figure 7.3.: Left: A live edge (v, v') in $\mathcal{G}^\ell$. Right: The gadget used to replace (v, v') in $\widehat{\mathcal{G}}$. The vertex named vv' is a newly added vertex in $\widehat{\mathcal{G}}$; v belongs to $\widehat{V}_1$, vv' belongs to $\widehat{V}_0$, but v' may belong to either $\widehat{V}_0$ or $\widehat{V}_1$.

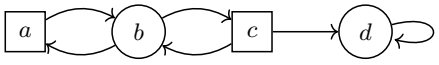


Figure 7.4.: Counterexample to the equality of strong transition fairness and strong fairness (compassion).

fixpoint computation technique of Long et al. (1994), they deduce a bound of $O(n^{k+1}k!)$ symbolic steps. We show in App. A.3 that this accelerated fixpoint computation can also be applied to (7.4) yielding a bound of $O(n^{k+2}k!)$ symbolic steps. (The additional complexity is because of an additional outermost ν-fixpoint.) Thus our algorithm is almost as efficient as the original algorithm for Rabin games without environment assumptions—*independent* of the number of strong transition fairness assumptions!

Comparison with a Naïve Solution: We show a naïve reduction from fair adversarial Rabin games to usual Rabin games. Suppose $\mathcal{G}^\ell = \langle \mathcal{G}, E^\ell \rangle$ is a game graph with live edges, $\mathcal{R} = \{\langle G_1, R_1 \rangle, \ldots, \langle G_k, R_k \rangle\}$ is a Rabin winning condition defined over $\mathcal{G}^\ell$, and φ is the corresponding LTL specification as defined in (7.3). Let $\widehat{\mathcal{G}} = \left\langle \widehat{V}, \widehat{V}_0, \widehat{V}_1, \widehat{E} \right\rangle$ be a 2-player game graph obtained by just replacing every live edge of $\mathcal{G}^\ell$ with a gadget shown in Fig. 7.3 and explained next. For every live edge $(v, v') \in E^\ell$ we introduce a new intermediate vertex named $vv' \in \widehat{V}$, and without loss of generality we assume that $vv' \in \widehat{V}_0$. (We could have equivalently used the convention that $vv' \in \widehat{V}_1$.) Then we replace the edge (v, v') with a pair of new edges $(v, vv') \in \widehat{E}$ and $(vv', v') \in \widehat{E}$; the rest remains the same as in $\mathcal{G}$. Assuming that $|E^\ell| = l$ and $|V| = n$, the number of vertices of $\widehat{\mathcal{G}}$ is $n + l$.

Intuitively, the event of the newly introduced vertices being reached in $\widehat{\mathcal{G}}$ simulates the event of the corresponding live edge being taken in $\mathcal{G}^\ell$, and vice versa. We are now ready to transfer the specification $\alpha \to \varphi$ to a new Rabin winning condition $\widehat{\mathcal{R}}$ for $\widehat{\mathcal{G}}$. First observe that $\alpha \to \varphi$ is equivalent to $\neg\alpha \vee \varphi$, and $\neg\alpha$ can be expressed in LTL as $\bigvee_{(v,v')\in E^\ell}(\Box\Diamond\{v\} \wedge \Diamond\Box\overline{\{vv'\}})$. and is therefore equivalent to the Rabin winning condition $\mathcal{R}^\ell \coloneqq \{\langle \{v\}, \{vv'\} \rangle \mid (v, v') \in E^\ell\}$. Since Rabin winning conditions are closed under union, we obtain the new Rabin condition $\widehat{\mathcal{R}} \coloneqq \mathcal{R} \cup \mathcal{R}^\ell$.

Once $\widehat{\mathcal{G}}$ and $\widehat{\mathcal{R}}$ are obtained, one can use the fixpoint algorithm of Piterman and Pnueli (2006) for "normal" 2-player Rabin games. This whole process yields a symbolic algorithm for fair adversarial Rabin games with $2(k + l) + 1$ alternations of fixpoint operators on a set of $(n + l)$ vertices that runs in time $O((n + l)^{k+l+1}(k + l)!)$. In contrast, our main theorem shows that we get a symbolic fixpoint expression with $2(k + 1)$ alternations that runs in $O(n^{k+2}k!)$ symbolic steps. In many applications, we expect $l = \Theta(n)$, for which our algorithm is significantly faster.

Remark As already mentioned in the introduction, not all strong fairness assumptions

(Streett assumptions) can be translated into live edges (see e.g., (Baier and Katoen, 2008, p.264)). As an example, consider the 2-player game graph depicted in Fig. 7.4. *Player* 0 and *Player* 1 vertices are indicated by a circle and a box, respectively. Now consider the following one-pair Streett assumption

$$\varphi_A \coloneqq \Box\Diamond\{a,b,c\} \rightarrow \Box\Diamond\{a\} = \Diamond\Box\{d\} \vee \Box\Diamond\{a\}. \tag{7.13}$$

This fairness assumption states that it is not possible for a game to infinitely stay inside the set $\{a,b,c\}$ if *Player* 0 decides to not transition from b to a anymore from some point onward. We see that we cannot model this behavior by a fair edge leaving a *Player* 1 (square) state. If we mark the edge (c,d) live, any fair play will transition to d no matter if a is visited infinitely often or not. Let us call this fair edge assumption α_A. Then we see that $\alpha_A \rightarrow \varphi_A$ but not vice versa.

7.2.4. Specialized Rabin Games

This section shows that the known fixpoint algorithms for Rabin chain, Parity, and Generalized Co-Büchi winning conditions allow for the same "syntactic transfomation" as in the Rabin case to get the right algoirthm for their fair adversarial version. We prove these claims by reducing the fixed point in (7.4) to the special cases induced by the aforementioned winning conditions.

We note that the fixpoint algorithm for fair adversarial Rabin games in (7.4) reduces to the normal fixed point for Rabin games if $E^\ell = \emptyset$. Therefore, our reductions of (7.4) to fixpoint algorithms for other winning conditions also proves these reductions in the usual case. We are not aware of such reductions proved elsewhere in the literature.

Fair Adversarial Rabin Chain Games: A *Rabin chain* winning condition Mostowski (1984) is a *Rabin* condition $\mathcal{R} = \{\langle G_1, R_1\rangle, \ldots, \langle G_k, R_k\rangle\}$, with the additional *chain condition*

$$R_1 \supseteq R_2 \supseteq \ldots \supseteq R_k \qquad \text{and} \qquad G_1 \supseteq G_2 \supseteq \ldots \supseteq G_k. \tag{7.14}$$

Intuitively, the fixpoint algorithm computing Z^* in (7.4) simplifies to a single permutation sequence, namely $p_1 = k$, $p_2 = k-1$, …, $p_k = 1$, if (7.14) holds. This is formalized in the following theorem which is proved in App. A.2.4.

Theorem 7.4 *Let $\mathcal{G}^\ell = \langle \mathcal{G}, E^\ell \rangle$ be a game graph with live edges and $\mathcal{R}$ be a Rabin condition over $\mathcal{G}$ with k pairs for which the chain condition* (7.14) *holds. Further, let*

$$Z^* \coloneqq \nu Y_0.\ \mu X_0.\ \nu Y_k.\ \mu X_k.\ \nu Y_{k-1}.\ \ldots \mu X_1.\ \textstyle\bigcup_{j=0}^{k} \widetilde{\mathcal{C}}_j, \tag{7.15a}$$

$$\text{where} \quad \widetilde{\mathcal{C}}_j \coloneqq \overline{R}_j \cap [(G_j \cap \mathit{Cpre}(Y_j)) \cup \mathit{Apre}(Y_j, X_j)] \tag{7.15b}$$

with $G_{p_0} \coloneqq \emptyset$ and $R_{p_0} \coloneqq \emptyset$. Then Z^* is equivalent to the winning region $\mathcal{W}$ of Player 0 *in the fair adversarial game over $\mathcal{G}^\ell$ for the winning condition φ in* (7.3)*. Moreover, the fixpoint algorithm runs in $O(n^{k+2})$ symbolic steps, and a memoryless winning strategy for Player* 0 *can be extracted from it.*

Fair Adversarial Parity Games: A *Parity* winning condition (Emerson and Jutla, 1989) is defined by a set $\mathcal{P} = \{C_1, C_2, \dots C_{2k}\}$ of colors, where each $C_i \subseteq V$ is the set of vertices of $\mathcal{G}$ with color i. Further, $\mathcal{P}$ partitions the state space, i.e., $\bigcup_{i\in[1;2k]} C_i = V$ and $C_i \cap C_j = \emptyset$ for all $i, j \in [1; 2k]$ with $i \neq j$. A play π satisfies the *Parity condition* $\mathcal{P}$ if π satisfies the LTL formula

$$\varphi \coloneqq \bigwedge_{i\in[1;k]} \left(\Box\Diamond C_{2i-1} \rightarrow \bigvee_{j\in[i;k]} \Box\Diamond C_{2j}\right). \tag{7.16}$$

That is, the maximal color visited infinitely often along π is even. A Parity winning condition $\mathcal{P}$ with $2k$ colors corresponds to the Rabin chain winning condition

$$\{\langle F_2, F_3\rangle, \dots, \langle F_{2k}, \emptyset\rangle\} \qquad \text{s.t. } F_i \coloneqq \bigcup_{j=i}^{2k} C_j, \tag{7.17}$$

which has k pairs. Due to $\mathcal{P}$ forming a partition of the state space one can further simplify the Rabin chain fixpoint algorithm in (7.15). Interestingly, the resulting fixpoint looks slightly different from the one we would obtain by a straightforward application of our syntactic transformation. While the usual fixpoint algorithm for parity games is as in the following

$$\begin{aligned} Z^* \coloneqq &\nu Y_{2k}.\ \mu X_{2k-1} \dots \nu Y_2.\ \mu X_1. \\ &(C_1 \cap \mathit{Cpre}(X_1)) \cup (C_2 \cap \mathit{Cpre}(Y_2)) \cup (C_3 \cap \mathit{Cpre}(X_3)) \dots \cup (C_{2k} \cap \mathit{Cpre}(Y_{2k})), \end{aligned} \tag{7.18}$$

the fixpoint algorithm for fair adversarial parity games looks slightly different:

Theorem 7.5 *Let $\mathcal{G}^\ell = \langle \mathcal{G}, E^\ell \rangle$ be a game graph with live edges and $\mathcal{P}$ be a Parity condition over $\mathcal{G}$ with $2k$ colors. Further, let*

$$\begin{aligned} Z^* \coloneqq &\nu Y_{2k}.\ \mu X_{2k-1} \dots \nu Y_2.\ \mu X_1. \\ &\cup (C_{2k} \cap \mathit{Cpre}(Y_{2k})) \cup ((C_1 \cup \dots \cup C_{2k-1}) \cap \mathit{Apre}(Y_{2k}, X_{2k-1})) \\ &\cup \ \dots \\ &\cup (C_4 \cap \mathit{Cpre}(Y_4)) \cup ((C_1 \cup C_2 \cup C_3) \cap \mathit{Apre}(Y_4, X_3)) \\ &\cup (C_2 \cap \mathit{Cpre}(Y_2)) \cup (C_1 \cap \mathit{Apre}(Y_2, X_1)). \end{aligned} \tag{7.19}$$

Then Z^* is equivalent to the winning region $\mathcal{W}$ of Player 0 in the fair adversarial game over $\mathcal{G}^\ell$ for the winning condition φ in (7.16). Moreover, the fixpoint algorithm runs in $O(n^{k+1})$ symbolic steps, and a memoryless winning strategy for Player 0 can be extracted from it.

Apart from the usual differences between (7.18) and (7.19), introduced by our syntactic transformation, the main difference is that every $\mathit{Apre}(Y_{2i}, X_{2i-1})$ in (7.19) is intersected with the union of *every* vertex with color $2i-1$ or smaller, whereas the $\mathit{Apre}(Y_{2i}, X_{2i-1})$-s in (7.18) are intersected with *only* vertices with color $2i-1$. The intuitive explanation is that fair adversarial games allow *Player* 0 more freedom in visiting vertices which do not directly contribute to winning as long as they are only transient. It turns out that it is necessary to allow transitions from the vertices with lower colors through $\mathit{Apre}(\cdot,\cdot)$, as

otherwise the fixpoint will not capture all the vertices from where it is possible to win the fair adversarial parity game.

Fair Adversarial (Generalized) Co-Büchi Games: A *Co-Büchi* winning condition is defined by a subset $A \subseteq V$ of vertices of $\mathcal{G}$. A play π satisfies the *Co-Büchi condition* A if π satisfies

$$\varphi \coloneqq \Diamond\Box A. \tag{7.20}$$

A *Generalized Co-Büchi* winning condition is defined by a set $\mathcal{A} = \{A_1, \dots A_r\}$, where each $A_i \subseteq V$ is a subset of vertices of $\mathcal{G}$. A play π satisfies the *Generalized Co-Büchi condition* $\mathcal{A}$ if π satisfies

$$\varphi \coloneqq \bigvee_{a\in[1;r]} \Diamond\Box A_a. \tag{7.21}$$

Generalized Co-Büchi winning conditions correspond to a Rabin condition $\mathcal{R}$ with r pairs s.t.

$$\forall j \in [1;r] \,.\, R_j \coloneqq \overline{A}_j \quad \text{and} \quad G_j \coloneqq V. \tag{7.22}$$

Intuitively, the fact that $G_j \coloneqq V$ for all j leads to a cancelation of all *Apre* terms in $\mathcal{C}_j$ and all terms become ordered, i.e., we have $\mathcal{C}_{p_{j+1}} \subseteq \mathcal{C}_{p_j}$ for every permutation sequence used in (7.4). As we take the union over all $\mathcal{C}_{p_j}$-s in (7.4a), the term $\mathcal{C}_{p_1}$ absorbs all others for every permutation sequence. Hence, for every permutation sequence we only have two terms left, one for $j = 0$ (over the artificially introduced Rabin pairs $G_{p_0} = R_{p_0} = \emptyset$) and one for the first choice p_1 made in this particular permutation. This is formalized in the following theorem which is proved in App. A.2.4.

Theorem 7.6 *Let $\mathcal{G}^\ell = \langle \mathcal{G}, E^\ell \rangle$ be a game graph with live edges and $\mathcal{A}$ be a generalized Co-Büchi winning condition $\mathcal{G}$ with r pairs. Further, let*

$$Z^* \coloneqq \nu Y_0.\ \mu X_0.\ \bigcup_{a\in[1;r]} \nu Y_a.\ \mathit{Apre}(Y_0, X_0) \cup (\overline{A}_a \cap \mathit{Cpre}(Y_a)). \tag{7.23}$$

Then Z^* is equivalent to the winning region $\mathcal{W}$ of Player 0 in the fair adversarial game over $\mathcal{G}^\ell$ for the winning condition φ in (7.21). Moreover, the fixpoint algorithm runs in $O(rn^2)$ symbolic steps, and a memoryless winning strategy for Player 0 can be extracted from it.

7.3. Generalized Rabin Games

In this section, we slightly generalize our main result, Thm. 7.1, to fair adversarial *generalized* Rabin games. That is, for each Rabin pair, we allow the goal set G_i to be a set of goal sets $G_j = \{\ {}^1G_j, \dots,\ {}^{m_j}G_j\}$. Then a play fulfills the winning condition if there exists one generalized Rabin pair $\langle G_i, R_i\rangle$ such that the play eventually remains in $\overline{R}_i$ and visits *all* sets ${}^lG_i$ infinitely often.

The motivation of this generalization is to show that our syntactic transformation also works for fair adversarial games with a *generalized reactivity winning condition of rank* 1 (GR(1) games for short) Piterman et al. (2006). *Generalized* Rabin games allow us to see a GR(1) winning condition as a particularly simple instantiation of a Rabin game as shown in Sec. 7.3.2.

7.3.1. Fair Adversarial Generalized Rabin Games

Generalized Rabin Conditions: A *generalized Rabin condition* is defined by a set $\widetilde{\mathcal{R}} = \{\langle G_1, R_1\rangle, \ldots, \langle G_k, R_k\rangle\}$ where each $G_j = \{\ {}^1G_j, \ldots,\ {}^{m_j}G_j\}$ is a finite set s.t. ${}^lG_j \subseteq V$ for all $j \in [1;k]$ and all $l \in [1;m_j]$. We say that $\widetilde{\mathcal{R}}$ has global index set $P = [1;k]$. A play π satisfies the *generalized Rabin condition* $\widetilde{\mathcal{R}}$ if π satisfies the LTL formula

$$\varphi := \bigvee_{j\in P} \left(\Diamond\Box\overline{R}_j \wedge \bigwedge_{l\in[1;m_j]} \Box\Diamond\ {}^lG_j\right). \tag{7.24}$$

Recalling the discussion of Sec. 7.2.1, we know that the proof of Thm. 7.1 fundamentally relies on the correctness of our transformation for safe Büchi (Thm. 7.2) and safe reachability (Thm. 7.3) games. Similarly, one needs to prove correctness of our syntactic transformation for safe *generalized* Büchi games in the case of generalized Rabin games.

Safe Generalized Büchi Games A *safe generalized Büchi condition* is defined by a tuple $\langle F^c, Q\rangle$ where $Q \subseteq V$ is a set of safe states and $\mathcal{F} = \{\ {}^1F, \ldots,\ {}^sF\}$ is a set of goal sets. A play π satisfies the *safe generalized Büchi condition* $\langle F^c, Q\rangle$ if π satisfies the LTL formula

$$\varphi := \Box Q \wedge \bigwedge_{l\in[1;s]} \Box\Diamond\ {}^lF. \tag{7.25}$$

Now we can apply our syntactic transformation to the usual fixpoint algorithm for solving safe generalized Büchi games and prove its correctness for all fair adversarial plays. This is formalized in the next theorem and proved in App. A.2.5.

Theorem 7.7 *Let $\mathcal{G}^\ell = \langle \mathcal{G}, E^\ell\rangle$ be a game graph with live edges and $\langle F^c, Q\rangle$ with $\mathcal{F} = \{\ {}^1F, \ldots,\ {}^sF\}$ a safe generalized Büchi winning condition. Further, let*

$$Z^* := \nu Y.\ \bigcap_{b\in[1;s]} \mu\ {}^bX.\ Q \cap \left[(\ {}^bF \cap \mathit{Cpre}(Y)) \cup \mathit{Apre}(Y,\ {}^bX)\right]. \tag{7.26}$$

Then Z^* is equivalent to the winning region $\mathcal{W}$ of Player 0 in the fair adversarial game over $\mathcal{G}^\ell$ for the winning condition φ in (7.25). Moreover, the fixpoint algorithm runs in $O(sn^2)$ symbolic steps, and a finite-memory winning strategy for Player 0 can be extracted from it.

Intuitively, the proof of Thm. 7.7 reduces to Thm. 7.2 in a similar manner as the proof of Thm. 7.2 reduces to Thm. 7.3. However, the challenge in proving Thm. 7.7 is to show that it is indeed sound to use the fixpoint variable Y which is actually the intersection of fixpoint variables X both within *Cpre* and *Apre*. The proof of this correctness essentially requires to show that upon termination we have $Y^* = {}^bX^*$ for all $b \in [1;s]$ (see App. A.2.5 for a formal proof).

The Symbolic Algorithm: By knowing that (7.26) allows to correctly solve safe generalized Büchi games, we can immediately generalize this observation to Rabin games. This is formalized in the following theorem which is an immediate consequence of Thm. 7.1 and Thm. 7.7.

Theorem 7.8 *Let $\mathcal{G}^\ell = \langle \mathcal{G}, E^\ell \rangle$ be a game graph with live edges and $\widetilde{\mathcal{R}}$ be a generalized Rabin condition over $\mathcal{G}$ with index set $P = [1;k]$. Further, let*

$$Z^* := \nu Y_0.\ \mu X_0.\ \bigcup_{p_1 \in P} \nu Y_{p_1}.\ \bigcap_{l_1 \in [1;m_{p_1}]} \mu\ {}^{l_1}X_{p_1}.\ \ldots\ldots$$

$$\bigcup_{p_k \in P \setminus \{p_1,\ldots,p_{k-1}\}} \nu Y_{p_k}.\ \bigcap_{l_k \in [1;m_{p_k}]} \mu\ {}^{l_k}X_{p_k}.\ \bigcup_{j=0}^{k} {}^{l_j}\mathcal{C}_{p_j}, \tag{7.27a}$$

$$\textit{where} \qquad {}^{l_j}\mathcal{C}_{p_j} := \Big(\textstyle\bigcap_{i=0}^{j} \overline{R}_{p_i}\Big) \cap \Big[\Big(\ {}^{l_j}G_{p_j} \cap \mathit{Cpre}(Y_{p_j})\Big) \cup \mathit{Apre}(Y_{p_j},\ {}^{l_j}X_{p_j})\Big] \tag{7.27b}$$

with[1] $p_0 = 0$, $G_{p_0} := \{\emptyset\}$ and $R_{p_0} := \emptyset$. Then Z^* is equivalent to the winning region $\mathcal{W}$ of Player 0 in the fair adversarial game over $\mathcal{G}^\ell$ for the winning condition φ in (7.24). Moreover, the fixpoint algorithm runs in $O(n^{k+2}k!m_1 \ldots m_k)$ symbolic steps, and yields a finite-memory winning strategy for Player 0.

The proof of Thm. 7.8 is almost identical to the proof of Thm. 7.1 in App. A.2.3, when using Thm. 7.7 instead of Thm. 7.2 in all appropriate places. This, yields a finite memory winning strategy by suitably "stacking" the individual finite-memory strategies constructed in the proof of Thm. 7.7. (See App. A.2.5 for a complete proof of Thm. 7.8.)

7.3.2. Fair Adversarial GR(1) Games

Within this section, we show how fair adversarial Rabin games can be reduced to fair adversarial games with GR(1) winning conditions.

GR(1) winning condition: A GR(1) winning condition is defined by two sets $\mathcal{A} = \{A_1, \ldots, A_r\}$ and $\mathcal{F} = \{F_1, \ldots, F_s\}$, where for every $i \in [1;r]$ and $j \in [1;s]$, $A_i, F_j \subseteq V$. A play π satisfies the GR(1) condition $(\mathcal{A}, \mathcal{F})$ if it satisfies the LTL formula

$$\varphi := \Big(\textstyle\bigwedge_{a \in [1;r]} \Box\Diamond A_a\Big) \rightarrow \Big(\bigwedge_{b \in [1;s]} \Box\Diamond F_b\Big) = \Big(\bigvee_{a \in [1;r]} \Diamond\Box \overline{A}_a\Big) \vee \Big(\bigwedge_{b \in [1;s]} \Box\Diamond F_b\Big). \tag{7.28}$$

By comparing φ in (7.28) with φ in (7.24), we see that a GR(1) condition $(\mathcal{A}, \mathcal{F})$ can be transformed into a generalized Rabin condition $\widetilde{\mathcal{R}}$ with $k = r + 1$ pairs, such that

$$\forall j \in [1;r]\ .\ R_j := A_j \quad \text{and} \quad G_j := \{V\}, \quad \text{and} \tag{7.29a}$$

$$R_k := \emptyset \quad \text{and} \quad G_k := \mathcal{F}. \tag{7.29b}$$

Fixpoint Algorithm: We first observe that the first r Rabin pairs with trivial goal sets actually correspond to a generalized Co-Büchi condition (compare (7.22)) which can be solved by the fixed point in Thm. 7.6 (see Sec. 7.2.4). Intuitively, the fixed point in Thm. 7.6 only needs to consider single indices form $P = [1;r]$ rather then full permutation sequences as in Thm. 7.1. By adding the last tuple $\langle G_k, R_k \rangle$ to the winning condition, we

[1] Again, the generalized Rabin pair $\langle G_{p_0}, R_{p_0} \rangle$ in (7.4) is artificially introduced and not part of $\widetilde{\mathcal{R}}$.

essentially need to consider two indices in each conjunct of (7.15), i.e., p_j (with $j \in [1;r]$) and p_k. In principle, we would need to consider both possible orderings of these two indices (compare (7.27)). However, by inspecting (7.29) we see that the sets corresponding to these indices always fulfill a (generalized) chain condition (compare (7.14)). That is, we have $R_j \supseteq R_k$ and $V = {}^1G_j \supseteq {}^bF$ for any $j \in [1;r]$ and $b \in [1;s]$. Hence, we only need to consider the permutation sequence $p_k p_j$ (compare (7.15)). Using this insight, along with some additional simplifications, we indeed yield the fixed point that we would obtain by simply applying our transformation to the well-known GR(1) fixed point (compare e.g. Piterman et al. (2006)). This observation is formalized in the next theorem and proved in App. A.2.5.

Theorem 7.9 *Let $\mathcal{G}^\ell = \langle \mathcal{G}, E^\ell \rangle$ be a game graph with live edges and $(\mathcal{A}, F^c)$ a GR(1) winning condition. Further, let*

$$Z^* = \nu Y_k. \bigcap_{b\in[1;s]} \mu\, {}^bX_k. \bigcup_{a\in[1;r]} \nu Y_a. \quad (F_b \cap \mathit{Cpre}(Y_k)) \cup \mathit{Apre}(Y_k,\ {}^bX_k) \cup (\overline{A}_a \cap \mathit{Cpre}(Y_a)). \tag{7.30}$$

Then Z^* is equivalent to the winning region $\mathcal{W}$ of Player 0 in the fair adversarial game over $\mathcal{G}^\ell$ for the winning condition φ in (7.28). Moreover, the fixpoint algorithm runs in $O(n^2rs)$ symbolic steps, and a finite-memory winning strategy for Player 0 can be extracted from it.

In particular, the strategy extraction is performed in the same way as by Piterman et al. (2006) for a "normal" GR(1) game.

7.4. Stochastic Generalized Rabin Games

We present an important application of our fixpoint algorithm in computing the almost sure winning region in $2^1/_2$-player games. For this, we reduce the problem of computing almost sure winning strategies in $2^1/_2$-player generalized Rabin games to the computation of winning strategies in fair adversarial generalized Rabin games. This yields a direct symbolic algorithm for solving $2^1/_2$-player generalized Rabin games.

Suppose $\mathcal{G}$ is a $2^1/_2$-player game graph and $\widetilde{\mathcal{R}}$ is a generalized Rabin winning condition. To obtain the reduced 2-player game graph, we simply reinterpret the random vertices as *Player* 1 vertices and the random edges as live edges. Let us first formalize this notion of the reduced game graph.

Definition 7.2 (Reduction to 2-player game with live edges) Let $\mathcal{G} = \langle V, V_0, V_1, V_r, E \rangle$ be a $2^1/_2$-player game graph. Define $\mathit{Derand}(\mathcal{G}) := \left\langle \left\langle \widetilde{V}, \widetilde{V}_0, \widetilde{V}_1, \widetilde{E} \right\rangle, E^\ell \right\rangle$ as follows:

- $\widetilde{V} = V$, $\widetilde{V}_0 = V_0$, $\widetilde{V}_1 = V_1 \cup V_r$, $\widetilde{E} = E$, and $E^\ell = E_r$.

It remains to show that the almost sure winning set of *Player* 0 in $\mathcal{G}$ for the generalized Rabin winning condition $\widetilde{\mathcal{R}}$ is the same as the winning set of *Player* 0 in the fair adversarial

game over $\mathit{Derand}(\mathcal{G})$ for the winning condition $\widetilde{\mathcal{R}}$. This is formalized in the following theorem, which is proved in App. A.2.6. The proof essentially shows that the random edges of $\mathcal{G}$ simulate the live edges of $\mathit{Derand}(\mathcal{G})$, and vice versa.

Theorem 7.10 *Let $\mathcal{G}$ be a $2^1/_2$-player game graph, $\widetilde{\mathcal{R}}$ be a generalized Rabin condition, $\varphi \subseteq V^\omega$ be the corresponding LTL specification (Eq. (7.24)) over the set of vertices V of $\mathcal{G}$, and $\mathit{Derand}(\mathcal{G})$ be the reduced 2-player game graph. Let $\mathcal{W} \subseteq \widetilde{V}$ be the set of all the vertices from where Player 0 wins the fair adversarial game over $\mathit{Derand}(\mathcal{G})$ for the winning condition φ, and $\mathcal{W}^{a.s.}$ be the almost sure winning set of Player 0 in the game graph $\mathcal{G}$ for the specification φ. Then, $\mathcal{W} = \mathcal{W}^{a.s.}$. Moreover, a winning strategy in $\mathit{Derand}(\mathcal{G})$ is also a winning strategy in $\mathcal{G}$, and vice versa.*

The above theorem generalizes (Glabbeek and Höfner, 2019, Thm. 11.1) from liveness properties to all LTL specifications on $2^1/_2$-player games. Together with our symbolic algorithm for fair adversarial Rabin games, the reduction implies a $O(n^{k+2}k!)$ algorithm for stochastic Rabin games for a game with n states and k Rabin pairs. This improves the previous best algorithm from Chatterjee et al. (2005), which reduces the problem to a normal 2-player game with $O(n(k+1))$ states and $k+1$ Rabin conditions, and therefore has a complexity of $O\left((n(k+1))^{k+2}(k+1)!\right)$.

Remark 5 *The idea underlying this section is to replace random edges with live edges to compute almost sure winning states. We recall again that probabilistic choice is different from (i.e., stronger than) strong transition fairness studied in this chapter. See Sec. 7.1 for an illustrative example in Fig. 7.1.*

7.5. Implementation and Experimental Evaluation

We have developed a C++-based tool Fairsyn, which implements the symbolic fair adversarial Rabin fixpoint from Eq. (7.4) using BDDs. We developed two versions of Fairsyn: A single-threaded version using the (single-threaded) CUDD library (Somenzi, 2019), and a multi-threaded version using the (multi-threaded) Sylvan library (van Dijk and van de Pol, 2015). In addition, Fairsyn implements a modified version of the well-known acceleration technique for fixpoint computations (Long et al., 1994). The original acceleration procedure uses excessive memory. We implemented a practical bounded memory version of the original algorithm that works well in practice; the main idea has been summarized in Sec. 7.5.1.

To show the effectiveness of our proposed symbolic algorithm for fair adversarial Rabin games, we performed various experiments with Fairsyn which fall into two different categories. First, in Sec. 7.5.2, we demonstrate the merits of utilizing parallelization and acceleration within Fairsyn. Second, in Sec. 7.5.3, we show the practical relevance of our algorithm by solving two large practical case-studies stemming from the areas of software engineering and control systems.

The experiments in Sec. 7.5.2 and Sec. 7.5.3 were performed using Sylvan-based Fairsyn on a computer equipped with a 3 GHz Intel Xeon E7 v2 processor with 48 CPU cores

and 1.5 TiB RAM. The experiments in Sec. 7.5.3 were performed using CUDD-based Fairsyn on a Macbook Pro (2015) laptop equipped with a 2.7 GHz Dual-Core Intel Core i5 processor with 16 GiB RAM.

7.5.1. The Accelerated Fixpoint Algorithm

Consider the fixpoint algorithm in (7.4). In the correctness proof of Thm. 7.1 discussed in App. A.2.3, we have been remembering so called configuration prefixes $\delta = p_0 i_0 \dots p_{j-1} i_{j-1}$ for some $j \leq k$ for every fixed-point variable X (see Eq. (A.13)). We denoted by $X^{i_j}_{\delta p_j}$ the set of states computed in the i_j-th iteration of the fixed-point over X_{p_j} after the fixed-point over Y_{p_j} has already terminated within the i_{j-1}-th iteration over $X_{p_{j-1}}$ after the fixed-point over $Y_{p_{j-1}}$ has terminated in the i_{j-2}-th iteration over $X_{p_{j-2}}$ and so forth.

In order to describe the accelerated implementation of (7.4), we do not assume that the fixed-points over Y-variables have already terminated, but additionally remember their counters m. This leads to configuration prefixes $\delta = p_0 m_0 i_0 \dots p_{j-1} m_{j-1} i_{j-1}$ and lets us define that $X^{m_j i_j}_{\delta p_j}$ is the set of states computed in the i_j-th iteration of the fixed-point over X_{p_j} during the m_j-th iteration over Y_{p_j}, computing the set $Y^{m_j}_{\delta p_j}$ and so forth.

Given two configuration prefixes $\delta = p_0 m_0 i_0 \dots p_{j-1} m_{j-1} i_{j-1}$ and $\delta' = p'_0 m'_0 i'_0 \dots p'_{j-1} m'_{j-1} i'_{j-1}$ we define $\delta <_m \delta'$ if $p_0 \dots p_{j-1} = p'_0 \dots p'_{j-1}$, $m_0 \dots m_{j-1} < m'_0 \dots m'_{j-1}$ (using the induced lexicographic order) and $i_0 \dots i_{j-1} = i'_0 \dots i'_{j-1}$. We define $\delta <_i \delta'$ similarly.

Now Piterman and Pnueli (2006) showed, based on a result of Long et al. (1994), that for every configuration prefix $\delta = p_0 m_0 i_0 \dots p_{j-1} m_{j-1} i_{j-1}$ the computation of $Y^0_{\delta p_j}$ can start from the *minimal* set $Y^{m_j}_{\delta' p_j}$ (instead of the entire set of vertices V) such that $\delta' p_j m_j <_m \delta p_j 0$. Dually, for every configuration prefix $\delta = p_0 m_0 i_0 \dots p_{j-1} m_{j-1} i_{j-1}$ the computation of $X^{m_j 0}_{\delta p_j}$ can start from the *maximal* set $X^{m_j i_j}_{\delta' p_j}$ (instead of the empty set) such that $\delta' p_j m_j i_j <_i \delta p_j m_j 0$.

Further, we see that for the innermost fixpoint, i.e. when $j = k$, it follows that for every computation prefix δ , there can be at most n iterations over both Y_{p_k} and X_{p_k}, where n is the total number of vertices. I.e., n different sets $Y^{m_k}_{\delta p_k}$ and $X^{m_k i_k}_{\delta p_k}$ have to be freshly computed for each δp_k and $\delta p_k m_k$ respectively. We see that there are $\mathcal{O}(n^{k+1}k!)$ different such permutation sequences. As the computation of the innermost fixpoint dominates the computation time, it is shown by Long et al. (1994) that this results in an overall worst-case computation time of $\mathcal{O}(n^{(k+1)+1}k!) = \mathcal{O}(n^{k+2}k!)$ (where n is the total number of vertices and k is the number of Rabin pairs).

