PERMISSIVE ASSUMPTIONS IN LOGICAL CONTROLLER SYNTHESIS FOR CYBER-PHYSICAL SYSTEMS

Thesis approved by
the Department of Computer Science
RPTU University Kaiserslautern-Landau
for the award of the Doctoral Degree
Doctor of Natural Sciences (Dr. rer. nat.)

to

Satya Prakash Nayak

Date of Defense:	26.03.2026
Dean:	Prof. Dr. Christoph Garth
Reviewer:	Dr. Anne-Kathrin Schmuck
Reviewer:	Prof. Dr. Bernd Finkbeiner
Reviewer:	Prof. Dr. Jean-François Raskin

DE-386

To Mama (my sister) ...

Summary

The synthesis of logical controllers that guarantee desired specifications is a central problem in the design of cyber-physical systems (CPS). In practice, such guarantees rely on assumptions about how the controller interacts with its environment. These assumptions restrict environment behavior to make synthesis feasible, but in existing approaches they are often overly restrictive, leading to conservative designs and limiting the range of behaviors that systems can safely accommodate.

This thesis rethinks the role of assumptions in logical controller synthesis by emphasizing their *permissiveness*—the ability to capture a wide range of admissible environment behaviors. We study permissive assumptions in two key settings: (a) interactions among multiple discrete components in distributed systems, and (b) interactions between high-level logical controllers and low-level physical dynamics in hybrid systems. In both settings, we develop theoretical and algorithmic foundations for computing and exploiting permissive assumptions to enable new design paradigms for logical controller synthesis.

For distributed systems, we define permissiveness as capturing all cooperative behaviors of other components that enable a controller to satisfy its specification. We present an algorithm for computing such assumptions in monolithic systems and extend it to distributed systems via a negotiation-based framework that iteratively constructs permissive assume-guarantee contracts for each component. These contracts enable decentralized synthesis and are applied to human-robot interaction, allowing robots to cooperate with humans whenever possible and request cooperation only when necessary.

For hybrid systems, we utilize permissive assumptions on the plant model—the abstract representation of physical dynamics—to address three key challenges. To enable seamless adaptation of controllers to changing logical contexts, i.e., changes in high-level goals or tasks, we introduce a novel synthesis framework that utilizes *persistent live groups*, a class of assumptions capturing liveness properties of continuous dynamics. To improve scalability to large or uncertain plant models, we develop *universal controllers* where decisions are conditioned on branching-time assumptions called *prophecies*, which are learned from representative models and efficiently verified at runtime on unseen plant models. Finally, to enhance robustness under uncertainty or partial violations of assumptions on the plant model, we introduce a robust semantics for branching-time temporal logics, enabling formal reasoning about controller behavior under such violations.

Overall, this work enables correctness-by-construction synthesis while avoiding unnecessary conservatism, resulting in CPS that are more robust, scalable, and responsive.

Zusammenfassung

Die Synthese logischer Regelungsoftware, die das gewünscht Verhalten eines technischen Systems sicherstellt, ist ein zentrales Problem beim Entwurf von cyber physikalischen Systemen. Meist basieren die resultierenden Verhaltensgarantien dann auf Annahmen darüber, wie das System mit seiner Umgebung interagiert. Bestehende Ansätze beruhen jedoch oft auf sehr restriktiven Annahmen, was zu konservativen Regelungsverhalten führt und das Umgebungsverhalten begrenzt was Systeme sicher tolerieren können.

Diese Arbeit entwickelt einen neunen Ansatz zum logischen Reglerentwurf, indem sie die Permissivität von Annahmen – das heißt die Fähigkeit ein breites Spektrum zulässiger Umgebungsverhalten zu modellieren – in den Vordergrund stellt. Konkret betrachten wir permissive Annamhen in zwei zentralen Bereichen: (a) Interaktionen zwischen mehreren diskreten Komponenten in verteilten Systemen und (b) Interaktionen zwischen übergeordneten logischen Reglern und der physikalischen Dynamik in hybriden Systemen. In beiden Anwendungsbereichen entwickeln wir theoretische und algorithmische Grundlagen für die Berechnung und Nutzung permissive Annahmen, um neue modellbasierte Entwurfsverfahren für logische Regelungssoftware zur Verfügung zu stellen.

Für verteilte Systeme definieren wir Permissivität als die Fähigkeit alle kooperativen Verhaltensweisen anderer Komponenten abzubilden die einen Reglerentwurf möglich machen. Wir präsentieren einen Algorithmus zur Berechnung solcher Annahmen in monolithischen Systemen und erweitern ihn auf verteilte Systeme mittels eines verhandlungsbasierten Ansatzes, das iterativ permissive Annahme-Garantie-Verträge für jede Komponente erstellt. Diese Verträge ermöglichen dezentrale Synthese und wurden von uns in der Mensch-Roboter-Interaktion angewendet.

Für hybride Systeme nutzen wir permissive Annahmen über das Streckenmodell—d.h. die abstrakte logische Repräsentation der physikalischen Streckendynamik—um drei zentrale Herausforderungen zu bewältigen. Um eine natlose Anpassung von REgelungssoftware an sich ändernde logische Kontexte, zu ermöglichen, führen wir einen neuartigen Synthese-Ansatz ein, das neuartige permissive Annahmen nutzt um Letztendlichkeitseigentschaften der Streckendynamik zu erfassen. Um die Skalierbarkeit auf große oder nicht genau bekannte Streckenmodelle zu erweitern, entwickeln wir so genannte *universelle Regler* deren Entscheidungen auf *Prophezeiungen* basieren, die mithilfe repräsentativer kleiner Modelle gelernt wurden. Um die Robustheit gegenüber Unsicherheiten oder teilweisen Verletzungen der Annahmen über das Streckenmodell zu erhöhen, führen wir eine robuste Semantik für Verzweigungszeitlogiken ein.

Acknowledgements

This thesis marks the conclusion of my work at MPI-SWS, a journey shaped by the support of many individuals across several institutions. I have done my best to mention everyone who helped me along the way, though it is difficult to fully capture everyone's contribution in such a brief note. If I have accidentally omitted anyone's name, please know that your help was nonetheless deeply appreciated.

The most significant impact on my development as a researcher has been made by my advisor, Anne-Kathrin Schmuck. I am profoundly grateful for her guidance, her patience as I navigated complex problems, and her belief in my potential. I have also been fortunate to collaborate with and be mentored by brilliant researchers such as Bernd Finkbeiner, Daniel Neider, and Martin Zimmermann, whose insights greatly enriched my work. I am also thankful to Rupak Majumdar and Björn Brandenburg for their rigorous and constructive feedback during our group meetings, which helped me refine my ideas and presentations.

My path toward research began at Rtapalli Vidyapitha. I want to thank the school and its founder, Bidyut Baran Das, for encouraging me toward this field. I owe a special debt of gratitude to my professors at the Chennai Mathematical Institute. I would like to thank B. Srivathsan, C. Aishwarya, and Prajakta Nimbhorkar for introducing me to the world of theoretical computer science and formal methods.

A special thanks to the administrative and technical staff at MPI-SWS, including Andreas, Corinna, Geraldine, Lisa-Marie, Mary-Lou, Ruth, Susanne, Tobias, Torsten, and Vera, for making the logistics of research much easier to navigate. I also want to thank Rose for guiding me in developing the skills required for writing clear scientific texts and delivering fluent presentations.

My time in Kaiserslautern has been defined by the friendships I formed here. These years were made better by the people who shared this journey with me through countless dinners, coffee breaks, and game nights. I want to thank Ali, Amir, Andrea, Andreea, Arabinda, Arash, Aris, Bite, Chris, Corto, Eirene, Eleni, Faezeh, Felix, Filip, Germano, Iason, Ivan, Kaushik, Khushraj, Kilian, Kimaya, Leo, Lia, Mahdi, Marin, Mehrdad, Michalis, Mihir, Mohammad, Munko, Nastaran, Nikhil, Nina, Oz, Pascal, Pavel, Pushpdeep, Ram, Rosa, Richard, Seungeon, Srinidhi, Stratis, and Xuan for making my time here more enjoyable. I am thankful to Annika, Ben, Ellen, Ezgi, Hasan, Ivi, Maria, Mariam, Numair, and Suhas for the amazing trips and memories that enriched my experience. I also want to thank Ana, Bala, Irmak, and Mahmoud for the random discussions

that provided a welcome break from research and made daily life less stressful. Most importantly, I want to thank my closest friends, Ashwani, Rajarshi, Ritam, and Sathiya, for the gym sessions, shared meals, etc., that provided a necessary balance to my work.

Finally, I want to thank my parents for everything they have done to help me reach this stage. I am lucky to have my wife, Rinu, whose patience and encouragement have been essential to my progress throughout these years. I also want to thank my extended family and friends for their wishes and for their support from afar. This thesis is dedicated to my sister, Mama, who has been my closest companion and a constant source of calm throughout my life.

Contents

Chapter 1

Introduction

Over the past decade, cyber-physical systems (CPS) are increasingly being deployed in the real world and have become central to many modern technologies. They are now commonly used in areas such as autonomous driving, industrial automation, and aerial robotics. Examples include self-driving cars on roads, robots assembling products, and drones being used for surveillance and inspection tasks. While these systems offer impressive capabilities and open up new possibilities, their growing use also raises concerns about the consequences of failures. There have been serious accidents where CPS failed in critical ways, e.g., autonomous vehicles causing fatal accidents [3, 5], aircraft crashes linked to software issues [1], and industrial robots causing harm to operators [192]. As these systems become more autonomous and complex, the consequences of such failures, both in terms of reliability and cost, become more significant.

Consequently, a fundamental challenge in CPS design is ensuring that the systems behave correctly, i.e., they do exactly what they are designed to do. For instance, autonomous vehicles must eventually reach their destinations without crashing, industrial robots must complete their tasks without harming workers, and drones must operate safely without colliding with buildings or people. This has led to a need for strong correctness guarantees across all possible operating scenarios. To address these challenges, formal verification has emerged as a powerful approach that enables designers to mathematically analyze and prove that a system satisfies its required specifications [175, 200].

While formal verification provides powerful tools for post-hoc analysis of correctness of an existing system, a more proactive approach is to ensure *correctness-by-construction*. These methodologies aim at the automated construction of systems that are guaranteed to satisfy their formal specifications in all possible scenarios. This eliminates the need for extensive post-hoc verification and allows for a more efficient design process.

Correctness-by-construction methodologies have roots in two largely independent communities: reactive synthesis and control theory. Reactive synthesis [178] is based on a logical specification of the system's desired behavior, typically given as a temporal logic formula, and aims to automatically construct a controller that satisfies this specification. In contrast, traditional control theory [183] is based on a mathematical model of the system's physical behavior and aims to design controllers based on this model to

ensure that the system behaves correctly in the physical world. CPS often require a combination of both viewpoints, e.g., the behavior of the robot is specified using temporal logic, while the physics of the robot's interaction with its *environment*, such as walls, humans, and other robots, as well as the physics of their interaction, is described by a *plant model*. This has led to many approaches [210, 56] that automatically synthesize *logical controllers* that enforce the desired logical specifications while respecting the physical behavior of the system as described by the plant model.

A common technique in such approaches is to model this interaction between the logical controller and its environment (as described by the plant model) along with the logical specification as a two-player *graph game* between the controller and the environment [178, 210, 56]. The winning condition of this game encodes the logical specification, and a winning strategy for the controller player corresponds to a logical controller that enforces the desired behavior. Hence, the synthesis of logical controllers in such settings reduces to computing winning strategies in these graph games.

To be able to compute such winning strategies, most of these approaches rely on a set of *assumptions* about how the controller interacts with its environment. These assumptions are typically used to restrict the behavior of the environment in a way that enables the synthesis of a logical controller to ensure the specification. For instance, in the context of industrial robots, an assumption might be that human workers will not enter the robot's workspace while it is operating. This assumption allows the logical controller for the robot to be synthesized without needing to account for the possibility of human interference. In *distributed systems*—which is composed of multiple interacting components—assumptions are used to restrict the behavior of other components. For example, in a workspace shared by multiple robots, a robot may make assumptions that nearby robots will follow certain protocols to avoid collisions. Without such assumptions, it may be impossible to guarantee that the robot can safely navigate the workspace. Moreover, in *hybrid systems*—which combine discrete logical controllers with continuous physical dynamics—assumptions are also used on the plant model of a CPS to abstract its continuous dynamics. For example, assumptions on the physical dynamics of a mobile robot may include that it can always move in a straight line without slipping, allowing the logical controller to be synthesized without needing to consider complex physical interactions like friction or uneven terrain. Such assumptions are crucial for enabling the synthesis of logical controllers that can ensure the desired specifications in these complex settings.

A key property that is often overlooked in existing approaches is the *permissiveness* of these assumptions. In many cases, the assumptions are overly restrictive and fail to admit the full range of behaviors that its environment may exhibit without violating the specification. For instance, in distributed systems, assumptions on other components often reduce their behavior to a single fixed strategy, which limits their flexibility and robustness. Similarly, in hybrid systems, assumptions on the plant model may fail to reflect the actual dynamics of the system, resulting in conservative controllers that perform poorly in real-world conditions.

This thesis addresses these limitations by proposing novel synthesis frameworks that

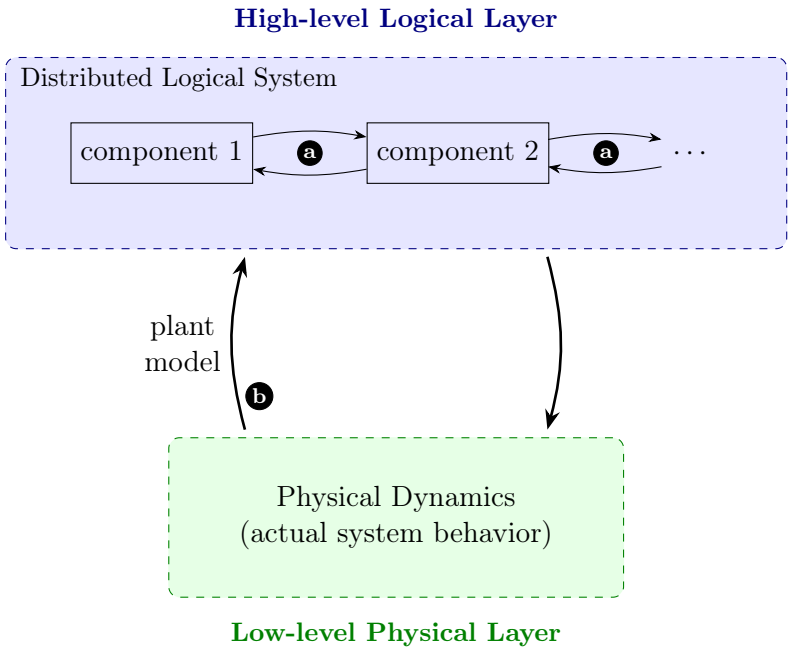


Figure 1.1: Two types of interaction in cyber-physical systems: (a) interaction between multiple discrete components at the high-level logical layer, and (b) interaction between the high-level logical layer and the low-level physical layer.

fundamentally rethink the role of assumptions in logical controller synthesis. The key insight is that assumptions play a central, multifaceted role in CPS design, and their *permissiveness*—a critical yet often overlooked property—which enables new design approaches that have not been explored in prior work. More specifically, we[1] focus on two distinct settings where permissive assumptions can significantly enhance the design of logical controllers: (a) assumptions in distributed logical systems, and (b) assumptions on the plant model in hybrid systems. These correspond to two distinct types of interaction in CPS, as illustrated in Figure 1.1: interaction between multiple discrete components at the high-level logical layer (shown with a), and interaction between the high-level logical layer and the low-level physical layer (shown with b). Building on these insights, we develop new algorithms and frameworks that compute and utilize permissive assumptions in both settings, leading to more flexible, robust, and efficient logical controllers for CPS.

In distributed logical systems a, we define permissiveness as the ability to capture *all cooperative behaviors* of other components that enable the controller to satisfy its specification. This leads to more flexible and decentralized designs, where each component can independently choose any strategy consistent with the assumptions. We present a novel algorithm for automatically computing such permissive assumptions in

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

monolithic systems—systems with a single controller interacting with its environment—modeled as two-player games. Building on this foundation, we extend our approach to distributed systems, where each component has its own local specification, modeled as multi-player games. This extension enables decentralized synthesis of logical controllers through negotiation frameworks that iteratively compute and refine assumptions until a set of permissive assume-guarantee contracts is established, one for each component, under which all components can satisfy their local specifications. We then apply these contracts to develop a novel framework for human-robot interaction, in which the robot leverages its guarantees to dynamically adapt its strategy, cooperating with the human whenever possible while requesting cooperation online only when necessary to ensure the human fulfills their assumptions.

For hybrid systems (b), we focus on the permissiveness of assumptions on the plant model—that is, assumptions derived from the underlying physical dynamics of the CPS. Utilizing permissive assumptions, we address three key challenges in synthesizing logical controllers for complex systems: (i) seamless integration of logical controllers with continuous dynamics, (ii) scalability to large or uncertain plant models, and (iii) robustness under uncertainty or partial violations of assumptions on the plant model.

First, we focus on *seamless integration* of logical controllers with the underlying continuous dynamics. In such systems, the logical controller needs to react to changes in the logical context, e.g., a change in the target location of a robot, a new task assigned to a drone, or a sudden obstacle appearing in the path of an autonomous vehicle. This challenge is particularly pronounced as such changes are triggered by the external environment and can occur at any time. To enable seamless reactivity to these context changes, we introduce a novel synthesis framework that utilizes a new class of assumptions on the plant model, called *persistent live groups*. Such assumptions capture rich liveness properties of the continuous dynamics, enabling the logical controller to adapt its strategy dynamically based on the current logical context.

Second, we tackle *scalability* to large plant models. Often, to capture the full range of behaviors of the physical dynamics, the plant model needs a huge state space. Consequently, the standard approaches to construct a game graph from the plant model suffers from state explosion, making them infeasible for large-scale systems. Even more importantly, when the plant model is uncertain or only known at runtime, then the usual state explosion becomes even more impractical. We address this by constructing a *universal controller* derived from the logical specification alone, whose decisions are conditioned by assumptions on the plant model, called *prophecies*. These prophecies are learned from a small set of representative plant models and expressed as branching-time temporal logic formulas, which approximate the possible future branching structures of the plant model (e.g., the ability to reach certain goal locations). This allows us to generalize the controller's behavior across different plant models by efficiently verifying the learned prophecies on unseen plant models at runtime, thereby improving efficiency and scalability.

Finally, we address *robustness* under uncertainty or partial violations of such assumptions on the plant model. Given that many assumptions on the plant model are

branching-time in nature, we leverage techniques from temporal logic to reason about the controller's behavior under uncertainty. In particular, we consider the setting where assumptions on the plant model are expressed as branching-time temporal logic formulas. To this end, we propose a new robust semantics for branching-time temporal logics, which allows us to formally reason about how logical controllers tolerate such uncertainty or violations.

1.1 Contributions

Based on the two main settings outlined above, we design the theoretical and algorithmic foundations for computing and utilizing permissive assumptions in logical controller synthesis. In particular, our contributions can be summarized as follows:

1. We develop a polynomial-time algorithm for computing permissive environment assumptions in two-player ω-regular games, expressed as local edge constraints in the game graph. These constraints capture all possible cooperative behaviors of the environment under which the controller can satisfy its specification.

2. We present negotiation-based distributed synthesis algorithms that compute permissive assume-guarantee contracts for each component in a multi-player game in order to satisfy the local specifications of each component. We show that our algorithm is sound and complete in case of cooperative components, and sound in case of rational components.

3. We introduce a novel framework for human-robot interaction based on assume-guarantee contracts, where the robot can dynamically adapt its strategy to cooperate with the human whenever possible and request human cooperation online only when necessary to guarantee progress. We demonstrate the effectiveness of our framework through simulations and a real-world case study with a robotic arm.

4. We introduce a novel synthesis framework for hybrid control systems which automatically designs a *novel interface* between the plant and the high-level logical controller based on a new form of assumptions on the plant model. These interfaces enable the resulting hybrid controllers to seamlessly react to changes in the logical context. We further analyze the complexity of solving games under such assumptions and provide a novel algorithm for solving them.

5. We propose a universal controller synthesis framework where a *generic* controller is derived solely from the logical specification, and its behavior is conditioned by assumptions on the plant model, called *prophecies*, expressed as branching-time temporal logic formulas. We show that these prophecies can be learned from a small set of representative plant models, which generalizes to unseen plant models via a verification step. This offers improved efficiency and explainability through small formulas as prophecies.

6. We propose a new robust semantics for branching-time temporal logics and analyze its expressiveness compared to existing logics. We show that standard problems such as satisfiability, model checking, and synthesis remain in the same complexity classes as their non-robust counterparts.

Our algorithms and frameworks build on established concepts from the reactive synthesis community, such as temporal logic, ω-regular graph games, and assume-guarantee contracts. We provide the proof of correctness and complexity results for all our algorithms. Furthermore, we implement these algorithms in several user-friendly, open-source prototype tools, and evaluate their performance empirically on a range of benchmarks from the literature. Where applicable, we also compare our implementations against existing tools to highlight the improvements in permissiveness achieved by our approaches.

The frameworks and algorithms presented in this thesis provide a principled way to preserve correctness-by-construction while avoiding unnecessary conservatism via permissive assumptions. In doing so, they enable autonomous systems to operate safely across a wider range of scenarios, to interact more effectively with dynamic and uncertain environments, and to make better use of their physical capabilities. Ultimately, this leads to the design of CPS that are not only provably correct, but also more robust, scalable, and responsive in the real world.

1.2 Intuitive Overview of the Results

In the following, we provide a brief overview of the main results of this thesis, organized according to the contributions mentioned in Section 1.1.

1.2.1 Permissive Assumptions in Two-Player Games

As our first contribution, we focus on the permissiveness of environment assumptions for monolithic systems, modelled as two-player games. As mentioned before, logical controller synthesis often relies on a game-theoretic formulation, modeling the interaction between a controller and its environment as a two-player game played over a finite graph. This graph is obtained from an abstract plant model that captures the interaction between the controller and its environment. The desired logical behaviors of the system are specified as ω-regular objectives for the controller player in the game. A solution of such a game is a set of decisions for the controller player, i.e., a *winning strategy*, that ensures satisfying the given ω-regular objective over the states of the game. This winning strategy is then used to design the logical controller for the system.

Traditionally, these two-player games are solved in a zero-sum fashion, i.e., assuming that the environment will behave arbitrarily and possibly adversarially. Although this approach results in robust system designs, it usually makes the environment too powerful to allow a winning strategy for the controller to exist. However, in reality, many of the application areas, especially in distributed systems where components are co-designed, actually account for some cooperation of the environment (e.g., other components). In this scenario it is useful to understand how the environment needs to cooperate to allow for

a winning strategy to exist. This can be formalized by environment assumptions, which are ω-regular properties that restrict the moves of the environment player in a synthesis game. Such assumptions can then be used as additional specifications in other components' synthesis problems to enforce the necessary cooperation (possibly in addition to other local requirements) or can be used to verify existing models of the environment.

For the reasons outlined above, the automatic computation of environment assumptions has received significant attention in the reactive synthesis community. It has been used in two-player games [57, 55], both in the context of monolithic system design [65, 148] as well as distributed system design [145, 95]. Furthermore, in the context of specification-repair in zero-sum games, multiple automated methods for repairing environment models exist [194, 103, 104, 152, 57].

All these works emphasize two desired properties of assumptions. They should be (i) *sufficient*, i.e., enable the controller player to win if the environment obeys its assumption and (ii) *implementable*, i.e., prevent the controller player to falsify the assumption and thereby vacuously win the game by not even respecting the original specification. However, in the context of synthesis of distributed systems, we argue that a third important property is often overlooked, which is the *permissiveness* of the assumptions. By permissiveness, we mean that the assumptions should retain all cooperatively winning plays in the game, i.e., all plays that satisfy the specification if both players cooperate. This notion is crucial in the setting of distributed systems, as here assumptions are generated *before* the model of every component is fixed. Therefore, assumptions need to retain *all* feasible ways of cooperation to allow for a solution to be discovered in a decentralized manner. Without permissiveness, the controller synthesis problem for the controller player may become infeasible, even though there exists a solution that satisfies the original specification.

We address this limitation by introducing the notion of *adequately permissive assumptions* (APAs)—assumptions that are not only sufficient and implementable, but also permissive. We present a polynomial-time algorithm to compute APAs for two-player games with *parity objectives*, which are a canonical representation of ω-regular objectives and more expressive than the classical LTL (linear temporal logic) specifications.

Our key insight is that APAs can be captured using local edge constraints on the game graph. In particular, we show that APAs can be expressed using three simple templates that can be directly extracted from a cooperative synthesis algorithm for the game. This leads to a polynomial-time algorithm for computing APAs for parity games, which is significantly more efficient than the existing approaches for computing environment assumptions [55, 57].

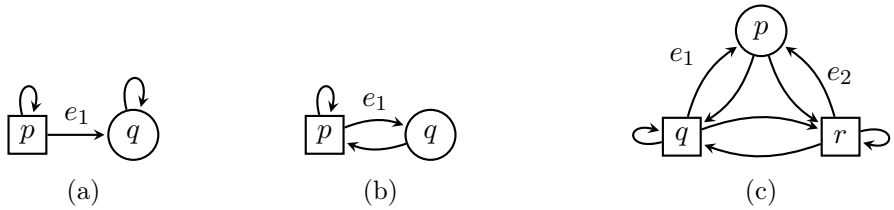


Figure 1.2: Game graphs with environment (squares) and controller (circles) vertices.

Expressing APAs using Simple Templates. To appreciate the simplicity of the assumption templates we use, let us illustrate them with some simple examples. Consider the game graphs depicted in Figure 1.2 where the controller and the environment player control the circle and square vertices, respectively. The game begins at some designated initial vertex and proceeds as follows: at each step, the owner of the current vertex chooses an outgoing edge to move to the next vertex. The game continues indefinitely, resulting in an (infinite) play, which is a sequence of vertices visited in the game graph. The controller player wins the game if the outcome play satisfies a given ω-regular specification.

For instance, in the game depicted in Figure 1.2a, consider the specification $\Phi = \Diamond\Box\{p\}$, i.e., the play should eventually only see vertex p. Here, it is easy to observe that the controller player can win the game by requiring the environment to disable edge e_1. This is captured by our first template type that marks e_1 as an *unsafe edge*.

In contrast, consider the game in Figure 1.2b with the same specification $\Phi = \Diamond\Box\{p\}$. Here, the controller player can win the game by requiring the environment to disable edge e_1 only finitely often. This is captured by our second template type that marks e_1 as a *co-live edge*.

Furthermore, consider the game in Figure 1.2c with the specification $\Phi = \Box\Diamond\{p\}$, i.e. vertex p should be seen infinitely often. Here, the controller player wins if whenever the source vertices of edges e_1 and e_2 are seen infinitely often, also one of these edges is taken infinitely often. This is captured by our third template type that marks the edge-group $\{e_1, e_2\}$ as a *live group*. For more general specifications like $\Phi = \Box\Diamond\{q\} \Rightarrow \Box\Diamond\{p\}$, i.e. if vertex q is seen infinitely often, then vertex p should be seen infinitely often, the above live group template is generalized to a *conditional live group* conditioned on set $\{q\}$ such that the live group templates needs to be satisfied only when the set $\{q\}$ is seen infinitely often. We show that the combination of these templates are sufficient to capture APAs for any parity game (and hence any ω-regular games).

Computing APAs for Parity Games. We present a polynomial-time algorithm to compute an APA for a given parity game. The algorithm is based on the following key observation about the templates: (a) The unsafe edges can be used to stop the environment from taking a move that leads to not satisfying the specification. This is done by marking an edge as unsafe if it goes out of the *cooperative winning region*, i.e., the set of vertices from which both players can ensure that the specification is satisfied. (b) The co-live edges can be used to ensure that the environment does not take a move

infinitely often that goes to the *co-Büchi region*, i.e., the set of vertices that has to be visited only finitely often to satisfy the specification. (c) The live groups can be used to ensure that the environment takes the right moves infinitely often (when enabled) in order to move towards a target region. Furthermore, the conditional live groups can be used to ensure moving towards the target region under certain conditions. With these observations, the algorithm starts by computing the cooperative winning region and the corresponding unsafe edges. Then, it iteratively computes the co-Büchi region and the corresponding co-live edges. Finally, as standard methods for solving parity games are based on the computation of attractors to reach certain target regions, the algorithm iteratively computes the corresponding conditional live groups for such attractors.

From the above observations, one can see that for safety games, only the unsafe edges are needed to express an APA. With a similar intuition, we also show that for other simpler classes of games, such as Büchi (resp. co-Büchi) games, an APA can be expressed using the combination of only unsafe edges and live groups (resp. co-live edges), and can be directly computed as post-processing of the standard algorithms for solving these games.

1.2.2 Negotiation Frameworks for Multi-Player Games

While Section 1.2.1 forms the basis by introducing adequately permissive assumptions (APAs) on environment in monolithic systems, we now utilize this concept to restrict the behavior of multiple interacting components in distributed systems (as shown in Figure 1.1 (top)) in order to synthesize controllers for each component. In particular, we consider the setting of distributed synthesis where each component has its own logical specification, and the goal is to synthesize a controller for each component such that all components satisfy their respective specifications when interacting with each other. Although the classical distributed synthesis problem is undecidable in general [176], a common and tractable variant assumes that the strategic interaction structure among components is known and represented by an abstract plant model.

One common approach to solve such distributed synthesis problems is to use *assume-guarantee reasoning*, where each component is synthesized under the assumption that other components behave in a certain way. This allows to reduce the distributed synthesis problem to computing an *assume-guarantee contract* for each component, which can then be used to synthesize the component's controller. This has lead to several works in distributed synthesis from both the control and reactive synthesis community [60, 44, 10, 35, 98, 31, 21, 76, 127, 151, 107].