Unfortunately, the memory requirement of this acceleration algorithm is enormous. To see this, observe that in order to warm-start the computation of $Y^0_{\delta p_j}$ with $\delta = p_0 m_0 i_0 \dots p_{j-1} m_{j-1} i_{j-1}$ we need to store the current minimal set w.r.t. the m-prefix for every combination of p- and i-prefixes that can occur in δ, which are $\mathcal{O}(n^{k+1}k!)$ many. Similarly, to warm-start the computation of $X^{m_j i_j}_{\delta p_j}$ we need to store the current minimal set w.r.t. the i-prefix for every combination of p- and m-prefixes that can occur in δ. This means that the memory required by the algorithm is $\mathcal{O}(n^{k+1}k!)$, which is prohibitively large for large values of n and k.

We implemented a *space-bounded* version of the acceleration algorithm, where for any given parameter M (chosen by the user), we stored only up to M values for each counter. Whenever the values of all the counters are less than M, we use the regular acceleration algorithm as outlined above. Otherwise, if any of the counters exceeds M, then we fall back to the regular initialization procedure of fixpoint algorithms, i.e. depending on whether it is an Y or an X variable, initialize it with V or $\emptyset$ respectively. As a result, the memory requirement of our accelerated fixpoint algorithm is given by $\mathcal{O}(M^{k+1}k!)$. This space-bounded acceleration algorithm made our implementation much faster and yet practically feasible, as has been demonstrated in Sec. 7.5.2 and Sec. 7.5.3.

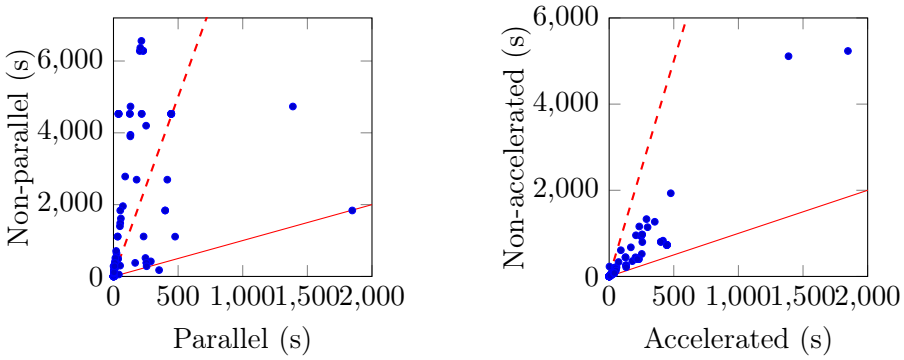
7.5.2. Performance Evaluation

This section discusses a benchmark suite used to empirically evaluate the merits of the two important aspects of Fairsyn, namely the parallelization and the acceleration. Our benchmark suite is build on transition systems taken from the Very Large Transition Systems (VLTS) benchmark suite (Garavel and Descoubes, 2003). For each chosen transition system, we randomly generated benchmark instances of fair adversarial Rabin games with up to 3 Rabin pairs. To transform a given transition systems into a fair adversarial Rabin game, we labeled (i) 50% of randomly chosen vertices as system vertices, (ii) the remaining vertices as environment vertices, (iii) up to 5% of randomly selected environment edges as live edges, and (iv) for every set in $\mathcal{R} = \{\langle G_1, R_1\rangle, \ldots, \langle G_k, R_k\rangle\}$ we randomly selected up to 5% of all vertices to be contained. We have summarized the relevant details of all the randomly generated instances of the fair adversarial Rabin games in Table A.1 and Table A.2 in App. A.4. In these examples, the number of vertices were 289–566,639, the number of BDD variables were 9–20, the number of transitions were 1224–3,984,160, and number of live edges were 1–42,757. For all benchmark instances with more than 4 live edges, the naïve version of Fairsyn which treats live edges as Streett conditions and transforms them into additional Rabin pairs as discussed in Sec. 7.2.3, did not terminate after 2 hours.

Merits of parallelization. We ran Fairsyn on 10 different benchmark instances with 1 or 2 Rabin pairs, and varied the number of parallel worker threads used in Fairsyn between 1–48, while keeping the acceleration enabled. The left scatter plot in Fig. 7.5 plots the computation times with 48 threads (parallel) versus the computation times with 1 thread (non-parallel). Observe that in almost all the experiments, the parallelized version outperforms the non-parallelized version (points above the solid red line). In addition, in many cases the speedup achieved due to the parallelization was more than one order of magnitude (points above the dashed red line).

A more fine-grained analysis of the benefits of parallelization is shown in Fig. 7.6.(a). Here computation time (in logarithmic scale) is plotted over the number of worker threads used. We observe that the saving due to parallelization is more significant for the curves lying in the top half which correspond to larger examples. This is due to the better utilization of the available pool of worker threads by the larger examples.

Merits of acceleration. We ran Fairsyn on 10 different benchmark instances with 1–3



(a) Comparison between the computation times for the non-parallel (1 worker thread) and parallel (48 worker threads) version of Fairsyn, with acceleration being enabled in both cases.

(b) Comparison between the computation times for the non-accelerated and the accelerated version of Fairsyn, with parallelization being enabled in both cases.

Figure 7.5.: Experimental evaluation of various algorithmic optimization on the VLTS benchmark examples. In both scatter plots, the points on the solid red line represent the same computation time. The points on the dashed red line represent an order of magnitude improvement.

Rabin pairs, and varied the acceleration parameter M between 2–15, while the number of worker threads was fixed to 48. The right scatter plot in Fig. 7.5 plots the computation times with $M = 15$ versus the computation times with no acceleration. Observe that in almost all the experiments, the accelerated version outperformed the non-accelerated version (points above the solid red line), and in many cases the achieved speedup is close to an order of magnitude (points near the dashed red line). See Fig. A.2 in App. A.4 for a zoomed-in version of Fig. 7.5.

A more fine-grained analysis of the benefits of acceleration is shown in Fig. 7.6.(b)–(e). Here we have plotted the total computation time (Plots (b),(d)) and the initialization time (Plots (c),(e)) in logarithmic scale over M for benchmark instances with 2 Rabin pairs (Plots (b),(c)) and 3 Rabin pairs (Plots (d),(e)). Plots for instances with 1 Rabin pair can be found in Fig. A.3 in App. A.4.

The plotted initialization time is needed by the accelerated algorithm for allocating memory to store intermediate fixpoint values. We observe that this initialization time grows exponentially with M, which is due to the $\mathcal{O}(M^{k+1}k!)$ space complexity of the acceleration algorithm. As a result, the computational savings due to the use of acceleration get undermined by the high initialization cost for large M. We note that, due to their random generation, the considered benchmark instances are not well structured. This results in low iteration numbers over involved fixed-point variables. Due to this, the allocated memory gets underutilized for large values of M. In the practically relevant examples discussed in Sec. 7.5.3 the game graph is naturally structured, resulting in a large number of fixpoint iterations and thereby showing superior performance for larger

values of M.

7.5.3. Practical Benchmarks

This section shows that Fairsyn is able to efficiently solve two practical case studies stemming from the areas of software engineering (Sec. 7.5.3) and control systems (Sec. 7.5.3).

Code-Aware Resource Management

We consider a case study introduced by Chatterjee et al. (2013). It considers the problem of synthesizing a *code-aware resource manager* for a network protocol, i.e., multi-threaded program running on a single CPU. The task of the resource manager is to grant different threads access to different shared synchronization resources (mutexes and counting semaphores). The specification is deadlock freedom across all threads at all time while assuming a *fair scheduler* (scheduling every thread always eventually) and *fair progress* in every thread (i.e., taking every existing execution branch always eventually). By making the resource manager *code-aware*, it can avoid deadlocks by utilizing its knowledge about the require and release characteristics of all treads for different resources.

Chatterjee et al. (2013) showed that the problem of synthesizing a code-aware resource manager can be approximated using a $1^1/_2$-player game[1] generated from the known require and release characteristics of all threads. We used Fairsyn to synthesize a code-aware resource manager for this problem, where the live edges model the aforementioned fairness conditions imposed on the scheduler and the threads.

Motivated by the case study conducted by Chatterjee et al. (2013), we consider a network protocol consisting of 3 threads and 2 queues of bounded capacity, as depicted in Fig. 7.7. The threads (shown as oval-shaped nodes) are called *generator*, *sender*, and *delay*, and the queues (shown as rectangular nodes) are called *broadcast* and *output*. The generator generates data packets and dispatches them to either the broadcast queue or the output queue. Packets from the broadcast queue are added to the output queue after a random delay, introduced by the delay thread. The purpose of this delay is to avoid packet collisions during broadcasting. The packets in the output queue are in transit and get processed by the sender process. The sender process attempts to transmit packets from the output queue via the network, and when the transmission fails, it adds the respective data packet back to the broadcast queue, so that another transmission attempt can be made after a delay. Access to all queues is protected by mutexes and semaphores. Each queue has one mutex and two semaphores, one for counting the number of empty places and another for counting the number of packets present.

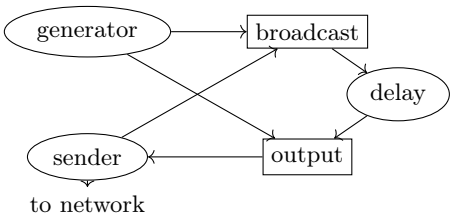
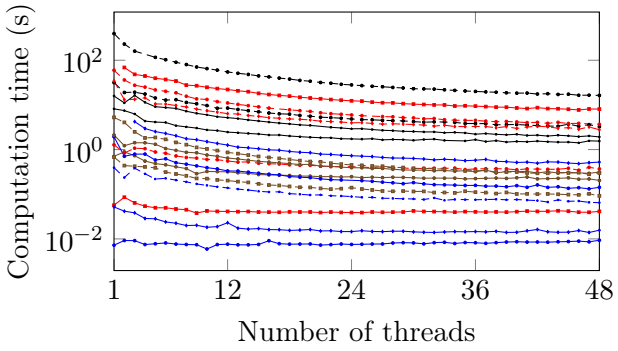
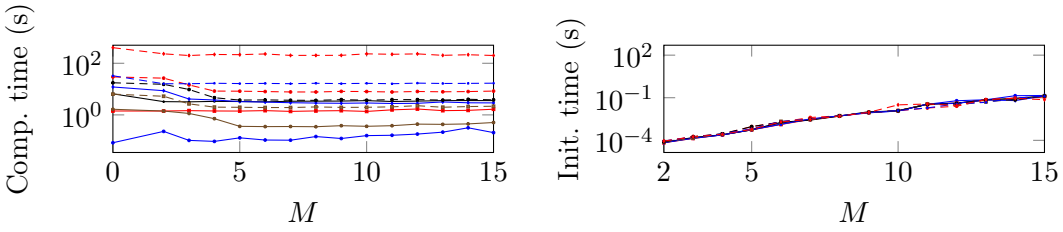


Figure 7.7.: Structure of network protocol.

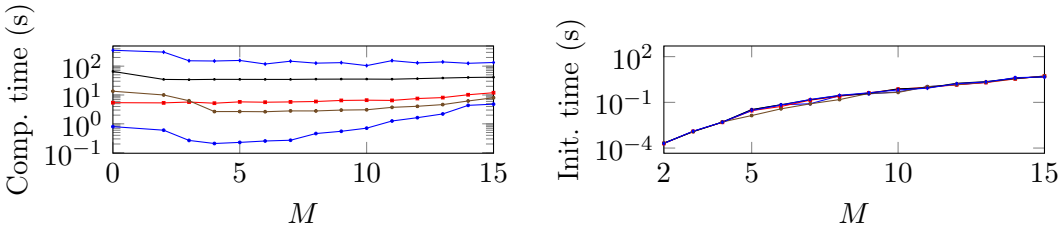
[1] A $1^1/_2$-player game is a $2^1/_2$-player game without any *Player* 1 vertices.



(a) Effect of parallelization on computation time, with the acceleration enabled.



(b) Effect of variation of the acceleration parameter M on the total computation time and initialization time (parallelization being enabled) for 2 Rabin pairs.



(c) Effect of variation of the acceleration parameter M on the total computation time and initialization time (parallelization being enabled) for 3 Rabin pairs.

Figure 7.6.: Effect of parallelization and the acceleration parameter M on the computation time. The computation times (Y-axis) is always shown in the logarithmic scale.

Broadcast Queue Capacity	Output Queue Capacity	Number of Vertices	Number of Transitions	Number of Live edges	Number of BDD variables	Time (seconds)
1	1	5,307,840	10,135,300	5,124,100	25	7.38
2	1	21,231,400	40,541,200	20,496,400	27	24.90
3	1	21,414,100	42,080,300	21,265,900	27	28.98
1	2	21,340,800	40,879,100	20,834,300	27	38.26
1	3	21,559,400	42,756,100	21,772,800	27	51.56
2	2	85,363,200	163,516,000	83,337,200	29	133.20
3	2	86,061,400	169,673,000	86,415,400	29	144.28
2	3	86,237,400	171,024,000	87,091,200	29	163.62
3	3	86,870,100	177,181,000	90,169,300	29	203.15

Table 7.1.: Performance of Fairsyn on the code-aware resource management benchmark experiment.

As discussed by Chatterjee et al. (2013), the outlined network protocol may deadlock when both queues are full, a transmission via sender fails, and the sender tries to insert the packet back to the broadcast queue. In this case, due to the output queue being full, the broadcast queue will not be able to make space for the incoming packet, leading to a deadlock situation. The correct strategy for the resource manager to prevent this deadlock is to ensure that the generator never adds packets to the broadcast queue if the output queue is full.

We used the parallel and accelerated version of Fairsyn with $M = 15$ to automatically synthesize the resource manager for the outlined network protocol case study. Indeed, Fairsyn was successful in discovering the outlined managing strategy. To showcase Fairsyn's performance on this case study, we report the number of vertices of the problem instance and Fairsyn's computation time to solve it for different queue capacities in Table 7.1; an extended version of the table with more number of cases has been included in Table A.3 in App. A.4. In all cases, Fairsyn was able to provide expected strategies within a reasonable amount of time. Note that treating the live edges as Streett conditions would result in a game with several million Rabin pairs, making all these examples go far beyond the scope of any synthesis tool for Rabin games.

Controller Synthesis for Stochastically Perturbed Dynamical Systems

In Chap. 5, we showed how synthesizing almost sure winning controller for controlled Markov processes (CMP) can be approximately solved by an abstract $2^1/_2$-player game. This abstract $2^1/_2$-player game can be directly solved by our fixpoint algorithms for fair adversarial games, through the reduction shown in Sec. 7.4. These form the basic ABCD procedure for the qualitative controller synthesis for CMPs, and is implemented in our tool `Mascot-SDS`. We summarize the experimental performance evaluation of `Mascot-SDS`

in Sec. 5.6. There we show that on different instances of an established case study from the literature, `Mascot-SDS` outperforms state-of-the art synthesis techniques by up to 2.5 orders of magnitude.

7.6. Related Work

The idea to consider strong transition fairness as a tractable fragment of *strong fairness (compassion)* assumptions is inspired by work on the synthesis of supervisory controllers for non-terminating processes by Thistle and Malhamé (1998). Here, a fixed-point algorithm for the general problem (Thistle, 1995) which manipulates Rabin automata, is shown to significantly simplify under a slightly weaker transition fairness assumption. While our algorithm shares the underlying intuition behind the simplification of Thistle and Malhamé (1998), it is syntactically very different due to the symbolic manipulation of sets of states rather then automata.

Aminof et al. (2004) studied fair CTL and LTL model checking where the fairness condition is given by a transition fairness with *all* edges of the transition system live. They show that CTL model checking under this all-live fairness condition, can be syntactically transformed to *non-fair* CTL model checking. A similar transformation is possible for fair model checking of Büchi, Rabin, and Streett formulas. The correctness of their transformation is based on reasoning similar to our *Apre* operator. For example, a state satisfies the CTL formula $\forall\Diamond p$ under fairness iff all paths starting from the state either eventually visits p or always visits states from which a visit to p is possible.

The fixpoint (7.9), as well as the *Apre* operator, are similar in structure to the solution of almost surely winning states in concurrent reachability games (de Alfaro et al., 1998); see Rem. 4.

For GR(1) winning conditions, Svorenová et al. (2015) presented a symbolic fixpoint algorithm for stochastic games (which can be modeled using fair adversarial games, see Sec. 7.4). While one can show that the output of their algorithm coincides with the output of our newly derived fixpoint algorithm in (7.30), their algorithm is structurally more involved. On a conceptual level, we feel our insight about simply "swapping" predecessor operators in the right manner is insightful even if one can also use their algorithm to find a solution to this problem.

The idea of the simple "predecessor operator swapping trick" shares resemblence with environmentally-friendly GR(1) synthesis, proposed by Majumdar et al. (2019). There, the authors show a direct symbolic algorithm to compute *Player* 0 strategies which do not win a given GR(1) game vacuously, by rendering the assumptions false. Recall that in GR(1) specifications, the assumption is represented using deterministic generalized Büchi conditions, but a direct relationship with transition fairness is not known. Hence, it is not clear if environmentally-friendly GR(1) synthesis problems can be reduced to fair adversarial games.

7.7. Conclusion

In this chapter, we presented an efficient symbolic fixpoint algorithm for computing the sure winning region in fair adversarial games with various different ω-regular winning conditions. Our algorithm is obtained through a simple syntactic transformation of the predecessor operator in the known algorithms for solving 2-player games. As a by-product, we also obtain direct symbolic algorithms for computing the almost sure winning region in $2^1/_2$-player games. We implemented our algorithm in the tool called `Fairsyn`, and demonstrate significant improvement in performance compared to the state-of-the-art.

8. Conclusion and Future Outlook

In this thesis, we have presented different correct-by-construction controller synthesis approaches which are faster, more modular, and support broader classes of problems compared to the state-of-the-art; following is a summary of the presented content.

Lazy multi-layered ABCD. Chap. 4 shows, using lazily computed multi-layered abstractions can give us significantly faster ABCD algorithms. The original Feedback Refinement Relations-based ABCD algorithms would use a single abstraction layer with uniform granularity. However, this would often require having a fine-grained abstraction for better precision, at the cost of larger computation time, so that the abstract synthesis does not return an empty solution. Chap. 4 shows how multiple abstractions of different granularities can gracefully address the precision versus computation time tradeoff. In our lazy multi-layered algorithms, during synthesis, the coarser abstraction layers are used as much as possible, and the finer abstraction layers are used only when necessary. We implemented this procedure in the tool `Mascot`, and on standard benchmarks, our algorithm showed up to one order of magnitude speedup.

Stochastic ABCD. In Chap. 5, we consider the controller synthesis problem for stochastic systems with ω-regular specifications. We show that the optimal controller, the one that maximizes the probability of satisfying the specification from every state, can be computed through a two-step decomposition. First, we need to compute the optimal almost sure winning controller that satisfies the specification with probability one and whose domain, known as the almost sure winning region, is as large as possible. Second, we need to compute a controller that maximizes the probability of *reaching* the almost sure winning region: The maximal reachability probability will give us a lower bound on the optimal satisfaction probability of the original ω-regular specification. While the second step can be solved using existing techniques, we present an ABCD approach to compute an under-approximation of the almost sure winning region and the associated almost sure winning controller. In our proposed ABCD approach, we obtain a finite $2\frac{1}{2}$-player game by discretizing the continuous state space and abstracting every (discrete-time) probabilistic continuous transition using a three-step game among the controller, the uncertainty introduced by the discretization, and the randomness from the noise. We symbolically implement this stochastic ABCD algorithm in the tool `Mascot-SDS`, and demonstrate improved performance over a contemporary tool independently developed around the same time as ours. The symbolic solution of $2\frac{1}{2}$-player games has been presented as a special case of the solution of fair adversarial games studied in Chap. 7.

Assume-guarantee distribution synthesis. Chap. 6 shows how to do modular synthesis of local controllers for a network of discrete systems, communicating through discrete/boolean variables. The systems negotiate a set of mutually beneficial assume-guarantee contracts, whose individual satisfactions will ensure overall satisfaction of the specifications. Technically speaking, the negotiation is an iterative procedure, where the systems repeatedly exchange minimal sets of assumptions that they require for satisfaction of their own specifications as well as the assumptions made by the other systems in the previous iterations. This gives us a sound, but incomplete, solution to the distributed synthesis problem, which is known to be undecidable. We implemented the negotiation algorithm in a prototype tool called `Agnes`, using which we showed the effectiveness of our approach on a couple of distributed synthesis examples.

Fair adversarial 2-player games. Fair adversarial games are 2-player games on finite graphs with a fairness assumption on the environment transitions. We show that winning strategy in fair adversarial games can be symbolically computed by introducing a simple syntactic transformation of the existing algorithms for 2-player games. The advantages of our algorithm are simplicity and computational performance compared to the state-of-the-art solution techniques. We demonstrate the benefits using several examples from the literature on controller synthesis (for stochastic systems), software engineering, and a number of well-established synthetic benchmarks in formal methods.

Future Work

We addressed several challenges of correct-by-design controller synthesis, but many are left open. Following is a summary of the problems that may be interesting to pursue as follow-up of this thesis in future:

Even better scalability. One of the constant focuses of research in this field has remained scalability. Even though we have seen many improvements, still lot more needs to be done to be able to *automatically* handle the vastness of the realistic models of cyber-physical systems having tens to hundreds of state variables, where the existing state-of-the-art tools can handle around 10 state variables at most. To be able to close this gap between reality and practice, not only we should keep building powerful tools, but also we should develop more compact abstraction methods, and focus on hierarchical and decentralized techniques.

Liveness assumptions with many components. One of the key components in tackling scalability issues is the use of system decomposition and using decentralized synthesis approaches. We have presented a negotiation procedure for mining assume-guarantee contracts from the local specifications of the system components. However, we used the restriction that the contracts be made up of only safety properties. In reality, there are situations, where live contracts would offer much more freedom to the systems, and the restriction of safe contracts would give no controller.

Another limitation of our negotiation algorithm is that we only consider a network of two systems. Generalizing this to arbitrarily many systems will help us arbitrarily scale the synthesis.

Generalization of transition fairness. In Chap. 7, we considered fair adversarial games, which are 2-player games with transition fairness assumption on certain live environment edges. The main drawback of transition fairness is that it is not preserved under synchronous product construction, i.e., the synchronization of a pair of live edges from two systems running in parallel might result in non-live fairness condition in the product. In reality, there are many real-life examples where such product construction is sometimes inevitable, before we can use the synthesis algorithm. So, it will be beneficial to investigate efficient algorithms for more general class of fairness assumptions which will be preserved under synchronous composition as well.

Handling unpredictability in real-world situations. When we talk about correct-by-construction techniques, the soundness of a controller depends on the assumed model of the system and the environment. In reality, the models, especially that of the environment, are not totally predictable. As a result, often we need to assume the worst-case model of the environment, which in many cases give us empty controller domain. To address this issue, we need to develop synthesis techniques which can dynamically adjust itself to unmodeled environmental behaviors at runtime. From the perspective of the abstraction-based works presented in this thesis, one of the challenges of a dynamic controller readjustment is that the reactive synthesis algorithms work best backwards in time. As a result, if any change in the model is detected at runtime, then the straightforward application of reactive synthesis would require complete recomputation of the abstraction from that point onward. Needless to say, this is computationally expensive, making practical usage implausible. We have done some initial work on a locally adaptive dynamic abstraction recomputation procedure (Bai et al., 2019), though it assumes that a global worst-case model of the disturbance is known (even if it is not enforced at all time).

To reason about unverifiable components in the loop. Often, a precise dynamic model of the system and the environment is either unavailable or is specified using a language that has so far limited support for formal reasoning (e.g. Simulink models, neural network). To tackle this issue, there is a surge of data-driven correct-by-construction techniques that provide statistical guarantees on the synthesized design (Hashimoto et al., 2020; Devonport et al., 2021; Salamati et al., 2021b,a; Lavaei et al., 2020b). The existing techniques use strong assumptions on the dynamics (Hashimoto et al., 2020; Salamati et al., 2021b), or does not support external disturbances (Devonport et al., 2021), or allows only a small subset of ω-regular properties as control specifications (Salamati et al., 2021a; Lavaei et al., 2020b). This is an active area of research, and further effort is needed in order to arrive at a satisfactory solution to the problem.

Bibliography

A. Abate, H. Blom, N. Cauchi, J. Delicaris, A. Hartmanns, M. Khaled, A. Lavaei, C. Pilch, A. Remke, S. Schupp, et al. Arch-comp20 category report: Stochastic models. *EPiC Series in Computing*, 74:76–106, 2020.

R. Alur and T. Henzinger. Reactive modules. *Formal Methods Syst. Des.*, 15(1):7–48, 1999.

R. Alur, T. Henzinger, O. Kupferman, and M. Vardi. Alternating refinement relations. In *CONCUR'97*, LNCS 1466, pages 163–178. Springer, 1998.

R. Alur, S. Moarref, and U. Topcu. Compositional and symbolic synthesis of reactive controllers for multi-agent systems. *Inf. Comput.*, 261(Part):616–633, 2018. doi: 10.1016/j.ic.2018.02.021. URL `https://doi.org/10.1016/j.ic.2018.02.021`.

A. D. Ames, P. Tabuada, B. Schürmann, W. Ma, S. Kolathaya, M. Rungger, and J. W. Grizzle. First steps toward formal controller synthesis for bipedal robots. In A. Girard and S. Sankaranarayanan, editors, *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control, HSCC'15, Seattle, WA, USA, April 14-16, 2015*, pages 209–218. ACM, 2015. doi: 10.1145/2728606.2728611. URL `https://doi.org/10.1145/2728606.2728611`.

A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada. Control barrier functions: Theory and applications. In *18th European Control Conference, ECC 2019, Naples, Italy, June 25-28, 2019*, pages 3420–3431. IEEE, 2019. doi: 10.23919/ECC.2019.8796030. URL `https://doi.org/10.23919/ECC.2019.8796030`.

B. Aminof, T. Ball, and O. Kupferman. Reasoning about systems with transition fairness. In F. Baader and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 11th International Conference, LPAR 2004, Montevideo, Uruguay, March 14-18, 2005, Proceedings*, volume 3452 of *Lecture Notes in Computer Science*, pages 194–208. Springer, 2004. doi: 10.1007/978-3-540-32275-7_14. URL `https://doi.org/10.1007/978-3-540-32275-7_14`.

i. B. Associated Press. Robot kills worker at volkswagen plant in germany. *The Guardian*. URL `https://www.theguardian.com/world/2015/jul/02/robot-kills-worker-at-volkswagen-plant-in-germany`.

Y. Bai and K. Mallik. Accurate abstractions for controller synthesis with non-uniform disturbances. In S. Lin, Z. Hou, and B. P. Mahony, editors, *Formal Methods and*

Software Engineering - 22nd International Conference on Formal Engineering Methods, ICFEM 2020, Singapore, Singapore, March 1-3, 2021, Proceedings, volume 12531 of *Lecture Notes in Computer Science*, pages 297–307. Springer, 2020. doi: 10.1007/978-3-030-63406-3_18. URL https://doi.org/10.1007/978-3-030-63406-3_18.

Y. Bai, K. Mallik, A. Schmuck, D. Zufferey, and R. Majumdar. Incremental abstraction computation for symbolic controller synthesis in a changing environment. In *58th IEEE Conference on Decision and Control, CDC 2019, Nice, France, December 11-13, 2019*, pages 6261–6268. IEEE, 2019. doi: 10.1109/CDC40024.2019.9030141. URL https://doi.org/10.1109/CDC40024.2019.9030141.

C. Baier and J. Katoen. *Principles of model checking*. MIT Press, 2008. ISBN 978-0-262-02649-9.

S. Bansal, M. Chen, S. L. Herbert, and C. J. Tomlin. Hamilton-jacobi reachability: A brief overview and recent advances. In *56th IEEE Annual Conference on Decision and Control, CDC 2017, Melbourne, Australia, December 12-15, 2017*, pages 2242–2253. IEEE, 2017. doi: 10.1109/CDC.2017.8263977. URL https://doi.org/10.1109/CDC.2017.8263977.

I. S. Bisht. Uk's 'protector' drone completes key tests. *The Defense Post*. URL https://www.thedefensepost.com/2021/12/02/uk-protector-drone-key-tests/.

F. Bruse, M. Falk, and M. Lange. The fixpoint-iteration algorithm for parity games. *arXiv preprint arXiv:1408.5961*, 2014.

O. L. Bulancea, P. Nilsson, and N. Ozay. Nonuniform abstractions, refinement and controller synthesis with novel bdd encodings. *IFAC-PapersOnLine*, 51(16):19–24, 2018.

J. Cámara, A. Girard, and G. Gössler. Synthesis of switching controllers using approximately bisimilar multiscale abstractions. In *HSCC*, pages 191–200, 2011a.

J. Cámara, A. Girard, and G. Gössler. Safety controller synthesis for switched systems using multi-scale symbolic models. In *CDC '11*, pages 520–525, 2011b.

K. Chandy and J. Misra. *Parallel Program Design: A Foundation*. Addison-Wesley Publishing Company, 1988.

K. Chatterjee and T. A. Henzinger. A survey of stochastic ω-regular games. *J. Comput. Syst. Sci.*, 78(2):394–413, 2012.

K. Chatterjee, M. Jurdziński, and T. A. Henzinger. Simple stochastic parity games. In *International Workshop on Computer Science Logic*, pages 100–113. Springer, 2003.

K. Chatterjee, L. de Alfaro, and T. A. Henzinger. The complexity of stochastic Rabin and Streett games. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, volume 3580 of *Lecture Notes in Computer Science*, pages 878–890. Springer, 2005.

K. Chatterjee, T. A. Henzinger, and B. Jobstmann. Environment assumptions for synthesis. In *International Conference on Concurrency Theory*, pages 147–161. Springer, 2008.

K. Chatterjee, L. De Alfaro, M. Faella, R. Majumdar, and V. Raman. Code aware resource management. *Formal Methods in System Design*, 42(2):146–174, 2013.

Y. Chen, M. Ahmadi, and A. D. Ames. Optimal safe controller synthesis: A density function approach. In *2020 American Control Conference, ACC 2020, Denver, CO, USA, July 1-3, 2020*, pages 5407–5412. IEEE, 2020. doi: 10.23919/ACC45564.2020.9147721. URL `https://doi.org/10.23919/ACC45564.2020.9147721`.

A. Condon. The complexity of stochastic games. *Inf. Comput.*, 96(2):203–224, 1992. doi: 10.1016/0890-5401(92)90048-K. URL `https://doi.org/10.1016/0890-5401(92)90048-K`.

S. Coogan and M. Arcak. Efficient finite abstraction of mixed monotone systems. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, pages 58–67. ACM, 2015.

S. Coogan, E. A. Gol, M. Arcak, and C. Belta. Traffic network control from temporal logic specifications. *IEEE Trans. Control. Netw. Syst.*, 3(2):162–172, 2016. doi: 10.1109/TCNS.2015.2428471. URL `https://doi.org/10.1109/TCNS.2015.2428471`.

E. L. Corronc, A. Girard, and G. Goessler. Mode sequences as symbolic states in abstractions of incrementally stable switched systems. In *Proceedings of the 52nd IEEE Conference on Decision and Control, CDC 2013, December 10-13, 2013, Firenze, Italy*, pages 3225–3230. IEEE, 2013. doi: 10.1109/CDC.2013.6760375. URL `https://doi.org/10.1109/CDC.2013.6760375`.

T. Dang and R. Testylier. Reachability analysis for polynomial dynamical systems using the bernstein expansion. *Reliable Computing*, 17(2):128–152, 2012.

L. De Alfaro. *Formal verification of probabilistic systems.* Number 1601. Citeseer, 1997.

L. de Alfaro and T. A. Henzinger. Concurrent omega-regular games. In *Proceedings Fifteenth Annual IEEE Symposium on Logic in Computer Science (Cat. No. 99CB36332)*, pages 141–154. IEEE, 2000.

L. de Alfaro, T. A. Henzinger, and O. Kupferman. Concurrent reachability games. In *39th Annual Symposium on Foundations of Computer Science, FOCS*, pages 564–575. IEEE Computer Society, 1998.

A. Devonport, A. Saoud, and M. Arcak. Symbolic abstractions from data: A PAC learning approach. *CoRR*, abs/2104.13901, 2021. URL `https://arxiv.org/abs/2104.13901`.

R. Dimitrova and R. Majumdar. Deductive control synthesis for alternating-time logics. In T. Mitra and J. Reineke, editors, *2014 International Conference on Embedded Software, EMSOFT 2014, New Delhi, India, October 12-17, 2014*, pages 14:1–14:10. ACM, 2014. doi: 10.1145/2656045.2656054. URL `https://doi.org/10.1145/2656045.2656054`.

M. Dutreix and S. Coogan. Specification-guided verification and abstraction refinement of mixed-monotone stochastic systems, 2019.

M. Dutreix, J. Huh, and S. Coogan. Abstraction-based synthesis for stochastic systems with omega-regular objectives. *arXiv preprint*, 2020. arXiv:2001.09236.

E. A. Emerson and C. S. Jutla. The complexity of tree automata and logics of programs. In *FoCS*, volume 88, pages 328–337, 1988.

E. A. Emerson and C. S. Jutla. On simultaneously determinizing and complementing omega-automata (extended abstract). In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science*, pages 333–342. IEEE Computer Society, 1989.

E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy. In *FOCS'91*, pages 368–377, 1991.

B. Finkbeiner and S. Schewe. Uniform distributed synthesis. In *20th IEEE Symposium on Logic in Computer Science (LICS 2005), 26-29 June 2005, Chicago, IL, USA, Proceedings*, pages 321–330. IEEE Computer Society, 2005. doi: 10.1109/LICS.2005.53. URL `https://doi.org/10.1109/LICS.2005.53`.

B. Finkbeiner and S. Schewe. Bounded synthesis. *Int. J. Softw. Tools Technol. Transf.*, 15(5-6):519–539, 2013. doi: 10.1007/s10009-012-0228-z. URL `https://doi.org/10.1007/s10009-012-0228-z`.

N. Francez. *Fairness*. Springer, Berlin, 1986.

D. M. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. In P. W. Abrahams, R. J. Lipton, and S. R. Bourne, editors, *Conference Record of the Seventh Annual ACM Symposium on Principles of Programming Languages, Las Vegas, Nevada, USA, January 1980*, pages 163–173. ACM Press, 1980. doi: 10.1145/567446.567462. URL `https://doi.org/10.1145/567446.567462`.

H. Garavel and N. Descoubes. Very large transition systems. 2003. URL `http://tinyurl.com/yuroxx`.

D. Gelles. Boeing 737 max: What's happened after the 2 deadly crashes. *The New York Times*. URL `https://www.nytimes.com/interactive/2019/business/boeing-737-crashes.html`.

A. Girard. Controller synthesis for safety and reachability via approximate bisimulation. *Automatica*, 48(5):947–953, 2012.

A. Girard and G. Gößler. Safety synthesis for incrementally stable switched systems using discretization-free multi-resolution abstractions. *Acta Informatica*, 57(1-2): 245–269, 2020. doi: 10.1007/s00236-019-00341-x. URL `https://doi.org/10.1007/s00236-019-00341-x`.

A. Girard and G. J. Pappas. Approximation metrics for discrete and continuous systems. *TAC*, 25(5):782–798, 2007.

A. Girard, G. Pola, and P. Tabuada. Approximately bisimilar symbolic models for incrementally stable switched systems. *TAC*, 55(1):116–126, 2010.

A. Girard, G. Gössler, and S. Mouelhi. Safety controller synthesis for incrementally stable switched systems using multiscale symbolic models. *TAC*, 61(6):1537–1549, 2016.

R. V. Glabbeek and P. Höfner. Progress, justness, and fairness. *ACM Comput. Surv.*, 52 (4), 2019.

E. Gradel and W. Thomas. *Automata, logics, and infinite games: a guide to current research*, volume 2500. Springer Science & Business Media, 2002.

F. Gruber, E. S. Kim, and M. Arcak. Sparsity-aware finite abstraction. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 2366–2371. IEEE, 2017.

Y. Gurevich and L. Harrington. Trees, automata, and games. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 60–65, 1982.

S. Haesaert and S. Soudjani. Robust dynamic programming for temporal logic control of stochastic systems. *CoRR*, abs/1811.11445, 2018. URL `http://arxiv.org/abs/1811.11445`.

S. Haesaert, S. E. Z. Soudjani, and A. Abate. Temporal logic control of general markov decision processes by approximate policy refinement. *CoRR*, abs/1712.07622, 2017. URL `http://arxiv.org/abs/1712.07622`.

K. Hashimoto, A. Saoud, M. Kishida, T. Ushio, and D. V. Dimarogonas. Learning-based safe symbolic abstractions for nonlinear control systems. *CoRR*, abs/2004.01879, 2020. URL `https://arxiv.org/abs/2004.01879`.

D. Henrion and M. Korda. Convex computation of the region of attraction of polynomial control systems. *IEEE Trans. Autom. Control.*, 59(2):297–312, 2014. doi: 10.1109/TAC.2013.2283095. URL `https://doi.org/10.1109/TAC.2013.2283095`.

H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, and M. Siegle. A tool for model-checking Markov chains. *STTT*, 4(2):153–172, 2003.

O. Hernández-Lerma and J. B. Lasserre. *Discrete-time Markov control processes*, volume 30 of *Applications of Mathematics*. Springer, 1996.

K. Hsu, R. Majumdar, K. Mallik, and A.-K. Schmuck. Lazy abstraction-based control for safety specifications. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 4902–4907. IEEE, 2018a.

K. Hsu, R. Majumdar, K. Mallik, and A.-K. Schmuck. Multi-layered abstraction-based controller synthesis for continuous-time systems. In *HSCC*, pages 120–129. ACM, 2018b.

K. Hsu, R. Majumdar, K. Mallik, and A.-K. Schmuck. Lazy abstraction-based control for reachability. *arXiv preprint arXiv:1804.02722*, 2018c.

K. Hsu, R. Majumdar, K. Mallik, and A. Schmuck. Lazy abstraction-based controller synthesis. In Y. Chen, C. Cheng, and J. Esparza, editors, *Automated Technology for Verification and Analysis - 17th International Symposium, ATVA 2019, Taipei, Taiwan, October 28-31, 2019, Proceedings*, volume 11781 of *Lecture Notes in Computer Science*, pages 23–47. Springer, 2019. doi: 10.1007/978-3-030-31784-3_2. URL `https://doi.org/10.1007/978-3-030-31784-3_2`.

O. Hussien and P. Tabuada. Lazy controller synthesis using three-valued abstractions for safety and reachability specifications. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 3567–3572. IEEE, 2018.

P. Jagtap, S. Soudjani, and M. Zamani. Formal synthesis of stochastic systems via control barrier certificates. *arXiv: eess.SY*, abs/1905.04585, 2019.

M. M. Jr., A. Davitian, and P. Tabuada. PESSOA: A tool for embedded controller synthesis. In T. Touili, B. Cook, and P. B. Jackson, editors, *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*, volume 6174 of *Lecture Notes in Computer Science*, pages 566–569. Springer, 2010. doi: 10.1007/978-3-642-14295-6_49. URL `https://doi.org/10.1007/978-3-642-14295-6_49`.

O. Kallenberg. *Foundations of modern probability*. Probability and its Applications. Springer Verlag, New York, 2002.

N. Kariotoglou, M. Kamgarpour, T. H. Summers, and J. Lygeros. The linear programming approach to reach-avoid problems for Markov decision processes. *J. Artif. Int. Res.*, 60 (1):263–285, Sept. 2017. ISSN 1076-9757.

M. Kazemi and S. Soudjani. Formal policy synthesis for continuous-space systems via reinforcement learning. *CoRR*, abs/2005.01319, 2020. URL `https://arxiv.org/abs/2005.01319`.

M. Khaled and M. Zamani. pfaces: an acceleration ecosystem for symbolic control. In N. Ozay and P. Prabhakar, editors, *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2019, Montreal, QC, Canada, April 16-18, 2019*, pages 252–257. ACM, 2019. doi: 10.1145/3302504.3311798. URL `https://doi.org/10.1145/3302504.3311798`.

E. S. Kim, M. Arcak, and S. A. Seshia. Symbolic control design for monotone systems with directed specifications. *Automatica*, 83:10–19, 2017.

M. Korda, D. Henrion, and C. N. Jones. Convex computation of the maximum controlled invariant set for polynomial control systems. *SIAM J. Control. Optim.*, 52(5):2944–2969, 2014. doi: 10.1137/130914565. URL `https://doi.org/10.1137/130914565`.

M. Korda, D. Henrion, and C. N. Jones. Controller design and value function approximation for nonlinear dynamical systems. *Autom.*, 67:54–66, 2016. doi: 10.1016/j.automatica.2016.01.022. URL `https://doi.org/10.1016/j.automatica.2016.01.022`.

I. Kottasová. A truck without a cab and driver takes to the road in sweden. *CNN Business*. URL `https://edition.cnn.com/2019/05/15/tech/einride-self-driving-trucks/index.html`.

D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27(3): 333 – 354, 1983. ISSN 0304-3975. Special Issue Ninth International Colloquium on Automata, Languages and Programming (ICALP) Aarhus, Summer 1982.

O. Kupferman and M. Y. Vardi. Synthesizing distributed systems. In *16th Annual IEEE Symposium on Logic in Computer Science, Boston, Massachusetts, USA, June 16-19, 2001, Proceedings*, pages 389–398. IEEE Computer Society, 2001. doi: 10.1109/LICS.2001.932514. URL `https://doi.org/10.1109/LICS.2001.932514`.

M. Lahijanian, S. B. Andersson, and C. Belta. Formal verification and synthesis for discrete-time stochastic systems. *IEEE Transactions on Automatic Control*, 60(8): 2031–2045, Aug 2015.

A. Lavaei, S. Soudjani, and M. Zamani. Compositional construction of infinite abstractions for networks of stochastic control systems. *Automatica*, 107:125 – 137, 2019. ISSN 0005-1098.

A. Lavaei, M. Khaled, S. Soudjani, and M. Zamani. AMYTISS: parallelized automated controller synthesis for large-scale stochastic systems. *CoRR*, abs/2005.06191, 2020a. URL `https://arxiv.org/abs/2005.06191`.

A. Lavaei, F. Somenzi, S. Soudjani, A. Trivedi, and M. Zamani. Formal controller synthesis for continuous-space mdps via model-free reinforcement learning. In *2020 ACM/IEEE 11th International Conference on Cyber-Physical Systems (ICCPS)*, pages 98–107. IEEE, 2020b.

K. Lesser and M. Oishi. Approximate safety verification and control of partially observable stochastic hybrid systems. *IEEE Transactions on Automatic Control*, 62(1):81–96, Jan 2017.

Y. Li and J. Liu. ROCS: A robustly complete control synthesis tool for nonlinear dynamical systems. In M. Prandini and J. V. Deshmukh, editors, *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (part of CPS Week), HSCC 2018, Porto, Portugal, April 11-13, 2018*, pages 130–135. ACM, 2018. doi: 10.1145/3178126.3178153. URL `https://doi.org/10.1145/3178126.3178153`.

J. Liu. Robust abstractions for control synthesis: Completeness via robustness for linear-time properties. In G. Frehse and S. Mitra, editors, *Proceedings of the 20th International*

Conference on Hybrid Systems: Computation and Control, HSCC 2017, Pittsburgh, PA, USA, April 18-20, 2017, pages 101–110. ACM, 2017. doi: 10.1145/3049797.3049826. URL https://doi.org/10.1145/3049797.3049826.

D. E. Long, A. Browne, E. M. Clarke, S. Jha, and W. R. Marrero. An improved algorithm for the evaluation of fixpoint expressions. In *International Conference on Computer Aided Verification*, pages 338–350. Springer, 1994.

R. Majumdar, N. Piterman, and A.-K. Schmuck. Environmentally-friendly GR(1) synthesis. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 229–246, Cham, 2019. Springer International Publishing.

R. Majumdar, K. Mallik, and S. Soudjani. Symbolic controller synthesis for Büchi specifications on stochastic systems. In *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*, pages 1–11, 2020a.

R. Majumdar, N. Ozay, and A. Schmuck. On abstraction-based controller design with output feedback. In A. Ames, S. A. Seshia, and J. Deshmukh, editors, *HSCC '20: 23rd ACM International Conference on Hybrid Systems: Computation and Control, Sydney, New South Wales, Australia, April 21-24, 2020*, pages 15:1–15:11. ACM, 2020b. doi: 10.1145/3365365.3382219. URL https://doi.org/10.1145/3365365.3382219.

O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems. In *STACS'95*, volume 900 of *LNCS*, pages 229–242. Springer, 1995.

K. Mallik, S. E. Z. Soudjani, A.-K. Schmuck, and R. Majumdar. Compositional construction of finite state abstractions for stochastic control systems. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 550–557. IEEE, 2017.

K. Mallik, A.-K. Schmuck, S. Soudjani, and R. Majumdar. Compositional synthesis of finite state abstractions. *IEEE Transactions on Automatic Control*, 2018.

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

D. A. Martin. The determinacy of blackwell games. *The Journal of Symbolic Logic*, 63 (4):1565–1581, 1998.

K. McMillan. Circular compositional reasoning about liveness. In *CHARME*, pages 342–346. Springer, 1999.

P. Meyer, H. Yin, A. H. Brodtkorb, M. Arcak, and A. J. Sørensen. Continuous and discrete abstractions for planning, applied to ship docking. *CoRR*, abs/1911.09773, 2019. URL http://arxiv.org/abs/1911.09773.

P. Miller. One small backflip for a robot is one giant leaping backflip for humankind. *Circuit Breaker*. URL https://www.theverge.com/circuitbreaker/2017/11/17/16671328/boston-dynamics-backflip-robot-atlas.

T. Moor, J. Raisch, and S. O'Young. Discrete supervisory control of hybrid systems based on l-complete approximations. *Discrete Event Dynamic Systems*, 12(1):83–107, 2002.

A. W. Mostowski. Regular expressions for infinite trees and a standard form of automata. In *Symposium on computation theory*, pages 157–168. Springer, 1984.

S. Mouelhi, A. Girard, and G. Gössler. Cosyma: a tool for controller synthesis using multi-scale abstractions. In *HSCC*, pages 83–88. ACM, 2013.

K. Namjoshi and R. Trefler. On the completeness of compositional reasoning methods. *ACM Trans. Comput. Log.*, 11(3):16:1–16:22, 2010.

P. Nilsson and N. Ozay. Incremental synthesis of switching protocols via abstraction refinement. In *53rd IEEE Conference on Decision and Control*, pages 6246–6253. IEEE, 2014.

P. Nilsson, O. Hussien, A. Balkan, Y. Chen, A. D. Ames, J. W. Grizzle, N. Ozay, H. Peng, and P. Tabuada. Correct-by-construction adaptive cruise control: Two approaches. *IEEE Trans. Control. Syst. Technol.*, 24(4):1294–1307, 2016. doi: 10.1109/TCST.2015.2501351. URL `https://doi.org/10.1109/TCST.2015.2501351`.

P. Nilsson, N. Ozay, and J. Liu. Augmented finite transition systems as abstractions for control synthesis. *Discrete Event Dynamic Systems*, 27(2):301–340, 2017.

N. Ozay, U. Topcu, and R. Murray. Distributed power allocation for vehicle management systems. In *CDC and ECC*, pages 4841–4848. IEEE, 2011.

N. Piterman and A. Pnueli. Faster solutions of Rabin and Streett games. In *21st Annual IEEE Symposium on Logic in Computer Science (LICS'06)*, pages 275–284, 2006.

N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of reactive (1) designs. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 364–380. Springer, 2006.

A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Annual ACM Symposium on Principles of Programming Languages*, pages 179–190. ACM Press, 1989.

A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthesize. In *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume II*, pages 746–757. IEEE Computer Society, 1990. doi: 10.1109/FSCS.1990.89597. URL `https://doi.org/10.1109/FSCS.1990.89597`.

G. Pola, A. Girard, and P. Tabuada. Approximately bisimilar symbolic models for nonlinear control systems. *Automatica*, 44(10):2508–2516, 2008.

S. Prajna and A. Jadbabaie. Safety verification of hybrid systems using barrier certificates. In R. Alur and G. J. Pappas, editors, *Hybrid Systems: Computation and Control, 7th International Workshop, HSCC 2004, Philadelphia, PA, USA,*

March 25-27, 2004, Proceedings, volume 2993 of *Lecture Notes in Computer Science*, pages 477–492. Springer, 2004. doi: 10.1007/978-3-540-24743-2_32. URL https://doi.org/10.1007/978-3-540-24743-2_32.

S. Prajna and A. Rantzer. Primal-dual tests for safety and reachability. In M. Morari and L. Thiele, editors, *Hybrid Systems: Computation and Control, 8th International Workshop, HSCC 2005, Zurich, Switzerland, March 9-11, 2005, Proceedings*, volume 3414 of *Lecture Notes in Computer Science*, pages 542–556. Springer, 2005. doi: 10.1007/978-3-540-31954-2_35. URL https://doi.org/10.1007/978-3-540-31954-2_35.

S. Prajna, A. Jadbabaie, and G. J. Pappas. A framework for worst-case and stochastic safety verification using barrier certificates. *IEEE Trans. Autom. Control.*, 52(8): 1415–1428, 2007. doi: 10.1109/TAC.2007.902736. URL https://doi.org/10.1109/TAC.2007.902736.

R. Price. A self-driving uber car hit and killed a woman in the first known autonomous-vehicle death. *Business Insider*. URL https://www.businessinsider.com/uber-arizona-woman-dies-after-behing-hit-self-driving-car-report-2018-3?r=DE&IR=T.

J.-P. Queille and J. Sifakis. Fairness and related properties in transition systems–a temporal logic to deal with fairness. *Acta Informatica*, 19(3):195–220, 1983.

M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969.

H. Ravanbakhsh and S. Sankaranarayanan. A class of control certificates to ensure reach-while-stay for switched systems. In D. Fisman and S. Jacobs, editors, *Proceedings Sixth Workshop on Synthesis, SYNT@CAV 2017, Heidelberg, Germany, 22nd July 2017*, volume 260 of *EPTCS*, pages 44–61, 2017. doi: 10.4204/EPTCS.260.6. URL https://doi.org/10.4204/EPTCS.260.6.

H. Reese. Uber's driverless rides in pittsburgh: What's happening and what it means. *TechRepublic*. URL https://www.techrepublic.com/article/ubers-driverless-rides-in-pittsburgh-whats-happening-and-what-it-means/.

G. Reissig, A. Weber, and M. Rungger. Feedback refinement relations for the synthesis of symbolic controllers. *TAC*, 62(4):1781–1796, 2017.

P. Roy, P. Tabuada, and R. Majumdar. Pessoa 2.0: a controller synthesis tool for cyber-physical systems. In M. Caccamo, E. Frazzoli, and R. Grosu, editors, *Proceedings of the 14th ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2011, Chicago, IL, USA, April 12-14, 2011*, pages 315–316. ACM, 2011. doi: 10.1145/1967701.1967748. URL https://doi.org/10.1145/1967701.1967748.

M. Rungger and M. Zamani. SCOTS: A tool for the synthesis of symbolic controllers. In *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*, HSCC '16, pages 99–104, New York, NY, USA, 2016. ACM.

A. Salamati, A. Lavaei, S. Soudjani, and M. Zamani. Data-driven verification and synthesis of stochastic systems through barrier certificates. *CoRR*, abs/2111.10330, 2021a. URL `https://arxiv.org/abs/2111.10330`.

A. Salamati, S. Soudjani, and M. Zamani. Data-driven verification of stochastic linear systems with signal temporal logic constraints. *Autom.*, 131:109781, 2021b. doi: 10.1016/j.automatica.2021.109781. URL `https://doi.org/10.1016/j.automatica.2021.109781`.

A. Saoud, A. Girard, and L. Fribourg. On the composition of discrete and continuous-time assume-guarantee contracts for invariance. In *ECC*, pages 435–440. IEEE, 2018a.

A. Saoud, A. Girard, and L. Fribourg. Contract based design of symbolic controllers for vehicle platooning. In M. Prandini and J. V. Deshmukh, editors, *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (part of CPS Week), HSCC 2018, Porto, Portugal, April 11-13, 2018*, pages 277–278. ACM, 2018b. doi: 10.1145/3178126.3187001. URL `https://doi.org/10.1145/3178126.3187001`.

A. Saoud, P. Jagtap, M. Zamani, and A. Girard. Compositional abstraction-based synthesis for interconnected systems: An approximate composition approach. *CoRR*, abs/2002.02014, 2020. URL `https://arxiv.org/abs/2002.02014`.

F. Somenzi. Cudd 3.0. 0. *URL http://vlsi. colorado. edu/˜ fabio/CUDD/html/. Also available at https://github. com/ivmai/cudd*, 2019.

S. Soudjani and A. Abate. Higher-Order Approximations for Verification of Stochastic Hybrid Systems. In S. Chakraborty and M. Mukund, editors, *Automated Technology for Verification and Analysis*, volume 7561 of *Lecture Notes in Computer Science*, pages 416–434. Springer Verlag, Berlin Heidelberg, 2012.

S. Soudjani and A. Abate. Adaptive and sequential gridding procedures for the abstraction and verification of stochastic processes. *SIAM Journal on Applied Dynamical Systems*, 12(2):921–956, 2013.

S. Soudjani, A. Abate, and R. Majumdar. Dynamic Bayesian networks for formal verification of structured stochastic processes. *Acta Informatica*, 54(2):217–242, Mar 2017.

S. E. Z. Soudjani, C. Gevaerts, and A. Abate. FAUST ^{2} : Formal abstractions of uncountable-state stochastic processes. In C. Baier and C. Tinelli, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, volume 9035 of *Lecture Notes in Computer Science*, pages 272–286. Springer, 2015. doi: 10.1007/978-3-662-46681-0_23. URL `https://doi.org/10.1007/978-3-662-46681-0_23`.

M. Svorenová, J. Kretínský, M. Chmelik, K. Chatterjee, I. Cerná, and C. Belta. Temporal logic control for stochastic linear systems using abstraction refinement of probabilistic games. In A. Girard and S. Sankaranarayanan, editors, *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control, HSCC'15, Seattle, WA, USA, April 14-16, 2015*, pages 259–268. ACM, 2015. doi: 10.1145/2728606.2728608. URL `https://doi.org/10.1145/2728606.2728608`.

P. Tabuada. *Verification and Control of Hybrid Systems - A Symbolic Approach*. Springer, 2009. ISBN 978-1-4419-0223-8. URL `http://www.springer.com/mathematics/applications/book/978-1-4419-0223-8`.

Y. Tazaki and J.-I. Imura. Bisimilar finite abstractions of interconnected systems. *HSCC'08*, pages 514–527, 2008.

J. G. Thistle. On control of systems modelled as deterministic Rabin automata. *Discrete Event Dyn. Systems*, 5(4):357–381, 1995.

J. G. Thistle and R. Malhamé. Control of ω-automata under state fairness assumptions. *Systems & control letters*, 33(4):265–274, 1998.

W. Thomas. On the synthesis of strategies in infinite games. In *STACS'95 Munich, Germany*, pages 1–13. 1995.

I. Tkachev and A. Abate. Regularization of Bellman equations for infinite-horizon probabilistic properties. In *Proceedings of the 15th ACM international conference on Hybrid Systems: computation and control*, pages 227–236, Beijing, PRC, April 2012.

I. Tkachev and A. Abate. Characterization and computation of infinite-horizon specifications over markov processes. *Theoretical Computer Science*, 515:1–18, 2014.

I. Tkachev, A. Mereacre, J.-P. Katoen, and A. Abate. Quantitative model-checking of controlled discrete-time Markov processes. *Information and Computation*, 253:1 – 35, 2017. ISSN 0890-5401.

T. van Dijk and J. van de Pol. Sylvan: Multi-core decision diagrams. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 677–691. Springer, 2015.

M. Y. Vardi. Automatic verification of probabilistic concurrent finite state programs. In *26th Annual Symposium on Foundations of Computer Science*, pages 327–338. IEEE, 1985.

A. P. Vinod and M. M. K. Oishi. Scalable underapproximative verification of stochastic LTI systems using convexity and compactness. In *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (Part of CPS Week)*, HSCC '18, pages 1–10, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5642-8.

A. Weber, M. Rungger, and G. Reissig. Optimized state space grids for abstractions. *TAC*, 2016.

P. Wieland and F. Allgöwer. Constructive safety using control barrier functions. *IFAC Proceedings Volumes*, 40(12):462–467, 2007.

T. Wongpiromsarn, U. Topcu, N. Ozay, H. Xu, and R. M. Murray. Tulip: a software toolbox for receding horizon temporal logic planning. In *Proceedings of the 14th international conference on Hybrid systems: computation and control*, pages 313–314, 2011.

L. Yang, A. Y. Karnik, B. Pence, M. T. B. Waez, and N. Ozay. Fuel cell thermal management: Modeling, specifications and correct-by-construction control synthesis. In *2017 American Control Conference, ACC 2017, Seattle, WA, USA, May 24-26, 2017*, pages 1839–1846. IEEE, 2017. doi: 10.23919/ACC.2017.7963220. URL `https://doi.org/10.23919/ACC.2017.7963220`.

Z. Yang, M. Wu, and W. Lin. An efficient framework for barrier certificate generation of uncertain nonlinear hybrid systems. *Nonlinear Analysis: Hybrid Systems*, 36:100837, 2020.

M. Zamani, P. M. Esfahani, A. Abate, and J. Lygeros. Symbolic models for stochastic control systems without stability assumptions. In *European Control Conference, ECC 2013, Zurich, Switzerland, July 17-19, 2013*, pages 4257–4262. IEEE, 2013. URL `http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=6669650`.

M. Zamani, P. M. Esfahani, R. Majumdar, A. Abate, and J. Lygeros. Symbolic control of stochastic systems via approximately bisimilar finite abstractions. *IEEE Trans. Autom. Control.*, 59(12):3135–3150, 2014. doi: 10.1109/TAC.2014.2351652. URL `https://doi.org/10.1109/TAC.2014.2351652`.

M. Zamani, A. Abate, and A. Girard. Symbolic models for stochastic switched systems: A discretization and a discretization-free approach. *Autom.*, 55:183–196, 2015. doi: 10.1016/j.automatica.2015.03.004. URL `https://doi.org/10.1016/j.automatica.2015.03.004`.

W. Zielonka. Perfect-information stochastic parity games. In *International Conference on Foundations of Software Science and Computation Structures*, pages 499–513. Springer, 2004.

D. Zonetti, A. Saoud, A. Girard, and L. Fribourg. A symbolic approach to voltage stability and power sharing in time-varying DC microgrids. In *18th European Control Conference, ECC 2019, Naples, Italy, June 25-28, 2019*, pages 903–909. IEEE, 2019. doi: 10.23919/ECC.2019.8796095. URL `https://doi.org/10.23919/ECC.2019.8796095`.

Appendices

A. Supplementary Material for Chap. 7

A.1. Example-Computation of the Rabin Fixed-Point

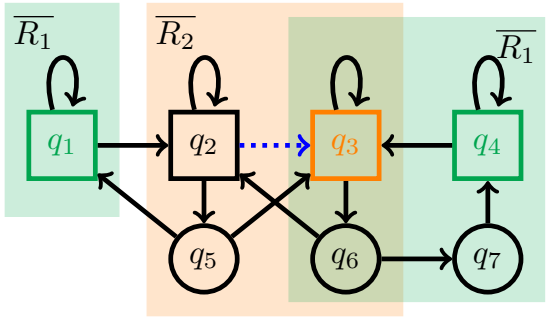


Figure A.1.: Example of a fair adversarial Rabin game with two pairs $\langle G_1, R_1\rangle = \langle\{q_1, q_4\}, \{q_2, q_5\}\rangle$ (G_1 and $\overline{R_1}$ are indicated in green) and $\langle G_2, R_2\rangle = \langle\{q_3\}, \{q_1, q_4, q_7\}\rangle$ (G_2 and $\overline{R_2}$ are indicated in orange), and one live edge $E^\ell = \{(q_2, q_3)\}$ (dashed blue).

Consider the game graph depicted in Fig. A.1, where circles and squares denote *Player* 0 and *Player* 1 vertices, respectively. We are given a Rabin condition with two pairs $\mathcal{R} = \{\langle G_1, R_1\rangle, \langle G_2, R_2\rangle\}$ s.t.

$$\overline{R_1} = \{q_1, q_3, q_4, q_6, q_7\} \qquad G_1 = \{q_1, q_4\} \qquad \overline{R_2} = \{q_2, q_3, q_5, q_6\} \qquad G_2 = \{q_3\}$$

which are indicated in green and orange, respectively, in Fig. A.1. The only live edge in the game graph is indicated in dashed blue from q_2 to q_3. We assert that *Player* 0 wins from every vertex. However, in the absence of the live edge, she wins only from $\{q_3, q_4, q_5, q_6, q_7\}$. (This is because *Player* 1 can force the game to stay forever in q_2 from the remaining states.)

We first flatten the algorithm in (7.4) for two Rabin pairs. This yields the following

algorithm:

$$\nu Y_0.\ \mu X_0. \tag{A.1a}$$
$$\big\{\nu Y_1.\mu X_1.\ \nu Y_2.\mu X_2. \tag{A.1b}$$
$$\quad Apre(Y_0, X_0)$$
$$\quad \cup\ \big(\overline{R}_1 \cap [(G_1 \cap Cpre(Y_1)) \cup (Apre(Y_1, X_1))]\big)$$
$$\quad \cup\ \big(\overline{R}_1 \cap \overline{R}_2 \cap [(G_2 \cap Cpre(Y_2)) \cup (Apre(Y_2, X_2))]\big)$$
$$\cup\ \nu Y_2'.\mu X_2'.\ \nu Y_1'.\mu X_1'. \tag{A.1c}$$
$$\quad Apre(Y_0, X_0)$$
$$\quad \cup\ \big(\overline{R}_2 \cap \big[\big(G_2 \cap Cpre(Y_2')\big) \cup \big(Apre(Y_2', X_2')\big)\big]\big)$$
$$\quad \cup\ \big(\overline{R}_1 \cap \overline{R}_2 \cap \big[\big(G_1 \cap Cpre(Y_1')\big) \cup \big(Apre(Y_1', X_1')\big)\big]\big)\ \big\}$$

We first consider the upper part of (A.1), i.e., the permutation sequence $\delta = 012$ (labeled by (A.1b)). We first recall that the computation is initialized with $Y_i^0 = V$ and $X_i^0 = \emptyset$ and we see from the structure of the game graph that $Cpre(V) = V$. Further, we see from the definition of $Apre$ that $Apre(\cdot, \emptyset) = \emptyset$. So, we have

$$X_2^1 = (\overline{R}_1 \cap G_1) \cup (\overline{R}_1 \cap \overline{R}_2 \cap G_2) = \{q_1, q_4\} \cup \{q_3\} = \{q_1, q_3, q_4\}.$$

As q_6 is the only other state in $\overline{R}_1 \cap \overline{R}_2 \cap G_2$ and q_6 does not have an edge to $\{q_1, q_3, q_4\}$ the iteration over X_2 terminates and we get $Y_2^1 = \{q_1, q_3, q_4\}$. As $q_3 \notin Cpre(Y_2^1)$ the last line of the upper part of (A.1) becomes the empty set and we terminate with $Y_2^* = X_2^* = (\overline{R}_1 \cap G_1) = \{q_1, q_4\}$. This gives $X_1^1 = \{q_1, q_4\}$ and resets Y_2 and X_2 to V and $\emptyset$, respectively. Therefore, we now get

$$X_2^1 = (\overline{R}_1 \cap G_1) \cup Apre(Q, X_1^1) \cup (\overline{R}_1 \cap \overline{R}_2 \cap G_2) = \{q_1, q_4\} \cup \{q_7\} \cup \{q_3\}.$$

Now, as $q_7 \in X_2^1$, also q_6 is added before X_2 terminates. This now gives $Y_2^1 = \{q_1, q_3, q_4, q_6, q_7\}$ and hence $q_3 \in Cpre(Y_2^1)$. As there are no other states in $\overline{R}_1 \cap \overline{R}_2 \cap G_2$ that can be added to this set, the iteration over X_2 terminates and we get $Y_2^2 = \{q_1, q_3, q_4, q_6, q_7\}$, which also terminates the iteration over Y_2, resulting in $X_1^2 = \{q_1, q_3, q_4, q_6, q_7\}$. As there are again no other states inside $\overline{R}_1$ that could be added, this iteration over X_1 terminates, giving $Y_1^1 = \{q_1, q_3, q_4, q_6, q_7\}$. Now we see that $Cpre(Y_1^1) = \{q_3, q_4, q_6, q_7\}$. As the exclusion of q_1 from Y_1 does not influence the reasoning about $\{q_3, q_4, q_6, q_7\}$ the iteration terminates with $Y_1^* = \{q_3, q_4, q_6, q_7\}$.

Now we consider the lower part of (A.1), i.e., the permutation sequence $\delta = 021$ (labeled by (A.1c)). Here, we get

$$X_1'^1 = (\overline{R}_2 \cap G_2) \cup (\overline{R}_1 \cap \overline{R}_2 \cap G_1) = \{q_3\} \cup \emptyset = \{q_3\}.$$

For the same reason as before we see again that the last line of the lower part of (A.1) becomes the empty set and we terminate with $Y_1'^* = X_1'^* = (\overline{R}_2 \cap G_2) = \{q_3\}$. This gives $X_2'^1 = \{q_3\}$ and resets Y_1' and X_1' to V and $\emptyset$, respectively. With this, we now get

$$X_1'^2 = (\overline{R}_2 \cap G_2) \cup Apre(Q, X_2'^1) \cup (\overline{R}_1 \cap \overline{R}_2 \cap G_1) = \{q_3\} \cup \{q_2, q_5\} \cup \emptyset.$$

Here, for the first time, the live edge from q_2 to q_3 comes into play. If this would not be a live edge, q_2 would not be added to X'_1, as in this case the environment could trap the game in q_2, and thereby prevent the second Rabin pair to hold. However, due to the edge from q_2 to q_3 being live, we know that the environment will always eventually transition from q_2 to q_3. With this, now also q_6 is added to X'_1, finally leading to a termination of the iteration over X'_2 with $\{q_2, q_3, q_5, q_6\}$ and hence $Y'^1_2 = \{q_2, q_3, q_5, q_6\}$. As $q_3 \in \mathit{Cpre}(Y'^1_2)$ the iteration over Y'_2 terminates with $Y'^*_2 = \{q_2, q_3, q_5, q_6\}$.

With both the upper and the lower part of (A.1) terminated, we can now take the union of $Y^*_1 = \{q_3, q_4, q_6, q_7\}$ and $Y'^*_2 = \{q_2, q_3, q_5, q_6\}$ to get $X^1_0 = \{q_2 \dots q_7\}$ (reaching the part of the formula labeled with (A.1a)). After this update of X_0 all inner fixpoint variables (in (A.1b) and (A.1c)) are reset, and the upper and lower expressions in (A.1) are re-evaluated. As $\mathit{Apre}(Q, X^1_0) = \{q_2 \dots q_7\}$, we see that every iteration over X_i in (A.1b) and (A.1c) is essentially initialized with a set containing $\{q_2 \dots q_7\}$. This implies that q_1 will actually remain within Y_1, leading to $Y^*_1 = V$, and with this $X^2_0 = V$. As this implies $Y^1_0 = V = Y^0_0$, the computation terminates with $Z^* = V$.