While control theory methods [76, 21] do not consider the full class of LTL specifications, reactive synthesis methods [60, 44, 10, 35, 98] do not fully utilize the available common abstract model of the interacting systems (i.e., the plant model), leading to either incomplete or overly restrictive contracts. In fact, several methods ignore the plant model altogether [98, 35] leading to incompleteness even when a feasible solution exists. Furthermore, most works restrict each component to have a single strategy or to have a specific property, leading to overly restrictive contracts.

To address these shortcomings, we introduce novel *negotiation frameworks*—where

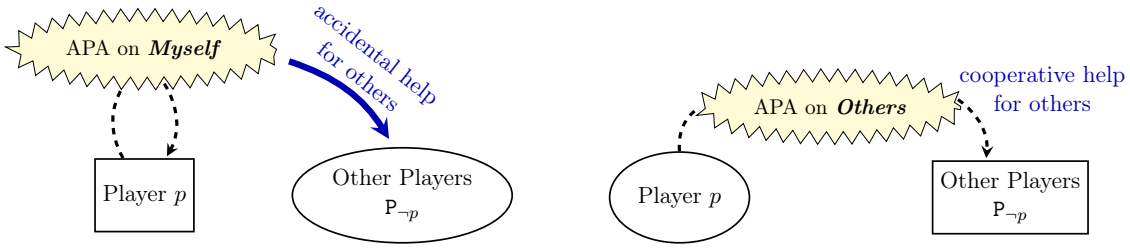


Figure 1.3: Illustration of utilizing APAs to encode help from Player p to other players $\mathtt{P}_{\neg p}$ in multi-player games for rationally interacting players (left) and fully cooperative players (right).

components iteratively negotiate the guarantees they provide to each other based on the assumptions they receive from each other—for distributed synthesis that compute *permissive* assume-guarantee contracts for each component. The key idea is to leverage the full structure of the abstract plant model, represented by a multi-player game graph, where each component is a player with an associated ω-regular specification on the game graph. We consider two interaction settings: (1) *fully cooperative* players, where each player can rely on others to help them satisfy their specifications, and (2) *rationally interacting* players, where each player is selfish and wants to satisfy their own specification while falsifying the specifications of others as much as possible. In both cases, we encode the assume-guarantee contract for each player as a *contracted specification* which are: (i) *sound*: if every player satisfies their contracted specification, then the resulting combined strategy profile is cooperatively/rationally winning, and (ii) *decentralized*: each player can satisfy their contracted specification locally, i.e., irrespective of the strategies of other players. We provide novel techniques to compute such contracted specifications using negotiation frameworks for both fully cooperating and rationally interacting players.

The key insight of our framework is to utilize *adequately permissive assumptions* (APA). Such assumptions allow encoding the cooperative (resp. the selfish) behavior of other components in the cooperative (resp. rational) case (as illustrated in Figure 1.3), which can then be used to iteratively refine the contracted specifications of each component through negotiation. Furthermore, as the APAs are encoded as local edge constraints, this allows to have efficient algorithms for computing the contracts.

In the following, we will give some intuition behind our negotiation frameworks for both settings with an illustrative example as depicted in Figure 1.4.

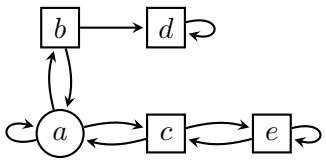


Figure 1.4: A 2-player game graph with Player 0 (circle) and Player 1 (square) vertices.

Contracted Specifications for Rational players. In the rational setting, we consider a multi-player game where each player $p \in \mathtt{P}$ has its own specification Φ_p and wants to satisfy Φ_p as the primary objective while falsifying the specifications of other players Φ_q for $q \neq p$ as much as possible. As mentioned earlier, the key insight is to use APAs to encode selfish behavior of other players. Intuitively, while these rationally interacting players are intrinsically selfish and are primed on achieving their own objective, through this selfishness they *might* – accidentally – help others. Towards computing a contracted specification over rational players, the key idea is to over-approximate the way in which players (accidentally) help each other by letting each Player p compute an APA Ψ_p *on themselves* to satisfy their own objective Φ_p (as illustrated in Figure 1.3 (left)). This can be done by considering a two-player game where Player p is the environment player and others are combined into one controller player, and then computing an APA on the environment player. Intuitively, Ψ_p collects all restrictions on Player p strategies (i.e., moves they *choose themselves*) s.t. the resulting play *can* satisfy Φ_p if others cooperate (somehow). It therefore *over-approximates* the set of all Player p strategies which could possibly form a rationally winning strategy profile with the other players. As a consequence, the intersection of all such assumptions for other players, i.e., $\Psi_{\neg p} = \bigwedge_{q \neq p} \Psi_q$, *under-approximates* the way in which other players (accidentally) help Player p. With this, we can form a *contracted specification* for Player p as $\varphi_p := \Psi_p \wedge (\Psi_{\neg p} \Rightarrow \Phi_p)$. Here, the first part Ψ_p ensures that Player p provides all the help that others might need from them, while the second part $\Psi_{\neg p} \Rightarrow \Phi_p$ ensures that Player p can satisfy their specification whenever others help them as per $\Psi_{\neg p}$.

With this, the main idea of the negotiation framework to compute *rationally contracted specifications* (RCS) is to iteratively refine these assumptions $(\Psi_p)_p$ on every player until the contracted specifications $(\varphi_p)_p$ form an RCS. If the current contracted specifications do not form an RCS, then there exists some player Player p who cannot satisfy their contracted specification φ_p locally and hence, needs more help from others. So, in the next iteration, we strengthen the assumptions by re-computing the APAs for restricted specifications that incorporates the knowledge on how other players are surely cooperating, as $\Psi_{\neg p}$ is an under-approximation of their (accidental) help. As a result, when the algorithm terminates, we show that the resulting contracted specifications form an RCS. Let us illustrate this with an example.

Example 1.1. Consider the game graph in Figure 1.4 with specification $\Phi_0 = \diamondsuit\square\{c, d, e\}$, i.e., the play should eventually only see vertices $\{c, d, e\}$, for Player 0 and $\Phi_1 = \diamondsuit\square\{c, e\}$, i.e., the play should eventually only see vertices $\{c, e\}$, for Player 1. Then first we observe that, starting from vertex a, neither player can satisfy their specification by themselves. Then the APA computed by Player 0 is $\Psi_0 = \Lambda_{\text{COLIVE}}(e_{aa})$, i.e., co-live edge e_{aa} should eventually not be taken. Similarly, the APA computed by Player 1 is $\Psi_1 = \Lambda_{\text{UNSAFE}}(e_{bd}) \wedge \Lambda_{\text{COLIVE}}(e_{ca})$, i.e., unsafe edge e_{bd} should never be taken and co-live edge e_{ca} should eventually not be taken. Intuitively, Ψ_0 ensures that the play does not get stuck in vertex a, and progress is made towards Player 0's goal set. Similarly, Ψ_1 ensures that the play does not reach a region from which satisfying Φ_0 becomes infeasible, and ensures progress towards vertex e happens. We note that Player 0 accidentally ends

up helping Player 1, by trying to leave a. Hence, in the next iteration, Player p actually can realize their contracted specification $\varphi_p = \Psi_p \wedge (\Psi_{\neg p} \Rightarrow \Phi_p)$, which forms an RCS. ⌟

Contracted Specifications for Cooperative players. Similar to the rationally contracted specifications, we can also compute *cooperatively contracted specifications* (CCS) by utilizing APAs. Here, a player can *rely* on other players to help. However, in order to give every player as much freedom as possible in (locally) choosing their strategy, we want to reduce this help to the necessary minimum. In the iterative algorithm for computing cooperatively contracted specifications every player therefore computes how *other* players *must help* her to make her own objective Φ_p realizable. Such an assumption $\Psi_{\neg p}$ on all other players $\mathtt{P}_{\neg p}$ can be formalized as an APA (as illustrated in Figure 1.3 (right)) in a two-player game where Player p is the controller player and others are combined into one environment player.

Similar to the rational setting, the main idea of the algorithm to compute a CCS is to iteratively refine these assumptions $(\Psi_p)_p$ on every player until the contracted specifications $(\varphi_p)_p$ given by $\varphi_p := \Psi_p \wedge (\Psi_{\neg p} \Rightarrow \Phi_p)$ forms a CCS. Notably, this shows that the computation of contracted specifications for cooperating players follows the same algorithmic structure as for rational players. However, the cooperative nature of the players leads to significant differences in both the interpretation and computation of these contracted specifications. In particular, when strengthening assumptions by re-computing APAs for restricted specifications, the cooperative setting allows us to leverage additional information—such as winning regions and co-Büchi regions computed during APA computation (as mentioned in Section 1.2.1)—to directly encode the restricted specifications as parity objectives over the same game graph. This direct encoding enables the algorithm to terminate in polynomial time, in contrast to the rational setting where termination is not guaranteed due to the fundamentally different interpretation of assumptions. Consequently, the negotiation framework for cooperative players is significantly more efficient than its rational counterpart. Let us illustrate the difference between rational and cooperative contracted specifications with an example.

Example 1.2. Consider the game graph in Figure 1.4 with specification $\Phi_0 = \square\diamondsuit\{c\}$, i.e., the play should infinitely often visit vertex c, for Player 0 and $\Phi_1 = \diamondsuit\square\{c, e\}$, i.e., the play should eventually only see vertices $\{c, e\}$, for Player 1. Again, starting from vertex a, neither player can satisfy the specifications on their own. Then, in the rational setting, to compute an RCS, the APA computed by Player 0 is $\Psi_0 = \Lambda_{\text{LIVE}}(\{e_{ac}\})$, i.e., the live edge e_{ac} should be taken infinitely often if source vertex a is visited infinitely often. Similarly, the APA computed by Player 1 is $\Psi_1 = \Lambda_{\text{UNSAFE}}(e_{bd}) \wedge \Lambda_{\text{COLIVE}}(e_{ca})$, i.e., unsafe edge e_{bd} should never be taken and co-live edge e_{ca} should eventually not be taken, as before. In the next iteration, Player 1 can satisfy her specification $\varphi_1 := \Psi_1 \wedge (\Psi_0 \Rightarrow \Phi_1)$ by herself, but Player 0 can not satisfy $\varphi_0 := \Psi_0 \wedge (\Psi_1 \Rightarrow \Phi_0)$. Moreover, the new APAs and specifications will remain the same in the next iterations, and hence, the algorithm will not terminate with an RCS. In fact, for the rational setting, no RCS exists for this game. This is because once the play reaches the vertex c, Player 1 can satisfy her specification and has no incentive to help Player 0.

However, in the cooperative setting, Player 0 computes the assumption (i.e., the help she needs from Player 1) $\Psi_1 = \Lambda_{\text{UNSAFE}}(e_{bd}) \wedge \Lambda_{\text{LIVE}}(\{e_{ec}\})$, i.e., unsafe edge e_{bd} should never be taken and live edge e_{ec} should be taken infinitely often if source vertex e is visited infinitely often. Similarly, Player 1 computes the assumption $\Psi_0 = \Lambda_{\text{COLIVE}}(e_{aa}, e_{ab})$ on Player 0, i.e., co-live edges e_{aa} and e_{ab} should eventually not be taken. As a result, both players can satisfy their respective specifications under these assumptions. Intuitively, Player 0 brings the play to c from a, and Player 1 ensures that the play does not leave $\{c, e\}$ infinitely often. Then due to Ψ_1, Player 1 also brings the play to c infinitely often. Hence, both players can satisfy their respective specifications, and hence, the algorithm terminates with a CCS. ⌟

1.2.3 Permissive Human-Robot Interaction

While Section 1.2.2 introduced the theoretical foundations of assume-guarantee contracts for distributed systems to enable permissive interactions among multiple components, we now apply these contracts to facilitate permissive interactions between humans and robots in human-robot interaction (HRI). In such domains, robots operate alongside humans who pursue their own goals, often without explicitly revealing them. Such scenarios arise increasingly across domains ranging from smart manufacturing and logistics to assistive robotics in healthcare and households. Here, the robot must not only plan its actions to satisfy its own task but also adapt online to human behavior that may be cooperative, indifferent, or even obstructive. At the same time, HRI becomes more effective and enjoyable when robots do not merely *react* to human actions, but respect and even leverage human behavior, while providing *feedback* to guide humans towards cooperation only when necessary. We address this challenge by introducing a novel framework for HRI that utilizes the permissive assume-guarantee contracts from Section 1.2.2 to enable robots to adapt online to human behavior while respecting the contract, and to provide feedback based on the contract's assumptions only when human behavior deviates from them.

We consider the setting where the robot is assigned a high-level temporal task, expressed in linear temporal logic (LTL), and the human simultaneously pursues an unknown strategic latent task. As an example, consider the simplified manipulation task depicted in Figure 1.5, where a Franka robotic arm takes turns with a human to place blocks in a 3×3 domain. The robot's task is to ensure that the majority of the cells is always eventually occupied with no adjacent cells filled, while the hidden human objective is to form a diagonal. If the domain of a logical task is restricted (as in this example), all possible strategic interactions of the human and the robot can be encoded in a two-player game graph. Particular states in this graph fulfil the robot's (resp. human's) objective and are intended to be visited always again by the robot (resp. the human). However, as the robot and the human are both able to move blocks, they can obstruct the satisfaction of each other's goal. Importantly, this might already happen if both have the same goal, e.g., forming a diagonal. Here, the human can persist in forming the diagonal south-west-to-north-east, while the robot persists to form the south-east-to-north-west diagonal, resulting in a livelock. This problem is amplified if robot and human objectives

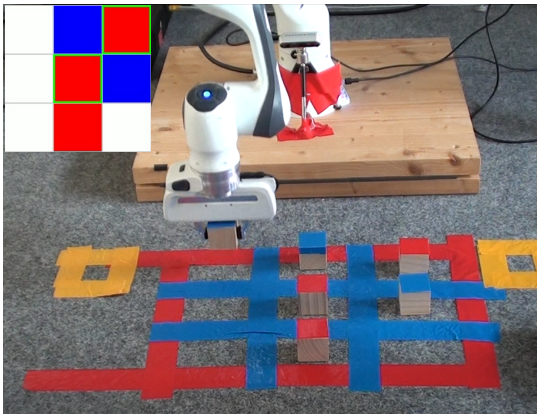


Figure 1.5: A simplified gridworld block-manipulation domain where a Franka Emika Panda robotic arm takes turns with a human to place blocks in a 3×3 grid. The robot places blue blocks, while the human places red blocks. The top-left inset illustrates the robot's feedback to the human, suggesting the removal of the block in cell $(1, 3)$ or $(2, 2)$.

differ, or are even incompatible.

To solve this problem, this work introduces a novel human-robot logical interactions (HRℓI) framework which enables robots to persistently satisfy their logical tasks, while ensuring human autonomy whenever possible and requesting cooperation only when necessary. Formally, the framework builds on our permissive assume-guarantee contracts for graph games from Section 1.2.2, which can compactly represent all cooperative behavior of human and robot that guarantee satisfaction of robot tasks. In particular, when pursuing a high-level LTL task φ, the robot strategy (i) adapts at runtime to the human within the strategy space allowed by the permissive contract (i.e., forming a south-west-to-north-east diagonal as pursued by the human instead of insisting on forming the other one) to maximize cooperation whenever possible, and (ii) provides strategic feedback to the human required by the contract only when human behavior deviates from the contract's assumptions and hence, the robot's adaptation alone cannot ensure progress towards the satisfaction of φ (e.g., if the human persistently removes the middle block obscuring to form a diagonal the robot will ask the human to stop taking this move). This minimizes human-robot interference, maximizes cooperation and preserves human autonomy whenever possible, while still guaranteeing the eventual satisfaction of robot tasks if the human eventually listens to the provided feedback.

We validate our approach both in simulation and on robotic hardware. In addition to the robotic manipulation domain implementation in Figure 1.5, we evaluate our framework in the *Overcooked-AI* simulation environment [53], a widely used benchmark for collaborative planning with multiple agents. In this domain, the human and the robot repeatedly perform cooking tasks with the objective of persistently producing soups. Each participant is assigned an independent LTL task that encodes a recipe specification. As in the gridworld domain, these tasks are private and not known to the other. The human behavior is simulated by a probabilistic strategy.

In this application scenario, the advantage of using LTL specifications over commonly used reachability tasks becomes apparent. As both agents should produce as many specified soups as possible, they can actually cooperate even if their specifications are conflicting—simply by 'taking turns' in producing the 'right' soup. We show that this intuitive cooperative behavior autonomously emerges via the online adaptation and feedback mechanism provided by our framework. To the best of our knowledge, the complexity of the emergent HRℓI behavior provided by our framework thereby vastly exceeds the capabilities of all existing approaches [158, 159, 118, 199, 224] while providing formal guarantees.

1.2.4 Seamless Reactivity of Hybrid Control

In contrast to Sections 1.2.1 to 1.2.3, which addresses high-level synthesis problems (Figure 1.1 (top)) over finite game graphs, this contribution focuses on controller synthesis where the underlying system exhibits non-linear continuous dynamics (Figure 1.1 (bottom)). Specifically, we consider the problem of synthesizing hybrid controllers—that are composed of a high-level logical controller that makes logical decisions and a low-level continuous controller that actuates the underlying continuous dynamics—that ensure the system satisfies a given linear temporal logic (LTL) specification.

One of the key challenges in this setting is the seamless integration of the high-level logical controller with the low-level continuous dynamics. This requires reasoning about the continuous dynamics while ensuring that the logical specification is satisfied. Such problems occur for instance in the control of autonomous robots deployed in warehouses [108], under-water inspection [123, 124], or in rescue and evacuation scenarios [83, 220]. In these applications, the robots need to strategically react to any logical context change, e.g., a newly arriving package that needs to be re-located in the warehouse, a leak in an oil pipeline that needs to be fixed under-water, or a door that got closed and needs to be re-opened to reach a target in a rescue scenario. These context changes are triggered by the external environment and can occur at any time. They must directly result in (high-level) strategic reactions of the robots that trigger new objectives of the low-level controller which, on the other hand, is able to correctly actuate non-trivial non-linear dynamical systems.

While many of the existing approaches to hybrid controller synthesis [30, 191, 120, 116, 221] do not even allow for such logical context changes by the external environment, those that do (see standard literature [210, 184] for an overview and tool papers [188, 51, 117, 125, 140] for tool support) typically rely on discretization-based abstraction techniques that exhaustively discretize the continuous dynamics of the system. This leads to a finite plant model that can be used in high-level synthesis algorithms (e.g., the ones presented in Sections 1.2.1 and 1.2.2), but it introduces significant conservatism and suffers from the curse of dimensionality.

To address this limitation, we introduce a novel synthesis framework for hybrid controllers that can react seamlessly to logical context changes triggered by the external environment, while avoiding exhaustive discretization of the continuous dynamics. The key idea is to construct an abstract plant model from the low-level continuous dynam-

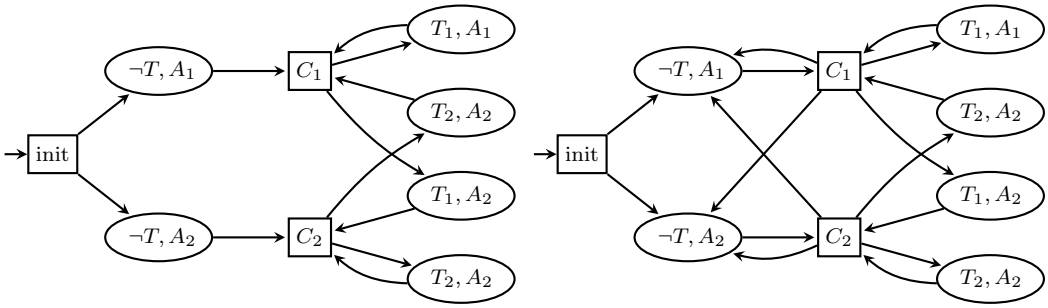


Figure 1.6: Two abstract game graphs, with Player 0 (circle) and Player 1 (square) vertices, to model the robot control problem: without persistent live groups (left) and with persistent live groups $(\mathtt{S}_i, \mathtt{C}_i, \mathtt{T}_i) = (V, E \cap (V \times \{C_i\}), \{(T_i, \cdot)\})$ for each $i \in [1;2]$ (right).

ics that also captures rich liveness properties of the low-level continuous controller, i.e., the ability of the low-level controller to reach certain regions if persistently applied. We achieve this by augmenting the abstract plant model with a new class of assumptions, called *persistent live groups*, which allow the high-level logical controller to reason about the continuous dynamics in a manner that (1) still keeps the resulting logical controller synthesis game computationally tractable, and (2) enables it to react seamlessly to changes in logical context. Note that, in contrast Sections 1.2.1 to 1.2.3, which utilize permissive assumptions to allow for more flexible behaviors of the environment or other logical components, here we use the permissiveness of persistent live groups to capture rich features of the low-level continuous controller.

Persistent Live Groups for Seamless Reactivity. To illustrate the need for persistent live groups in continuous control systems, consider a simple robot control scenario where the robot must react to dynamically changing target locations. Suppose there are two non-overlapping target regions, T_1 and T_2, and assume the existence of low-level controllers C_1 and C_2 that ensure the robot reaches T_i whenever the corresponding controller C_i is applied persistently. The high-level synthesis problem is to design a strategy to trigger the low level controllers C_i in response to the currently activated target (signaled by proposition A_i set to 'true' by the environment), such that the target region T_i is actually reached, if the respective target is persistently activated. While the correct strategy is obvious in this case – choose C_i iff A_i is true – constructing a correct abstract plant model, i.e., a game graph, that returns this strategy and allows to conclude that T_i is actually reached under this strategy, is surprisingly cumbersome.

Two possible abstract game graphs are depicted in Figure 1.6. Starting from the initial vertex (i.e., the one with label "init"), the game alternates between the environment (Player 1) and the controller (Player 0). From each square vertex, Player 1, representing the environment, moves to one of its neighboring vertices (i.e., one of the circle vertices), whose label indicates the currently activated target (i.e., which A_i is 'true') and whether the robot is currently in one of the target regions T_i or in the non-target region $\neg T$. Then,

from each circle vertex, Player 0, representing the controller, chooses which controller C_i to activate by moving to one of the neighboring square vertices. In favor of readability, we only depict the Player 0 edges corresponding to the strategy of choosing C_i iff A_i is activated.

In Figure 1.6 (left) the controller only allows the environment to activate a new target after it reached a target location, while Figure 1.6 (right) models the more realistic scenario that the environment might activate a different target at any point, which then allows Player 0 to activate the other controller, even if the robot is still in $\neg T$. However, with the additional edges in Figure 1.6 (right), Player 1 can force the play to loop between state $(\neg T, A_i)$ and C_i, thereby preventing the play from ever reaching T_i even if A_i is persistently active. In the physical system, however, we know that the robot will reach T_i if C_i is persistently used. Intuitively, this assumption can be expressed by a *persistent live group* which models that in the source region $\mathtt{S}_i = V$ (complete vertex set in this case), if Player 0 persistently chooses edges from $\mathtt{C}_i = E \cap (V \times \{C_i\})$ (representing the choice of controller C_i), then the play will eventually reach the target region $\mathtt{T}_i = \{(T_i, \cdot)\}$.

In terms of the abstraction, these persistent live groups capture the fact that progress will eventually be made without requiring to *spell this out* via the state space of the abstraction, avoiding any explicit time or space discretization. Augmenting such assumptions allows to directly synthesize the correct strategy from the high-level game. Furthermore, we show that solving such augmented games can be done in similar complexity as solving the original game.

Two-layered Approach for Controller Synthesis. Utilizing such persistent live groups, we present a novel two-layered synthesis framework for hybrid controllers that can react seamlessly to logical context changes triggered by the external environment, while ensuring satisfaction of a given LTL specification. In particular, starting from a non-linear dynamical system and an LTL specification φ, we use the following key steps. First, we solve a high-level logical game induced by the specification. Then, we build a top-down interface which allows us to translate strategic choices from the logical level into certified low-level controllers that actuate these choices in the continuous dynamics. Afterwards, we build a bottom-up interface to extract relevant information about the dynamics of these low-level controllers into a plant model augmented with persistent live groups. We then incorporate the augmented plant model into the logical synthesis game, thereby resulting in a game augmented with persistent live groups. Finally, we solve the augmented game to obtain a winning strategy for the controller player, which defines a hybrid controller that reacts to logical context changes by switching between different low-level controllers.

Solving Games under Persistent Live Groups. While the above framework outlines the overall synthesis approach, a key technical challenge is to solve the resulting games augmented with persistent live groups. As standard algorithms for solving parity games are based on attractor computations, we first show that one can compute attractors in games under persistent live groups assumptions in polynomial time. The

idea is to iteratively increase the set of attractor states by utilizing one of the persistent live groups. Based on this result, one can use the standard attractor-based algorithms (e.g., Zielonka's algorithm [225]) to solve such augmented games. Furthermore, we also show that one can also obtain a quasi-polynomial time algorithm for solving such games by extending the quasi-polynomial time algorithm given by Lehtinen et al. [137] and Parys [171] for solving parity games. This shows that parity games under persistent live groups assumptions can be solved in quasi-polynomial time, which is the best known complexity for solving parity games in general.

1.2.5 Universal Safety Controller with Learned Prophecies

While Section 1.2.4 presented a synthesis framework that incorporates assumptions on the plant model to ensure seamless reactivity in hybrid systems, we now turn to another synthesis framework that considers the problem at discrete level where the abstract plant model is assumed to be part of the input. As mentioned earlier, the standard approaches to logical controller synthesis in this setting typically rely on constructing a game graph from the specification and the plant model. These approaches are, however, known to face major scalability challenges: exploring all plant behaviors in a model can lead to an intractable explosion in the state space during synthesis. More critically, if the plant is only available at runtime or changes over time, traditional state exploration methods quickly become infeasible.

These severe limitations have motivated the introduction of *Universal Safety Controllers (USCs)* [96], which shift the focus from plant-specific synthesis to the computation of a *universal controller* derived from the specification alone, whose decisions are conditioned by so-called *prophecies*, i.e., assumptions on the future behaviors of the plant model. As a result, USCs promise two major advantages: (1) generalization: USCs provide a correct solution for all realizable plant models, and (2) computational efficiency: by focusing on the specification, USC synthesis promises to reduce the computational burden of utilizing a complete plant model. The latter computational benefit, however, is only achieved if prophecies are small and concise. As prophecies describe branching-time properties of the plant (e.g., the existence of certain paths contingent on the controller's actions that satisfy essential properties), Finkbeiner et al. [96] use tree automata to encode prophecies to capture all possible future branching structure of the plant potentially relevant for the specification. While preserving correctness and completeness, this technique renders both prophecy synthesis and prophecy verification computationally very intense, as tree automata are notoriously difficult to handle in practice.

Our contribution addresses this limitation of the automata-based approach by introducing the first *learning-based* algorithm for prophecy construction. Concretely, we obtain prophecies in computation tree logic (CTL) [69], which we learn from a small set of representative (nominal) plant models drawn from the application domain. As CTL is a well-established branching-time temporal logic, it allows us to express rich properties of the plant model while being amenable to efficient learning algorithms [70]. Hence, the resulting USC offers improved efficiency and explainability through small and concise CTL prophecies, which remain generalizable and human-readable. Note that in contrast

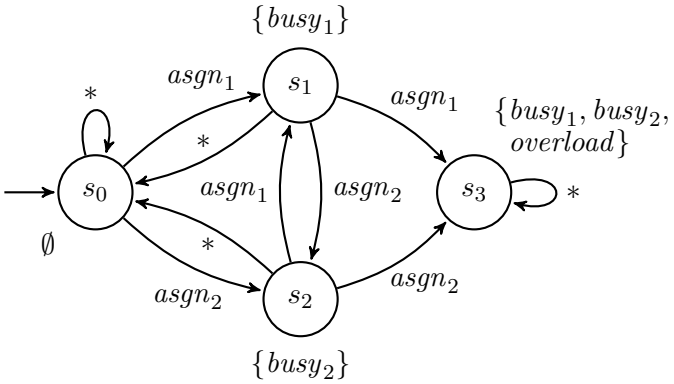


Figure 1.7: A plant that models the occupancy of processors in the load balancer example. We use $*$ to represent edges that are taken whenever no other edge guard is satisfied.

Sections 1.2.1 to 1.2.4, where the assumptions used are linear-time properties expressed in LTL, here, we leverage the branching-time nature of CTL to express prophecies that capture the necessary branching structure of the plant.

A Load Balancer Example. As a motivating example, we consider the problem of synthesizing a simple load-balancing scheduling controller, which assigns incoming tasks to two processing units (CPUs). The load-balancing controller is specified with the following temporal formula:

$$\varphi \coloneqq \square(task \rightarrow \bigcirc(asgn_1 \vee asgn_2)) \wedge \square \neg overload, \tag{1.1}$$

where *task* models the arrival of a new task and is controlled by the *environment*, $asgn_i$ assigns an incoming task to either cpu_1 or cpu_2 and is controlled by the controller, and *overload* means that a task was assigned to a busy processor, which is controlled by the *plant*. The specification states that there should never be an *overload*, i.e., the controller should never assign a task to a processor that is already busy with another task.

Ultimately, we are interested in applying the scheduler to a concrete plant model, such as the one described in Figure 1.7. In this particular plant model, each of the processors is busy for exactly one time-step once assigned a task. An example of a controller that satisfies our specification for this plant, and would typically be found by a standard synthesis algorithm, is the round-robin controller, which alternates between assigning new tasks to the first and second CPU. The disadvantage of this controller, however, is that it *only* works correctly for our *specific* plant. By contrast, a universal controller is immediately applicable to an entire set of plants.