Despite all states being winning, we see that *Player* 0 has to play appropriately to enforce winning. Intuitively, from state q_5 she must go to q_3 and from q_6 she has to consistently either (i) always go to q_2 or (ii) always go to q_7. If she picks option (i), the play is won by satisfying the second Rabin pair, i.e., always eventually visiting q_3 while remaining within $\overline{R_2}$. If she picks option (ii), it is up to the environment whether the game is won by satisfying the first or the second Rabin pair. Intuitively, if the environment plays such that either (a) the game eventually remains in q_4 or (b) the edges (q_4, q_3) and (q_3, q_6) are taken infinitely often, the game fulfills the first Rabin condition. If, however, (c), the environment decides to trap the game in q_3, the game is won by satisfying the second Rabin pair. This influence of the environment on the selection of the satisfied Rabin pair intuitively requires the evaluation of all possible permutation sequences in the evaluation of the fixpoint algorithm. We will see later that for Rabin pairs which are ordered by inclusion (corresponding to the special case of a Rabin-chain condition), no permutation is required.

We comment that the strategy construction outlined in Thm. A.4 provided in App. A.2.3 chooses to enforce a transition from q_6 to q_7 (see Example A.1 in App. A.2.3 for a detailed discussion).

A.2. Detailed Proofs

A.2.1. General Lemmas

We first introduce some useful general lemmas.

Lemma A.1 *If* $Y \supseteq X$ *then* $\mathit{Cpre}(Y) \cup \mathit{Apre}(Y, X) = \mathit{Cpre}(Y)$.

PROOF The claim follows from the following derivation

$$\begin{aligned} \mathit{Cpre}(Y) \cup \mathit{Apre}(Y,X) &= \mathit{Cpre}(Y) \cup \mathit{Cpre}(X) \cup \Big(\mathrm{Lpre}^{\exists}(X) \cap \mathrm{Pre}_1^{\forall}(Y)\Big) \\ &= \mathit{Cpre}(Y) \cup \Big(\mathrm{Lpre}^{\exists}(X) \cap \mathrm{Pre}_1^{\forall}(Y)\Big) \\ &= \Big(\mathit{Cpre}(Y) \cup \mathrm{Lpre}^{\exists}(X)\Big) \cap \Big(\mathit{Cpre}(Y) \cup \mathrm{Pre}_1^{\forall}(Y)\Big) \\ &= \Big(\mathit{Cpre}(Y) \cup \mathrm{Lpre}^{\exists}(X)\Big) \cap \mathit{Cpre}(Y) \\ &= \mathit{Cpre}(Y) \end{aligned}$$

where the second line follows from $\mathit{Cpre}(X) \subseteq \mathit{Cpre}(Y)$ (as $X \subseteq Y$) and the forth line follows as $\mathit{Cpre}(Y) = \mathrm{Pre}_0^{\exists}(Y) \cup \mathrm{Pre}_1^{\forall}(Y) \supseteq \mathrm{Pre}_1^{\forall}(Y)$. □

Lemma A.2 *If $Y \subseteq X$ then $\mathit{Apre}(Y,X) = \mathit{Cpre}(X)$.*

PROOF The claim follows from the following derivation

$$\begin{aligned} \mathit{Apre}(Y,X) &= \mathit{Cpre}(X) \cup \Big(\mathrm{Lpre}^{\exists}(X) \cap \mathrm{Pre}_1^{\forall}(Y)\Big) \\ &= \Big(\mathit{Cpre}(X) \cup \mathrm{Lpre}^{\exists}(X)\Big) \cap \Big(\mathit{Cpre}(X) \cup \mathrm{Pre}_1^{\forall}(Y)\Big) \\ &= \Big(\mathit{Cpre}(X) \cup \mathrm{Lpre}^{\exists}(X)\Big) \cap \mathit{Cpre}(X) \\ &= \mathit{Cpre}(X) \end{aligned}$$

where the fourth line follows as $\mathit{Cpre}(X) = \mathrm{Pre}_0^{\exists}(X) \cup \mathrm{Pre}_1^{\forall}(X) \supseteq \mathrm{Pre}_1^{\forall}(Y)$ as $Y \subseteq X$. □

Lemma A.3 *Let $f(X,Y)$ and $g(X,Y)$ be two functions which are monotone in both $X \subseteq V$ and $Y \subseteq V$. Further, let*

$$\begin{aligned} Z_a :=& \nu Y_a.\ \mu X_a.\ \nu Y_b.\ \mu X_b.\ f(X_a, Y_a) \cup g(X_b, Y_b) \\ Z_b :=& \nu Y_a.\ \mu X_a.\ \nu Y_b.\ \mu X_b.\ g(X_a, Y_a) \cup f(X_b, Y_b) \\ Z_c :=& \nu Y_c.\ \mu X_c.\ f(X_c, Y_c) \end{aligned}$$

Then it holds that
(i) $Z_c \subseteq Z_a$ *and*
(ii) $Z_c \subseteq Z_b$.
If, in addition, $g(X,Y) \subseteq f(X,Y)$ for all $X, Y \subseteq V$, then it holds that
(iii) $Z_a = Z_c$ *and*
(iv) $Z_b = Z_c$.

PROOF We prove all claims separately:
▶ *(i)* "$Z_c \subseteq Z_a$" : First, consider a stage of the fixed point evaluation where Y_a and X_a have their initialization value $Y_a^0 := V$ and $X_a^{00} := \emptyset$ (here, the notation X_a^{lk} refers to the value of X_a computed in the k'th iteration over X_a using the value for Y_a computed in the

l'th iteration over Y_a). Then we see that $X_a^{01} = Y_b^{00*}$ where $Y_b^{00*} = f(\emptyset, V) \cup g(Y_b^{00*}, Y_b^{00*})$. We therefore see that $X_a^{01} \supseteq X_c^{01} = f(\emptyset, V)$. With this, it follows from the monotonicity of f and g that $Y_a^{01} = X_a^{0*} \supseteq X_c^{0*} = Y_c^1$. With this, we see that $X_a^{m1} \supseteq X_c^{m1}$ for all $m > 0$ and therefore $Z_a = Y_a^* \supseteq Y_c^* = Z_c$.
▶ *(ii)* "$Z_c \subseteq Z_b$" : Consider arbitrary values Y_a^m and X_a^{mn} and assume that Y_b and X_b have their initialisation value, i.e., $Y_b^{mn0} := V$ and $X_b^{mn00} := \emptyset$. Then we have

$$X_b^{mn01} = g(X_a^{mn}, Y_a^m) \cup f(\emptyset, V) \supseteq X_c^{01}.$$

Using the same reasoning as in the previous part, we see that this implies $Y_b^{mn*} \supseteq Y_c^* = Z_c$. As this holds for any m and n it also holds when the fixed point over Y_a and X_a is obtained, i.e., when we have $Z_a = Y_a^* = Y_b^{***}$, which proves the statement.
▶ *(iv)* "$Z_c \supseteq Z_b$" : First, observe that for the initialization values $Y_a^0 = Y_b^{000} = V$ and $X_a^{00} = X_b^{0000} = \emptyset$ we have $g(\emptyset, V) \subseteq f(\emptyset, V)$. We therefore have

$$Y_b^{00*} = X_b^{00**} = f(X_b^{00**}, Y_b^{00*}) = Z_c$$

Now it remains to show, that the outer fixed-point cannot add any additional states. First, observe that $X_a^{01} = Y_b^{00*}$ and

$$X_b^{0100} = g(X_a^{01}, V) \cup f(\emptyset, V) \subseteq f(X_a^{01}, V) \cup f(\emptyset, V) = f(X_a^{01}, V)$$

Now it follows from the famous acceleration result of Long et al. (1994) that warm-starting the inner fixed-point computation with X_a^{01} yields the same inner fixed-point. With this, we see that $X_a^{0n} = Z_c$ for all n, implying $Y_a^0 = X_a^{0*} = Z_c$. As $Z_b = Y_a^* \subseteq Y_a^0$, this proves the claim.
▶ *(iii)* "$Z_c \supseteq Z_a$" : As $g(X, Y) \subseteq f(X, Y)$ for all $X, Y \subseteq V$ it follows from the monotonicity of g and f that

$$Z_a \subseteq \nu Y_a.\ \mu X_a.\ \nu Y_b.\ \mu X_b.\ f(X_a, Y_a) \cup f(X_b, Y_b)$$

with this, it follows from (iv) that $Z_a \subseteq Z_c$, what proves the claim. □

A.2.2. Additional Proofs for Sec. 7.2

Proof of Thm. 7.3

Theorem A.1 (Thm. 7.3 restated for convenience) *Let $\mathcal{G}^\ell = \langle \mathcal{G}, E^\ell \rangle$ be a game graph with live edges and $\langle T, Q \rangle$ be a safe reachability winning condition. Further, let*

$$Z^* \coloneqq \nu Y.\ \mu X.\ T \cup (Q \cap Apre(Y, X)). \tag{A.2}$$

Then Z^* is equivalent to the winning region of Player 0 in the fair adversarial game over $\mathcal{G}^\ell$ for the winning condition ψ in (7.8). Moreover, the fixpoint algorithm runs in $O(n^2)$ symbolic steps, and a memoryless winning strategy for Player 0 can be extracted from it.

We denote by Y^m the m-th iteration over the fixpoint variable Y in (A.2), where $Y^0 = V$. Further, we denote by X^{mi} the set computed in the i-th iteration over the fixpoint variable X in (A.2) during the computation of Y^m where $X^{m0} = \emptyset$. Then it follows form (A.2) that

$$X^{m1} = X^{m0} \cup T \cup (Q \cap \mathit{Apre}(Y^{m-1}, X^{m0})) = \emptyset \cup T \cup (Q \cap \mathit{Apre}(Y^m, \emptyset)) = T,$$
$$X^{m2} = X^{m1} \cup T \cup (Q \cap \mathit{Apre}(Y^{m-1}, X^{m1})) = T \cup (Q \cap \mathit{Apre}(Y^{m-1}, X^{m1})) \supseteq X^{m1},$$

and therefore, in general,

$$X^{mi+1} = T \cup (Q \cap \mathit{Apre}(Y^{m-1}, X^{mi})) \supseteq X^{mi}.$$

With this, the fixed point over X corresponds to the set $X^{m*} = \bigcup_{i>0} X^{mi} = X^{mi^{\uparrow}}$, where $i^{\uparrow}$ is the iteration where the fixed point over X^{mi} is attained.

Now consider the computation of Y. Here we have $Y^0 = V$ and $Y^m = Y^{m-1} \cap X^{m*} \subseteq Y^{m-1}$ where equality holds when a fixed point is reached. Hence, in particular we have $Y^* = X^{**} = Z^*$. For simplicity we denote X^{*i} by X^i.

Strategy construction. In order to construct a winning strategy for *Player* 0 from (A.2), we construct a ranking over V by choosing

$$\mathrm{rank}(v) = i \;\Leftrightarrow\; v \in X^i \setminus X^{i-1} \quad \text{and} \quad \mathrm{rank}(v) = \infty \;\Leftrightarrow v \notin Z^*. \tag{A.3}$$

As $X^0 = \emptyset$, $X^1 = T$ (from above) and $Z^* = \bigcup_{i>0} X^i$, it follows that $\mathrm{rank}(v) = 1$ iff $v \in T$ and $1 < \mathrm{rank}(v) < \infty$ iff $v \in Z^* \setminus T$. Using this ranking we define a *Player* 0 strategy $\rho_0 : V_0 \to V$ s.t.

$$\rho_0(v) = \min_{(v,w)\in E} \mathrm{rank}(w)\,. \tag{A.4}$$

We next show that this player 0 strategy is actually winning w.r.t. ψ (in (7.8)) in every fair adversarial play over $\mathcal{G}^\ell$.

Soundness. To prove soundness, we need to show $Z^* \subseteq \mathcal{W}$. That is, we need to show that for all $v \in Z^*$ there exists a strategy for player 0 s.t. the goal set T is eventually reached along all live compliant plays π starting at v while staying in Q. We choose ρ_0 in (A.4) and show that the claim holds.

First, it follows from the definition of *Apre* that for a vertex $v \in Z^*$ exactly one of the following cases holds:
(a) $v \in T$ and hence $\mathrm{rank}(v) = 1$,
(b) $v \in (V_0 \cap Z^*) \setminus T$, i.e., $1 < \mathrm{rank}(v) < \infty$ and $v \in Q$ and there exists a $v' \in E(v)$ with $\mathrm{rank}(v') < \mathrm{rank}(v)$,
(c) $v \in ((V_1 \setminus V^\ell) \cap Z^*)) \setminus T$, i.e., $1 < \mathrm{rank}(v) < \infty$ and $v \in Q$ and for all $v' \in E(v)$ it holds that $\mathrm{rank}(v') < \mathrm{rank}(v)$, or
(ℓ) $v \in (V^\ell \cap Z^*) \setminus T$, i.e., $1 < \mathrm{rank}(v) < \infty$ and $v \in Q$ and there exists a $v' \in E^\ell(v)$ with $\mathrm{rank}(v') < \mathrm{rank}(v)$ *and* $E(v) \subseteq Z^*$.
We see that $\rho_0(v)$ chooses one existentially quantified edge in (b) vertices. In all other cases player 1 chooses the successor.

Further, we see that any play π which starts in $\pi(0) = v \in Z^*$ and obeys ρ_0 has the property that $\pi(k) \in Z^* \setminus T$ implies $\pi(k) \in Q$ and $\pi(k+1) \in Z^*$ for all $k \geq 0$. This, in turn, means that for any such state $v = \pi(k) \in Z^* \setminus T$ as well as for its successor $\pi(k+1)$ a rank is defined, i.e., $\pi(k) \in X^i$ for some $0 < i < \infty$ and exactly one of the cases (b)-(ℓ) applies. We call a vertex for which case (α) applies, an (α) vertex.

Now observe that the above reasoning implies that whenever an (a) vertex is hit along a play π the claim holds. We therefore need to show that any play starting in $v \in Z^*$ eventually reaches an (a) vertex. First, consider a play in which no (ℓ) vertex occurs. Then constantly hitting (b) and (c) vertices always reduces the rank of visited states (as we assume that π obays ρ_0 in (A.4)). As the maximal rank is finite, we see that we must eventually hit a state with rank 1, which is an (a) state.

Note that the same argument holds when only a finite number of (ℓ) vertices is visited along π. In this case we know that from some time onward no more (ℓ) vertex occurs. As the last (ℓ) vertex has a finite rank, there can only be a finite sequence of (b) and (c) vertices afterwards until finally an (a) vertex is reached.

We are therefore left with showing that on every path with an infinite number of (ℓ) vertices, eventually an (a) vertex will be reached. We prove this claim by contradiction. I.e., we show that there cannot exist a path with infinitely many (ℓ) vertices and no (a) vertex.

We first show that infinitely many (ℓ) vertices and no (a) vertices in π imply that vertices with rank 2 can only occur finitely often along π.

▶ Recall that the construction of ρ_0 ensures that whenever we visit a state $v \in V_0 \cap Z^*$ with $\text{rank}(v) = 2$ we will surely visit a state with rank 1 afterwards, implying the occurrence of a vertex labeled (a). As no (a) labeled vertices are assumed to occur along π, no (b) vertices with $\text{rank}(v) = 2$ occur along π.

▶ Now assume that $v \in V_1 \cap Z^*$ with $\text{rank}(v) = 2$. If v is a (c) vertex all successor states will have rank 1. With the same reasoning as before, this cannot occur.

▶ Now assume that $v \in V_1 \cap Z^*$ with $\text{rank}(v) = 2$ is labeled with (ℓ). In this case there surely exists a successor v' of v s.t. $(v, v') \in E^\ell$ and $\text{rank}(v') = 1$. But there *might* also exist another successor v'' of v (i.e., $(v'' \in E(v))$ s.t. $\text{rank}(v'') > 1$. If there does not exists such a successor v'', all successors have rank 1 and we again cannot visit v.

▶ Now assume that $v \in V_1 \cap Z^*$ with $\text{rank}(v) = 2$, labeled with (ℓ) and there exists a successor $v'' \in E(v)$ s.t. $\text{rank}(v'') > 1$. Now let us assume that such a state v is visited infinitely often along π. As π is a fair adversarial play over G we know that visiting v infinitely often along π implies that v' with $(v, v') \in E^\ell$ and $\text{rank}(v') = 1$ (which surely exists by the definition of $Apre$) will also be visited infinitely often along π. This is again a contradiction to the above hypothesis and implies that such v's can only be visited finitely often.

▶ As V is a finite set, the set of states with rank 2 is finite. Hence, the occurrence of infinitely many states with rank 2 along π implies that one of the above cases must occur infinitely often, which gives a contradiction to the above hypothesis. Using the same arguments, we can inductively show that states with any fixed rank can only occur finitely often if states with rank 1 (i.e., (a)-labeled vertices) never occur. As the maximal rank is

finite (due to the finiteness of V) this contradicts the assumption that π is an infinite play.

We therefore conclude that along any *infinite* fair adversarial play π with infinitely many vertices labeled by (ℓ) we will eventually see a vertex labeled by (a).

Completeness. We now show that the fixpoint in (A.2) is complete, i.e., that every state in $\overline{Z}^* := V \setminus Z^*$ is loosing for *Player* 0. In particular, we show that from every vertex $v \in \overline{Z}^*$ *Player* 1 has a memoryless strategy ρ_1 s.t. all fair adversarial plays compliant with ρ_1 satisfy

$$\overline{\psi} := \neg\psi = \neg(Q\mathcal{U}T) = \Box\neg T \vee \neg T\mathcal{U}\neg Q \tag{A.5}$$

and are hence loosing for *Player* 0.

In order to prove the latter claim we fist compute $\overline{Z}^* := V \setminus Z^*$ by negating the fixed-point formula in (A.2). For this, we define $\overline{X}^* := V \setminus X$, $\overline{Y}^* := V \setminus Y$ and use the negation rule of the μ-calculus, i.e., $\neg(\mu X.f(X)) = \nu\overline{X}.V \setminus f(X)$ along with common De-Morgan laws. This results in the following derivation.

$$\overline{Z}^* = \mu\overline{Y}.\ \nu\overline{X}.\ \overline{T} \cap (\overline{Q} \cup V \setminus \mathit{Apre}(Y, X))$$

where

$$\begin{aligned}
&V \setminus \mathit{Apre}(Y, X)\\
&= V \setminus \Big[\mathit{Cpre}(X) \cup \Big(\mathrm{Lpre}^{\exists}(X) \cap \mathrm{Pre}_1^{\forall}(Y)\Big)\Big]\\
&= [V \setminus \mathit{Cpre}(X)] \cap \Big[V \setminus \Big(\mathrm{Lpre}^{\exists}(X) \cap \mathrm{Pre}_1^{\forall}(Y)\Big)\Big]\\
&= \Big[\mathrm{Pre}_1^{\exists}(\overline{X}) \cup \mathrm{Pre}_0^{\forall}(\overline{X})\Big] \cap \Big[V_0 \cup (V_1 \setminus V^\ell) \cup \Big(V^\ell \setminus \Big(\mathrm{Lpre}^{\exists}(X) \cap \mathrm{Pre}_\ell^{\forall}(Y)\Big)\Big)\Big]\\
&= \Big[\mathrm{Pre}_1^{\exists}(\overline{X}) \cup \mathrm{Pre}_0^{\forall}(\overline{X})\Big] \cap \Big[V_0 \cup (V_1 \setminus V^\ell) \cup \Big(\mathrm{Lpre}^{\forall}(\overline{X}) \cup \mathrm{Pre}_\ell^{\exists}(\overline{Y})\Big)\Big]\\
&= \mathrm{Pre}_0^{\forall}(\overline{X}) \cup \mathrm{Pre}_{1\setminus\ell}^{\exists}(\overline{X}) \cup \Big[\mathrm{Pre}_1^{\exists}(\overline{X}) \cap \Big(\mathrm{Lpre}^{\forall}(\overline{X}) \cup \mathrm{Pre}_\ell^{\exists}(\overline{Y})\Big)\Big]\\
&= \mathrm{Pre}_0^{\forall}(\overline{X}) \cup \mathrm{Pre}_{1\setminus\ell}^{\exists}(\overline{X}) \cup \Big[\mathrm{Pre}_\ell^{\exists}(\overline{X}) \cap \Big(\mathrm{Lpre}^{\forall}(\overline{X}) \cup \mathrm{Pre}_\ell^{\exists}(\overline{Y})\Big)\Big]\\
&= \mathrm{Pre}_0^{\forall}(\overline{X}) \cup \mathrm{Pre}_{1\setminus\ell}^{\exists}(\overline{X}) \cup \mathrm{Lpre}^{\forall}(\overline{X}) \cup \mathrm{Pre}_\ell^{\exists}(\overline{Y}).
\end{aligned}$$

The last line in the above derivation follows from the observation that $\mathrm{Lpre}^{\forall}(\overline{X}) \subseteq \mathrm{Pre}_l^{\exists}(\overline{X})$ and $\overline{Y} \subseteq \overline{X}$ for all iterations of the fixed-point. The additionally introduced pre-operators

are defined in close analogy to (2.8) and (7.2) as follows:

$$\begin{aligned}
\mathrm{Pre}_1^{\exists}(S) &\coloneqq \{v \in V_1 \mid E(v) \cap S \neq \emptyset\},\\
\mathrm{Pre}_0^{\forall}(S) &\coloneqq \{v \in V_0 \mid E(v) \subseteq S\},\\
\mathrm{Pre}_{1\setminus\ell}^{\exists}(S) &\coloneqq \{v \in V_1 \setminus V^\ell \mid E(v) \cap S \neq \emptyset\},\\
\mathrm{Pre}_{\ell}^{\exists}(S) &\coloneqq \{v \in V^\ell \mid E(v) \cap S \neq \emptyset\},\\
\mathrm{Pre}_{\ell}^{\forall}(S) &\coloneqq \{v \in V^\ell \mid E(v) \subseteq S\},\\
\mathrm{Lpre}^{\forall}(S) &\coloneqq \{v \in V^\ell \mid E^\ell(v) \subseteq S\}.
\end{aligned}$$

With this, we can conclude that

$$\overline{Z}^* = \mu\overline{Y}.\ \nu\overline{X}.\ \overline{T} \cap \left(\overline{Q} \cup \mathrm{Pre}_0^{\forall}(\overline{X}) \cup \mathrm{Pre}_{1\setminus\ell}^{\exists}(\overline{X}) \cup \mathrm{Lpre}^{\forall}(\overline{X}) \cup \mathrm{Pre}_{l}^{\exists}(\overline{Y})\right). \tag{A.6}$$

where $\overline{T} = V \setminus T$ and $\overline{Q} = V \setminus Q$.

Now denote by $\overline{Y}^m$ the m-th iteration over the fixpoint variable $\overline{Y}$ in (A.6), where $\overline{Y}^0 = \emptyset$. Further, we denote by $\overline{X}^{mi}$ the set computed in the i-th iteration over the fixpoint variable $\overline{X}$ in (A.6) during the computation of $\overline{Y}^m$ where $\overline{X}^{m0} = V$. After termination of the inner fixed point over $\overline{X}^{mi}$ we have by construction that $\overline{Y}^m = \overline{X}^{m*}$ and therefore

$$\overline{Y}^m = \overline{T} \cap \left(\overline{Q} \cup \mathrm{Pre}_0^{\forall}(\overline{Y}^m) \cup \mathrm{Pre}_{1\setminus\ell}^{\exists}(\overline{Y}^m) \cup \mathrm{Lpre}^{\forall}(\overline{Y}^m) \cup \mathrm{Lpre}^{\exists}(\overline{Y}^{m-1})\right). \tag{A.7}$$

Similar to the soundness proof, we define a ranking over V induced by the iterations of the smallest fixed-point, which now is $\overline{Y}$:

$$\overline{\mathrm{rank}}(v) = m \leftrightarrow v \in \overline{Y}^m \setminus \overline{Y}^{m-1} \quad \text{and} \quad \overline{\mathrm{rank}}(v) = \infty \ \leftrightarrow\ v \notin \overline{Z}^*.$$

This ranking can now be used to define a memoryless *Player* 1 strategy $\rho_1 : V_1 \to V$ s.t.

$$\rho_1(v) = \min_{(v,w)\in E} \overline{\mathrm{rank}}(w)\,. \tag{A.8}$$

Towards proving that ρ_1 is winning for $\overline{\psi}$ in (A.5) we first observe that for every vertex $v \in \overline{Z}^*$ exactly one of the following cases holds:
(a) $v \in (V_0 \cap \overline{Z}^* \cap \overline{T})$, i.e., $\overline{\mathrm{rank}}(v) < \infty$ and $v \in \overline{Q}$ or for all $v' \in E(v)$ it holds that $\overline{\mathrm{rank}}(v') \leq \overline{\mathrm{rank}}(v)$,
(b) $v \in ((V_1 \setminus V^\ell) \cap \overline{Z}^* \cap \overline{T}))$, i.e., $\overline{\mathrm{rank}}(v) < \infty$ and $v \in \overline{Q}$ or there exists $v' \in E(v)$ s.t. $\overline{\mathrm{rank}}(v') \leq \overline{\mathrm{rank}}(v)$, or
($\ell_\forall$) $v \in (V^\ell \cap Z^* \cap \overline{T})$ and $\overline{\mathrm{rank}}(v) < \infty$ and $v \in \overline{Q}$ or for all $v' \in E^\ell(v)$ holds that $\overline{\mathrm{rank}}(v') \leq \overline{\mathrm{rank}}(v)$
($\ell_\exists$) $v \in (V^\ell \cap Z^* \cap \overline{T})$ and $\overline{\mathrm{rank}}(v) > 1$ (and $\overline{\mathrm{rank}}(v) < \infty$), and ($\ell_\forall$) does not hold, but there exists a $v' \in E(v)$ s.t. $\overline{\mathrm{rank}}(v') < \overline{\mathrm{rank}}(v)$.

Using this observation, we now show that every fair adversarial play π compliant with ρ_1 satisfies $\overline{\psi}$ in (A.5), that is, either stays in $\overline{T}$ forever, or eventually visits $\overline{Q}$ before visiting T.

First, observe that for every node $v \in \overline{Z}^*$ one of the cases (a),(b),($\ell_\forall$), or ($\ell_\exists$) holds. If v is an (a) vertex, we see that either $v \in \overline{Q}$ or for all choices of *Player* 0 (i.e., for any *Player* 0 strategy), the play remains in $\overline{Z}^* \subseteq \overline{T}$. Further, it is obvious that ρ_1 ensures, that whenever a (b) vertex is seen, the play remains in $\overline{Z}^* \subseteq \overline{T}$ if we do not already have $v \in \overline{Q}$. The same is true for ($\ell_\forall$) vertexes.

Now consider a fair adversarial play π that is compliant with ρ_1 and $\pi(0) \in \overline{Z}^* \subseteq \overline{T}$. Then it follows from the above intuition that for all visits to (a),(b),($\ell_\forall$) we have two cases: (i) Either $\overline{\psi}$ is immediately true on π by visiting $\overline{Q}$ (and having been in $\overline{Z}^* \subseteq \overline{T}$ in all previous time steps). In this case the suffix of π is irrelevant, because *Player* 0 has already lost (by visiting $\overline{Q}$ without seeing T). Or (ii) the play remains in $\overline{Z}^* \subseteq \overline{T}$. Now observe that this is also true for infinite visits to (a),(b),($\ell_\forall$) vertexes. As π is fair adversarial, visiting a ($\ell_\forall$) vertex infinitely often, implies that all live edges are taking infinitely often, which all ensure that the play remains in $\overline{Z}^* \subseteq \overline{T}$ or is immediately lost by visiting $\overline{Q}$. Therefore, the only interesting case occurs if π visits ($\ell_\exists$) vertexes. If such a vertex is visited finitely often, ρ_1 ensures that the play stays in $\overline{Z}^* \subseteq \overline{T}$. However, if they are visited infinitely often, a live edge that leaves $\overline{Z}^*$ will also be taken infinitely often. Hence, in order to ensure that π is loosing for *Player* 0, we need to show that ρ_1 enforces that ($\ell_\exists$) vertexes are only visited finitely often.

To see this, let v be an ($\ell_\exists$) vertex and observe that $\overline{\mathrm{rank}}(v)$ is finite and larger than 1. At the first visit of π to v, ρ_1 decreases the rank as it chooses by definition one of the existentially quantified successors $v' \in E^\ell(v)$ with $\overline{\mathrm{rank}}(v') < \overline{\mathrm{rank}}(v)$. Now observe that for all other cases (a),(b),($\ell_\forall$) either $\overline{Q}$ is visited and the play is immediately loosing for *Player* 0 or the play is kept in $\overline{Z}^* \subseteq \overline{T}$ and the strategy ρ_1 never increases the rank. As every vertex has a unique rank, ρ_1 ensures that every ($\ell_\exists$) vertex is visited at most once along every compliant fair adversarial play that remains in $\overline{Z}^* \subseteq \overline{T}$. This proves the claim.

Proof of Thm. 7.2

Theorem A.2 (Thm. 7.2 restated for convenience) *Let $\mathcal{G}^\ell = \langle \mathcal{G}, E^\ell \rangle$ be a game graph with live edges and $Q, G \subseteq V$ be two state sets over $\mathcal{G}$. Further, let*

$$Z^* \coloneqq \nu Y.\ \mu X.\ Q \cap [(G \cap \mathit{Cpre}(Y)) \cup (\mathit{Apre}(Y, X))]\,. \tag{A.9}$$

Then Z^* is equivalent to the winning region of Player 0 in the fair adversarial game over $\mathcal{G}^\ell$ for the winning condition ψ in (7.5). Moreover, the fixpoint algorithm runs in $O(n^2)$ symbolic steps, and a memoryless winning strategy for Player 0 can be extracted from it.

In order to simplify the proof of Prop. A.2.2, we first prove the following lemma.

Lemma A.4 *Let $Q, G \subseteq V$ and*

$$Z^* \coloneqq \nu Y.\mu X.Q \cap [(G \cap \mathit{Cpre}(Y)) \cup \mathit{Apre}(Y, X)] \tag{A.10a}$$

$$\widetilde{Z}^* \coloneqq \nu \widetilde{Y}.\nu Y.\mu X.Q \cap \left[\left(G \cap \mathit{Cpre}(\widetilde{Y})\right) \cup \mathit{Apre}(Y, X)\right]. \tag{A.10b}$$

Then $Z^* = \widetilde{Z}^*$.

PROOF To prove the claim we consider a third version of the fixed-point equation, namely

$$\check{Z}^* \coloneqq \nu\widetilde{Y}.\nu Y.\mu X.Q \cap \left[\left(G \cap \mathit{Cpre}(\widetilde{Y})\right) \cup (G \cap \mathit{Cpre}(Y)) \cup \mathit{Apre}(Y,X)\right].$$

Then it immediately follows from the monotonicity of all involved functions that $\widetilde{Z}^* \subseteq \check{Z}^*$. It further follows from Lem. A.3 (iv) that $Z^* = \check{Z}^*$. It therefore remains to show that $\check{Z}^* \subseteq \widetilde{Z}^*$ to prove the claim. We actually show $\check{Z}^* \subseteq \widetilde{Z}^*$.

Let $\widetilde{Y}^0 = Y^{00} = V$. Then it immediately follows that the computation of X^{00*} returns the same set for both fixed-points. It further follows that $Y^{0n} \subseteq \widetilde{Y}^0$, which implies $(G \cap \mathit{Cpre}(Y^{0n})) \subseteq (G \cap \mathit{Cpre}(\widetilde{Y}^0))$ and therefore the set $\widetilde{Y}^1$ coincides for both fixed-points. Now recall from Long et al. (1994) that warm-starting the inner fixed-point computation with the largest fixed-point retained from previous values of outer fixed-point variables, does not change the resulting fixed-point. With this, we can use $Y^{10} = \widetilde{Y}^1$ and observe that this implies that the computation of $\widetilde{Y}^2$ becomes again identical for both fixed-points. Re-applying this argument until termination shows, that indeed $\check{Z}^* \subseteq \widetilde{Z}^*$.□

With Lem. A.4 in place, we can use (A.10b) instead of (A.9) to prove Thm. 7.2. Further, let us define $Z^*(\langle T, Q\rangle)$ to be the set of states computed by the fixpoint algorithm in (7.9). Then we know that upon termination we have

$$\widetilde{Z}^* = \widetilde{Y}^* = Z^*(\left\langle Q \cap G \cap \mathit{Cpre}(\widetilde{Y}^*), Q \right\rangle). \tag{A.11}$$

Now we will use (A.11) to prove soundness and completeness of Thm. 7.2.

Soundness Let us now define $T := Q \cap G \cap \mathit{Cpre}(\widetilde{Y}^*))$. Pick any state $v \in \widetilde{Z}^*$ and the strategy ρ_0 defined as in (A.4) over the sets X^i computed in the last iteration over X when computing $Z^*(\langle T, Q\rangle)$. Further, let π be an arbitrary fair adversarial play starting in v and being compliant with ρ_0. Then we need to show that π fulfills ψ in (7.5).

Using (A.11) and the fact that $v \in \widetilde{Z}^*$ we know from Thm. 7.3 that π fulfills $Q\mathcal{U}T$. That is, there exists a $k \in \mathbb{N}$ s.t. $\pi(i) \in Q$ for all $i < k$ and $\pi(k) \in T = Q \cap G \cap \mathit{Cpre}(\widetilde{Y}^*))$. With this we know that (a) $\pi(k) \in Q$, (b) $\pi(k) \in G$ and (c) $v \in \mathit{Cpre}(\widetilde{Y}^*)$. Now we have two cases: (c.1) If $\pi(k) \in V^1$, then it follows from the definition of *Cpre* that $E(\pi(k)) \subseteq \widetilde{Y}^*$. As $\widetilde{Y}^* = \widetilde{Z}^*$, we know $\pi(k+1) \in \widetilde{Z}^*$. (c.2) If $\pi(k) \in V^0$ we know that $\mathrm{rank}(\pi(k)) = min_{v' \in E(\pi(k))} \mathrm{rank}(v')$. Now recall that $\widetilde{Z}^* = \widetilde{Y}^* = Y^* = \bigcup_{i>0} X^i$. Hence, any state with rank $0 < n < \infty$ is contained in $\widetilde{Z}^*$ and hence, we have $\pi(k+1) \in \widetilde{Z}^*$. With this, we can successively re-apply Thm. 7.3 to $\pi(k+1)$. This shows that G is visited infinitely often along π while π always remains within Q.