Previous work: automata-based USC synthesis. The previous approach to universal controller synthesis, UNICON [96], takes as input only a specification: the USC is computed independently of any specific plant. For each possible output of the controller, the synthesis algorithm computes the exact condition on the plant under which

this output is correct. In our example, both $asgn_1$ and $asgn_2$ might be correct outputs, depending on whether the respective CPU is available, and also depending on the *future* availability of the CPU. For example, if, in a hypothetical plant, cpu_1 would become permanently busy if used in the first step, it would only be correct to start with $asgn_2$. UNICON computes a tree automaton that recognizes *exactly those* plants where $asgn_1$ (or $asgn_2$, respectively) does not lead to an immediate *overload* and where, additionally, a control strategy exists for the state reached by executing $asgn_1$ (or $asgn_2$, respectively). The tree automaton is non-trivial and difficult to interpret; we omit depicting it here.

New approach: learning-based USC synthesis. Our new learning-based approach, UCLEARN, takes as input both the specification and a set of *nominal* plants which are representative of the plants the controller will be applied to. Similar to UNICON, the prophecies express conditions on the plants for which a certain output should be applied. However, instead of *characterizing* precisely the set of plants for which a particular output is correct, our learned prophecies *separate* plants for which a control output is correct from those where the output is incorrect. This relaxation introduces a freedom of choice in the selection of the separating condition. The precise condition computed by UNICON is also one of the separating conditions, but typically, there are separating conditions that are much simpler, easier to verify, and easier to understand. We find such a condition, expressed as a temporal formula, using a learning algorithm for CTL.

Figure 1.8 shows the controller synthesized by UCLEARN for the specification of the load-balancing controller together with the plant from Figure 1.7 as the (single) nominal plant. The controller has three states. The initial state q_0 represents the case that no task has been received yet. Here, no_asgn is a correct decision independently of the plant. The outputs $asgn_1$ and $asgn_2$ are allowed as long as the respective CPU is not busy. State q_1 represents the case that a task has been received and still needs to be assigned to a CPU. The difference to q_0 is that no_asgn is no longer an option. Finally, q_2 represents the case that an overload has occurred. This will never happen with the plant from Figure 1.7, although it might happen on plants which the synthesis algorithm has not seen. In q_2, no controller output is correct, because the specification has already been violated.

The synthesized controller is not only correct for the given plant, it also generalizes to other plants that have sufficient CPU availability and use *busy* like in our example. UCLEARN thus produces controllers that are more general than the plant-specific controllers computed by classic algorithms from reactive synthesis and supervisory control, and simpler than the universal controllers produced by UNICON.

1.2.6 Robust Computation Tree Logic

In contrast to Sections 1.2.4 and 1.2.5, where we utilized assumptions on the plant model to address challenges in controller synthesis, now, we turn to the problem of ensuring the *robustness* of logical controllers with respect to violations of such assumptions. In practice, it is often the case that the plant model is not entirely known at design time,

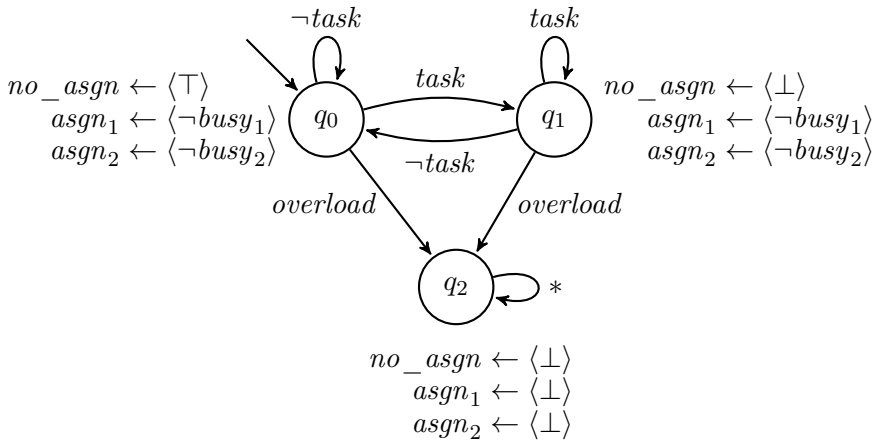


Figure 1.8: An approximate USC with CTL prophecies. Whenever the plant satisfies the prophecy, the controller outputs the respective variable.

and thus, assumptions made about it may be violated at runtime. It is therefore crucial to ensure that the system behaves reasonably even when the assumptions are not fully satisfied—enabling the system to operate reliably under more permissive and uncertain physical behaviors.

A common method for handling robustness against violation of such assumptions is to encode them as environment assumptions within the system's specifications, typically using the implication $\Phi_e \Rightarrow \Phi_s$, where Φ_e represents the environment assumption and Φ_s represents the system guarantee. This formulation is similar to the one used for contracted specifications in distributed systems in Section 1.2.2, where we employed implications to encode the specification of a component under assumptions on other components. While such implications pose no issue in Section 1.2.2—since the assumptions are, by design, implementable by the respective components—this approach has a well-known drawback in the general case: the implication is trivially satisfied when the environment assumption Φ_e is violated, regardless of the system's behavior. Consequently, the system may behave arbitrarily when the environment assumption is violated, which is clearly undesirable in practice.

To address this, many approaches in the literature have focused on making the specifications robust to violations of the environment assumption. For instance, Bloem et al. [36], Tarraf et al. [214], Doyen et al. [82], Ehlers et al. [85], and Tabuada et al. [212, 211] have provided different ways of introducing robustness for specifications in Linear Temporal Logic (LTL). However, all of these approaches require additional assumptions or quantitative information from the designer, which is often tedious and hard to obtain.

This drawback has motivated Tabuada and Neider [213] to introduce a new logic, named robust LTL (rLTL), which provides robustness without relying on any additional assumptions or input from a designer beyond an LTL formula. rLTL achieves this by

employing *multi-valued semantics* that quantify how well a system satisfies a specification under different degrees of violations of the environment assumption. Among rLTL's main features are its ease of use (one simply "dots" temporal operators in existing LTL formulas) and the fact that adding robustness does not change the asymptotic complexity of the model checking, runtime monitoring, and synthesis problems [213, 168, 169, 19, 18, 20, 153, 153, 165]. Inspired by this logic, there have been several works introducing robust extensions of different classes of temporal logics [168, 169, 157, 226].

However, these approaches primarily address linear-time properties, which limits their applicability to branching-time properties that can express more complex plant behavior (including the branching-time properties required for prophecies in Section 1.2.5). For instance, a robot operating in a building might rely on the assumption that it can always move from room A to room B in one time step. This assumption cannot be expressed in LTL but can be naturally stated in branching time logic such as CTL or CTL* as: $\forall\Box(A \Rightarrow \exists\bigcirc B)$, meaning that from any reachable state, if the robot is in room A, there exists a path where the robot can move to room B in the next step.

To address this gap, we develop robust extensions of CTL and CTL*, which we call robust CTL (rCTL) and robust CTL* (rCTL*), which are inspired by the notion of robustness in rLTL. Similar to rLTL, our new logics employ multi-valued semantics to track the degree of violations of a specification. Furthermore, our semantics is guided by two objectives: first, the syntax of rCTL and rCTL* is similar to CTL and CTL*, respectively; second, the notion of robustness in these logics is intrinsic rather than extrinsic, i.e., robustness does not rely on the designers to provide quantitative information about the specification, such as the number of violations permitted, ranks, cost, etc.

Demonstration of Robustness in Branching-time Logics. To illustrate the intuition behind our notion of robustness, consider a specification $\Phi_e \Rightarrow \Phi_s$ for a robot deployed in an office-like environment. As before, the environment assumption on the plant model is $\Phi_e = \forall\Box(A \Rightarrow \exists\bigcirc B)$, which states that from every reachable state, if the robot is in room A, then there exists a path where it can move to room B in one time step. The system guarantee is $\Phi_s = \forall\Box T$, which states that the robot should be able to perform task T in all reachable states at all times. In an ideal setting, we would expect the following robustness behavior:

- If the environment (i.e., the plant model) satisfies the assumption Φ_e, then the robot should satisfy its guarantee Φ_s.
- If the assumption Φ_e is temporarily violated, e.g., due to office workers obstructing the robot's path from room A to B for a finite amount of time, such that the environment only satisfies $\forall\Diamond\Box(A \Rightarrow \exists\bigcirc B)$, then the robot should eventually be able to perform its task reliably again. That is, it should satisfy $\forall\Diamond\Box T$.
- If the environment violates the assumption infinitely often or eventually always, i.e., satisfies $\forall\Box\Diamond(A \Rightarrow \exists\bigcirc B)$ or $\forall\Diamond(A \Rightarrow \exists\bigcirc B)$, then the robot should satisfy a degraded form of the guarantee $\forall\Box\Diamond T$ or $\forall\Diamond T$, respectively.

Table 1.1: Summary of our results (in gray) and comparison to other logics. All problems are complete for the respective complexity class.

	Model Checking	Satisfiability	Synthesis
CTL	PTIME	EXPTIME	EXPTIME
rCTL	PTIME	EXPTIME	EXPTIME
LTL	PSPACE	PSPACE	2EXPTIME
rLTL	PSPACE	PSPACE	2EXPTIME
CTL*	PSPACE	2EXPTIME	2EXPTIME
rCTL*	PSPACE	2EXPTIME	2EXPTIME

We show that the semantics of rCTL and rCTL* indeed capture such a notion of robustness.

Expressiveness and Complexity Results. After having introduced rCTL and rCTL*, we analyze their expressive power and compare them to existing logics such as LTL, rLTL, CTL, and CTL*. Our key results are that rCTL is more expressive than CTL, while rCTL* has the same expressive power as CTL*.

We also study the complexity of the verification, satisfiability, and synthesis problems for rCTL and rCTL*. For verification, we consider the model checking problem, which asks whether a given system satisfies a specification expressed in rCTL or rCTL*. We provide efficient model-checking algorithms for rCTL and rCTL* to demonstrate that both logics can be effectively used for verification. We establish that the rCTL model checking problem is PTIME-complete and that the rCTL* model checking problem is PSPACE-complete, which is the same asymptotic complexity as CTL and CTL* model checking, respectively.

Moreover, we show that the satisfiability and reactive synthesis problems for rCTL and rCTL* specifications match the exact asymptotic complexity of their non-robust counterparts, i.e., EXPTIME-completeness for rCTL and 2EXPTIME-completeness for rCTL*. Thus, robustness can be added to branching-time logics "for free". Table 1.1 shows an overview over our complexity results.

1.3 List of Publications

The contents of this thesis are based on the following publications co-authored with several collaborators.

[11] Ashwani Anand, Kaushik Mallik, Satya Prakash Nayak, and Anne-Kathrin Schmuck. **Computing Adequately Permissive Assumptions for Synthesis**. *29th International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (TACAS 2023)

[166] Satya Prakash Nayak and Anne-Kathrin Schmuck. **Most General Winning Secure Equilibria Synthesis in Graph Games**. *30th International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (TACAS 2024)

[16] Ashwani Anand, Anne-Kathrin Schmuck, and Satya Prakash Nayak. **Contract-Based Distributed Logical Controller Synthesis**. *27th ACM International Conference on Hybrid Systems: Computation and Control* (HSCC 2024)

[15] Ashwani Anand, Satya Prakash Nayak, and Anne-Kathrin Schmuck. **Strategy Templates - Robust Certified Interfaces for Interacting Systems**. *22nd International Symposium on Automated Technology for Verification and Analysis* (ATVA 2024)

[109] Oz Gitelson, Satya Prakash Nayak, Ritam Raha, and Anne-Kathrin Schmuck. **Maximal Adaptation, Minimal Guidance: Permissive Reactive Robot Task Planning with Humans in the Loop**. *under review* (2026)

[162] Satya Prakash Nayak, Lucas N Egidio, Matteo Della Rossa, Anne-Kathrin Schmuck, and Raphael M Jungers. **Context-triggered abstraction-based control design**. *IEEE Open Journal of Control Systems* (OJ-CSYS 2023)

[197] Anne-Kathrin Schmuck, K. S. Thejaswini, Irmak Saglam, and Satya Prakash Nayak. **Solving Two-Player Games Under Progress Assumptions**. *25th International Conference on Verification, Model Checking, and Abstract Interpretation* (VMCAI 2024)

[97] Bernd Finkbeiner, Niklas Metzger, Satya Prakash Nayak, and Anne-Kathrin Schmuck. **Universal Safety Controllers with Learned Prophecies**. *40th AAAI Conference on Artificial Intelligence* (AAAI 2026)

[163] Satya Prakash Nayak, Daniel Neider, Rajarshi Roy, and Martin Zimmermann. **Robust Computation Tree Logic**. *14th International Symposium on NASA Formal Methods* (NFM 2022)

[164] Satya Prakash Nayak, Daniel Neider, Rajarshi Roy, and Martin Zimmermann. **Robust Computation Tree Logic**. *Innovations in Systems and Software Engineering* (ISSE 2025)

Note that most of these publications (i.e., all except [162]) list authors in alphabetic or randomized order (generated using an author randomization tool by `https://www.aeaweb.org/` with publicly verifiable records).

1.4 Organization of the Contents

The remainder of this thesis is organized as follows, and their inter-dependencies are illustrated in Figure 1.9. Chapter 2 introduces the basic concepts and notations used

throughout the thesis. All subsequent chapters build upon these preliminaries to present our contributions (outlined in Section 1.2) in detail. Chapters 3 to 6 focus on permissive assumptions in distributed logical systems, forming Part A of the thesis, while Chapters 7 to 9 address permissive assumptions on the plant model, forming Part B.

More specifically, Chapter 3 discusses permissive assumptions in two-player games and presents an algorithm for computing them, which is based on work published in the proceedings of 29th International Conference on Tools and Algorithms for the Construction and Analysis of Systems [11]. Building on these results, Chapters 4 and 5 propose negotiation-based distributed synthesis algorithms for computing permissive assume-guarantee contracts in multi-player games. While Chapter 4 considers rational players, based on work published in the 30th International Conference on Tools and Algorithms for the Construction and Analysis of Systems [166], Chapter 5 focuses on cooperative players, based on work published in the proceedings of 27th ACM International Conference on Hybrid Systems: Computation and Control [16] and in the proceedings of 22nd International Symposium on Automated Technology for Verification and Analysis [15]. Chapter 6 utilizes these contracts to design a novel framework for permissive human-robot interaction, based on work which is currently under review. Part B begins with Chapter 7, which introduces a new synthesis framework that enables seamless reactivity in hybrid systems by augmenting persistent live groups as assumptions on the plant model, based on work published in IEEE Open Journal of Control Systems [162] and in the proceedings of 25th International Conference on Verification, Model Checking, and Abstract Interpretation [197]. Chapter 8 presents a universal controller synthesis framework that leverages learned prophecies to generalize controller behavior across different plant models, based on work to be published in the proceedings of 40th AAAI Conference on Artificial Intelligence [97]. Chapter 9 introduces robust semantics for branching-time temporal logics and analyzes their expressiveness and complexity, based on work published in the proceedings of 14th International Symposium on NASA Formal Methods [163] and in Innovations in Systems and Software Engineering [164]. Finally, Chapter 10 concludes the thesis and outlines potential directions for future research.