Completeness Let $\mathcal{W} \subseteq V$ be the set of states from which *Player* 0 has a winning strategy w.r.t. ψ in (7.5). In order to prove completeness, we need to show that $\mathcal{W} \subseteq Z^*$.

Recall, that for all states $v \in \mathcal{W}$ there exists a strategy ρ_0 s.t. all compliant fair adversarial plays π fulfill ψ. Now consider the weaker LTL formula $\widetilde{\psi} := Q\mathcal{U}(Q \cap G)$ and let $\widetilde{\mathcal{W}}$ be the winning state set for $\widetilde{\psi}$. Then we know by construction that $\widetilde{\psi}$ holds for $\pi(0)$ and for every $\pi(k) \subseteq Q \cap G$ while π always remains in Q. We can therefore strengthen $\widetilde{\psi}$ to $\widetilde{\psi} := Q\mathcal{U}(Q \cap G \cap \mathit{Cpre}(\widetilde{\mathcal{W}}))$ and see that still $\psi \to \widetilde{\psi}$ and therefore $\mathcal{W} \subseteq \widetilde{\mathcal{W}}$.

Now observe that it follows from Thm. 7.3 that $\widetilde{\mathcal{W}} = Z^*(\left\langle Q \cap G \cap \mathit{Cpre}(\widetilde{\mathcal{W}}), Q \right\rangle)$. It further follows from the monotonicity of the fixed-point that $\widetilde{Z}^*$ is the *largest* set of states s.t. equality holds in (A.11). We therefore have to conclude that $\widetilde{\mathcal{W}} \subseteq \widetilde{Z}^*$. As we have shown that $\mathcal{W} \subseteq \widetilde{\mathcal{W}}$, the claim is proved.

A.2.3. Proof of Thm. 7.1

Theorem A.3 (Thm. 7.1 restated for convenience) *Let $\mathcal{G}^\ell = \langle \mathcal{G}, E^\ell \rangle$ be a game graph with live edges and $\mathcal{R}$ be a Rabin condition over $\mathcal{G}$ with index set $P = [1; k]$. Further, let*

$$\begin{aligned} Z^* \coloneqq \nu Y_{p_0}.\mu X_{p_0}.\textstyle\bigcup_{p_1 \in P} \nu Y_{p_1}.\mu X_{p_1}. \\ \textstyle\bigcup_{p_2 \in P \setminus \{p_1\}} \nu Y_{p_2}.\mu X_{p_2}. \\ \vdots \\ \textstyle\bigcup_{p_k \in P \setminus \{p_1,\ldots,p_{k-1}\}} \nu Y_{p_k}.\mu X_{p_k}.\left[\textstyle\bigcup_{j=0}^{k} \mathcal{C}_{p_j}\right], \end{aligned}$$

where

$$\mathcal{C}_{p_j} \coloneqq \bigcap_{i=0}^{j} \overline{R}_{p_i} \cap \left[\left(G_{p_j} \cap \mathit{Cpre}(Y_{p_j})\right) \cup \left(\mathit{Apre}(Y_{p_j}, X_{p_j})\right)\right],$$

with $p_0 = 0$, $G_{p_0} \coloneqq \emptyset$ and $R_{p_0} \coloneqq \emptyset$. Then Z^* is equivalent to the winning region $\mathcal{W}$ of Player 0 in the fair adversarial game over $\mathcal{G}^\ell$ for the winning condition φ in (7.3). Moreover, the fixpoint algorithm runs in $O(n^{k+2}k!)$ symbolic steps, and a memoryless winning strategy for Player 0 can be extracted from it.

This section contains the proof of Thm. 7.1 which is inspired by the proof of Piterman and Pnueli (2006) for "normal" Rabin games. We first give a construction of a ranking induced by the fixpoint algorithm in (7.4) in Sec. A.2.3, and use this ranking to define a memoryless *Player* 0 strategy. As part of the soundness proof for Thm. 7.1 in Sec. A.2.3, we then show that this extracted strategy is indeed a winning strategy of *Player* 0 in the *fair adversarial game* over $\mathcal{G}^\ell$ w.r.t. φ. Further, we show in Sec. A.2.3 that the fixpoint algorithm in (7.4) is also complete, that is $\mathcal{W} \subseteq Z^*$. Intuitively, completeness shows that if Z^* is empty, there indeed exists no live-sufficient winning strategy (with arbitrary memory) for the given fair adversarial Rabin game. Additional lemmas and proofs can be found in App. A.2.3. The time complexity of the algorithm is proven separately in App. A.3.

Strategy Extraction

Our strategy extraction is adapted from the ranking of Piterman and Pnueli (2006, Sec. 3.1). Recall, that we consider the set of Rabin pairs $\mathcal{R} = \{\langle G_1, R_1 \rangle, \ldots, \langle G_k, R_k \rangle\}$ with index set $P = \{1, \ldots, k\}$ and the artificial Rabin pair $\langle G_0, R_0 \rangle$ s.t. $G_0 = R_0 = \emptyset$. A

permutation of the index set P is an one-to-one and onto function from P to P; as usual, we write $p_1 \dots p_k$ to denote the permutation mapping i to p_i, for $i = 1, \dots, k$. We define $\Pi(P)$ to be the set of all permutations over P. The *configuration domain* of the Rabin condition $\mathcal{R}$ is defined as

$$D(\mathcal{R}) := \{p_0 i_0 p_1 i_1 \dots p_k i_k \mid i_j \in [0; n],\ p_0 = 0,\ p_1 \dots p_k \in \Pi(P)\} \cup \{\infty\} \tag{A.13}$$

where $n < \infty$ is a natural number which is larger then the maximal number of iterations needed in any instance of the fixed point computation in (7.4) which is known to be finite. If $\mathcal{R}$ is clear from the context, we write D instead of $D(\mathcal{R})$.

Intuition: We first explain the intuition behind the chosen ranking. For this we consider the definition of ranks for states $v \in Z^*$ in an iterative fashion. First, consider the last iteration over X_{p_0} converging to the fixed point $Z^* = Y^*_{p_0} = \bigcup_{i_0 > 0} X^{i_0}_{p_0}$ where $X^0_{p_0} := \emptyset$. By flattening (7.4) we see that for all $i_0 > 0$ we have

$$X^{i_0}_{p_0} = Apre(Y^*_{p_0}, X^{i_0-1}_{p_0}) \cup \mathcal{A}_{p_0 i_0} \tag{A.14a}$$

where $\mathcal{A}_{p_0 i_0}$ collects all remaining terms of the fixpoint algorithm in (7.4) and will be specified later. For now, we want to assign a "minimal rank" to all states added to Z^* via the first term in (A.14a). Let us assume that the right "minimal rank" for these states is

$$d = p_0 i_0 p_1 0 \dots p_k 0 \quad \text{with} \quad p_1 < p_2 < \dots < p_k \text{ and } i_0 > 0.$$

We assign this rank to v iff $v \in Apre(Y^*_{p_0}, X^{i_0-1}_{p_0}) \setminus X^{i_0-1}_{p_0}$, i.e., if v is not already added to the fixed point in a previous iteration. The intuition behind this rank choice is that we want to remember that we have added v to Z^* in the i_0's computation over X_{p_0}, which sets the counter for p_0 in d to i_0. We keep all other counters at 0 because there is no actual contribution of terms involving variables X_{p_i} for $p_i \in P$ for the "adding" of v.

Now recall that

$$X^{i_0}_{p_0} = \bigcup_{p_1 \in P} Y^*_{p_1} = \bigcup_{p_1 \in P} \bigcup_{i_1 > 0} X^{i_1}_{p_1}.$$

Further, we know that

$$Apre(Y^*_{p_0}, X^{i_0-1}_{p_0}) \subseteq X^{i_1}_{p_1} \quad \text{for all} \quad p_1 \in P \text{ and } i_1 > 0. \tag{A.14b}$$

Hence, any state added to the fixed point via $X^{i_0}_{p_0}$ (which is not contained in $X^{i_0-1}_{p_0}$) is either added via $Apre(Y^*_{p_0}, X^{i_0}_{p_0})$ or via any other remaining term within $X^{i_1}_{p_1}$ for at least one p_1 and $i_1 > 0$. So let us explore the ranking in the latter case.

For this, let us proceed by going over all $X^{i_1}_{p_1}$ in increasing order over P, i.e, we start with selecting $p_1 = 1$. Further, we remember that we compute the next iteration over X_{p_1} (i.e., $X^{i_1}_{p_1}$ given $X^{i_1-1}_{p_1}$) as part of computing the set $X^{i_0}_{p_0}$. I.e., we remember the *computation-prefix* $\delta = p_0 i_0$ in the computation of $X^{i_1}_{p_1}$. To make δ explicit, we denote $X^{i_1}_{p_1}$ by $X^{i_1}_{\delta p_1}$. Now, we again consider the last iteration over $X_{\delta p_1}$ converging to the fixed point $Y^*_{\delta p_1}$ (for the currently considered computation-prefix δ). Then we have

$$X^{i_1}_{\delta p_1} = \underbrace{Apre(Y^*_{p_0}, X^{i_0-1}_{p_0})}_{=:S_\delta} \cup \underbrace{\overline{R}_{p_1} \cap \Big[\big(G_{p_1} \cap Cpre(Y^*_{\delta p_1})\big) \cup Apre(Y^*_{\delta p_1}, X^{i_1-1}_{\delta p_1})\Big]}_{=:\mathcal{C}_{\delta p_1 i_1}} \cup \mathcal{A}_{\delta p_1 i_1}.$$

We now want to assign the "minimal rank" to all states that are added to the fixed point via $\mathcal{C}_{\delta p_1 i_1}$. The immediate choice of this rank is

$$d = p_0 i_0 p_1 i_1 p_2 0 \dots p_k 0 = \delta p_1 i_1 p_2 0 \dots p_k 0 \quad \text{with } p_2 < \dots < p_k \quad \text{and } i_0, i_1 > 0. \tag{A.14c}$$

(Note that we do not necessarily have $p_1 < p_2$!)

We only want to assign this rank to states that are actually added to the fixed point via $\mathcal{C}_{\delta p_1 i_1}$, i.e., do not already have a rank assigned. First, all states $v \in S_\delta$ already have an assigned rank (as discussed before). Second, for $i_1 > 1$ all states in $\mathcal{C}_{\delta p_1 i_1 - 1}$ have already an assigned rank. But, third, also all states that have been added by considering a different $X_{\tilde{p}_1}$ with $\tilde{p}_1 \in P$ being smaller then the currently considered p_1 also have an already assigned rank.

Now consider the ranking choices suggested in (A.14b) and (A.14c). Then we see that all already assigned ranks are *smaller* (in terms of the lexicographic order over D) than the one in (A.14c). To see this, first consider a state $v \in S_\delta$. Either, $v \in X_{p_0}^{i_0 - 1}$ in which case its 0'th counter is smaller then i_0 (i.e., $i_0 - 1 < i_0$) or v has been added via S_δ, in which case the 0'th counter is equivalent but the first counter is 0 and therefore smaller then i_1 in (A.14c) (as, $i_1 > 0$). Now consider a state $v \in X_{\tilde{p}_1}$ with $\tilde{p}_1 < p_1$. In this case we see that 0'th counter is equivalent but the first permutation index is smaller (as $\tilde{p}_1 < p_1$).

We can therefore avoid specifying exactly in which set v should *not* be contained to be a newly added state. We can simply collect all possible rank assignments for every state and then, post-process this set to select the smallest rank in this set. Let us now generalize this idea to all possible configuration prefixes.

Proposition A.1 *Let $\delta = p_0 i_0 \dots p_{j-1} i_{j-1}$ be a* configuration prefix*, $p_j \in P \backslash \{p_1, \dots, p_{j-1}\}$ the next permutation index and $i_j > 0$ a counter for p_j. Then the flattening of* (7.4) *for this configuration prefix is given by*

$$X_{\delta p_j}^{i_j} = \underbrace{S_\delta \cup \mathcal{C}_{\delta p_j i_j}}_{S_{\delta p_j i_j}} \cup \mathcal{A}_{\delta p_j i_j} \tag{A.15a}$$

where

$$Q_{p_0 \dots p_a} := \bigcap_{b=0}^{a} \overline{R}_{p_b}, \tag{A.15b}$$

$$\mathcal{C}_{\delta p_a i_a} := \left(Q_{\delta p_a} \cap G_{p_a} \cap \mathit{Cpre}(Y_{\delta p_a}^*)\right) \cup \left(Q_{\delta p_a} \cap \mathit{Apre}(Y_{\delta p_a}^*, X_{\delta p_a}^{i_a - 1})\right), \tag{A.15c}$$

$$S_{p_0 i_0 \dots p_a i_a} := \bigcup_{b=0}^{a} \mathcal{C}_{p_0 i_0 \dots p_b i_b}, \tag{A.15d}$$

$$\mathcal{A}_{\delta p_j i_j} := \bigcup_{p_{j+1} \in P \backslash \{p_1, \dots, p_j\}} \bigcup_{i_{j+1} > 0} \left(X_{\delta p_j i_j p_{j+1}}^{i_{j+1}} \setminus S_{\delta p_j i_j}\right) \tag{A.15e}$$

As this flattening follows directly from the structure of the fixpoint algorithm in (7.4) and the definition of $\mathcal{C}_{p_j}$ in (7.4b), the proof is omitted.

Using the flattening of (7.4) in (A.15) we can define a ranking function induced by (7.4) as follows.

Definition A.1 Given the premises of Prop. A.1, we define $\underline{\gamma} := p_{j+1}0p_{j+2}0 \dots p_k 0$ with $p_{j+1} < p_{j+2} < \dots < p_k$ to be the minimal configuration post-fix. Then we define the rank-set $R : V \to 2^D$ s.t. (i) $\infty \in R(v)$ for all $v \in V$, and (ii) $\delta p_j i_j \underline{\gamma} \in R(v)$ iff $v \in S_{\delta p_j i_j}$. The ranking function $\text{rank} : V \to D$ is defined s.t. $\text{rank} : v \mapsto \min\{R(v)\}$.

Based on the ranking in Def. A.1 we define a memory-less player 0 strategy ρ_0, s.t. $\rho_0(v)$ forces progress to a state reachable from v which has minimal rank compared to all other successors of v. We prove Thm. A.4 in Sec. A.2.3.

Theorem A.4 *Given the premises of Prop. A.1, the memoryless player* 0 *strategy* $\rho_0 : V^0 \cap Z^* \to V^1$ *s.t.*

$$\rho_0(v) := \min_{(v,w)\in E}(\text{rank}(w)), \tag{A.16}$$

is a winning strategy for player 0 *in the* fair adversarial game *over* $\mathcal{G}^\ell$ *w.r.t.* φ.

Example A.1 Consider the Rabin game depicted in Fig. A.1 and discussed in App. A.1. Here, the strategy construction outlined in Thm. A.4 enforces a transition from q_6 to q_7 and a transition from q_5 to q_3. This is observed by noting that $\text{rank}(q_2) = 002012$ and $\text{rank}(q_7) = 001121$ where $\text{rank}(q_7) < \text{rank}(q_2)$. In addition, $\text{rank}(q_1) = 011021$ and $\text{rank}(q_3) = 001121$, where $\text{rank}(q_3) < \text{rank}(q_1)$.

Soundness

We now show why the fixpoint algorithm in (7.4) is *sound*, i.e., why $Z^* \subseteq \mathcal{W}$ in Thm. 7.1 holds. In addition, we also show that Thm. A.4 holds.

We prove soundness by an induction over the nesting of fixed points in (7.4) from inside to outside. In particular, we iteratively consider instances of the flattening in (A.15), starting with $j = k$ as the base case, and doing an induction from "$j+1$" to "j". To this end, we consider a local winning condition which refers to the current configuration-prefix $\delta = p_0 i_0 \dots p_{j-1} i_{j-1}$ in (A.15), namely

$$\psi_{\delta p_j} := \left(\begin{array}{ll} & Q_{\delta p_j} \mathcal{U} S_\delta \\ \vee & \Box Q_{\delta p_j} \wedge \Box\Diamond G_{p_j} \\ \vee & \Box Q_{\delta p_j} \wedge \left(\bigvee_{i \in P \setminus \{p_0,\dots,p_j\}} \left(\Diamond\Box \overline{R}_i \wedge \Box\Diamond G_i \right) \right) \end{array} \right). \tag{A.17}$$

Further, we denote by $\mathcal{W}_{\delta p_j}$ the set of states for which player 0 wins the *fair adversarial game* over $\mathcal{G}^\ell$ w.r.t. $\psi_{\delta p_j}$ in (A.17).

By recalling that for $p_j = p_0 = 0$ we have $Q_{p_0} = V$, $S_\varepsilon = \emptyset$ and $G_{p_0} = \emptyset$, we see that for $j = 0$ the condition in (A.17) simplifies to

$$\psi_{p_0} = \bigvee_{i \in P} \left(\Diamond\Box \overline{R}_i \wedge \Box\Diamond G_i \right).$$

This implies that ψ_{p_0} is equivalent to φ in (7.3). Given this observation, the proof of soundness in Thm. 7.1 proceeds by inductively showing that

$$X^{i_j}_{\delta p_j} \subseteq \mathcal{W}_{\delta p_j} \tag{A.18}$$

for any configuration prefix δ, next permutation index p_j and counter $i_j > 0$. Thereby, we ultimately also prove this claim for $p_j = p_0 = 0$ where δ is the empty string and $Y^*_{p_0} = \bigcup_{i_0>0} X^{i_0}_{p_0}$ coincides with Z^* in (7.4), which proves the statement.

With this insight the proof of Thm. A.4 as well as the soundness part of Thm. 7.1 reduce to the following proposition.

Proposition A.2 *For all $j \in [0,k]$, computation-prefixes $\delta = p_0 i_0 \dots p_{j-1} i_{j-1}$, next permutation index $p_j \in P \setminus \{p_0, \dots, p_{j-1}\}$, counter $i_j > 0$ and state $v \in X^{i_j}_{\delta p_j}$ the strategy ρ_0 in* (A.16) *wins the fair adversarial game over $\mathcal{G}^\ell$ w.r.t. $\psi_{\delta p_j}$ in* (A.17).

To see why Prop. A.2 holds, we consider the computation of $X^{i_j+1}_{\delta p_j}$ in (A.15a) and observe that the states in $X^{i_j+1}_{\delta p_j}$ can be clustered based on their rank induced via Def. A.1 as follows (see Sec. A.2.3 for a full proof).

Proposition A.3 *Given the premisses of Prop. A.2, let*

$$\underline{\gamma} = p_{j+1} 0 p_{j+2} 0 \dots p_k 0 \qquad \textit{with} \quad p_{j+1} < p_{j+2} < \dots < p_k, \quad \textit{and}$$
$$\overline{\gamma} = p_{j+1} n p_{j+2} n \dots p_k n \qquad \textit{with} \quad p_k < p_{k-1} < \dots < p_{j+1}$$

be the minimal *and* maximal *post-fix, respectively. Then, for all $v \in X^i_{\delta p_j}$ exactly one of the following cases holds:*
(a) $v \in S_\delta$ *and* $\mathrm{rank}(v) \leq \delta p_j 0 \underline{\gamma}$,
(b) $v \in Q_{\delta p_j} \cap G_{p_j} \cap \mathit{Cpre}(Y^*_{\delta p_j})$ *and* $\mathrm{rank}(v) = \delta p_j 1 \underline{\gamma}$,
(c) $v \in Q_{\delta p_j} \cap \mathit{Apre}(Y^*_{\delta p_j}, X^{i_j-1}_{\delta p_j})$ *and* $\mathrm{rank}(v) = \delta p_j i_j \underline{\gamma}$ *s.t.* $i_j > 1$, *or*
(d) $v \in \mathcal{A}_{\delta p_j i_j}$ *and there exists* $\underline{\gamma} < \gamma' \leq \overline{\gamma}$ *s.t.* $\mathrm{rank}(v) = \delta p_j i_j \gamma'$.

Using Prop. A.3 we prove Prop. A.2 by an induction over j.

Proof (Proof of Prop. A.2) **Base case:** First, for $j = k$ the last line of (A.17) disappears. Then the proof reduces to Thm. 7.3 and Thm. 7.2 in the following way. First, we fix all fixpoint variables $Y^*_{p_0 \dots p_l}$ and $X^{i_l}_{p_0 \dots p_l}$ for $l < j$ as well as $Y^*_{\delta p_j}$. With this, we see that $T := S_\delta \cup (Q_{\delta p_j} \cap G_{p_j} \cap \mathit{Cpre}(Y^*_{\delta p_j}))$ becomes a fixed set of states and (A.15a) reduces to

$$X^{i_j}_{\delta p_j} = T \cup (Q_{\delta p_j} \cap \mathit{Apre}(Y^*_{\delta p_j}, X^{i_j-1}_{\delta p_j}))$$

where we know that $X^{i_j}_{\delta p_j} \subseteq Y^*_{\delta p_j}$. Further, it follows form Prop. A.3 that for all $X^{i_j}_{\delta p_j}$ the ranking only differs by the i_j count. Hence, we can replace ρ_0 in (A.16) by the simpler strategy ρ_0 in (A.4) that only consideres the i_j count as the rank of states in $Y^*_{\delta p_j} = \bigcup_{i_j>0} X^{i_j}_{\delta p_j}$. With this it follows from Thm. 7.3 that for any fair adversarial play

π compliant with ρ_0 in (A.16) and starting in $X^{i_j}_{\delta p_j}$ for some $i_j \geq 0$ it holds that $Q_{\delta p_j} \mathcal{U} T$. This implies that whenever such a play π eventually reaches a state in $S_\delta \subseteq T$ the first line of (A.17) holds.

Now assume that π does not reach a state in $S_\delta \subseteq T$. Then it reaches a state in $Q_{\delta p_j} \cap G_{p_j} \cap \mathit{Cpre}(Y^*_{\delta p_j})$ and therefore has a successor state $v' \in Y^*_{\delta p_j} = \bigcup_{i_j > 0} X^{i_j}_{\delta p_j}$. Hence, $v' \in X^{i_j}_{\delta p_j}$ for some $i_j \geq 0$. By repeatedly applying this argument we see that π either eventually reaches a state in $S_\delta \subseteq T$ or it remains infinitely in $\mathcal{C}_{\delta p_j}$.. In the latter case, it follows from Thm. 7.2 that the second line of (A.17) holds.

Induction step: For the induction step (from "$j+1$" to "j") we first analyze the assumption. I.e., we know that for the longer computation prefix $\delta' = \delta p_j i_j$ and any next permutation index p_{j+1} we have that $Y^*_{\delta' p_{j+1}} \subseteq \mathcal{W}_{\delta' p_{j+1}}$ for all $p_{j+1} \in P \setminus \{p_1, \ldots, p_j\}$. Now recall that (A.15e) implies

$$\mathcal{A}_{\delta p_j i_j} = \bigcup_{p_{j+1} \in P \setminus \{p_1, \ldots, p_j\}} Y^*_{\delta' p_{j+1}} \setminus S_{\delta p_j i_j}$$

and therefore, we know that for all $v \in \mathcal{A}_{\delta p_j i_j}$ there exists a p_{j+1} s.t. $v \in \mathcal{W}_{\delta' p_{j+1}}$. That is, any fair adversarial play starting in v that is compliant with ρ_0 in (A.16) fulfills (A.17).

Therefore, whenever a fair adversarial play π starting in $X^{i_j}_{\delta p_j}$ visits a vertex $v \in \mathcal{A}_{\delta p_j i_j}$ (i.e., case (d) holds), we know that π could possibly come back to a state $v \in S_{\delta' p_{j+1}} = S_\delta \cup \mathcal{C}_{\delta p_j i_j}$ (via the first line of $\psi_{\delta' p_{j+1}}$).

In this case, Prop. A.3 ensures that the i_j count of the rank of states always stays constant while the play stays in $\mathcal{A}_{\delta p_j i_j}$. Therefore, one can ignore these finite sequences of (d) vertices in π while applying the ranking arguments of Thm. 7.3 and Thm. 7.2. I.e., we can conclude that in this case either the first or the second line of (A.17) holds for π. It remains to show that π fulfills the last line of (A.17) if π eventually stays within $\mathcal{A}_{\delta p_j i_j}$ forever. First, observe that this is only possible if S_δ is not visited along π. Hence, we know that $Q_{\delta p_j}$ holds along π until $\mathcal{A}_{\delta p_j i_j}$ is entered and never left. Further, as $\mathcal{A}_{\delta p_j i_j}$ is assumed to be never left after some time $k > 0$, we know that from that time onward there exists no p_{j+1} s.t. $S_{\delta' p_{j+1}}$ is visited again by π. This implies that for all vertices $\pi(k')$ with $k' > k$ the last two lines of $\psi_{\delta' p_{j+1}}$ (denoted $\psi'_{\delta' p_{j+1}}$) must be true for at lease one p_{j+1}. Hence, π fulfills the property

$$\Psi_{\delta p_j} := \Box Q_{\delta p_j} \wedge \Diamond \underbrace{\left(\bigvee_{p_{j+1} \in P \setminus \{p_1, \ldots, p_j\}} \psi'_{\delta' p_{j+1}} \right)}_{\Psi'_{\delta p_j}} \tag{A.19a}$$

With this, it remains to show that $\Psi_{\delta p_j}$ implies that the last line of (A.17) is true for π. In particular, we can show that both statements are equivalent, i.e.,

$$\Psi_{\delta p_j} = \Box Q_{\delta p_j} \wedge \bigvee_{p_{j+1} \in P \setminus \{p_1, \ldots, p_j\}} \left(\Diamond \Box \overline{R}_{p_{j+1}} \wedge \Box \Diamond G_{p_{j+1}} \right) \tag{A.19b}$$

Equation (A.19) is proved in Sec. A.2.3. This conclues the proof. □

Completeness

We now show why the fixpoint algorithm in (7.4) is complete, i.e., why $\mathcal{W} \subseteq Z^*$ in Thm. 7.1 holds.

We also prove completeness by an induction over the nesting of fixed points in (7.4) from inside to outside. In particular, we iteratively consider the fixed points $Y^*_{\delta p_j}$ and show that $Y^*_{\delta p_j} \subseteq \mathcal{W}_{\delta p_j}$. As $\psi_{\delta p_j}$ simplifies to φ in (7.3) for $p_j = p_0 = 0$, we ultimately show that $\mathcal{W} \subseteq Z^*$ in Thm. 7.1. With this insight the proof of the completness part of Thm. 7.1 reduces to the following proposition.

Proposition A.4 *For all $j \in [0,k]$, computation-prefixes $\delta = p_0 i_0 \dots p_{j-1} i_{j-1}$ and next permutation index $p_j \in P \setminus \{p_0, \dots, p_{j-1}\}$ it holds that $\mathcal{W}_{\delta p_j} \subseteq Y^*_{\delta p_j}$.*

Proof The proof proceeds by a nested induction over j starting with $j = k$.

Base case: Recall that for $j = k$ the last line of (A.17) disappears. Hence, for any state $v \in \mathcal{W}_{\delta p_j}$ either the first or the second line of (A.17) holds. Then the proof reduces to Thm. 7.3 and Thm. 7.2 in the following way.

First, we fix all fixpoint variables $Y^*_{p_0 \dots p_l}$ and $X^{il}_{p_0 \dots p_l}$ for $l < j$ as well as $Y^*_{\delta p_j}$. With this, we see that $T := S_\delta \cup (Q_{\delta p_j} \cap G_{p_j} \cap \mathit{Cpre}(Y^*_{\delta p_j}))$ becomes a fixed set of states and (A.15a) reduces to

$$Y^*_{\delta p_j} = Z^*(\langle T, Q_{\delta p_j} \rangle)$$

where $Z^*(\langle T, Q \rangle)$ is the set of states computed by the fixpoint algorithm in (7.9).

Then it follows from Thm. 7.3 that any state $v \in V$ for which there exists a fair adversarial play π that is winning for the winning condition $Q_{\delta p_j} \mathcal{U} T$ is contained in $Y^*_{\delta p_j}$. If, indeed the first line of (A.17) holds for π, this ensures that the claim holds.

Now assume that $Q_{\delta p_j} \mathcal{U} T$ holds for π but S_δ is never reached. Hence, $Q_{\delta p_j} \mathcal{U} (Q_{\delta p_j} \cap G_{p_j} \cap \mathit{Cpre}(Y^*_{\delta p_j}))$ holds for π. With this, it follows form Thm. 7.2 that any state $v \in V$ for which there exists a fair adversarial play π for which the second line of (A.17) holds is contained in $Y^*_{\delta p_j}$, proving the claim in this case.

Induction Step: For the induction from "$j+1$" to "j" we first analyze the assumption. I.e., we know that for the longer computation prefix $\delta' = \delta p_j$ and any next permutation index p_{j+1} we have that $\mathcal{W}_{\delta' p_{j+1}} \subseteq Y^*_{\delta' p_{j+1}}$. Further, observe that $\Psi'_{\delta p_j} \subseteq \bigcup_{p_{j+1} \in P \setminus \{p_1, \dots, p_j\}} \mathcal{W}_{\delta' p_{j+1}} \setminus S_{\delta p_j i_j}$ by construction. We therefore have

$$\Psi'_{\delta p_j} \subseteq \bigcup_{p_{j+1} \in P \setminus \{p_1, \dots, p_j\}} Y^*_{\delta' p_{j+1}} \setminus S_{\delta p_j i_j} = \mathcal{A}_{\delta p_j i_j}.$$

With this observation, we see that any fair adversarial play π which fulfills the last line of (A.17) also fulfills the weaker condition $Q_{\delta p_j} \mathcal{U} \mathcal{A}_{\delta p_j i_j}$. Therefore, the claim follows from the same reasoning as in the base case by re-defining T to $T := S_\delta \cup (Q_{\delta p_j} \cap G_{p_j} \cap \mathit{Cpre}(Y^*_{\delta p_j})) \cup \mathcal{A}_{\delta p_j i_j}$. □

Additional Lemmas and Proofs

In this section we provide additional lemmas and proofs to support the proof of Thm. 7.1 and Thm. A.4.

Proof of Prop. A.3

Lemma A.5 *Given the premisses of Prop. A.3, it holds for all* $v \in X^{i_j}_{\delta p_j}$ *that*

(i) $v \in S_\delta$ *iff* $\mathrm{rank}(v) \leq \delta p_j 0 \underline{\gamma}$

(ii) $v \in X^{i_j}_{\delta p_j}$ *iff* $\mathrm{rank}(v) \leq \delta p_j i_j \overline{\gamma}$

(iii) $v \in Y^{*}_{\delta p_j}$ *iff* $\mathrm{rank}(v) \leq \delta p_j n \overline{\gamma}$

(iv) $v \in \mathcal{A}_{\delta p_j i_j}$ *iff there exists* $\underline{\gamma} < \gamma' \leq \overline{\gamma}$ *s.t.* $\mathrm{rank}(v) = \delta p_j i_j \gamma'$

Proof (Proof of Lem. A.5) We prove all claims separately.
(i) It immediately follows from Def. A.1 (i) that $\delta p_j 0 \underline{\gamma} \in R(v)$ iff $v \in S_\delta$. If it is the minimal element in $R(v)$ then $\mathrm{rank}(v) = \delta p_j 0 \underline{\gamma}$, if not, there exists a smaller element in $R(v)$, and then $\mathrm{rank}(v) < \delta p_j 0 \underline{\gamma}$ from the definition of rank.
(ii) First, observe, that for $j = k$ it follows from (A.15a) that $X^{i_k}_{\delta p_k} = S_\delta p_k i_k$ and therefore from (i) that $v \in X^{i_k}_{\delta p_k}$ iff $\mathrm{rank}(v) \leq \delta p_k i_k$. Now we do an induction, assuming that for any $p_{j+1} \in P \setminus \{p_0, \ldots, p_j\}$ and $0 < i_{j+1} \leq n$ it holds that $v \in X^{i_{j+1}}_{\delta p_{j+1}}$ iff $\mathrm{rank}(v) \leq \delta' p_{j+1} i_{j+1} \overline{\gamma'}$ (where δ' goes up to index j and γ' starts only at index $j+2$. Now recall that

$$X^{i_j}_{\delta p_j} = \bigcup_{p_{j+1} \in P \setminus \{p_0, \ldots, p_j\}} Y^{*}_{\delta p_{j+1}} = \bigcup_{p_{j+1} \in P \setminus \{p_0, \ldots, p_j\}} \bigcup_{i_{j+1} > 0} X^{i_{j+1}}_{\delta p_j i_j p_{j+1}}.$$

Hence, $v \in X^{i_j}_{\delta p_j}$ iff there exists $p_{j+1} \in P \setminus \{p_0, \ldots, p_j\}$ and $0 < i_{j+1} \leq n$ s.t. $v \in X^{i_{j+1}}_{\delta p_j i_j p_{j+1}}$. Now we know that for any choice of p_{j+1} and i_{j+1} we have $\mathrm{rank}(v) \leq \delta' p_j i_j p_{j+1} i_{j+1} \overline{\gamma'}$. Now the worst case, in terms of the lexicographic ordering over D is that $p_{j+1} = \max(()P \setminus \{p_0, \ldots, p_j\})$ and $i_{j+1} = n$. Hence, we know that $\mathrm{rank}(v) \leq \delta p_j i_j \overline{\gamma}$.
(iii) As $Y^{*}_{\delta p_j} = \bigcup_{i_j > 0} X^{i_j}_{\delta p_j}$ it follows that there exists $0 < i_j \leq n$ s.t. $v \in X^{i_j}_{\delta p_j}$ and (from (ii)) therefore $\mathrm{rank}(v) \leq \delta p_j i_j \overline{\gamma}$. Again, the worst case is $i_j = n$, giving $\mathrm{rank}(v) \leq \delta p_j n \overline{\gamma}$.
(iv) It follows from (A.15a) that $v \in \mathcal{A}_{\delta p_j i_j}$ iff $v \in X^{i_j}_{\delta p_j} \setminus S_{\delta p_j i_j}$. Hence, it follows from (i) and (ii) that $\mathrm{rank}(v) > \delta p_j 0 \underline{\gamma}$ and $\mathrm{rank}(v) \leq \delta p_j i_j \overline{\gamma}$ which is true iff there exists $\underline{\gamma} < \gamma' \leq \overline{\gamma}$ s.t. $\mathrm{rank}(v) = \delta p_j i_j \gamma'$, which proves the statement. □

Given these properties of the ranking function, we are ready to prove the suggested case split in Prop. A.3.

Proof (Proof of Prop. A.3) We call a vertex $v \in V$ that fulfills cases (α) in either Lem. A.5 or Prop. A.3 an (α)-vertex. First, observe that cases (i) and (iv) in Lem. A.5 coincide with cases (a) and (d), respectively, in Prop. A.3. Further, recall that $X^1_{\delta p_j} = \emptyset$. Therefore, $X^1_{\delta p_j}$ only contains (a)-,(b)- and (d)-vertices, as $Apre(\cdot, \emptyset) = \emptyset$. Now we know

from (ii) that for any $v \in X^1_{\delta p_j}$ we have $\text{rank}(v) \leq \delta p_j 1\overline{\gamma}$. Now excluding the rankings for (a)- and (d)-vertices we obtain that (b)-vertices must have rank $\text{rank}(v) \leq \delta p_j 1\underline{\gamma}$. Similarly, for every $i_j > 1$ we know that $X^{i_j}_{\delta p_j}$ contains (a)-, (b)-, (c)- and (d)- vertices. Now excluding (a)-, (b)- and (d)- vertices yields $\text{rank}(v) \leq \delta p_j i_j \underline{\gamma}$ for all (c)-vertices. □

Proof of (A.19)

Given the notation in Sec. A.2.3 we prove that the equality in (A.19) holds.

First recall that

$$\Psi'_{\delta' p_{j+1}} := \begin{pmatrix} & \Box Q_{\delta' p_{j+1}} \wedge \Box\Diamond G_{p_{j+1}} \\ \vee & \Box Q_{\delta' p_{j+1}} \wedge \left(\bigvee_{i \in \widetilde{P}_{\setminus j+1}} \left(\Diamond\Box \overline{R}_i \wedge \Box\Diamond G_i \right) \right) \end{pmatrix}, \tag{A.20}$$

where $\widetilde{P}_{\setminus j+1} := P \setminus \{p_1, \ldots, p_{j+1}\}$.

For the insertion of (A.20) into (A.19a) we have the following observations. First, observe that $\Diamond(B \vee C) = \Diamond B \vee \Diamond C$, i.e., we can distribute the eventuality operator preceding $\Psi'_{\delta' p_{j+1}}$ over both lines. Second, we can re-order the preceeding disjunction over p_{j+1} in (A.19a) and the disjunction between the two lines of (A.20). This yields to the following condition

$$\begin{aligned} \Psi_{\delta p_j} =& \Box Q_{\delta p_j} \wedge \left(\bigvee_{p_{j+1} \in \widetilde{P}_{\setminus j}} (\Diamond \lambda_1) \vee \bigvee_{p_{j+1} \in \widetilde{P}_{\setminus j}} (\Diamond \lambda_2) \right) \\ =& \underbrace{\left(\Box Q_{\delta p_j} \wedge \bigvee_{p_{j+1} \in \widetilde{P}_{\setminus j}} (\Diamond \lambda_1) \right)}_{=:\Psi_1} \vee \underbrace{\left(\Box Q_{\delta p_j} \wedge \bigvee_{p_{j+1} \in \widetilde{P}_{\setminus j}} (\Diamond \lambda_2) \right)}_{=:\Psi_2}, \end{aligned} \tag{A.21}$$

where λ_i denotes the i-th line of the conjunction in (A.20).

Now let us investigate the terms Ψ_1 and Ψ_2 in (A.21) separately. For Ψ_1, observe that $\Diamond\Box\Diamond A = \Box\Diamond A$ and $\Diamond(\Box A \wedge \Box B) = \Diamond\Box A \wedge \Diamond\Box B$. Further we have $Q_{\delta' p_{j+1}} = Q_{\delta p_j} \wedge \overline{R}_{j+1} \subseteq Q_{\delta p_j}$ and hence

$$\Psi_1 = \Box Q_{\delta p_j} \wedge \bigvee_{p_{j+1} \in \widetilde{P}_{\setminus j}} \left(\Diamond\Box (Q_{\delta p_j} \wedge \overline{R}_{p_{j+1}}) \wedge \Box\Diamond G_{p_{j+1}} \right)$$

By using the equatlity $\Diamond\Box(A \wedge B) = \Diamond\Box A \wedge \Diamond\Box B$ and the fact that $Q_{\delta p_j}$ is independent of the choice of p_{j+1} we get

$$\begin{aligned} \Psi_1 =& \Box Q_{\delta p_j} \wedge \Diamond\Box Q_{\delta p_j} \wedge \bigvee_{p_{j+1} \in \widetilde{P}_{\setminus j}} \left(\Diamond\Box \overline{R}_{p_{j+1}} \wedge \Box\Diamond G_{p_{j+1}} \right) \\ =& \Box Q_{\delta p_j} \wedge \bigvee_{p_{j+1} \in \widetilde{P}_{\setminus j}} \left(\Diamond\Box \overline{R}_{p_{j+1}} \wedge \Box\Diamond G_{p_{j+1}} \right). \end{aligned} \tag{A.22}$$

To analyze Ψ_2 in (A.21), recall that the eventuality operator $\Diamond$ distributes over disjunctions. We can therefore move the inner disjunction over i outside and get

$$\Psi_2 = \Box Q_{\delta p_j} \wedge \bigvee_{p_{j+1} \in \widetilde{P}_{\setminus j}} \left(\bigvee_{i \in \widetilde{P}_{\setminus j+1}} \left[\Diamond \left(\Box Q_{\delta' p_{j+1}} \wedge \left(\Diamond \Box \overline{R}_i \wedge \Box \Diamond G_i \right) \right) \right] \right)$$

Now observe that $\left(\Diamond \Box \overline{R}_i \wedge \Box \Diamond G_i \right) = \Diamond \left(\Box \overline{R}_i \wedge \Box \Diamond G_i \right)$ and $\Diamond(\Box A \wedge \Diamond B) = \Diamond \Box A \wedge \Diamond B$. Additionally using $Q_{\delta' p_{j+1}} = Q_{\delta p_j} \wedge \overline{R}_{p_{j+1}} \subseteq Q_{\delta p_j}$ we get

$$\Psi_2 = \Box Q_{\delta p_j} \wedge \bigvee_{p_{j+1} \in \widetilde{P}_{\setminus j}} \left(\bigvee_{i \in \widetilde{P}_{\setminus j+1}} \left[\Diamond \Box (Q_{\delta p_j} \wedge \overline{R}_{p_{j+1}}) \wedge \left(\Diamond \Box \overline{R}_i \wedge \Box \Diamond G_i \right) \right] \right)$$

Now we can do the same trick as in the simplification of Ψ (see (A.22)) to remove the $Q_{\delta p_j}$ term inside the disjunction and get

$$\Psi_2 = \Box Q_{\delta p_j} \wedge \bigvee_{p_{j+1} \in \widetilde{P}_{\setminus j}} \left(\bigvee_{i \in \widetilde{P}_{\setminus j+1}} \left[\Diamond \Box \overline{R}_{p_{j+1}} \wedge \left(\Diamond \Box \overline{R}_i \wedge \Box \Diamond G_i \right) \right] \right) \tag{A.23}$$

To see how we can simplify (A.23), let us assume that the set $\widetilde{P}_{\setminus j}$ contains three elements, e.g., $\{a, b, c\}$. Then we can expand (A.23) to

$$\begin{aligned}
& \Diamond \Box \overline{R}_a \wedge \left(\Diamond \Box \overline{R}_b \wedge \Box \Diamond G_b \right) \\
\vee\ & \Diamond \Box \overline{R}_a \wedge \left(\Diamond \Box \overline{R}_c \wedge \Box \Diamond G_c \right) \\
\vee\ & \Diamond \Box \overline{R}_b \wedge \left(\Diamond \Box \overline{R}_a \wedge \Box \Diamond G_a \right) \\
\vee\ & \Diamond \Box \overline{R}_b \wedge \left(\Diamond \Box \overline{R}_c \wedge \Box \Diamond G_c \right) \\
\vee\ & \Diamond \Box \overline{R}_c \wedge \left(\Diamond \Box \overline{R}_b \wedge \Box \Diamond G_b \right) \\
\vee\ & \Diamond \Box \overline{R}_c \wedge \left(\Diamond \Box \overline{R}_a \wedge \Box \Diamond G_a \right)
\end{aligned}$$

Now, we can re-order terms and get

$$\begin{aligned}
& \left(\Diamond \Box \overline{R}_b \wedge \Box \Diamond G_b \right) \wedge \left(\Diamond \Box \overline{R}_a \vee \Diamond \Box \overline{R}_c \right) \\
\vee\ & \left(\Diamond \Box \overline{R}_c \wedge \Box \Diamond G_c \right) \wedge \left(\Diamond \Box \overline{R}_a \vee \Diamond \Box \overline{R}_b \right) \\
\vee\ & \left(\Diamond \Box \overline{R}_a \wedge \Box \Diamond G_a \right) \wedge \left(\Diamond \Box \overline{R}_b \vee \Diamond \Box \overline{R}_c \right)
\end{aligned}$$

Generalizing this observation, we get the following formula equivalent to (A.23)

$$\Psi_2 = \Box Q_{\delta p_j} \wedge \bigvee_{p_{j+1} \in \widetilde{P}_{\setminus j}} \left(\left(\Diamond \Box \overline{R}_{p_{j+1}} \wedge \Box \Diamond G_{p_{j+1}} \right) \right) \wedge \bigvee_{j \in \widetilde{P}_{\setminus j+1}} \Diamond \Box \overline{R}_j \tag{A.24}$$

Now recall that $A \wedge B \Rightarrow A$ for any choice of A and B. With this one can verify that $\Psi_2 \Rightarrow \Psi_1$ as the term after the disjuction over p_{j+1} in (A.24) implies the term after the disjuction over p_{j+1} in (A.22). Hence, the set of states which fulfill Ψ_1 in (A.22) is always larger then the set of states which fulfill Ψ_2 (A.24)). As both terms are connected by a conjunction in (A.21), we can ignore Ψ_2 in (A.21) and obtain

$$\Psi_{\delta p_j} = \Psi_1 = \Box Q_{\delta p_j} \wedge \bigvee_{p_{j+1} \in \widetilde{P}_{\setminus j}} \left(\Diamond \Box \overline{R}_{p_{j+1}} \wedge \Box \Diamond G_{p_{j+1}} \right). \tag{A.25}$$

This concludes the proof of (A.19) as (A.25) coincides with (A.19b).

A.2.4. Additional Proofs for Sec. 7.2.4

Fair Adversarial Rabin Chain Games

Theorem A.5 (Thm. 7.4 restated for convenience) *Let $\mathcal{G}^\ell = \langle \mathcal{G}, E^\ell \rangle$ be a game graph with live edges and $\mathcal{R}$ be a Rabin condition over $\mathcal{G}$ with k pairs for which the chain condition* (7.14) *holds. Further, let*

$$Z^* \coloneqq \nu Y_0.\ \mu X_0.\ \nu Y_k.\ \mu X_k.\ \nu Y_{k-1}.\ \ldots \mu X_1.\ \textstyle\bigcup_{j=0}^{k} \widetilde{\mathcal{C}}_j, \tag{A.26a}$$

$$\text{where } \widetilde{\mathcal{C}}_j \coloneqq \overline{R}_j \cap [(G_j \cap \mathit{Cpre}(Y_j)) \cup \mathit{Apre}(Y_j, X_j)]$$

with $G_{p_0} \coloneqq \emptyset$ and $R_{p_0} \coloneqq \emptyset$.

Then Z^* is equivalent to the winning region $\mathcal{W}$ of Player 0 *in the fair adversarial game over $\mathcal{G}^\ell$ for the winning condition φ in* (7.3)*. Moreover, the fixpoint algorithm runs in $O(n^{k+2})$ symbolic steps, and a memoryless winning strategy for Player* 0 *can be extracted from it.*

In this section we prove Thm. 7.4. That is, we prove that for Rabin chain conditions, the fixpoint computing Z^* in (7.4) simplifies to the one in (A.26). This is formalized in the next proposition.

Proposition A.5 *Given the premisses of Thm. 7.4 let Z^* be the fixed point computed by* (7.4) *and $\widetilde{Z}^*$ the fixed point computed by* (A.26)*. Then $Z^* = \widetilde{Z}^*$.*

If Prop. A.5 holds, we imediately see that Thm. 7.4 directly follows from Thm. 7.1. It therefore remains to prove Prop. A.5.

Similar to the soundness and completeness proof for Thm. 7.1 we prove Prop. A.5 by an induction over the nesting of fixpoints in (7.4) form inside to outside. Here, however we do not need to explicitly refer to counters i_j as in Prop. 7.4. Hence, we can look at permutation prefixes instead of configuration prefixes. We have the following proposition.

Proposition A.6 *Let P be the index set of the Rabin chain condition $\mathcal{R}$ in Thm. 7.4. Further, for any $j \in [0; k]$ let $\delta \coloneqq p_0 p_1 \ldots p_{j-1}$ be a* permutation prefix*, $\widetilde{P}_{\setminus \delta} \coloneqq P \setminus$*

$\{p_0, \dots, p_{j-1}\}$ the reduced index set and $q_0 := p_j \in \widetilde{P}_{\setminus\delta}$ the current permutation index. Further, define[1]

$$\begin{aligned} Z^*_{\delta p_j} :=& \nu Y_{q_0}.\ \mu X_{q_0}. \\ & \bigcup_{q_1 \in \widetilde{P}_{\setminus \delta p_j}} \nu Y_{q_1}.\ \mu X_{q_1}. \\ & \vdots \\ & \bigcup_{q_n \in \widetilde{P}_{\setminus \delta p_j} \setminus \{q_1, \dots, q_{n-1}\}} \nu Y_{q_n}.\ \mu X_{q_n}.\ S_\delta \cup \left[\bigcup_{\ell=0}^{n} \mathcal{C}_{\delta q_\ell}\right] \end{aligned} \tag{A.27a}$$

where $n := k - j$,

$$\mathcal{C}_{\delta q_j} := Q_\delta \cap \bigcap_{i=0}^{\ell} \overline{R}_{q_i} \cap \left[(G_{q_\ell} \cap \mathit{Cpre}(Y_{q_\ell})) \cup (\mathit{Apre}(Y_{q_\ell}, X_{q_\ell}))\right], \tag{A.27b}$$

$Q_\delta := \bigcap_{i=0}^{j} \overline{R}_{p_i}$ and $S_{p_0 \dots p_{j-1}} := \bigcup_{b=0}^{j-1} \mathcal{C}_{p_0 \dots p_b}$.
Then it holds that

$$Z^*_{\delta p_j} = \nu Y_{r_0}.\ \mu X_{r_0}.\ \nu Y_{r_1}.\ \mu X_{r_1}.\ \dots \nu Y_{r_n}.\ \mu X_{r_n}.\ S_\delta \cup \left[\bigcup_{\ell=0}^{n} \widetilde{\mathcal{C}}_{\delta r_\ell}\right], \tag{A.28a}$$

where

$$\widetilde{\mathcal{C}}_{\delta r_\ell} := Q_{\delta p_j} \cap \overline{R}_{r_\ell} \cap \left[(G_{r_\ell} \cap \mathit{Cpre}(Y_{r_\ell})) \cup (\mathit{Apre}(Y_{r_\ell}, X_{r_\ell}))\right] \tag{A.28b}$$

with $r_i \in \widetilde{P}_{\setminus \delta p_j}$ for all $i \in [1; n]$ such that $r_1 > r_2 > \dots > r_n$ and $r_0 = q_0 = p_j$.

It should be noted that Prop. A.6 needs to hold for any choice of j and δ. Further, we have slightly abused notation by not specifying the values of the fixpoint parameters used within S_δ. This is, however, not relevant for the proof of Prop. A.6 and we should interpret S_δ as a term computed by an arbitrary choice of the involved fixpoint parameters.

Now, it should be obvious that for the choice $j = 0$ we get $\delta = \varepsilon$ and $S_\delta = \emptyset$. Further, we see that in this case, we have $\widetilde{P}_{\setminus \delta p_0} = P$ which implies that $Z^*_{p_0}$ in (A.27) coincides with Z^* in (7.4). Further, as $\widetilde{P}_{\setminus \delta p_0} = P$ we must have $r_1 = k$, $r_2 = k - 1$, …, $r_k = 1$ and $r_0 = p_0 = 0$ to fulfill the requirements on r. Further $Q_{p_0} = \overline{R_0} = Q$. Therefore $Z^*_{p_0}$ in (A.28) coincides with Z^* in (A.26) in this case. Hence, proving Prop. A.6 for any j (including $j = 0$), immediately proves Prop. A.5.

In the remainder of this section we prove Prop. A.6 by an induction over j, starting with $j = k$ as the base case. Now observe that for $j = k$ we have $\widetilde{P}_{\setminus \delta p_j} = \emptyset$ and hence both (A.27) and (A.28) reduce to a two-nested fixed point over the variables Y_{q_0}, X_{q_0} and Y_{r_0}, X_{r_0}, respectively, where $r_0 = q_0 = p_k$ by definition. Further, we see that $\mathcal{C}_{\delta q_0} = \widetilde{\mathcal{C}}_{\delta r_0}$ by definition, which immediately proves the claim of Prop. A.6 for the base case.

In the remainder of this section we prove the induction step from "j" to "$j - 1$" in a series of definitions and lemmas.

[1] Observe that $\delta p_j = p_0 \dots p_{j-1} p_j$ is itself a permutation prefix.

Definition A.2 Let $\tilde{P} \subseteq \mathbb{N}$ be a set of n indices and $\beta = q_1 \dots q_n$ with $q_i \in \tilde{P}$ and $q_i \neq q_j$ for all $j \neq i$ a full permutation sequence of the elements from $\tilde{P}$. For $1 \leq j \leq l \leq n$ we call $\beta_{jl} = q_j q_{j+1} \dots q_j$ a *maximal decreasing sub-sequence* of β if (i) $q_j < q_{j+1} < \dots < q_l$, (ii) $q_{j-1} > q_j$ or $j = 1$, and (iii) $q_l > q_{l+1}$ or $l = n$.

We see that, by definition, the first maximally decreasing sub-sequences of a permutation sequence β starts with q_1. Intuitively, decreasing sub-sequences allow to immediately utilize the properties in (7.14) to simplify the fixpoint expression.

Lemma A.6 *Let δ, $\widetilde{P}_{\setminus\delta}$ and $q_0 = p_j$ as in Prop. A.6, $\beta = q_1 \dots q_n$ a full permutation sequence of $\widetilde{P}_{\setminus\delta p_j}$ and $\beta_{jl} = q_j q_{j+1} \dots q_j$ a* maximal decreasing sub-sequence *of β. Then*

$$\nu Y_{q_j}.\ \mu X_{q_j}.\ \dots \nu Y_{q_l}.\ \mu X_{q_l}.\ \textstyle\bigcup_{i=j}^{l} \mathcal{C}_{\delta q_i} = \nu Y_{q_j}.\ \mu X_{q_j}.\ \mathcal{C}_{\delta q_j} \tag{A.29}$$

Proof Let $\alpha := q_0 \dots q_{j-1}$ and observe that

$$\begin{aligned}
\mathcal{C}_{\delta q_j} &= Q_{\delta\alpha} \cap \left[\left(\overline{R}_j \cap G_{q_j} \cap \mathit{Cpre}(Y_{q_j}) \right) \cup \left(\overline{R}_j \cap \mathit{Apre}(Y_{q_j}, X_{q_j}) \right) \right] \\
\mathcal{C}_{\delta q_{j+1}} &= Q_{\delta\alpha} \cap \left[\left(\overline{R}_j \cap \overline{R}_{j+1} \cap G_{q_{j+1}} \cap \mathit{Cpre}(Y_{q_j}) \right) \cup \left(\overline{R}_j \cap \overline{R}_{j+1} \cap \mathit{Apre}(Y_{q_j}, X_{q_j}) \right) \right] \\
&= Q_{\delta\alpha} \cap \left[\left(\overline{R}_j \cap G_{q_{j+1}} \cap \mathit{Cpre}(Y_{q_j}) \right) \cup \left(\overline{R}_j \cap \mathit{Apre}(Y_{q_j}, X_{q_j}) \right) \right],
\end{aligned}$$

where the simplification of $\mathcal{C}_{\delta q_{j+1}}$ follows from $\overline{R}_j \subseteq \overline{R}_{j+1}$ (see (7.14)). So $\mathcal{C}_{\delta q_j}$ and $\mathcal{C}_{\delta q_{j+1}}$ really only differ by the G_{q_j} (resp. $G_{q_{j+1}}$) term in the first term of the disjunct. As $G_{q_j} \supseteq G_{q_{j+1}}$ (see (7.14)) and all terms in the first part of the disjunct are intersected, we see that $\mathcal{C}_{\delta q_j} \supseteq \mathcal{C}_{\delta q_{j+1}}$. With this it follows from case (iii) in Lem. A.3 that

$$\nu Y_{q_j}.\ \mu X_{q_j}.\nu Y_{q_{j+1}}.\ \mu X_{q_{j+1}}.\ \mathcal{C}_{\delta q_j} \cup \mathcal{C}_{\delta q_{j+1}} = \nu Y_{q_j}.\ \mu X_{q_j}.\ \mathcal{C}_{\delta q_j}.$$

Applying this argument to all $i \in [j; l]$ proves the claim. □

Definition A.3 We say that a permutation sequence β has *chain index m* if it contains m *maximal decreasing* sub-sequences. For $\beta = q_1 \dots q_n$ with chain index m we define its reduction $\beta_\downarrow$ as $\beta_\downarrow := r_1 ... r_m$ such that $r_m = q_j$ if β_{jl} is the m'th maximally decreasing sub-sequence of β.

Lemma A.7 *Let δ, $\widetilde{P}_{\setminus\delta}$ and $q_0 = p_j$ as in Prop. A.6, $\beta = q_1 \dots q_n$ a full permutation sequence of $\widetilde{P}_{\setminus\delta p_j}$ with chain index m and $\beta_\downarrow := r_1 ... r_m$. Then*

$$\begin{aligned}
&\nu Y_{q_0}.\ \mu X_{q_0}.\ \nu Y_{q_1}.\ \mu X_{q_1}.\ \dots \nu Y_{q_n}.\ \mu X_{q_n} \bigcup_{j=0}^{n} \mathcal{C}_{\delta q_j} \\
&\qquad = \nu Y_{r_0}.\ \mu X_{r_0}.\ \nu Y_{r_1}.\ \mu X_{r_1}.\ \dots \nu Y_{r_m}.\ \mu X_{r_m} \bigcup_{l=0}^{m} \mathcal{C}_{\delta q_l}
\end{aligned} \tag{A.30}$$

where $q_0 = r_0 = p_j$.

PROOF First, observe that by construction we always have $r_1 = q_1$. Hence, $Q_{\delta\alpha}$ in the proof of Lem. A.6 reduces to $Q_{\delta q_1}$ in this case. Further, consider $r_2 = q_j$ and observe that in this case $Q_{\delta\alpha} = Q_\delta \cap \bigcap_{i=0}^{j-1} \overline{R}_{q_i} = Q_{\delta q_0} \cap \overline{R}_{q_1} = Q_{\delta p_j} \cap \overline{R}_{r_1}$ as $q_1 \dots q_{j-1}$ is a maximal decreasing sub-sequence by construction. Iteratively re-applying this argument along with Lem. A.6 for every $l \in [1, m]$ therefore proves the claim. □

Now observe that we can re-apply Lem. A.7 to $\beta_\downarrow$ and reduce it even more. That means, $\beta_\downarrow$ could now again have maximal decreasing sub-sequences and we therefore can reduce it to $(\beta_\downarrow)_\downarrow$. This might again be reduceable and so forth. We therefore define the *maximal reduced permutation sequence* $\beta_\Downarrow = (((\beta_\downarrow)_\downarrow)\dots)_\downarrow = r_1 \dots r_n$ such that $r_1 > r_2 > \dots r_n$, i.e. the chain index of $\beta_\Downarrow$ is equivalent to its length. With this, we have the following result.

Lemma A.8 *Let δ, $\widetilde{P}_{\setminus\delta}$ and $q_0 = p_j$ as in Prop. A.6, $\beta = q_1 \dots q_n$ a full permutation sequence of $\widetilde{P}_{\setminus\delta p_j}$ and $\beta_\Downarrow := r_1 ... r_m$ its maximal reduced permutation sequence. Then*

$$\begin{aligned} &\nu Y_{q_0}.\ \mu X_{q_0}.\ \nu Y_{q_1}.\ \mu X_{q_1}.\ \dots \nu Y_{q_n}.\ \mu X_{q_n} \bigcup_{j=0}^{n} \mathcal{C}_{\delta q_j} \\ &\quad = \nu Y_{r_0}.\ \mu X_{r_0}.\ \nu Y_{r_1}.\ \mu X_{r_1}.\ \dots \nu Y_{r_m}.\ \mu X_{r_m} \bigcup_{l=0}^{m} \widetilde{\mathcal{C}}_{\delta q_l} \end{aligned} \tag{A.31}$$

PROOF It follows from the definition of $\beta_\Downarrow$ and repeatably applying Lem. A.7 that

$$\begin{aligned} &\nu Y_{q_0}.\ \mu X_{q_0}.\ \nu Y_{q_1}.\ \mu X_{q_1}.\ \dots \nu Y_{q_n}.\ \mu X_{q_n} \bigcup_{j=0}^{n} \mathcal{C}_{\delta q_j} \\ &\quad = \nu Y_{r_0}.\ \mu X_{r_0}.\ \nu Y_{r_1}.\ \mu X_{r_1}.\ \dots \nu Y_{r_m}.\ \mu X_{r_m} \bigcup_{l=0}^{m} \mathcal{C}_{\delta r_l} \end{aligned}$$

Now we have by definition that $r_0 = q_0$ and $r_1 = q_1$ and therefore $\mathcal{C}_{\delta r_0} = \widetilde{\mathcal{C}}_{\delta r_0}$ and $\mathcal{C}_{\delta r_1} = \widetilde{\mathcal{C}}_{\delta r_1}$ by definition. Now recall that $r_1 > r_2$, hence $\overline{R}_{r_1} \cap \overline{R}_{r_2} = \overline{R}_{r_2}$. Iteratively applying this argument gives $\mathcal{C}_{\delta r_l} = \widetilde{\mathcal{C}}_{\delta r_l}$ for all $l \in [1, n]$, what proves the claim. □

Note that the only full permutation sequence of $\widetilde{P}_{\setminus\delta p_j}$ with *chain index* n is the one where $q_1 > q_2 > \dots > q_n$, giving $\beta_\downarrow = \beta_\Downarrow = \beta$. Hence, the sequence $r_1 \dots r_n$ used in (A.28) is actually the *maximal permutation sequence* of $\widetilde{P}_{\setminus\delta p_j}$. We see that all other full permutation sequences γ of $\widetilde{P}_{\setminus\delta p_j}$ have *chain index* m such that $1 \leq m < n$. As the $\widetilde{\mathcal{C}}$ terms in (7.15b) do not depend on the history of permutation sequences from $\widetilde{P}_{\setminus\delta p_j}$, we see that any term constructed for a non-maximal permutation sequence is contained in the term constructed for the maximal permutation sequence. This is formalized in the next lemma.

Lemma A.9 *Let δ, $\widetilde{P}_{\backslash\delta}$ and $q_0 = p_j$ as in Prop. A.6 and let $\beta = r_1...r_n$ be the maximal permutation sequence of $\widetilde{P}_{\backslash\delta p_j}$, that its $\beta = \beta_{\Downarrow}$. Further, let $\gamma \neq \beta$ be a full permutation sequence of $\widetilde{P}_{\backslash\delta p_j}$ such that $\gamma_{\Downarrow} = s_1 \dots s_m$ with $m < n$. Then*

$$\nu Y_{r_1}.\ \mu X_{r_1}.\ \dots \nu Y_{r_n}.\ \mu X_{r_n} \bigcup_{l=1}^{n} \widetilde{\mathcal{C}}_{\delta r_l} \tag{A.32}$$

$$\subseteq \nu Y_{s_1}.\ \mu X_{s_1}.\ \dots \nu Y_{s_m}.\ \mu X_{s_m} \bigcup_{l=1}^{m} \widetilde{\mathcal{C}}_{\delta s_l} \tag{A.33}$$

Proof As β is a full permutation sequence of $\widetilde{P}_{\backslash\delta p_j}$ we know that for any $i \in [1; m]$ there exists one $j \in [1; n]$ such that $s_i = r_j$. Further, as $\widetilde{\mathcal{C}}$ does not depend on the history of the permutation sequence β and γ we see that $\widetilde{\mathcal{C}}_{\delta s_i} = \widetilde{\mathcal{C}}_{\delta r_j}$ in this case. As $m < n$ we see that the first line of (A.33) contains the fixpoint variables and $\widetilde{\mathcal{C}}$ terms of the second line of (A.33). We can therefore apply Lem. A.3 (i) and (ii) which immediately proves the claim. □

Using this result, we are finally ready to prove the induction step of Prop. A.6.

Proof (Proof of Prop. A.6) Recall that Prop. A.6 trivially holds for $j = k$ which constitutes the base case of an induction over j. Now let us prove the induction step. Hence, let us assume that Prop. A.6 holds for j. Now consider "$j - 1$", i.e., consider the permutation prefix $\delta' = p_0 \dots p_{j-2}$ and pick any $p_{j-1} \in P_{\delta'}$. By the induction hypothesis, we know that Prop. A.6 holds for $\delta = p_0 \dots p_{j-1}$ and any choice of $p_j \in \widetilde{P}_{\backslash\delta}$. That is, $Z^*_{\delta p_j}$ can be computed using (A.28). With this, the fixpoint algorithm in (A.27) for δ' and p_{j-1} simplifies to

$$Z^*_{\delta' p_{j-1}} = Z^*_{\delta} = \nu Y_{p_{j-1}}.\ \mu X_{p_{j-1}}.\ \bigcup_{p_j \in \widetilde{P}_{\backslash\delta}} Z^*_{\delta p_j}.$$

Here, for any choice $p_j \in \widetilde{P}_{\backslash\delta}$, the term $Z^*_{\delta p_j}$ is given by (A.28) where $r_0 = p_j$ and $\beta_{p_j} = r_1 \dots r_n$ being the *maximal permutation sequence* of $\widetilde{P}_{\backslash\delta p_j}$. Now observe that for $j > 0$ and any choice of p_j we see that $\gamma = r_0 \dots r_n$ is actually a permutation sequence of $\widetilde{P}_{\backslash\delta}$, but not necessarily the maximal one. However, observe that the maximal permutation sequence β of $\widetilde{P}_{\backslash\delta}$ (that is $\beta = \beta_{\Downarrow}$) is actually defined by $\beta = \tilde{p}_j \beta_{\tilde{p}_j}$ for $\tilde{p}_j := \max(()\widetilde{P}_{\backslash\delta})$. With this, we can apply Lem. A.9 to see that $Z^*_{\delta p_j} \subseteq Z^*_{\delta \tilde{p}_j}$ for all $p_j \in \widetilde{P}_{\backslash\delta}$. With this we obtain

$$Z^*_{\delta' p_{j-1}} = Z^*_{\delta} = \nu Y_{p_{j-1}}.\ \mu X_{p_{j-1}}.\ Z^*_{\delta \tilde{p}_j}.$$

One can now verify that this allows us to choose $r_0 = p_{j-1}$, $r_1 = \tilde{p}_j$ and $r_2 \dots r_{n+1} = \beta_{\tilde{p}_j}$ and have $r_1 > r_2 > \dots r_{n+1}$. Hence, $Z^*_{\delta' p_{j-1}}$ can be written in the form of (A.28), which proves the statement. □

Fair Adversarial Parity Games

We now consider a *parity* winning condition with a set $\mathcal{P} = \{C_1, C_2, \dots C_{2k}\}$, where each $C_i \subseteq V$ is the set of vertices of $\mathcal{G}$ with color i. Further, $\mathcal{P}$ partition's the set of vertices, i.e., $\bigcup_{i\in[1,2k]} C_i = V$ and $C_i \cap C_j = \emptyset$ for all $i, j \in [0, 2k-1]$ such that $i \neq j$.

Theorem A.6 (Thm. 7.5 restated for convenience) *Let $\mathcal{G}^\ell = \langle \mathcal{G}, E^\ell \rangle$ be a game graph with live edges and $\mathcal{P}$ be a parity condition over $\mathcal{G}$ with $2k$ colors. Further, let*

$$\begin{aligned} Z^* :=& \nu Y_{2k}.\ \mu X_{2k-1}. \dots \nu Y_2.\ \mu X_1. \\ & \cup (C_{2k} \cap \mathit{Cpre}(Y_{2k})) \cup ((C_1 \cup \dots \cup C_{2k-1}) \cap \mathit{Apre}(Y_{2k}, X_{2k-1})) \\ & \cup \ \dots \\ & \cup (C_4 \cap \mathit{Cpre}(Y_4)) \cup ((C_1 \cup C_2 \cup C_3) \cap \mathit{Apre}(Y_4, X_3)) \\ & \cup (C_2 \cap \mathit{Cpre}(Y_2)) \cup (C_1 \cap \mathit{Apre}(Y_2, X_1)) \end{aligned} \tag{A.34}$$

Then Z^* is equivalent to the winning region $\mathcal{W}$ of Player 0 in the fair adversarial game over $\mathcal{G}^\ell$ for the winning condition φ in (7.16). Moreover, the fixpoint algorithm runs in $O(n^{k+1})$ symbolic steps, and a memoryless winning strategy for Player 0 can be extracted from it.