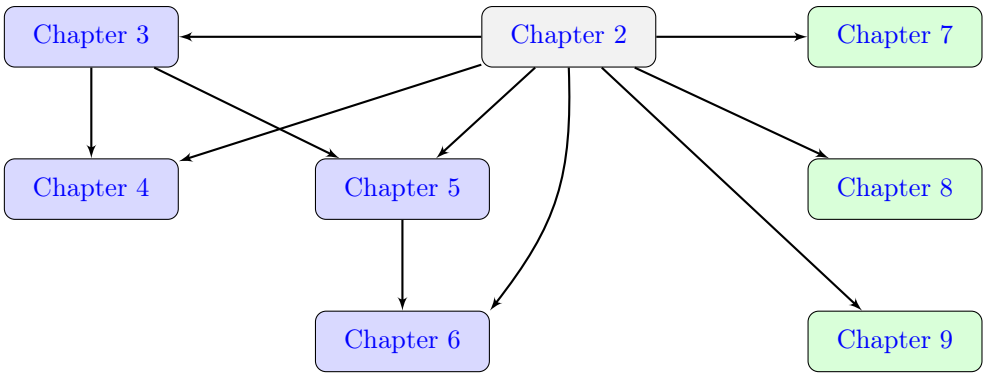


Figure 1.9: Dependencies between the chapters in this thesis. Chapters in blue (left) form Part A on permissive assumptions in distributed logical systems, and chapters in green (right) form Part B on permissive assumptions on the plant model.

Chapter 2

Preliminaries

In this chapter, we introduce some notation and concepts that will be used throughout this thesis. To specify system properties, we first define linear-time formalism such as linear temporal logic (LTL) and ω-regular languages in Section 2.2, followed by branching-time formalism such as computation tree logic (CTL) and CTL* in Section 2.3. We then present the notation and preliminary concepts on logical controller synthesis in Section 2.4. Next, to solve the logical controller synthesis problem, we introduce two-player games in Section 2.5, which are widely used to model the interaction between a controller and its environment. Finally, we extend these concepts to multi-player games in Section 2.5.4 to model the interaction among multiple components in a distributed system.

2.1 Notation

Numbers and Intervals. We use $\mathbb{R}$ to denote the set of real numbers, $\mathbb{R}_+$ to denote the set of non-negative real numbers, and $\mathbb{R}^n$ to denote the n-dimensional Euclidean space. We use $\mathbb{N}$ to denote the set of natural numbers including zero. Given two natural numbers $a, b \in \mathbb{N}$ with $a < b$, we use $[a; b]$ to denote the set $\{n \in \mathbb{N} \mid a \leq n \leq b\}$. For any given set $[a; b]$, we write $i \in_{\text{even}} [a; b]$ and $i \in_{\text{odd}} [a; b]$ as shorthand for $i \in [a; b] \cap \{0, 2, 4, \ldots\}$ and $i \in [a; b] \cap \{1, 3, 5, \ldots\}$, respectively.

Sets. Given a set S, we use 2^S to denote the power set of S, i.e., the set of all subsets of S, and $|S|$ to denote the cardinality of S. For two sets A and B, the Cartesian product $A \times B$ is defined as the set $\{(a, b) \mid a \in A, b \in B\}$.

Relations and Functions. For two sets A and B, a relation $R \subseteq A \times B$, and an element $a \in A$, we write $R(a)$ to denote the set $\{b \in B \mid (a, b) \in R\}$ and $src(R)$ to denote the set $\{a \in A \mid \exists b \in B : (a, b) \in R\}$. A function $f : A \to B$ denotes a mapping from every element $a \in A$ to a unique element $b \in B$, which we write as $f(a) = b$. Given a function $f : A \to B$ and a subset $A' \subseteq A$, we write $f(A')$ to denote the set

$\{b \in B \mid \exists a \in A' : f(a) = b\}$, and we write $f|_{A'}$ to denote the restriction of f to A', i.e., $f|_{A'} : A' \to B$ such that for every $a \in A'$, $f|_{A'}(a) = f(a)$.

Languages. Let Σ be a finite alphabet. A *word* over Σ is a finite or infinite sequence $w = a_0 a_1 \ldots$ of symbols from Σ. The notations Σ^* and Σ^ω denote the set of finite and infinite words over Σ, respectively, and Σ^∞ is equal to $\Sigma^* \cup \Sigma^\omega$. We write ϵ to denote the empty word. Given two words $u \in \Sigma^*$ and $v \in \Sigma^\infty$, the concatenation of u and v is written as the word uv. For a word $w \in \Sigma^\infty$, we use the following notation:

- $|w|$ to denote its length, where $|w| = \infty$ if w is an infinite word;
- $dom(w)$ to denote its domain, i.e., $dom(w) = [0; |w| - 1]$ if w is a finite word and $dom(w) = \mathbb{N}$ if w is an infinite word;
- $w[i]$ to denote the i-th symbol of w for $i \in dom(w)$;
- $w[i..]$ to denote the suffix of w from index i on for $i \in dom(w)$;
- $w|_X$ to denote its projection onto a subset $X \subseteq \Sigma$, obtained by removing all symbols in w that are not in X.

2.2 Linear-Time Properties

To specify the desired behavior of systems over time, we use various linear-time properties. These properties are often expressed using linear temporal logic (LTL) formulas or ω-automata that accept infinite traces satisfying the properties. In this section, we introduce the basic concepts of traces, LTL, ω-automata, and their equivalences.

2.2.1 Atomic Propositions and Traces

Atomic Propositions. An *atomic proposition* is a basic statement or variable that can be either true or false. Throughout this thesis, we define and use different sets of atomic propositions to specify various system behaviors and properties.

Traces. Let $\mathtt{AP}$ be a finite set of atomic propositions and $\Sigma = 2^{\mathtt{AP}}$ be the alphabet consisting of all subsets of $\mathtt{AP}$. A *trace* over $\mathtt{AP}$ is an infinite word $\gamma \in \Sigma^\omega$, where each $\gamma[i]$ for $i \geq 0$ is a subset of $\mathtt{AP}$ representing the set of atomic propositions that are true at the i-th position of the trace. For a subset of atomic propositions $\mathtt{AP}' \subseteq \mathtt{AP}$, we write $\gamma|_{\mathtt{AP}'}$ to denote the trace γ' obtained by restricting each position of γ to the propositions in $\mathtt{AP}'$, i.e., $\gamma'[i] = \gamma[i] \cap \mathtt{AP}'$ for all $i \geq 0$.

With a slight abuse of notation, we also refer to a trace γ with singleton sets at each position, i.e., $\gamma[i] = \{a_i\}$ for each $i \geq 0$, as the infinite sequence $a_0 a_1 \ldots \in \mathtt{AP}^\omega$.

Linear-Time Properties over Traces. A linear-time property usually specifies the desired behavior of a system over time in terms of the traces it can produce. More precisely, a linear-time property over $\mathtt{AP}$ defines a set of traces over $\mathtt{AP}$ that satisfies the property.

2.2.2 Linear Temporal Logic

Linear Temporal Logic (LTL) [177] is a widely used formalism for specifying linear-time properties of systems. LTL extends propositional logic with temporal operators that allow reasoning about the ordering of events over time in infinite traces.

Syntax. LTL formulas over a set of atomic propositions $\mathtt{AP}$ are given by the grammar

$$\Phi ::= \mathsf{True} \mid a \mid \Phi \vee \Psi \mid \neg\Phi \mid \bigcirc \Phi \mid \Phi \,\mathbf{U}\, \Psi,$$

where $a \in \mathtt{AP}$ is an atomic proposition and Ψ is an LTL formula. Here, 'True' denotes the formula that is always true, '$\vee$' denotes disjunction, '$\neg$' denotes negation, '$\bigcirc$' denotes the next operator, and '$\mathbf{U}$' denotes the until operator. In addition, the truth value 'False' and standard boolean operators such as conjunction '$\wedge$' and implication '$\Rightarrow$' can be derived as follows: $\mathsf{False} \coloneqq \neg\mathsf{True}$, $\Phi \wedge \Psi \coloneqq \neg(\neg\Phi \vee \neg\Psi)$, and $\Phi \Rightarrow \Psi \coloneqq \neg\Phi \vee \Psi$. Moreover, the well-known temporal operators eventually '$\Diamond$', always '$\Box$' and weak until '$\mathbf{W}$' can be derived as follows: $\Diamond \Phi \coloneqq \mathsf{True}\,\mathbf{U}\,\Phi$, $\Box\Phi \coloneqq \neg\Diamond\neg\Phi$, and $\Phi\,\mathbf{W}\,\Psi \coloneqq (\Phi\,\mathbf{U}\,\Psi) \vee \Box\Phi$.

We also use a set of LTL formulas $\{\Phi_1, \Phi_2, \ldots, \Phi_k\}$ as an LTL formula which represents the disjunction of all formulas in it.

Semantics. The semantics of LTL formulas are defined over traces. A trace $\gamma \in (2^{\mathtt{AP}})^\omega$ is defined to *satisfy* an LTL formula Φ over $\mathtt{AP}$, written as $\gamma \models \Phi$, recursively as follows:

- $\gamma \models a$ if $a \in \gamma[0]$;
- $\gamma \models \Phi \vee \Psi$ if $\gamma \models \Phi$ or $\gamma \models \Psi$;
- $\gamma \models \neg\Phi$ if $\gamma \not\models \Phi$;
- $\gamma \models \bigcirc\Phi$ if $\gamma[1..] \models \Phi$;
- $\gamma \models \varphi\,\mathbf{U}\,\psi$ if there exists $i \geq 0$ such that $\gamma[i..] \models \psi$ and for all $0 \leq j < i$, we have $\gamma[j..] \models \varphi$,

In addition, we have the following derived semantics:

- $\gamma \models \Phi \wedge \Psi$ if $\gamma \models \Phi$ and $\gamma \models \Psi$;
- $\gamma \models \Phi \Rightarrow \Psi$ if $\gamma \not\models \Phi$ or $\gamma \models \Psi$;
- $\gamma \models \Diamond\Phi$ if there exists $i \geq 0$ such that $\gamma[i..] \models \Phi$;
- $\gamma \models \Box\Phi$ if for all $i \geq 0$, $\gamma[i..] \models \Phi$;

- $\gamma \models \varphi \,\mathbf{W}\, \psi$ if either $\gamma \models \varphi \,\mathbf{U}\, \psi$ or for all $i \geq 0$, we have $\gamma[i..] \models \varphi$.

An LTL formula Φ over $\mathtt{AP}$ thus defines the set of traces over $\mathtt{AP}$ that satisfy Φ, called the *language* of Φ and denoted by $\mathcal{L}(\Phi) = \{\gamma \in (2^{\mathtt{AP}})^\omega \mid \gamma \models \Phi\}$.

2.2.3 ω-regular Properties

We also use ω-regular properties to specify linear-time properties of systems. Traditionally, ω-regular properties are represented using ω-automata that accept infinite traces satisfying the properties.

Formally, an *ω-automaton* over a finite alphabet $\Sigma = 2^{\mathtt{AP}}$ is a tuple $\mathcal{A} = (Q, q_0, \delta, \Omega)$ which consists of

- a finite set of states Q,
- an initial state $q_0 \in Q$,
- a transition function $\delta \colon Q \times \Sigma \to Q$, and
- an acceptance condition $\Omega \subseteq Q^\omega$.

The unique *run* of $\mathcal{A}$ from state q on some trace $\gamma \in \Sigma^\infty$, denoted by $\mathit{run}(\mathcal{A}, q, \gamma)$, is a sequence of states $\rho \in Q^\infty$ with $|\rho| = |\gamma| + 1$, $\rho[0] = q$, and $\delta(\rho[i], \gamma[i]) = \rho[i+1]$ for all $i \in \mathit{dom}(\gamma)$. A run ρ is *accepting* if $\rho \in \Omega$. The language $\mathcal{L}(\mathcal{A})$ is the set of all traces γ for which the unique run $\mathit{run}(\mathcal{A}, q_0, \gamma)$ is accepting. Furthermore, we write $\mathcal{A}(q) = (Q, q, \delta, \Omega)$ to denote the automaton with the same transition function and acceptance condition but with initial state q, and $\mathtt{reachable}(\mathcal{A})$ to denote the set of states that are reachable from the initial state q_0 in $\mathcal{A}$, i.e., there exists a run ρ of $\mathcal{A}$ with $\rho[0] = q_0$ and $\rho[k] = q$ for some $k \geq 0$.

There are various types of ω-automata depending on the acceptance condition Ω used. In this thesis, we consider these acceptance conditions to be specified using LTL formulas over Q, i.e., $\Omega = \mathcal{L}(\Phi)$ for some LTL formula Φ over the set of atomic propositions Q. We mainly focus on the ω-regular properties defined by the following LTL specifications:

- *safety specification* $\Phi = \Box F$ is given by a set $F \subseteq Q$ of safe states, requiring accepting runs to only visit states in F.
- *reachability specification* $\Phi = \Diamond F$ is given by a set $F \subseteq Q$ of target states, requiring accepting runs to visit states in F at least once.
- *Büchi specification* $\Phi = \Box\Diamond F$ is given by a set $F \subseteq Q$ of target states, requiring accepting runs to visit states in F infinitely many times.
- *co-Büchi specification* $\Phi = \Diamond\Box F$ is given by a set $F \subseteq Q$ of target states, requiring accepting runs to events only visit states in F.
- *generalized Büchi specification* $\Phi = \bigwedge_{i=1}^{n} \Box\Diamond F_n$ is given by a set $\{F_1, F_2, \ldots, F_n\}$ of sets $F_i \subseteq Q$ for all $1 \leq i \leq n$, requiring accepting runs to visit states in each F_i infinitely many times.

- *parity specification* $\Phi = \mathit{Parity}(\mathbb{P})$ is given by a priority function $\mathbb{P}\colon Q \to [0;d]$ such that $\mathbb{P}$ defines a partition $\{\mathbb{P}[i] \mid i \in [0;d]\}$ of the states Q with $\mathbb{P}[i] = \{q \in Q \mid \mathbb{P}(q) = i\}$ being the states with priority i. In this case, $\mathit{Parity}(\mathbb{P})$ is defined as follows

$$\mathit{Parity}(\mathbb{P}) \coloneqq \bigwedge_{i \in_{\text{odd}}[0;d]} \left(\Box\Diamond \mathbb{P}[i] \implies \bigvee_{j \in_{\text{even}}[i+1;k]} \Box\Diamond \mathbb{P}[j] \right),$$

  which requires that the highest priority appearing infinitely often along the accepting runs to be even.

We refer to ω-automata with safety, reachability, Büchi, co-Büchi, generalized Büchi, and parity acceptance conditions as *safety automata*, *Büchi automata*, *co-Büchi automata*, *generalized Büchi automata*, and *parity automata*, respectively.

2.2.4 LTL to ω-automata

It is well-known that an LTL formula can be translated into an equivalent ω-automaton with respect to the language they define.

Proposition 2.1. *For every LTL formula Φ over a set of atomic propositions $\mathtt{AP}$, there exists a parity automaton $\mathcal{A}$ over the alphabet $2^{\mathtt{AP}}$ such that $\mathcal{L}(\Phi) = \mathcal{L}(\mathcal{A})$. We write $\mathit{LTL}\textsc{to}\mathit{A}\textsc{ut}(\Phi)$ to denote the procedure that returns such an automaton $\mathcal{A}$.*

Safety LTL. An LTL specification Φ is said to be *safety* if the language $\mathcal{L}(\Phi)$ can be expressed by a safety automaton. With this, we also use $\text{LTL}\textsc{to}\text{A}\textsc{ut}(\Phi)$ to denote the procedure that translates a safety LTL specification Φ into an equivalent safety automaton $\mathcal{A}$ with $\mathcal{L}(\Phi) = \mathcal{L}(\mathcal{A})$.