Proof A parity winning condition $\mathcal{P}$ with $2k$ colors corresponds to the Rabin chain winning condition

$$\{\langle F_2, F_3\rangle, \dots, \langle F_{2k}, \emptyset\rangle\} \quad \text{s.t.} \quad F_i := \bigcup_{j=i}^{2k} C_j, \tag{A.35}$$

which has k pairs. Translating the Rabin chain condition induced by $\mathcal{P}$ in (A.35) into a Rabin condition as in Thm. 7.1 we get the tuple $\mathcal{R} = \{\langle G_1, R_1\rangle, \dots, \langle G_k, R_k\rangle\}$ such that

$$R_i = F_{2i+1} = \textstyle\bigcup_{j=2i+1}^{2k} C_j \tag{A.36a}$$

$$\overline{R}_i = \textstyle\bigcup_{j=1}^{2i} C_j \tag{A.36b}$$

$$G_i = F_{2i} = \textstyle\bigcup_{j=2i}^{2k} C_j \tag{A.36c}$$

$$\overline{R}_i \cap G_i = C_{2i} \tag{A.36d}$$

First, observe that $R_0 = G_0 = \emptyset$ have been artificially introduced, and result in $\widetilde{\mathcal{C}}_0 = \mathit{Apre}(Y_0, X_0)$. Further, as we have assumed that $\mathcal{P}$ is such that $\bigcup_{i\in[1,2k]} C_i = V$, we can equivalently write

$$\widetilde{\mathcal{C}}_0 = \left(\bigcup_{j=1}^{2k} C_j\right) \cup \mathit{Apre}(Y_0, X_0) = ((C_1 \cup \dots \cup C_{2k}) \cap \mathit{Apre}(Y_0, X_0))$$

For $j > 0$, by using (A.36) we observe that the definition of $\widetilde{\mathcal{C}}_j$ in (7.15b) can be written as

$$\begin{aligned}\widetilde{\mathcal{C}}_j &= (C_{2j} \cap \mathit{Cpre}(Y_j)) \cup \left(\left(\bigcup_{l=1}^{2j} C_l\right) \cap \mathit{Apre}(Y_j, X_j)\right) \\ &= (C_{2j} \cap \mathit{Cpre}(Y_j)) \cup (C_1 \cap \mathit{Apre}(Y_j, X_j)) \cup \ldots \cup (C_{2j} \cap \mathit{Apre}(Y_j, X_j))\,.\end{aligned}$$

With this, we obtain the following fixpoint equation

$$\begin{aligned}Z^* :=& \nu Y_0.\ \mu X_0.\ \nu Y_k.\ \mu X_k. \ldots \nu Y_1.\ \mu X_1. \\ & ((C_1 \cup \ldots \cup C_{2k}) \cap \mathit{Apre}(Y_0, X_0)) \\ & \cup (C_{2k} \cap \mathit{Cpre}(Y_k)) \cup ((C_1 \cup \ldots \cup C_{2k}) \cap \mathit{Apre}(Y_k, X_k)) \\ & \cup\ \ldots \\ & \cup (C_2 \cap \mathit{Cpre}(Y_1)) \cup ((C_1 \cup C_2) \cap \mathit{Apre}(Y_1, X_1))\end{aligned} \tag{A.37}$$

Now consider Lem. A.3 and let us define

$$\begin{aligned}g(X_0, Y_0) :=& ((C_1 \cup \ldots \cup C_{2k}) \cap \mathit{Apre}(Y_0, X_0)) \\ f(X_k, Y_k) :=& (C_{2k} \cap \mathit{Cpre}(Y_k)) \cup ((C_1 \cup \ldots \cup C_{2k}) \cap \mathit{Apre}(Y_k, X_k)).\end{aligned}$$

It is immediately obvious that $g(X, Y) \subseteq f(X, Y)$ for all X and Y. We can therefore apply Lem. A.3 (iv) and observe that the computation remains unchanged if we remove the fixpoint varibales X_0 and Y_0.

Now changing subscripts of iteration variables gives the following FP equation.

$$\begin{aligned}Z^* :=& \nu Y_{2k}.\ \mu X_{2k-1}. \ldots \nu Y_2.\ \mu X_1. \\ & \cup (C_{2k} \cap \mathit{Cpre}(Y_{2k})) \cup ((C_1 \cup \ldots \cup C_{2k}) \cap \mathit{Apre}(Y_{2k}, X_{2k-1})) \\ & \cup\ \ldots \\ & \cup (C_2 \cap \mathit{Cpre}(Y_2)) \cup ((C_1 \cup C_2) \cap \mathit{Apre}(Y_2, X_1))\end{aligned} \tag{A.38}$$

Now we recall from Lem. A.1 and Lem. A.2 that for all j such that $k \geq j \geq 1$ we have

$$(C_{2j} \cap \mathit{Cpre}(Y_j)) \cup (C_{2j} \cap \mathit{Apre}(Y_j, X_j)) = (C_{2j} \cap \mathit{Cpre}(Y_j)).$$

This yields

$$\begin{aligned}Z^* :=& \nu Y_{2k}.\ \mu X_{2k-1}. \ldots \nu Y_2.\ \mu X_1. \\ & \cup (C_{2k} \cap \mathit{Cpre}(Y_{2k})) \cup ((C_1 \cup \ldots \cup C_{2k-1}) \cap \mathit{Apre}(Y_{2k}, X_{2k-1})) \\ & \cup\ \ldots \\ & \cup (C_2 \cap \mathit{Cpre}(Y_2)) \cup (C_1 \cap \mathit{Apre}(Y_2, X_1))\end{aligned} \tag{A.39}$$

□

Remark For the reduction of "normal" Rabin chain games to parity games we would need to further simplify (A.39) for the special case where all $\mathit{Apre}(Y, X)$ are substituted by with $\mathit{Cpre}(Y)$. In this case, however, we observe that in any valid iteration it always holds that

$X_{i+1} \subseteq Y_i$ for all even i and $X_{j+2} \subseteq X_j$ for all odd j. We can therefore remove all terms for particular colors that have already appeared in inner fixpoint computations. Doing this yields the normal fixed-point for parity games presented in (7.18). For fair-adversarial parity games, this simplification is not possible due to the dependence of *Apre* on both Y and X.

Fair Adversarial Generalized Co-Büchi Games

Theorem A.7 (Thm. 7.6 restated for convenience) *Let $\mathcal{G}^\ell = \langle \mathcal{G}, E^\ell \rangle$ be a game graph with live edges and $\mathcal{A}$ be a generalized Co-Büchi winning condition $\mathcal{G}$ with r pairs. Further, let*

$$Z^* := \nu Y_0.\ \mu X_0.\ \bigcup_{a \in [1;r]} \nu Y_a.\ \mathit{Apre}(Y_0, X_0) \cup (\overline{A}_a \cap \mathit{Cpre}(Y_a)). \tag{A.40}$$

Then Z^* is equivalent to the winning region $\mathcal{W}$ of Player 0 in the fair adversarial game over $\mathcal{G}^\ell$ for the winning condition φ in (7.21). Moreover, the fixpoint algorithm runs in $O(rn^2)$ symbolic steps, and a memoryless winning strategy for Player 0 can be extracted from it.

In this section we prove Thm. 7.6. That is, we prove that for generalized Co-Büchi conditions, the fixpoint computing Z^* in (7.4) simplifies to the one in (A.40). This is formalized in the next proposition.

Proposition A.7 *Let $\mathcal{R} = \{\langle G_1, R_1 \rangle, \ldots, \langle G_k, R_k \rangle\}$ be a Rabin condition such that (7.22) holds. Further let Z^* be the fixed point computed by (7.4) and $\widetilde{Z}^*$ the fixed point computed by (A.40). Then $Z^* = \widetilde{Z}^*$.*

Proof Now consider the flattening of (7.4) in (A.15) for $\widetilde{\mathcal{R}}$. Then we see that for all $j > 0$ we have

$$\begin{aligned} \mathcal{C}_{\delta p_j i_j} &:= \left(Q_{\delta p_j} \cap \mathit{Cpre}(Y^*_{\delta p_j})\right) \cup \left(Q_{\delta p_j} \cap \mathit{Apre}(Y^*_{\delta p_j}, X^{i_j-1}_{\delta p_j})\right) \\ &= Q_{\delta p_j} \cap \left(\mathit{Cpre}(Y^*_{\delta p_j}) \cup \mathit{Apre}(Y^*_{\delta p_j}, X^{i_j-1}_{\delta p_j})\right) \end{aligned}$$

and we always have $X^{i_j-1}_{\delta p_j} \subseteq Y^*_{\delta p_j}$. With this, it follows from Lem. A.1 that

$$\mathcal{C}_{\delta p_j i_j} = Q_{\delta p_j} \cap \mathit{Cpre}(Y^*_{\delta p_j}) \tag{A.41}$$

for all δ, p_j and i_j with $j > 0$.

Now observe that for $\delta' = \delta p_j i_j$ and all $p_{j+1} \in P \setminus \{p_0, \ldots, p_j\}$ we have

$$Q_{\delta' p_{j+1}} = Q_{\delta p_j} \cap \overline{R}_{p_{j+1}} \subseteq Q_{\delta p_j}.$$

It further follows from the structure of the fixed point in (7.4) that

$$Y^*_{\delta p_j} = \bigcup_{i_j > 0} X^{i_j}_{\delta p_j} = \bigcup_{i_j > 0} \bigcup_{p_{j+1} \in P \setminus p_0, \ldots, p_j} Y^*_{\delta' p_{j+1}}$$

and therefore

$$Y^*_{\delta' p_{j+1}} \subseteq Y^*_{\delta p_j}.$$

With this we get

$$\mathcal{C}_{\delta' p_{j+1} i_{j+1}} \subseteq \mathcal{C}_{\delta p_j i_j}$$

for all δ, p_j and i_j with $j > 0$. Then it follows from Lem. A.3 (iii) that for every permutation sequence $\delta = p_0 p_1 \dots p_k$ the union over all $\mathcal{C}'s$ terms simplifies to two terms, one for $j = 0$ and one for $j = 1$. Using this insight, we see that for the particular Rabin condition $\widetilde{\mathcal{R}}$ the fixpoint algorithm in (7.4) simplifies to

$$\nu Y_0.\ \mu X_0.\ \bigcup_{p_1 \in P} \nu Y_{p_1}.\ \mu X_{p_1}.\ \mathcal{C}_{p_0} \cup \mathcal{C}_{p_1}. \tag{A.42}$$

Now recalling that $\mathcal{C}_{p_1}$ simplifies to $\overline{A}_a \cap \mathit{Cpre}(Y_a)$ for $a = p_1$ (see (A.41)) if (7.22) holds, and that $\mathcal{C}_{p_0} = \mathit{Apre}(Y_0, X_0)$ as $R_0 = Q_0 = \emptyset$, we see that (A.42) coincides with (A.40).□

A.2.5. Additional Proofs for Sec. 7.3

Proof of Thm. 7.7

Theorem A.8 (Thm. 7.7 restated for convenience) *Let $\mathcal{G}^\ell = \langle \mathcal{G}, E^\ell \rangle$ be a game graph with live edges and $\langle F^c, Q \rangle$ with $\mathcal{F} = \{ {}^1F, \dots, {}^sF \}$ a safe generalized Büchi winning condition. Further, let*

$$Z^* := \nu Y.\ \bigcap_{b \in [1;s]} \mu\ {}^bX.\ Q \cap \Big[({}^bF \cap \mathit{Cpre}(Y)) \cup \mathit{Apre}(Y,\ {}^bX) \Big]. \tag{A.43}$$

Then Z^* is equivalent to the winning region $\mathcal{W}$ of Player 0 in the fair adversarial game over $\mathcal{G}^\ell$ for the winning condition φ in (7.25). Moreover, the fixpoint algorithm runs in $O(sn^2)$ symbolic steps, and a finite-memory winning strategy for Player 0 can be extracted from it.

Our goal is to prove Thm. 7.7 by a reduction to Thm. 7.2 and Thm. 7.3. We therefore first show that a similar construction of an extended fixed point $\widetilde{Z}$ as in (A.10) within the proof of Thm. 7.2 also works for the generalized case. This is formalized in the following proposition.

Proposition A.8 *Given the premisses of Thm. 7.7, let*

$$Z^* := \nu Y.\ \bigcap_{b \in [1;s]} \mu\ {}^bX.\ Q \cap \Big[({}^bF \cap \mathit{Cpre}(Y)) \cup \mathit{Apre}(Y,\ {}^bX) \Big] \tag{A.44a}$$

and

$$\widetilde{Z}^* := \nu \widetilde{Y}.\ \bigcap_{b \in [1;s]} \nu\ {}^b\widetilde{Y}.\ \mu\ {}^b\widetilde{\mathcal{X}}.\ Q \cap \Big[({}^bF \cap \mathit{Cpre}(\widetilde{Y})) \cup \mathit{Apre}(\ {}^b\widetilde{Y},\ {}^b\widetilde{\mathcal{X}}) \Big]. \tag{A.44b}$$

Then $\widetilde{Z}^* = Z^*$.

However, as in (A.44) a conjunction is used to update Y, the proof is not as straight forward as for (A.10). We first show for both equations (A.44a) and (A.44b) that, upon termination, we have $Y^* = {}^bX^*$ for all $b \in [1;s]$. Both claims are formalized in Lem. A.10 and Lem. A.11, respectively.

Lemma A.10 *Given the premises of Prop. A.8, let* ${}^bX^i$ *be the set computed in the i-th iteration over the fixpoint variable* ${}^bX$ *in* (A.44a) *during the last iteration over* Y*, i.e.,* $Y = Z^*$ *already. Further, we define* ${}^bX^0 = \emptyset$ *and* ${}^bX^* := \bigcup_{i>0} {}^bX^i$*. Then it holds that* $Z^* = {}^bX^*$ *for all* $b \in [1;s]$*.*

Proof We fix $Y = Z^*$ and $b \subseteq [1;s]$ and observe from (A.44a) that

$$ {}^bX^0 = ({}^bF \cap \mathit{Cpre}(Z^*)) $$

and therefore

$$\begin{aligned} {}^bX^1 &= {}^bX^0 \cup ({}^bF \cap \mathit{Cpre}(Z^*)) \cup \mathit{Apre}(Z^*, {}^bX^0) \\ &= ({}^bF \cap \mathit{Cpre}(Z^*)) \cup \mathit{Apre}(Z^*, {}^bX^0) \supseteq {}^bX^0 \end{aligned}$$

With this, we have in general that

$$\begin{aligned} {}^bX^{i+1} &= {}^bX^i \cup ({}^bF \cap \mathit{Cpre}(Z^*)) \cup \mathit{Apre}(Z^*, {}^bX^i) \\ &= ({}^bF \cap \mathit{Cpre}(Z^*)) \cup \mathit{Apre}(Z^*, {}^bX^i) \end{aligned}$$

which implies ${}^bX^{i+1} \supseteq {}^bX^i$. Hence, ${}^bX^* := \bigcup_{i\in[0,i_{max}]} {}^bX^i = {}^bX^{i_{max}}$, and therefore, in particular

$$ {}^bX^* = ({}^bF \cap \mathit{Cpre}(Z^*)) \cup \mathit{Apre}(Z^*, {}^bX^*). \tag{A.45}$$

By recalling that $Z^* = \bigcap_b {}^bX^*$ we see that $Z^* \subseteq {}^bX^*$.

For the inverse direction, we use the observation $Z^* \subseteq {}^bX^*$ together with Lem. A.2 to see that $\mathit{Apre}(Z^*, {}^bX^*) = \mathit{Cpre}({}^bX^*)$. With this $({}^bF \cap \mathit{Cpre}(Z^*)) \subseteq \mathit{Cpre}(Z^*) \subseteq \mathit{Cpre}({}^bX^*) = \mathit{Apre}(Z^*, {}^bX^*)$ and hence (A.45) reduces to

$$ {}^bX^* = \mathit{Cpre}({}^bX^*) \supseteq \mathit{Cpre}(Z^*). $$

As the last equality holds for all $b \subseteq [1;s]$ we see that

$$ Z^* = \bigcap_b {}^bX^* = \bigcap_b \mathit{Cpre}({}^bX^*) \supseteq \mathit{Cpre}(Z^*). \tag{A.46}$$

We can now use (A.46) to proof that $Z^* \supseteq {}^bX^*$ also holds. To show this, we pick a vertex $v \in {}^bX^*$ and prove that $v \in Z^*$. To that end, observe that either (i) $v \in ({}^bF \cap \mathit{Cpre}(Z^*)) \subseteq \mathit{Cpre}(Z^*) \subseteq Z^*$ which immediately proves the statement, or (ii) $v \in \mathit{Apre}(Z^*, {}^bX^*)$. If (ii) holds we again have two cases. Either (a) $v \in \mathit{Cpre}({}^bX^*)$ which implies that there exists a finite sequence $\mathit{Cpre}(\mathit{Cpre}(\ldots \mathit{Cpre}({}^bX^1)\ldots))$ where ${}^bX^1 = {}^bF \cap \mathit{Cpre}(Z^*) \subseteq \mathit{Cpre}(Z^*) \subseteq Z^*$ and therefore $v \in \mathit{Cpre}(\mathit{Cpre}(\ldots \mathit{Cpre}(Z^*)\ldots)) \subseteq Z^*$. Finally we could have (b) that $v \in \mathrm{Pre}_l^{\exists}({}^bX^*) \cap \mathrm{Pre}_1^{\forall}(Z^*) \subseteq \mathrm{Pre}_1^{\forall}(Z^*) \subseteq \mathit{Cpre}(Z^*) \subseteq Z^*$, which again proves the statement. □

Lemma A.11 *Given the premises of Prop. A.8, let* ${}^{b}Y^{i}$ *be the set computed in the i-th iteration over the fixpoint variable* ${}^{b}Y$ *in* (A.44b) *during the last iteration over* Y*, i.e.,* $Y = \widetilde{Z}^*$ *already. Further, we define* ${}^{b}Y^{0} = V$ *and* ${}^{b}Y^{*} := \bigcap_{i>0} {}^{b}Y^{i}$*. Then it holds that* $\widetilde{Z}^* = {}^{b}Y^{*}$ *for all* $b \in [1; s]$*.*

Proof Recall that $\widetilde{Z}^* = \bigcap_b {}^{b}Y^{*}$ from the structure of the fixpoint algorithm in (A.44b). To prove $\widetilde{Z}^* = {}^{b}Y^{*}$ for all $b \in [1; s]$ it therefore suffices to show that ${}^{b}Y^{*} = {}^{b'}Y^{*}$ for any two $b, b' \in [1; s]$ s.t. $b \neq b'$.

Towards this goal, recall from Thm. 7.3 that ${}^{b}Y^{*}$ is exactly the set of states from which player 0 can win a fair adversarial reachability game with target ${}^{b}T := {}^{b}F \cap \mathit{Cpre}(\widetilde{Z}^*)$. However, every state $v \in {}^{b}T$ allows player 0 to force the game to a state $v' \in \widetilde{Z}^* = \bigcap_{b'} {}^{b'}Y^{*}$. Therefore, by definition player 0 has a strategy to reach a state $v' \in {}^{b'}Y^{*}$ from any state $v \in {}^{b}Y^{*}$ for any $b' \in [1; s]$ s.t. $b \neq b'$. As, however ${}^{b'}Y^{*}$ is defined as the winning region of player 0 w.r.t. the goal set ${}^{b'}T := {}^{b'}F \cap \mathit{Cpre}(\widetilde{Z}^*)$, we know that there actually exists a player 0 strategy to drive the game from any $v \in {}^{b}Y^{*}$ to ${}^{b'}T$, and therefore, by definition ${}^{b}Y^{*} \subseteq {}^{b'}Y^{*}$. As this inclusion holds mutually for all $b, b' \in [1; s]$ s.t. $b \neq b'$ we have that ${}^{b}Y^{*} = {}^{b'}Y^{*}$. With this, it immediately follows that $\widetilde{Z}^* = {}^{b}Y^{*}$ for all $b \in [1; s]$. □

With Lem. A.10 and Lem. A.11 in place, it remains to show that the retained fixed-points are indeed equivalent, which is achieved by the following lemma.

Lemma A.12 *Given the premises of Prop. A.8 it holds that*
(i) $Z^* \not\subset \widetilde{Z}^*$*, and*
(i) $\widetilde{Z}^* \not\subset Z^*$

Proof We show both claims by contradiction.
► *(i)* Assume $Z^* \subset \widetilde{Z}^*$. As $Y^0 = V$ and $Z^* = Y^k$ for some $k > 0$ this implies that there exists an $i > 0$ s.t. $Y^i \supseteq \widetilde{Z}^* \supset Y^{i+1}$. As $Y^{i+1} = \bigcap_b {}^{b}X^{i*}$, this implies the existence of a $b \in [1; s]$ s.t. $\widetilde{Z}^* \supset {}^{b}X^{i*}$, where

$$ {}^{b}X^{i*} = \mu\, {}^{b}X.Q \cap \Big[({}^{b}F \cap \mathit{Cpre}(Y^i)) \cup \mathit{Apre}(Y^i,\ {}^{b}X) \Big] $$

On the other hand,

$$ \widetilde{Z}^* = {}^{b}\widetilde{Y}^{**} = {}^{b}X^{***} = \mu\, {}^{b}X.Q \cap \Big[({}^{b}F \cap \mathit{Cpre}(\widetilde{Z}^*)) \cup \mathit{Apre}(\widetilde{Z}^*,\ {}^{b}X) \Big] $$

As $Y^i \supseteq \widetilde{Z}^*$ it follows from monotonicity of all involved functions that ${}^{b}X^{i*} \supseteq {}^{b}X^{***}$ which yields a contradiction.
► *(ii)* Now we assume $\widetilde{Z}^* \subset Z^*$. As $\widetilde{Y}^0 = V$ and $\widetilde{Z}^* = \widetilde{Y}^k$ for some $k > 0$ this implies that there exists an $i > 0$ s.t. $\widetilde{Y}^i \supseteq Z^* \supset \widetilde{Y}^{i+1}$.

As $Y^{i+1} = \bigcap_b {}^{b}Y^{i*}$, this implies the existence of $b \in [1; s]$ s.t. $Z^* \supset {}^{b}Y^{i*}$. We recall that

$$ {}^{b}Y^{i*} = \nu\, {}^{b}Y.\mu\, {}^{b}X.Q \cap \Big[({}^{b}F \cap \mathit{Cpre}(\widetilde{Y}^i)) \cup \mathit{Apre}({}^{b}Y^i,\ {}^{b}X) \Big] $$

Now observe that ${}^{b}Y^{i0} = V \supseteq Z^*$. Hence, for $Z^* \supset {}^{b}Y^{i*}$ to be true there must exists a j s.t. $Y^{ij} \supseteq Z^* \supset Y^{ij+1}$, where

$$ {}^{b}Y^{ij+1} = {}^{b}X^{ij*} = \mu\, {}^{b}X.Q \cap \Big[({}^{b}F \cap \mathit{Cpre}(\widetilde{Y}^i)) \cup \mathit{Apre}({}^{b}Y^{ij}, {}^{b}X) \Big] . $$

Now it is however easy to see that it follows from monotonicity again that we have $Y^{ij} \supseteq \widetilde{Z}^*$ whenever $Y^{ij} \supseteq Z^*$, which yields the intended contradiction. □

Using Prop. A.8 we know that (A.44a) and (A.44b) compute the same set. Hence, we can use (A.44b) instead of (A.43) to prove Thm. 7.7. This allows us to simply reduce the proof of Thm. 7.7 to Thm. 7.2 and Thm. 7.3 as formalized below.

Proof (Proof of Thm. 7.7) **Soundness & Completeness:** Let us define $Z^*(\langle T, Q\rangle)$ to be the set of states computed by the fixpoint algorithm in (7.9). Then it follows from (A.44b) that

$$ \widetilde{Z}^* = \nu Y. \bigcap_{b\in[1;s]} Z^*(\Big\langle Q \cap {}^{b}F \cap \mathit{Cpre}(Y), Q \Big\rangle). $$

In particular, it follows from Lem. A.11 that

$$ \widetilde{Z}^* = Z^*(\Big\langle Q \cap {}^{b}F \cap \mathit{Cpre}(\widetilde{Z}^*), Q \Big\rangle)\ \forall b \in [1;s]. $$

Now let us define ${}^{b}\mathcal{W}$ to be the fair adversarial winning state set for

$$ {}^{b}\psi = \Box Q \wedge \Box\Diamond\, {}^{b}F. $$

With this, it follows from Thm. 7.2 that $\widetilde{Z}^* = {}^{b}\mathcal{W}$ for all $b \in [1;s]$. Therefore, we obviously have $\bigcap_{b\in[1;s]} {}^{b}\mathcal{W} = \widetilde{Z}^*$. Now let $\mathcal{W}$ be the fair adversarial winning set w.r.t.

$$ \psi = \Box Q \wedge \bigwedge_{b\in[1;s]} \Box\Diamond({}^{b}F). $$

(compare (7.24)). Then we always have $\mathcal{W} \subseteq \bigcap_{b\in[1;s]} {}^{b}\mathcal{W}$ which immediately implies $\mathcal{W} \subseteq \widetilde{Z}^*$. However, as ${}^{a}\mathcal{W} = {}^{b}\mathcal{W}$ for all $a, b \in [1;s]$, we know that ψ holds for all $v \in \widetilde{Z}^*$, hence $Z^* \subseteq \mathcal{W}$.

Strategy construction: We can define a rank function for every b as in (A.4) within the proof of Thm. 7.3 (see App. A.2.2), i.e.,

$$ {}^{b}\mathrm{rank}(v) = i \quad \text{iff} \quad v \in {}^{b}X^{i} \setminus {}^{b}X^{i-1}. \tag{A.47} $$

Then, we have a different strategy, ${}^{b}\rho_0$, which is defined via (A.4) (see App. A.2.2) using the corresponding ${}^{b}\mathrm{rank}$ function. With this, we define a new strategy ρ which circles through all possible goal sets in a pre-defined order. That is

$$ \rho_0(v, b) = \begin{cases} {}^{b}\rho_0(v) & v \notin {}^{b}F \\ {}^{b^+}\rho_0(v) & v \in {}^{b}F \end{cases} \tag{A.48} $$

where $b^+ = b+1$ if $b < s$ and $b^+ = 1$ if $b = s$.

The strategy in (A.48) is obviously winning for ψ in (7.24) as every ${}^{b}\rho_0$ is a winning strategy for ${}^{b}\psi$ (from Thm. 7.2) and upon reaching ${}^{b}F$ we know that the respective state v is also contained in $Cpre(\widetilde{Z}^*)$ where $\widetilde{Z}^* = {}^{b^+}Y^*$. Now it follows from the definition of $Cpre$ that $Cpre({}^{b^+}Y^*) \subseteq {}^{b^+}Y^*$, hence, allowing to apply ${}^{b^+}\rho_0$ upon reaching ${}^{b}F$. □

Proof for Thm. 7.8

Theorem A.9 (Thm. 7.8 restated for convenience) *Let $\mathcal{G}^\ell = \langle \mathcal{G}, E^\ell \rangle$ be a game graph with live edges and $\widetilde{\mathcal{R}}$ be a generalized Rabin condition over $\mathcal{G}$ with index set $P = [1;k]$. Further, let*

$$\begin{aligned} Z^* :=& \nu Y_0.\ \mu X_0. \\ & \bigcup_{p_1 \in P} \nu Y_{p_1}. \bigcap_{l_1 \in [1;m_{p_1}]} \mu\, {}^{l_1}X_{p_1}. \\ & \quad \ddots \\ & \bigcup_{p_k \in P \setminus \{p_1,\dots,p_{k-1}\}} \nu Y_{p_k}. \bigcap_{l_k \in [1;m_{p_k}]} \mu\, {}^{l_k}X_{p_k}. \bigcup_{j=0}^{k} {}^{l_j}\mathcal{C}_{p_j}, \end{aligned} \tag{A.49a}$$

where

$${}^{l_j}\mathcal{C}_{p_j} := \left(\bigcap_{i=0}^{j} \overline{R}_{p_i} \right) \cap \Big[\Big({}^{l_j}G_{p_j} \cap Cpre(Y_{p_j}) \Big) \cup Apre(Y_{p_j},\ {}^{l_j}X_{p_j}) \Big]$$

with $p_0 = 0$, $G_{p_0} := \{\emptyset\}$ and $R_{p_0} := \emptyset$. Then Z^* is equivalent to the winning region $\mathcal{W}$ of Player 0 in the fair adversarial game over $\mathcal{G}^\ell$ for the winning condition φ in (7.24). Moreover, the fixpoint algorithm runs in $O(n^{k+2}k!m_1 \dots m_k)$ symbolic steps, and a finite-memory winning strategy for Player 0 can be extracted from it.

We show how the proof of Thm. 7.1 in App. A.2.3 needs to be adapted in order to prove the generalized version of Thm. 7.1, namely Thm. 7.8, instead.

Strategy Construction: Similar to the finite-memory strategy constructed for generalized Büchi games in App. A.2.5, the strategy for generalized Rabin games needs to remember the index of all the goal sets currently "chaised" for each permutation index up to p_j. To formalize this, we define the set of full goal chain sequences for a given generalized Rabin specification $\widetilde{\mathcal{R}}$ by

$$\Phi(\widetilde{\mathcal{R}}) := \{\ell_0\ell_1 \dots \ell_k \mid \ell_0 = 1,\ \ell_j \in [0;m_j]\}. \tag{A.50}$$

If $\widetilde{\mathcal{R}}$ is clear from the context we simply write Φ. Given a goal chain prefix $\phi := \ell_0\ell_1 \dots \ell_{j-1}$ we can now construct a ranking for each such prefix, using the flattening of (A.49) instead of (7.4). This yields the following proposition which follows from Prop. A.1 by simply annotating all terms with the goal chain prefix ϕ.

Proposition A.9 *Let $\delta = p_0 i_0 \dots p_{j-1} i_{j-1}$ be a* configuration prefix*, $\phi := \ell_0 \ell_1 \dots \ell_{j-1}$ a* goal chain prefix*, $p_j \in P \setminus \{p_1, \dots, p_{j-1}\}$ the next permutation index, $\ell_j \in [1; m_{p_j}]$ the next goal set and $i_j > 0$ a counter for p_j. Then the flattening of* (A.49) *for this configuration and goal prefix is given by*

$$ {}^{\phi\ell_j}X^{i_j}_{\widetilde{\delta p_j}} = \underbrace{{}^{\phi}S_\delta \cup {}^{\ell_j}\mathcal{C}_{\delta p_j i_j}}_{{}^{\phi\ell_j}S_{\delta p_j i_j}} \cup {}^{\phi\ell_j}\mathcal{A}_{\delta p_j i_j} \tag{A.51a} $$

where

$$ Q_{p_0 \dots p_a} := \bigcap_{b=0}^{a} \overline{R}_{p_b}, \tag{A.51b} $$

$$ {}^{\ell_a}\mathcal{C}_{\delta p_a i_a} := \Big(Q_{\delta p_a} \cap {}^{\ell_a}G_{p_a} \cap Cpre(Y^*_{\delta p_a})\Big) \cup \Big(Q_{\delta p_a} \cap Apre(Y^*_{\delta p_a}, {}^{\ell_a}X^{i_a-1}_{\delta p_a})\Big) $$

$$ {}^{\ell_0 \dots \ell_a}S_{p_0 i_0 \dots p_a i_a} := \bigcup_{b=0}^{a} {}^{\ell_b}\mathcal{C}_{p_0 i_0 \dots p_b i_b}, \tag{A.51c} $$

$$ {}^{\phi\ell_i}A_{\delta p_j i_j} := \textstyle\bigcup_{p_{j+1} \in P \setminus \{p_1, \dots, p_j\}} \Big(\bigcap_{\ell_{j+1} \in [1; m_{p_{j+1}}]} \Big(\bigcup_{i_{j+1} > 0} \Big({}^{\phi\ell_j\ell_{j+1}}X^{i_{j+1}}_{\delta p_j i_j p_{j+1}} \setminus {}^{\phi\ell_i}S_{\delta p_j i_j}\Big)\Big)\Big). \tag{A.51d} $$

Again we see that this flattening follows directly from the structure of the fixpoint algorithm in (A.49) and the definition of ${}^{l_j}\mathcal{C}_{p_j}$ in (7.27b). Using the flattening of (A.49) in (A.51) we can define a ranking function for each goal chain prefix ϕ identical to Def. A.1. That is, given the premises of Prop. A.9, we define ${}^{\phi\ell_j}R : V \to 2^{\widetilde{D}}$ s.t. (i) $\infty \in {}^{\phi\ell_j}R(v)$ for all $v \in V$, and (ii) $\delta p_j i_j \underline{\gamma} \in {}^{\phi\ell_j}R(v)$ iff $v \in {}^{\phi\ell_j}S_{\delta p_j i_j}$. The ranking function ${}^{\phi}\mathrm{rank} : V \to D$ is then again defined as in Def. A.1 s.t. ${}^{\phi}\mathrm{rank} : v \mapsto \min\{{}^{\phi}R(v)\}$. Similarly, we can define a memoryless winning strategy for every fixed goal sequence ϕ as in (A.16). That is,

$$ {}^{\phi}\rho_0(v) := \min_{(v,w) \in E}({}^{\phi}\mathrm{rank}(w)). \tag{A.52} $$

Now, similar to the proof of Thm. 7.7 (see Sec. 7.7) we can "stack" these memoryless winning strategies to define a new strategy with finite memory which circles through all possible goal sets in a pre-defined order. That is

$$ \rho_0(v, \phi\ell_j) := \begin{cases} {}^{\phi\ell_j}\rho_0(v) & v \notin {}^{\ell_j}F \\ {}^{\phi\ell_j^+}\rho_0(v) & v \in {}^{\ell_j}F \end{cases} \tag{A.53} $$

where $\ell_j^+ := \ell_j + 1$ if $\ell_j < m_{p_j}$ and $\ell_j^+ := 1$ if $\ell_j = m_{p_j}$.