2.3 Branching-Time Properties

While linear-time properties describe the behavior of a system along a single computation path or trace, branching-time properties allow reasoning about multiple possible future paths that can be taken from a given state. To formalize branching-time properties, we use Kripke structures to model the system's states and transitions, and we introduce computation tree logic (CTL) and CTL* as temporal logics for specifying these properties.

2.3.1 Kripke Structures

A *Kripke structure* $\mathcal{K} = (S, I, R, L)$ over a set $\mathtt{AP}$ of atomic propositions consists of

- a finite set of states S,
- a set of initial states $I \subseteq S$,

- a transition relation $R \subseteq S \times S$ such that for all states s there exists a state s' satisfying $(s, s') \in R$, and
- a labeling function $\ell \colon S \to 2^{\mathtt{AP}}$.

We define the size of $\mathcal{K}$ as $|S|$. The set $\mathrm{post}(s) = \{s' \in S \mid (s, s') \in R\}$ contains all successors of $s \in S$. A *path* of the Kripke structure $\mathcal{K}$ is an infinite sequence $\rho \in S^\omega$ of states such that $\rho[i+1] \in \mathrm{post}(\rho[i])$ for each $i \geq 0$. A trace generated by a path ρ is the infinite sequence of labels $L(\rho[0])L(\rho[1]) \cdots \in (2^{\mathtt{AP}})^\omega$. For a state s, let $\mathrm{paths}(s)$ denote the set of all paths starting from s, and let $\mathit{Traces}(\mathcal{K})$ denote the set of all traces generated by paths starting from initial states in I.

With this, one can also define satisfaction of linear-time properties in Kripke structures as follows: a Kripke structure $\mathcal{K}$ satisfies a linear-time property Φ, denoted by $\mathcal{K} \models \Phi$, if all traces in $\mathit{Traces}(\mathcal{K})$ satisfy Φ.

2.3.2 Computation Tree Logic

Computation tree logic (CTL) is a branching-time temporal logic that allows reasoning about both the states and paths in a Kripke structure.

Syntax. CTL formulas are classified into state and path formulas. Intuitively, state formulas express properties of states, whereas path formulas express temporal properties of paths. For ease of notation, while using CTL formulas, we denote state formulas and path formulas by Greek capital letters and Greek lowercase letters, respectively. CTL state formulas over $\mathtt{AP}$ are given by the grammar

$$\Phi ::= \mathsf{True} \mid a \mid \Phi \vee \Psi \mid \neg\Phi \mid \exists\varphi \mid \forall\varphi,$$

where $a \in \mathtt{AP}$, Ψ is a state formula, and φ is a path formula. CTL path formulas are given by the grammar

$$\varphi ::= \bigcirc \Phi \mid \Phi \,\mathbf{U}\, \Psi,$$

where Φ and Ψ are state formulas.

In addition, as in LTL, we also include the usual derived operators that define the additional state formulas False, $\Phi \wedge \Psi$, and $\Phi \Rightarrow \Psi$, and the additional path formulas $\Diamond \Phi$, $\Box \Phi$, and $\Phi \,\mathbf{W}\, \Psi$.

Semantics. The semantics of CTL formulas are defined with respect to a Kripke structure $\mathcal{K} = (S, I, R, L)$ over a set $\mathtt{AP}$ of atomic propositions. The satisfaction relation $\models$ for CTL state formulas (including derived ones) is defined as follows for all states $s \in S$:

- $s \models a$ if $a \in L(s)$, for $a \in \mathtt{AP}$,
- $s \models \Phi \vee \Psi$ if $s \models \Phi$ or $s \models \Psi$,
- $s \models \Phi \wedge \Psi$ if $s \models \Phi$ and $s \models \Psi$,

- $s \models \neg\Phi$ if $s \not\models \Phi$,
- $s \models \Phi \Rightarrow \Psi$ if $s \not\models \Phi$ or $s \models \Psi$,
- $s \models \exists\varphi$ if there exists a path $\rho \in \text{paths}(s)$ such that $\rho \models \varphi$,
- $s \models \forall\varphi$ if for all paths $\rho \in \text{paths}(s)$, we have $\rho \models \varphi$.

The satisfaction relation for CTL path formulas is defined as follows for all paths $\rho \in \text{paths}(s)$:

- $\rho \models \bigcirc \Phi$ if $\rho[1] \models \Phi$,
- $\rho \models \Diamond \Phi$ if there exists $i \geq 0$ such that $\rho[i] \models \Phi$,
- $\rho \models \Box \Phi$ if for all $i \geq 0$, we have $\rho[i] \models \Phi$,
- $\rho \models \Phi \,\mathbf{U}\, \Psi$ if there exists $i \geq 0$ such that $\rho[i] \models \Psi$ and for all $0 \leq j < i$, we have $\rho[j] \models \Phi$,
- $\rho \models \Phi \,\mathbf{W}\, \Psi$ if either $\rho \models \Phi \,\mathbf{U}\, \Psi$ or for all $i \geq 0$, we have $\rho[i] \models \Phi$.

A CTL state formula Φ is *satisfied* in the Kripke structure $\mathcal{K}$, denoted by $\mathcal{K} \models \Phi$, if for all initial states $s \in I$, we have $s \models \Phi$.

2.3.3 CTL*

CTL* is a more expressive temporal logic that extends CTL by allowing arbitrary boolean combinations of path and state formulas. The syntax of CTL* is more complex than that of CTL, as it allows for nested path quantifiers and a richer set of operators.

Syntax. Unlike CTL, CTL* allows path quantifiers $\exists$ and $\forall$ to be arbitrarily nested with temporal operators. The syntax of CTL* state formulas is the same as in CTL. Moreover, CTL* path formulas are similar to LTL formulas. Consequently, CTL* state formulas over $\texttt{AP}$ are formed according to the grammar

$$\Phi ::= \mathsf{True} \mid a \mid \Phi \vee \Psi \mid \neg\Phi \mid \exists\varphi \mid \forall\varphi,$$

where $a \in \texttt{AP}$, Ψ is a state formula, and φ is a path formula. CTL* path formulas are given by the grammar

$$\varphi ::= \Phi \mid \varphi \vee \psi \mid \neg\varphi \mid \bigcirc\varphi \mid \varphi \,\mathbf{U}\, \psi,$$

where Φ is a state formula, and φ and ψ are path formulas.

In addition, the usual derived operators are included in both state and path formulas as in CTL and LTL.

Semantics. The semantics of CTL* formulas are also defined with respect to a Kripke structure $\mathcal{K} = (S, I, R, L)$ over a set $\mathtt{AP}$ of atomic propositions. While the satisfaction relation for CTL* state formulas is similar to that of CTL state formulas, the satisfaction relation for CTL* path formulas is similar to that of LTL formulas. Formally, the satisfaction relation $\models$ for CTL* state formulas is defined as follows for all states $s \in S$:

- $s \models a$ if $a \in L(s)$, for $a \in \mathtt{AP}$,
- $s \models \Phi \vee \Psi$ if $s \models \Phi$ or $s \models \Psi$,
- $s \models \Phi \wedge \Psi$ if $s \models \Phi$ and $s \models \Psi$,
- $s \models \neg\Phi$ if $s \not\models \Phi$,
- $s \models \Phi \Rightarrow \Psi$ if $s \not\models \Phi$ or $s \models \Psi$,
- $s \models \exists\varphi$ if there exists a path $\rho \in \text{paths}(s)$ such that $\rho \models \varphi$,
- $s \models \forall\varphi$ if for all paths $\rho \in \text{paths}(s)$, we have $\rho \models \varphi$.

Similarly, the satisfaction relation $\models$ for CTL* path formulas is defined as follows for all paths $\rho \in \text{paths}(s)$:

- $\rho \models \Phi$ if $\rho[0] \models \Phi$,
- $\rho \models \varphi \vee \psi$ if $\rho \models \varphi$ or $\rho \models \psi$,
- $\rho \models \neg\varphi$ if $\rho \not\models \varphi$,
- $\rho \models \bigcirc\varphi$ if $\rho[1..] \models \varphi$,
- $\rho \models \varphi \,\mathbf{U}\, \psi$ if there exists $i \geq 0$ such that $\rho[i..] \models \psi$ and for all $0 \leq j < i$, we have $\rho[j..] \models \varphi$,
- $\rho \models \Diamond\varphi$ if there exists $i \geq 0$ such that $\rho[i..] \models \varphi$,
- $\rho \models \Box\varphi$ if for all $i \geq 0$, we have $\rho[i..] \models \varphi$,
- $\rho \models \varphi \,\mathbf{W}\, \psi$ if either $\rho \models \varphi \,\mathbf{U}\, \psi$ or for all $i \geq 0$, we have $\rho[i..] \models \varphi$.

A CTL* state formula Φ is *satisfied* in the Kripke structure $\mathcal{K}$, denoted by $\mathcal{K} \models \Phi$, if for all initial states $s \in I$, we have $s \models \Phi$.

2.4 Logical Controller Synthesis

Logical controller synthesis aims to automatically construct a controller that ensures satisfaction of a given specification while obeying the dynamics of the underlying system represented by a plant model. To formally define logical controller synthesis, we consider three processes: a controller $\mathtt{c}$, its environment $\mathtt{e}$, and a plant $\mathtt{p}$. The interaction among

these processes is given by an *architecture* that specifies how the processes communicate through their input and output propositions. The *implementation of the plant* represents the plant model and the *implementation of the controller* represents the controller to be synthesized. With this setting, let us introduce the notation and preliminaries for logical controller synthesis.

Architectures. Given three processes $\{\mathtt{c}, \mathtt{p}, \mathtt{e}\}$, an *architecture* over a set of atomic propositions $\mathtt{AP}$ is a tuple $(I_\mathtt{e}, O_\mathtt{e}, I_\mathtt{c}, O_\mathtt{c}, I_\mathtt{p}, O_\mathtt{p})$ consisting of the input and output propositions of each process. Throughout the thesis, we assume that the sets of output propositions of all processes form a partition of $\mathtt{AP}$, i.e., $O_\mathtt{c} \cup O_\mathtt{p} \cup O_\mathtt{e} = \mathtt{AP}$, and the input propositions of each process are output propositions of other processes, i.e., $I_i = \bigcup_{j \neq i} O_j$ for all $i \in \{\mathtt{c}, \mathtt{p}, \mathtt{e}\}$.

Implementations. We model an *implementation* for a process as a Moore machine $\mathcal{M} = (S, s_0, \tau, o)$ over its inputs I and outputs O, consisting of

- a finite set of states S,
- an initial state $s_0 \in S$,
- a transition function $\tau : S \times 2^I \to S$ over inputs, and
- an output labeling function $o : S \to 2^O$.

For an implementation $\mathcal{M} = (S, s_0, \tau, o)$, we write $\mathcal{M}(s)$ to denote the implementation (S, s, τ, o) with the same transition and output labeling function but with initial state s.

For a finite input sequence $\gamma = \alpha_0 \alpha_1 \cdots \alpha_{k-1} \in (2^I)^*$, $\mathcal{M}$ produces a finite path $s_0 s_1 \cdots s_k$ and an output sequence $o(s_0)o(s_1)\cdots o(s_k) \in (2^O)^*$ such that $\tau(s_j, \alpha_j) = s_{j+1}$. We write $\mathit{out}(\mathcal{M}, \gamma)$ to denote $o(s_k)$. Similarly, for an infinite input sequence $\gamma \in (2^I)^\omega$, $\mathcal{M}$ produces an infinite path $s_0 s_1 \cdots$ and an infinite output sequence $o(s_1)o(s_1)\cdots \in (2^O)^\omega$. We write $\mathit{Traces}(\mathcal{M})$ to denote the set of all traces $\gamma \in \Sigma^\omega$ such that for input sequence $\gamma|_I$, $\mathcal{M}$ produces the output sequence $\gamma|_O$. We say that $\mathcal{M}$ satisfies a linear-time specification φ, denoted by $\mathcal{M} \models \varphi$, if $\gamma \models \varphi$ holds for all trace $\gamma \in \mathit{Traces}(\mathcal{M})$.

For simplicity, we refer to the plant implementations as *plants* and the controller implementations as *controllers*. The set of all plant implementations and controller implementations are `Plants` and `Contrs`, respectively.

Parallel Composition. The parallel composition of a controller $\mathcal{M}_\mathtt{c} = (S^\mathtt{c}, s_0^\mathtt{c}, \tau^\mathtt{c}, o^\mathtt{c})$ and a plant $\mathcal{M}_\mathtt{p} = (S^\mathtt{p}, s_0^\mathtt{p}, \tau^\mathtt{p}, o^\mathtt{p})$, denoted by $\mathcal{M}_\mathtt{c} \parallel \mathcal{M}_\mathtt{p}$, is an implementation (S, s_0, τ, o) over inputs $O_\mathtt{e}$ and outputs $O_\mathtt{c} \cup O_\mathtt{p}$ with

- $S = S^\mathtt{c} \times S^\mathtt{p}$,
- $s_0 = (s_0^\mathtt{c}, s_0^\mathtt{p})$,
- $\tau((s^\mathtt{c}, s^\mathtt{p}), \sigma) = (\tau^\mathtt{c}(s^\mathtt{c}, \sigma \cup o^\mathtt{p}(s^\mathtt{p})), \tau^\mathtt{p}(s^\mathtt{p}, \sigma \cup o^\mathtt{c}(s^\mathtt{c})))$ for every $\sigma \in 2^{O_\mathtt{e}}$, and
- $o((s^\mathtt{c}, s^\mathtt{p})) = o^\mathtt{c}(s^\mathtt{c}) \cup o^\mathtt{p}(s^\mathtt{p})$.

Moore Machine to Kripke Structure. A Moore machine $\mathcal{M} = (S, s_0, \tau, o)$ over inputs I and outputs O can be transformed into an equivalent Kripke structure $\mathcal{K} = (S', I, R, L)$ over $\mathtt{AP} = I \cup O$, where:

- $S' = S \times 2^I$ is the set of states,
- $I = \{(s_0, \sigma) \mid \sigma \in 2^I\}$ is the set of initial states,
- $R \subseteq S' \times S'$ is defined by $((s, \sigma), (s', \sigma')) \in R$ if $\tau(s, \sigma) = s'$, and
- $L : S' \to 2^{\mathtt{AP}}$ is defined by $L(s, \sigma) = \sigma \cup o(s)$.

This ensures that the set of traces generated by the Kripke structure $\mathcal{K}$ is equivalent to the set of traces produced by the Moore machine $\mathcal{M}$, i.e., $\mathit{Traces}(\mathcal{K}) = \mathit{Traces}(\mathcal{M})$. Hence, the satisfaction of branching-time specifications by Kripke structures can be directly applied to Moore machines, i.e., for a CTL or CTL* specification Φ, we define $\mathcal{M} \models \Phi$ if and only if $\mathcal{K} \models \Phi$.