Using this goal chain dependent ranking function, the proof of soundness and completeness of (A.49) along with the proof that ρ_0 in (A.53) is indeed a winning strategy for player 0 in the fair adversarial generalized Rabin game, follows exactly the same lines as the proof in App. A.2.3. That is, we iteratively consider instances of the flattening in

(A.51), starting with $j = k$ as the base case, and doing an induction from "$j+1$" to "j". To this end, we consider a *generalized* local winning condition which refers not only to the current configuration-prefix $\delta = p_0 i_0 \dots p_{j-1} i_{j-1}$ but also to the current goal chain prefix $\phi := \ell_0 \dots \ell_{j-1}$. Hence, (A.17) gets modified to

$$
{}^{\phi}\psi_{\delta p_j} := \left(\begin{array}{ll} & Q_{\delta p_j} \mathcal{U}\ {}^{\phi}S_{\delta} \\ \vee & \Box Q_{\delta p_j} \wedge \bigwedge_{\ell_j \in [1;m_{p_j}]} \Box\Diamond\ {}^{\ell_j}G_{p_j} \\ \vee & \Box Q_{\delta p_j} \wedge \left(\bigvee_{i \in \widetilde{P}_{\setminus j}} \left(\Diamond\Box \overline{R}_i \wedge \bigwedge_{b \in [1;m_i]} \Box\Diamond\ {}^{b}G_i \right) \right) \end{array} \right) \tag{A.54}
$$

where $\widetilde{P}_{\setminus j} = P \setminus \{p_0, \dots, p_j\}$. With this, it becomes obvious that the proof of soundness, completness and the winning strategy for Thm. 7.8 follows exactly the same reasoning as in App. A.2.3 while additionally using Thm. 7.7 to reason about the conjunction over goal sets.

The only remaining part to be shown concerns the last line of ${}^{\phi}\psi_{\delta p_j}$. For this, we recall from App. A.2.3 that the induction step from "$j+1$" to "j" relies on the fact that

$$
{}^{\phi\ell_j}\Psi_{\delta p_j} := \Box Q_{\delta p_j} \wedge \Diamond \left(\bigvee\nolimits_{p_{j+1} \in P \setminus \{p_1, \dots, p_j\}} {}^{\phi'}\psi'_{\delta' p_{j+1}} \right) \tag{A.55}
$$

is indeed equivalent to the last line of ${}^{\phi}\psi_{\delta p_j}$, where ${}^{\phi'}\psi'_{\delta' p_{j+1}}$ denotes the last two lines of ${}^{\phi'}\psi_{\delta' p_{j+1}}$ with $\phi' := \phi\ell_j$ and $\delta' := \delta p_j$.

For (non-generalized) Rabin games this equivalence is proved in App. A.2.3. It can be seen by inspection within this proof, that using a conjunction over goal sets instead of a single goal set within the second and third line of ${}^{\phi}\psi_{\delta p_j}$ does not change any step in the derivation. Therefore, the same derivation can be used in the generalized case and is therefore omitted. This concludes the proof of Thm. 7.8.

Proof of Thm. 7.9

Theorem A.10 (Thm. 7.9 restated for convenience) *Let $\mathcal{G}^{\ell} = \langle \mathcal{G}, E^{\ell} \rangle$ be a game graph with live edges and $(\mathcal{A}, F^c)$ a GR(1) winning condition. Further, let*

$$
Z^* = \nu Y_k. \bigcap_{b \in [1;s]} \mu\ {}^{b}X_k. \bigcup_{a \in [1;r]} \nu Y_a. \ (F_b \cap \mathit{Cpre}(Y_k)) \cup \mathit{Apre}(Y_k,\ {}^{b}X_k) \cup (\overline{A}_a \cap \mathit{Cpre}(Y_a)).
$$

Then Z^* is equivalent to the winning region $\mathcal{W}$ of Player 0 in the fair adversarial game over $\mathcal{G}^{\ell}$ for the winning condition φ in (7.28). Moreover, the fixpoint algorithm runs in $O(n^2 rs)$ symbolic steps, and a finite-memory winning strategy for Player 0 can be extracted from it.

Within this section we proof Thm. 7.9. That is, we prove that for GR(1) winning conditions, the fixpoint computing Z^* in (A.49) simplifies to the one in (7.30). This is formalized in the next proposition.

Proposition A.10 *Let $\widetilde{\mathcal{R}}$ be a generalized Rabin condition with k pairs s.t.* (7.29) *holds for $r := k-1$. Further let Z^* be the fixed point computed by* (A.49) *and $\tilde{Z}^*$ the fixed point computed by* (7.30)*. Then $Z^* = \tilde{Z}^*$.*

If Prop. A.10 holds, we immediately see that Thm. 7.9 directly follows from Thm. 7.8. It therefore remains to prove that Prop. A.10 holds.

Proof First, consider an arbitrary permutation sequence $\delta = p_0 \dots p_k$. Then we know that there exists exactly one $j > 0$ s.t. $p_j = k$ and all other indices come from the set $[1;r]$. We can therefore define $\gamma' = p_1 \dots p_{j+1}$ and $\gamma'' = p_{j+1} \dots p_k$ s.t. $p_i \in [1;r]$ for all $i \neq j$. We note that $\gamma' = \varepsilon$ if $j = 1$ and $\gamma'' = \varepsilon$ if $j = k$. With this we have $\delta = p_0\gamma' p_j \gamma''$.

By inspecting (7.29) we see that the first r pairs of the generalized Rabin condition induced by the GR(1) specification actually form a Generalized Co-Büchi condition (compare (7.22) in Sec. 7.2.4). Hence, given a permutation sequence $\delta = p_0\gamma' p_j \gamma''$ we can use the same reasoning as in the proof of Thm. 7.6 in App. A.2.4 to see that

$$\mathcal{C}_{p_1} \supseteq \dots \supseteq \mathcal{C}_{p_{j-1}} \text{ and } \mathcal{C}_{p_{j+1}} \supseteq \dots \supseteq \mathcal{C}_{p_k}. \tag{A.56}$$

Now recall from the proof of Thm. 7.4 in App. A.2.4 that these inclusions allow to recursively apply Lem. A.3 to delete all $\mathcal{C}$ terms which are included in either $\mathcal{C}_{p_1}$ or $\mathcal{C}_{p_{j+1}}$ along with the fixpoint variables used within these terms (compare Lem. A.6 where now γ' and γ'' are interpreted as decreasing sub-sequences). Applying these simplifications to (A.49) (in exactly the same manner as these simplifications where applied to (7.4) in the proof of Thm. 7.4) results in a simpler fixpoint algorithm where all permutation sequences have the form $\delta = 0q_1kq_2$ with $q_1 \neq q_2$ and $q_1, q_2 \in [1;r]$ (here q_1 and q_2 correspond to p_1 and p_{j+1} in (A.56), and k corresponds to p_j).

Now we can inspect (7.29) again to see that $R_i \supseteq R_k$ and $G_i \supseteq {}^b G_{p_j}$ for all $i \in [1;r]$ and $b \in [1;s]$. This can be understood as a "generalized Rabin chain condition" (compare (7.14) in Sec. 7.2.4). Hence, we can apply Lem. A.6 one more time, now to the "decreasing sub-sequence" q_1k within every permutation sequence. Again, utilizing this argument iteratively in (A.49) yields a simpler fixpoint algorithm which only contains permutation sequences $\delta = 0ka$ with $a \in [1;r]$. This proves that Z^* is equivalent to the set

$$\nu Y_0.\ \mu X_0.\ \nu Y_k.\ \bigcap_{b\in[1;s]} \mu\ {}^b X_0.\ \bigcup_{a\in[1;r]} \nu Y_a.\ \mu X_a.\quad \mathcal{C}_{p_0} \cup\ {}^b\mathcal{C}_k \cup \mathcal{C}_a.$$

Now inserting the simplifications for terms from the generalized Co-Büchi part (see (A.41) in App. A.2.4) and using $R_0 = G_0 = \emptyset$, we obtain

$$\begin{aligned}&\nu Y_0.\ \mu X_0.\ \nu Y_k.\ \bigcap_{b\in[1;s]} \mu\ {}^b X_0.\ \bigcup_{a\in[1;r]} \nu Y_a.\\ &\quad Apre(Y_0, X_0) \cup (\ {}^b F \cap Cpre(Y_k)) \cup Apre(Y_k,\ {}^b X_k) \cup (\overline{A}_a \cap Cpre(Y_a)).\end{aligned}$$

Now we can apply Lem. A.3 (iii) again to remove the first occurrence of the *Apre* term to obtain the same expression as in (7.30). This concludes the proof. □

A.2.6. Additional Proofs for Sec. 7.4

Preliminaries

$1\frac{1}{2}$-player game: A special case of $2\frac{1}{2}$-player game graphs is a *Markov Decision Process* (MDP) or $1\frac{1}{2}$-*player game*, which is obtained by assuming that every *Player* 0 vertex in V_0 has only one outgoing edge.[1] Analogously to the $2\frac{1}{2}$-player games, for a given $1\frac{1}{2}$-player game graph $\mathcal{G}$, we use the notation $P^{\rho_1}_{v^0}(\mathcal{G} \models \varphi)$ to denote the probability of occurrence of the event $\mathcal{G} \models \varphi$ when the runs initiate at v^0 and when *Player* 1 uses the strategy ρ_1.

Role of end components in $1\frac{1}{2}$-player game: Limiting behaviors in a $1\frac{1}{2}$-player game can be characterized using the structure of the underlying game graph. We summarize one key technical argument in the following.

Let $\mathcal{G} = \langle V, V_0, V_1, V_r, E\rangle$ be a $1\frac{1}{2}$-player game graph. A set of vertices $U \subseteq V$ is called *closed* if (1) for every $v \in U \cap V_r$, $E(v) \subseteq U$, and (2) for every $v \in U \cap (V_0 \cup V_1)$, $E(v) \cap U \neq \emptyset$. A closed set of vertices U induces a *subgame graph* $(V', V'_0, V'_1, V'_r, E')$, denoted by $\mathcal{G} \downarrow U$, which is itself a $1\frac{1}{2}$-player game graph and is defined as follows:

- $V' = U$,
- $V'_0 = U \cap V_0$,
- $V'_1 = U \cap V_1$,
- $V'_r = U \cap V_r$, and
- $E' = E \cap (U \times U)$.

A set of vertices $U \subset V$ of a $1\frac{1}{2}$-player game graph $\mathcal{G}$ is an *end component* if (a) U is closed, and (b) the subgame graph $\mathcal{G} \downarrow U$ is strongly connected.

Denote the set of all end components of $\mathcal{G}$ by $\mathcal{E} \subset 2^V$. The next lemma states that under every strategy ρ_1 (being memoryless or not) of *Player* 1 in the $1\frac{1}{2}$-player game, the set of states visited infinitely often along a play is an end component with probability one.

Lemma A.13 *(De Alfaro, 1997, Thm. 3.2) For every $1\frac{1}{2}$-player game graph, for every vertex $v \in V$, and every Player 1 strategy ρ_1,*

$$P^{\rho_1}_v\left(\mathcal{G} \models \bigvee_{U \in \mathcal{E}} \left(\Diamond\Box U \wedge \bigwedge_{u \in U} \Box\Diamond u\right)\right) = 1. \tag{A.57}$$

This lemma implies the following corollary, which is motivated by similar claim for Rabin winning conditions in the literature Chatterjee et al. (2005).

[1] Alternatively, we could also define $1\frac{1}{2}$-player game graphs by restricting the outgoing edges from the *Player* 1 vertices; our choice is actually tailored for the content of the rest of the section.

Corollary A.1 *For a given $1\frac{1}{2}$-player game, for a given vertex $v \in V$, and for a given Player 1 strategy ρ_1, a generalized Rabin condition $\widetilde{\mathcal{R}} = \{\langle G_1, R_1\rangle, \ldots, \langle G_k, R_k\rangle\}$ is satisfied almost surely if and only if for every end component U reachable from v^0, there is a $j \in \{1, 2, \ldots, k\}$ such that $U \cap R_j = \emptyset$ and for every $l \in [1; m_j]$, $U \cap {}^lG_j \neq \emptyset$.*

Proof of Thm. 7.10

Theorem A.11 (Thm. 7.10 restated for convenience) *Let $\mathcal{G}$ be a $2\frac{1}{2}$-player game graph, $\widetilde{\mathcal{R}}$ be a generalized Rabin condition, $\varphi \subseteq V^\omega$ be the corresponding LTL specification (Eq. (7.24)) over the set of vertices V of $\mathcal{G}$, and Derand($\mathcal{G}$) be the reduced two-player game graph. Let $\mathcal{W} \subseteq \widetilde{V}$ be the set of all the vertices from where Player 0 wins the fair adversarial game over Derand($\mathcal{G}$) for the winning condition φ, and $\mathcal{W}^{a.s.}$ be the almost sure winning set of Player 0 in the game graph $\mathcal{G}$ for the specification φ. Then, $\mathcal{W} = \mathcal{W}^{a.s.}$. Moreover, a winning strategy in Derand($\mathcal{G}$) is also a winning strategy in $\mathcal{G}$, and vice versa.*

We define the fairness constraint on the random edges of $\mathcal{G}$ as per Eq. (7.1):

$$\varphi^\ell := \wedge_{(v,v')\in E_r} \Box\Diamond v \rightarrow \Box\Diamond(v \wedge \bigcirc v').$$

We first show that $\mathcal{W} \subseteq \mathcal{W}^{a.s.}$. Consider an arbitrary initial vertex $v^0 \in \mathcal{W}$ and an arbitrary strategy ρ_1 of *Player* 1 in $\mathcal{G}$. Let ρ_0^* be a corresponding winning strategy for *Player* 0 from v^0 for the fair adversarial game over *Derand*($\mathcal{G}$) for the winning condition φ. By definition, ρ_0^* realizes the specification φ, whenever the adversary satisfies the strong fairness condition on the live edges in *Derand*($\mathcal{G}$). On the other hand, the live edges in *Derand*($\mathcal{G}$) are exactly the random edges in $\mathcal{G}$. In other words, we already know that if we apply the *same* strategy ρ_0^* to $\mathcal{G}$, then $\inf_{\rho_1 \in R_1} P_{v^0}^{\rho_0^*,\rho_1}(\mathcal{G} \models \varphi^\ell \rightarrow \varphi) = 1$.

We first show that the random edges E_r also satisfy the strong fairness condition φ^ℓ *almost surely*; actually we show that the probability of violation of φ^ℓ in $\mathcal{G}$ is 0. Consider the following:

$$\begin{aligned}
P_{v^0}^{\rho_0^*,\rho_1}\left(\mathcal{G} \models \neg\varphi^\ell\right) &= P_{v^0}^{\rho_0^*,\rho_1}\left(\mathcal{G} \models \neg \bigwedge_{(v,v')\in E_r} \Box\Diamond v \rightarrow \Box\Diamond(v \wedge \bigcirc v')\right) \\
&= P_{v^0}^{\rho_0^*,\rho_1}\left(\mathcal{G} \models \bigvee_{(v,v')\in E_r} \Box\Diamond v \wedge \Diamond\Box\neg(v \wedge \bigcirc v')\right) \\
&\leq \sum_{(v,v')\in E_r} P_{v^0}^{\rho_0^*,\rho_1}\left(\mathcal{G} \models \Box\Diamond v \wedge \Diamond\Box\neg(v \wedge \bigcirc v')\right).
\end{aligned}$$

We show that the right-hand side of the last inequality equals to 0 by proving that for every $(v, v') \in E_r$,

$$P_{v^0}^{\rho_0^*,\rho_1}\left(\mathcal{G} \models \Box\Diamond v \wedge \Diamond\Box\neg(v \wedge \bigcirc v')\right) = 0.$$

Consider any arbitrary $(v, v') \in E_r$ and assume that the probability of taking the edge (v, v') from v is p_1. Let π be a play on $\mathcal{G}$ and $(i_0, i_1, i_2, \ldots)$ be the infinite sequence of

time indices when the vertex v is visited. For every i_k, the probability of *not* visiting v' for the next l time steps $(i_{k+1}+1, \ldots, i_{k+l}+1)$ is given by $(1-p)^l$, which converges to 0 as l approaches ∞. This proves that for every i_k, eventually there will be a v' at (i_k+1) with probability 1; in other words v' will be visited infinitely often with probability 1. Hence, it follows that $\sum_{(v,v')\in E_r} P_{v^0}^{\rho_0^*,\rho_1}(\mathcal{G} \models \Box\Diamond v \wedge \Diamond\Box\neg(v \wedge \bigcirc v')) = 0$, which in turn establishes that $P_{v^0}^{\rho_0^*,\rho_1}\left(\mathcal{G} \models \neg\varphi^\ell\right) = 0$.

Now consider the following derivation:

$$\begin{aligned} P_{v^0}^{\rho_0^*,\rho_1}(\mathcal{G} \models \varphi^\ell \to \varphi) = P_{v^0}^{\rho_0^*,\rho_1}(\mathcal{G} \models \neg\varphi^\ell \vee \varphi) &\leq P_{v^0}^{\rho_0^*,\rho_1}(\mathcal{G} \models \neg\varphi^\ell) + P_{v^0}^{\rho_0^*,\rho_1}(\mathcal{G} \models \varphi) \\ &= 0 + P_{v^0}^{\rho_0^*,\rho_1}(\mathcal{G} \models \varphi) = P_{v^0}^{\rho_0^*,\rho_1}(\mathcal{G} \models \varphi). \end{aligned}$$

Since we know that $P_{v^0}^{\rho_0^*,\rho_1}(\mathcal{G} \models \varphi^\ell \to \varphi) = 1$, hence it follows that $P_{v^0}^{\rho_0^*,\rho_1}(\mathcal{G} \models \varphi) = 1$.

Next, we show that $\mathcal{W} \supseteq \mathcal{W}^{a.s.}$. Consider an arbitrary initial vertex $v^0 \in \mathcal{W}^{a.s.}$. Let ρ_0^* be a corresponding almost sure winning strategy for *Player* 0 from v^0 in the $2\frac{1}{2}$-player game $\mathcal{G}$ with the specification φ. We show that *Player* 0 wins the fair adversarial game over $\mathit{Derand}(\mathcal{G})$ for the winning condition φ from vertex v^0 using the strategy ρ_0^*.

Let $\rho_1 \in R_1$ be any arbitrary *Player* 1 strategy in the game $\mathit{Derand}(\mathcal{G})$ such that the unique resultant play $\pi = (v^0, v^1, \ldots)$ due to ρ_0^* and ρ_1 satisfies the fairness assumption. We use the notation $\mathrm{Inf}(\pi)$ to denote the set of infinitely occurring vertices along the play π, i.e., $\mathrm{Inf}(\pi) := \{w \in V \mid \forall m \in \mathbb{N}_0 \,.\, \exists n > m \,.\, v^n = w\}$. First we show that (i) the set of vertices $\mathrm{Inf}(\pi)$ forms an end component in $\mathcal{G}$, and moreover (ii) there exists a *Player* 1 strategy ρ_1' in the game $\mathcal{G}$ such that $P_{v^0}^{\rho_0^*,\rho_1'}(\mathcal{G} \models \mathrm{Inf}(\pi)) > 0$. Claim (i) follows by observing the following:

- For all $v \in \mathrm{Inf}(\pi) \cap V_r$, $V_r(v) \subseteq \mathrm{Inf}(\pi)$, as otherwise in $\mathit{Derand}(\mathcal{G})$ there would be a vertex in $E^\ell(v)$ and outside $\mathrm{Inf}(\pi)$ which would be visited infinitely many times due to infinitely many visits to v.
- For every $v \in \mathrm{Inf}(\pi) \cap (V_0 \cup V_1)$, $E(v) \neq \emptyset$, as otherwise in $\mathit{Derand}(\mathcal{G})$ the play π would reach a dead-end.
- The subgame graph $\mathcal{G} \downarrow \mathrm{Inf}(\pi)$ is strongly connected, as otherwise in $\mathit{Derand}(\mathcal{G})$ there would be two vertices $u, v \in \mathrm{Inf}(\pi)$ so that v would not be reachable from u, contradicting the assumption that both u and v are visited infinitely often by π.

Claim (ii) follows by defining a strategy $\rho_1' \equiv \rho_1$ on $\mathcal{G}$. Now observe that for every edge (v, v') chosen by *Player* 1 from a vertex $v \in \mathrm{dom}(()E^\ell)$ in $\mathit{Derand}(\mathcal{G})$, there exists a corresponding positive probability edge (v, v') in $\mathcal{G}$. Since $\mathrm{Inf}(\pi)$ is entered by π after finite time steps, hence the Claim (ii) follows.

Now, from Cor. A.1 it follows that there is a $j \in \{1, 2, \ldots, k\}$ such that $\mathrm{Inf}(\pi) \cap R_j = \emptyset$ and for every $l \in \{1, \ldots, m_j\}$, $\mathrm{Inf}(\pi) \cap {}^lG_j \neq \emptyset$. Thus the play π satisfies the generalized Rabin condition $\widetilde{\mathcal{R}}$. Since this holds for any arbitrary *Player* 1 strategy, hence $\mathcal{W} \supseteq \mathcal{W}^{a.s.}$ and ρ^* is the corresponding winning strategy for *Player* 0.

A.3. The Accelerated Fixpoint Algorithm

Consider the fixpoint algorithm in (7.4). In the correctness proof of Thm. 7.1 discussed in App. A.2.3, we have been remembering so called configuration prefixes $\delta = p_0 i_0 \dots p_{j-1} i_{j-1}$ for some $j \leq k$ for every fixed-point variable X (see Eq. (A.13)). We denoted by $X^{i_j}_{\delta p_j}$ the set of states computed in the i_j'th iteration of the fixed-point over X_{p_j} after the fixed-point over Y_{p_j} has already terminated within the i_{j-1}th iteration over $X_{p_{j-1}}$ after the fixed-point over $Y_{p_{j-1}}$ has terminated in the i_{j-2}th iteration over $X_{p_{j-2}}$ and so forth.

In order to describe the accelerated implementation of (7.4), we do not assume that the fixed-points over Y-variables have already terminated, but additionally remember their counters m. This leads to configuration prefixes $\delta = p_0 m_0 i_0 \dots p_{j-1} m_{j-1} i_{j-1}$ and lets us define that $X^{m_j i_j}_{\delta p_j}$ is the set of states computed in the i_jth iteration of the fixed-point over X_{p_j} during the m_jth iteration over Y_{p_j}, computing the set $Y^{m_j}_{\delta p_j}$ and so forth.

Given two configuration prefixes $\delta = p_0 m_0 i_0 \dots p_{j-1} m_{j-1} i_{j-1}$ and $\delta' = p'_0 m'_0 i'_0 \dots p'_{j-1} m'_{j-1} i'_{j-1}$ we define $\delta <_m \delta'$ if $p_0 \dots p_{j-1} = p'_0 \dots p'_{j-1}$, $m_0 \dots m_{j-1} < m'_0 \dots m'_{j-1}$ (using the induced lexicographic order) and $i_0 \dots i_{j-1} = i'_0 \dots i'_{j-1}$. We define $\delta <_i \delta'$ similarly.

Now Piterman and Pnueli (2006) showed, based on a result of Long et al. (1994), that for every configuration prefix $\delta = p_0 m_0 i_0 \dots p_{j-1} m_{j-1} i_{j-1}$ the computation of $Y^0_{\delta p_j}$ can start from the *minimal* set $Y^{m_j}_{\delta' p_j}$ (instead of the entire set of vertices V) such that $\delta' p_j m_j <_m \delta p_j 0$. Dually, for every configuration prefix $\delta = p_0 m_0 i_0 \dots p_{j-1} m_{j-1} i_{j-1}$ the computation of $X^{m_j 0}_{\delta p_j}$ can start from the *maximal* set $X^{m_j i_j}_{\delta' p_j}$ (instead of the empty set) such that $\delta' p_j m_j i_j <_i \delta p_j m_j 0$.

Further, we see that for the innermost fixpoint, i.e. when $j = k$, it follows that for every computation prefix δ , there can be at most n iterations over both Y_{p_k} and X_{p_k}, where n is the total number of vertices. I.e., n different sets $Y^{m_k}_{\delta p_k}$ and $X^{m_k i_k}_{\delta p_k}$ have to be freshly computed for each δp_k and $\delta p_k m_k$ respectively. We see that there are $\mathcal{O}(n^{k+1}k!)$ different such permutation sequences. As the computation of the innermost fixpoint dominates the computation time, it is shown by Long et al. (1994) that this results in an overall worst-case computation time of $\mathcal{O}(n^{(k+1)+1}k!) = \mathcal{O}(n^{k+2}k!)$ (where n is the total number of vertices and k is the number of Rabin pairs).

Unfortunately, the memory requirement of this acceleration algorithm is enormous. To see this, observe that in order to warm-start the computation of $Y^0_{\delta p_j}$ with $\delta = p_0 m_0 i_0 \dots p_{j-1} m_{j-1} i_{j-1}$ we need to store the current minimal set w.r.t. the m-prefix for every combination of p- and i-prefixes that can occur in δ, which are $\mathcal{O}(n^{k+1}k!)$ many. Similarly, to warm-start the computation of $X^{m_j i_j}_{\delta p_j}$ we need to store the current minimal set w.r.t. the i-prefix for every combination of p- and m-prefixes that can occur in δ. This means that the memory required by the algorithm is $\mathcal{O}(n^{k+1}k!)$, which is prohibitively large for large values of n and k.

We implemented a *space-bounded* version of the acceleration algorithm, where for any given parameter M (chosen by the user), we stored only up to M values for each counter. Whenever the values of all the counters are less than M, we use the regular acceleration algorithm as outlined above. Otherwise, if any of the counters exceeds M, then we fall

back to the regular initialization procedure of fixpoint algorithms, i.e. depending on whether it is an Y or an X variable, initialize it with V or $\emptyset$ respectively. As a result, the memory requirement of our accelerated fixpoint algorithm is given by $\mathcal{O}(M^{k+1}k!)$. This space-bounded acceleration algorithm made our implementation much faster and yet practically feasible, as has been demonstrated in Sec. 7.5.

A.4. Supplementary Results for the Experiments

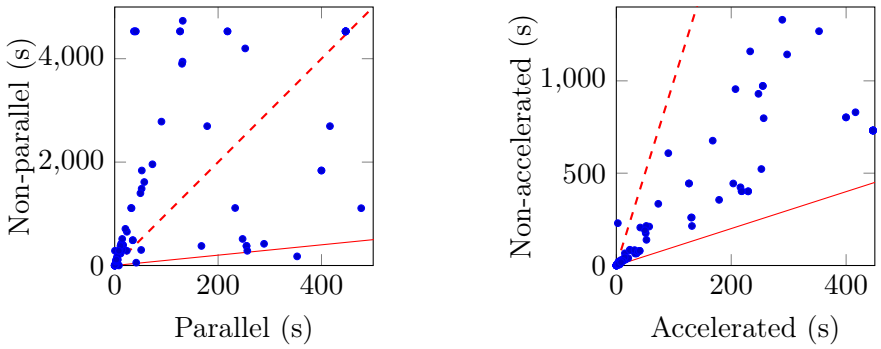


Figure A.2.: Zoomed-in version of Fig. 7.5. (Left) Comparison between the computation times for the non-parallel (1 worker thread) and parallel (48 worker threads) version of Fairsyn, with acceleration being enabled in both cases. (Right) Comparison between the computation times for the non-accelerated and the accelerated version of Fairsyn, with parallelization being enabled in both cases. (Both) The points on the solid red line represent the same computation time. The points on the dashed red line represent an order of magnitude improvement.

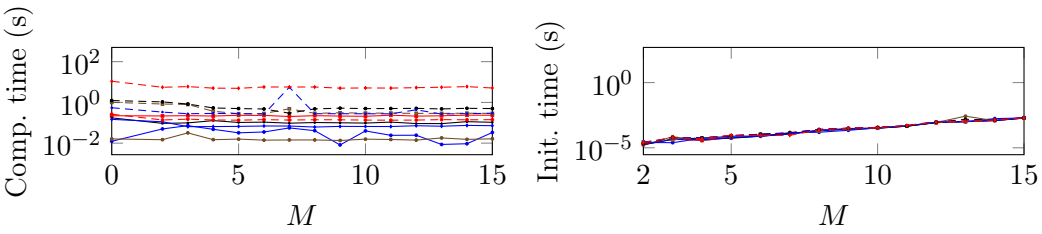


Figure A.3.: (Left) Effect of variation of the acceleration parameter M on the total computation time (parallelization being enabled) for the VLTS benchmark examples with 1 Rabin pair. (Right) Effect of variation of the acceleration parameter M on the initialization time for the VLTS benchmark examples with 1 Rabin pair. The computation time (Y-axis) in both the plots are shown in the logarithmic scale.

Number of Vertices	Number of Transitions	Number of Live Edges	Number of BDD Variables
289	1224	17	9
289	1224	25	9
289	1224	13	9
1952	2387	1	11
1952	2387	5	11
1952	2387	25	11
1183	4464	16	11
1183	4464	49	11
1183	4464	9	11
3995	14,552	39	12
3995	14,552	139	12
3995	14,552	153	12
5121	9392	1	13
5121	9392	54	13
5121	9392	73	13
8879	24,411	473	14
8879	24,411	397	14
7119	38,424	626	14
7119	38,424	835	14
7119	38,424	597	14
10,849	56,156	241	14
10,849	56,156	482	14
18,746	73,043	1585	15
18,746	73,043	1729	15
18,746	73,043	575	15
25,216	25,216	137	15
25,216	25,216	595	15
25,216	25,216	373	15
40,006	60,007	1130	16
40,006	60,007	865	16
52,268	292,823	107	16
52,268	292,823	3254	16
65,537	524,293	13,727	17
65,537	524,293	25,229	17
66,929	569,322	23,290	17
66,929	569,322	13,698	17
69,753	359,575	11,071	17
69,753	359,575	5058	17
83,435	259,488	1682	17
83,435	259,488	2707	17
96,878	282,880	6225	18
96,878	282,880	585	18

Table A.1.: Details of the fair adversarial Rabin games randomly generated from the VLTS benchmark suite. Continued to Table A.2.

Number of Vertices	Number of Transitions	Number of Live Edges	Number of BDD Variables
116,456	364,596	8316	17
116,456	364,596	7774	17
142,471	925,429	19,259	18
142,471	925,429	3304	18
164,865	1,619,200	13,407	18
164,865	1,619,200	24,868	18
166,463	518,976	13,633	18
166,463	518,976	4155	18
214,140	683,205	13,588	18
214,140	683,205	12,113	18
371,804	641,565	3413	19
371,804	641,565	12,151	19
386,496	1,171,870	26,247	19
386,496	1,171,870	17,823	19
566,639	3,984,160	7109	20
566,639	3,984,160	42,757	20

Table A.2.: Continued from Table A.1. Details of the fair adversarial Rabin games randomly generated from the VLTS benchmark suite.

Broad. Queue Size	Out. Queue Size	Number of Vertices	Number of Transitions	Number of Live Edges	Num. of BDD Variables	Time (seconds)
1	1	5,307,840	10,135,300	5,124,100	25	7.37
2	1	21,231,400	40,541,200	20,496,400	27	24.90
3	1	21,414,100	42,080,300	21,265,900	27	28.97
1	2	21,340,800	40,879,100	20,834,300	27	38.25
1	3	21,559,400	42,756,100	21,772,800	27	51.55
4	1	84,925,400	162,165,000	81,985,500	29	57.70
5	1	85,295,700	165,243,000	83,524,600	29	65.01
6	1	85,656,300	168,321,000	85,063,700	29	73.19
7	1	86,007,400	171,399,000	86,602,800	29	77.97
1	4	85,363,200	163,516,000	83,337,200	29	92.56
1	5	85,808,000	167,270,000	85,214,200	29	113.18
2	2	85,363,200	163,516,000	83,337,200	29	133.20
1	6	86,237,400	171,024,000	87,091,200	29	135.67
3	2	86,061,400	169,673,000	86,415,400	29	144.27
1	7	86,651,500	174,778,000	88,968,200	29	145.76
8	1	339,702,000	648,659,000	327,942,000	31	149.68
2	3	86,237,400	171,024,000	87,091,200	29	163.62
9	1	340,447,000	654,815,000	331,020,000	31	174.29
10	1	341,183,000	660,972,000	334,098,000	31	197.02
3	3	86,870,100	177,181,000	90,169,300	29	203.15
1	8	341,453,000	654,066,000	333,349,000	31	248.38
1	9	342,350,000	661,574,000	337,103,000	31	283.85
1	10	343,232,000	669,082,000	340,857,000	31	331.78
7	2	345,587,000	691,003,000	351,818,000	31	567.26
4	2	341,453,000	654,066,000	333,349,000	31	710.78
2	4	341,453,000	654,066,000	333,349,000	31	806.74
5	2	342,868,000	666,378,000	339,505,000	31	852.37
6	2	344,246,000	678,691,000	345,661,000	31	936.04
2	5	343,232,000	669,082,000	340,857,000	31	1034.57
4	3	344,950,000	684,098,000	348,365,000	31	1071.52
2	7	346,606,000	699,113,000	355,873,000	31	1111.64
7	3	348,693,000	721,035,000	366,834,000	31	1312.88
2	6	344,950,000	684,098,000	348,365,000	31	1336.35
5	3	346,233,000	696,410,000	354,521,000	31	1351.31
3	4	344,246,000	678,691,000	345,661,000	31	1632.63
6	3	347,480,000	708,723,000	360,677,000	31	1667.54
8	2	1,365,810,000	2,616,260,000	1,333,400,000	33	2478.13
9	2	1,368,660,000	2,640,890,000	1,345,710,000	33	2783.77

Table A.3.: Experimental evaluation for the code-aware resource management case study (extended table).

Curriculum Vitae

Research Interests

Formal verification and control of dynamical systems, which are either continuous or discrete, stochastic or non-stochastic, and with a distributed or a monolithic structure.

Education

[2016–]	Doctoral student (Computer Science), Max Planck Institute for Software Systems, Germany. Advisor: Rupak Majumdar.	
[2013–2015]	M.Tech in System and Control, (first division with distinction, ranked first in the dept.), Department of Electrical Engineering, IIT Roorkee, India.	CGPA: 9.69/10.
[2008–2012]	B.Tech in Electrical Engineering, Meghnad Saha Institute of Technology, India.	CGPA: 8.62/10.