Product with Automaton. Given an automaton $\mathcal{A} = (Q, q_0, \delta, \Omega)$ and a Moore machine $\mathcal{M}$ over inputs I and outputs O with $2^{I \cup O} \subseteq \Sigma$, we define their product $\mathcal{A} \times \mathcal{M} = (Q \times S, (q_0, s_0), \delta', \Omega')$ as a partial automaton with acceptance condition Ω' given by the set of runs whose corresponding run is accepting by Ω, and a partial transition function $\delta' : (Q \times S) \times \Sigma \to (Q \times S)$ such that $\delta'((q, s), \sigma)$ is defined if and only if $\sigma \cap O = o(s)$ and in that case $\delta'((q, s), \sigma) = (\delta(q, \sigma), \tau(s, \sigma))$.

We write $\mathit{Runs}(\mathcal{A}, q, \mathcal{M})$ to denote the set of runs ρ of $\mathcal{A}$ from state q for which there exists a corresponding run ρ' of $\mathcal{A} \times \mathcal{M}$ with $\rho'[i] = (\rho[i], s_i)$ for each $i \geq 0$. We write $\mathtt{reachable}(\mathcal{A} \times \mathcal{M})$ to denote the set of states (q, s) that are reachable from the initial state (q_0, s_0) in $\mathcal{A} \times \mathcal{M}$, i.e., there exists a run ρ of $\mathcal{A} \times \mathcal{M}$ with $\rho[0] = (q_0, s_0)$ and $\rho[k] = (q, s)$ for some $k \geq 0$.

Synthesis Problems. With the above definitions, logical controller synthesis aims to synthesize a controller that satisfies a given specification while enforcing the dynamics of a given plant. This can be formally defined as follows.

Problem 2.1 (Logical Controller Synthesis)**.** *Given a specification Φ over an architecture, and a fixed plant $\mathcal{M}_{\mathtt{p}} \in \mathtt{Plants}$, the* logical controller synthesis *problem is to find a controller $\mathcal{M}_{\mathtt{c}} \in \mathtt{Contrs}$ that is* correct *for the plant $\mathcal{M}_{\mathtt{p}}$, i.e., $\mathcal{M}_{\mathtt{c}} \parallel \mathcal{M}_{\mathtt{p}} \models \Phi$.*

A plant is said to be *admissible* if there exists a controller that is correct for the plant.

Reactive synthesis is a special case of logical controller synthesis where there is no explicit plant model. More formally, the architecture in reactive synthesis consists of only two processes, i.e., a controller $\mathtt{c}$ and its environment $\mathtt{e}$. For simplicity, we just consider I and O to denote the input and output propositions of the controller $\mathtt{c}$, respectively. In this case, an implementation of the controller $\mathcal{M}_{\mathtt{c}}$ is a Moore machine over inputs I and outputs O. With this setting, reactive synthesis can be defined as follows.

Problem 2.2 (Reactive Synthesis)**.** *Given a specification* Φ *over an architecture, the* reactive synthesis *problem is to find a controller* $\mathcal{M}_c$ *such that* $\mathcal{M}_c \models \Phi$*.*

We refer to reactive synthesis problem where the specification is given in LTL, CTL, or CTL* as reactive synthesis for LTL, CTL, or CTL*, respectively.

2.5 Games on Graphs

A standard approach to solving the logical controller synthesis problem (as given in Problem 2.1) is to reduce it to a two-player graph game between the controller and its environment. Intuitively, the game graph models all possible interactions between the controller and its environment, possibly including restrictions imposed by the plant model (as defined in Section 2.4), while the winning condition captures the desired logical specification. Hence, solving the logical controller synthesis problem reduces to finding a winning strategy for the controller in the corresponding game. In this section, we recall the definitions of two-player graph games with ω-regular winning conditions in Section 2.5.1. We then formalize the key insight that both the logical controller synthesis problem and the reactive synthesis problem (as given in Problems 2.1 and 2.2) can be reduced to solving parity games in Section 2.5.2. We present standard results for solving games with various ω-regular winning conditions in Section 2.5.3. Finally, we extend these definitions to multiplayer games that are used to model systems with multiple components in Section 2.5.4.

2.5.1 Two-Player Games

Throughout the thesis, unless stated otherwise, we refer to games and game graphs as two-player games and two-player game graphs, respectively, as formalized below.

Game Graphs. A (two-player) *game graph* is a tuple $G = (V, E, v_0, L)$ where

- $V = V_0 \cup V_1$ is a finite set of *vertices* partitioned into Player 0 vertices V_0 and Player 1 vertices V_1,
- $E \subseteq V \times V$ is a set of *edges*,
- $v_0 \in V$ is the *initial vertex*, and
- $L : V \to 2^{\mathtt{AP}}$ is a *labeling function* that assigns to each vertex a set of atomic propositions from $\mathtt{AP}$.

We say the game graph G is *alternating* if for every edge $(u, v) \in E$, it holds that $u \in V_0$ implies $v \in V_1$ and vice versa.

Without loss of generality, we assume that for every $v \in V$ there exists $v' \in V$ s.t. $(v, v') \in E$. We write E_0 and E_1 to denote the edges from Player 0 and Player 1 vertices, respectively, i.e., $E_0 = E \cap (V_0 \times V)$ and $E_1 = E \cap (V_1 \times V)$. For a set of vertices $V' \subseteq V$,

we write $\mathsf{pre}(V') = \{v \in V \mid \exists v' \in V' : (v, v') \in E\}$ to denote the set of predecessors of vertices in V' (and $E(V')$ to denote the set of successors as defined in Section 2.1).

Throughout the thesis, in many examples, we write e_{uv} to denote an edge $(u, v) \in E$. Furthermore, we sometimes drop the initial vertex v_0 and/or the labeling function L from the notation of a game graph if it is not relevant in the context, i.e., we write (V, E) or (V, E, v_0) or (V, E, L) instead of (V, E, v_0, L).

Plays. A *play* is an infinite sequence of vertices $\rho = u_0 u_1 \dots$ with $(u_j, u_{j+1}) \in E$ for all $j \geq 0$. Furthermore, a *play from a vertex* v is a play that starts from v, i.e., $\rho = v u_1 u_2 \dots$. We collect all plays in the set $\mathtt{plays}(G)$ and all plays from a vertex v in the set $\mathtt{plays}_v(G)$. A *play prefix* or *history* $\mathtt{h} = v v_1 \cdots v_k$ is a prefix of some play $\rho = \mathtt{h} v_{k+1} \dots \in V^\omega$.

Winning Conditions. Given a game graph G, we consider *winning conditions* (or simply *objectives*) specified using an LTL formula Φ over the vertex set V. In this case the set of desired infinite plays is given by the semantics of Φ over G, which is an ω-regular language $\mathcal{L}(G, \Phi) = \mathcal{L}(\Phi) \cap \mathtt{plays}(G)$. If the game graph G is clear from the context, we simply write $\mathcal{L}(\Phi)$ instead of $\mathcal{L}(G, \Phi)$.

We specifically consider safety conditions $\Box I$, reachability conditions $\Diamond I$, Büchi conditions $\Box\Diamond I$, co-Büchi conditions $\Diamond\Box I$, and parity winning conditions $\mathit{Parity}(\mathbb{P})$ for some subset $I \subseteq V$ and priority function $\mathbb{P} : V \to \mathbb{N}$ as specified in Section 2.2.3.

We also consider various types of specifications that are specified using LTL formulas over the set $V \cup E$, where edges $e = (u, v) \in E$ are used as syntactic sugar for $u \wedge \bigcirc v$.

Games. A (two-player) *game* is a pair $\mathcal{G} = (G, \Phi)$ where

- G is a game graph, and
- Φ is a *winning condition* over G.

A game with a safety, reachability, Büchi, co-Büchi, and parity winning condition is called a *safety game*, *reachability game*, *Büchi game*, *co-Büchi game*, and *parity game*, respectively.

Strategies. A *strategy* of Player p, $p \in \{0, 1\}$, is a function $\pi_p \colon V^* V_p \to V$ such that for every $\mathtt{h} v \in V^* V_p$, it holds that $\pi_p(\mathtt{h} v) \in E(v)$. A *strategy profile* is a tuple (π_0, π_1) of strategies, one for each player. A strategy is *memoryless* if it only depends on the last vertex of the play prefix, i.e., $\pi_p(\mathtt{h} v) = \pi_p(v)$ for all $\mathtt{h} v \in V^* V_p$. In this case, we write $\pi_p \colon V_p \to V$.

Given a strategy π_p, we say that the play $\rho = v_0 v_1 \dots$ is a π_p-play if $v_{k-1} \in V_p$ implies $v_k = \pi_p(v_0 \dots v_{k-1})$ for all $k \in \mathit{dom}(\rho)$. A play that is both a π_0-play and a π_1-play is referred to as a (π_0, π_1)-play. We collect all π_p-plays and (π_0, π_1)-plays in the sets $\mathtt{plays}(\pi_p)$ and $\mathtt{plays}(\pi_0, \pi_1)$, respectively. Furthermore, we write $\mathtt{plays}_v(\pi_p)$ and $\mathtt{plays}_v(\pi_0, \pi_1)$ for the corresponding sets of plays starting from $v \in V$.

Winning. Given a game graph G and an objective Φ, the following definitions capture the standard notions of winning for Φ.

- A play ρ is *winning* for Φ if $\rho \in \mathcal{L}(\Phi)$.
- A Player p strategy π_p is *winning for* Φ *from* a vertex v, denoted by $\pi_p \models_v \Phi$, if $\texttt{plays}_v(\pi_p) \subseteq \mathcal{L}(\Phi)$.
- A Player p strategy π_p is *winning for* Φ, denoted by $\pi_p \models \Phi$, if it is winning for Φ from the initial vertex v_0.

We collect all vertices from which Player p has a winning strategy for Φ in the region $\langle\langle p \rangle\rangle \Phi \subseteq V$.

Furthermore, given a (two-player) game $\mathcal{G} = (G, \Phi)$, the following definitions capture the standard notions of winning in zero-sum and cooperative settings.

- A play ρ is *winning* if $\rho \in \mathcal{L}(\Phi)$.
- A strategy π is *winning* (for Player 0) *from* a vertex v if $\pi \models_v \Phi$. Such a vertex $v \in V$ is said to be *winning*.
- A strategy is *winning* (for Player 0) if $\pi \models \Phi$.
- A strategy is *winning for Player* 1 *from* a vertex v if $\pi \models_v \neg\Phi$.
- A strategy is *winning for Player* 1 if $\pi \models \neg\Phi$.
- A strategy profile (π_0, π_1) is *cooperatively winning from* a vertex v, if $\texttt{plays}_v(\pi_0, \pi_1) \subseteq \mathcal{L}(\Phi)$. Such a vertex $v \in V$ is said to be *cooperatively winning*.
- A strategy profile (π_0, π_1) is *cooperatively winning* if it is cooperatively winning from the initial vertex v_0.

We collect all winning vertices and cooperatively winning vertices in the regions $\langle\langle 0 \rangle\rangle \Phi$ and $\langle\langle 0, 1 \rangle\rangle \Phi$, respectively. Furthermore, a strategy is called *uniformly winning* if it is winning from all winning vertices and a strategy profile is called *uniformly cooperatively winning* if it is cooperatively winning from all cooperatively winning vertices.

Restricted Game. Given a game graph $G = (V, E, v_0, L)$ and a subset of vertices $U \subseteq V$ containing the initial vertex v_0, the *restriction* of G to U is defined as $G|_U = (U, E', v_0, L|_U)$ where $E' = E \cap (U \times U)$ and $L|_U$ is the restriction of the labeling function L to U, i.e., $L|_U(u) = L(u)$ for all $u \in U$.

Analogously, given a game $\mathcal{G} = (G, \Phi)$ and a subset of vertices $U \subseteq V$ containing the initial vertex v_0, the *restriction* of $\mathcal{G}$ to U is defined as $\mathcal{G}|_U = (G|_U, \Phi|_U)$ where $\Phi|_U$ is the restriction of the winning condition Φ to plays over U, i.e., $\mathcal{L}(\Phi|_U) = \mathcal{L}(\Phi) \cap U^\omega$. For instance, if Φ is a parity winning condition $\mathit{Parity}(\mathbb{P})$ for some priority function $\mathbb{P} : V \to \mathbb{N}$, then $\Phi|_U$ is the parity winning condition $\mathit{Parity}(\mathbb{P}|_U)$ where $\mathbb{P}|_U$ is the restriction of $\mathbb{P}$ to U, i.e., $\mathbb{P}|_U(u) = \mathbb{P}(u)$ for all $u \in U$.

2.5.2 Reductions to Parity Games

Games with arbitrary ω-regular winning conditions and both the logical controller synthesis problem and the reactive synthesis problem (as given in Problems 2.1 and 2.2) with LTL specifications can be reduced to solving parity games. This is formalized in the following paragraphs.

ω-regular Games to Parity Games. It is well-known that every game with an arbitrary ω-regular winning condition can be reduced to a parity game (possibly with a larger vertex set) [93]. Specifically, given a game $\mathcal{G} = (G, \Phi)$ with an ω-regular winning condition Φ, one can construct a parity game $\mathcal{G}' = (G', \mathit{Parity}(\mathbb{P}))$ with a bijection between the plays of $\mathcal{G}$ and $\mathcal{G}'$ such that a play ρ in $\mathcal{G}$ satisfies Φ if and only if the corresponding play ρ' in $\mathcal{G}'$ satisfies $\mathit{Parity}(\mathbb{P})$.

LTL to Parity Games. Another well-known result is that the reactive synthesis problem for LTL specifications (as introduced in Section 2.4) can be reduced to solving parity games. More specifically, given an LTL formula Φ over a set $\mathtt{AP} = I \cup O$ of atomic propositions, one can construct an alternating parity game $\mathcal{G} = (G, \mathit{Parity}(\mathbb{P}))$ such that every winning strategy for Player 0 in $\mathcal{G}$ corresponds to a controller that satisfies Φ.

The intuition behind this reduction is as follows. An alternating game graph $G = (V, E, v_0, L)$ captures the interaction between the controller (Player 0) and its environment (Player 1) as follows: starting from the initial vertex $v_0 \in V_1$, in each step of a play, first Player 1 chooses the propositions from $\mathtt{AP}_1 = I$ by choosing a successor vertex in V_0, and then Player 0 chooses the propositions from $\mathtt{AP}_0 = O$ by choosing a successor vertex in V_1. This is captured by the labeling function L such that it only assigns propositions from $\mathtt{AP}_{1-p}$ to vertices in V_p, i.e., $L(v) \subseteq \mathtt{AP}_{1-p}$ for every $v \in V_p$ and $p \in \{0, 1\}$. Hence, every play ρ from the initial vertex in the game generates a trace over $\mathtt{AP}$ that captures the sequence of propositions chosen by both players, i.e., the trace γ *generated* by a play ρ is defined such that $\gamma[i] = L(\rho[2i+1]) \cup L(\rho[2i+2])$ for all $i \geq 0$. With this and the equivalence between LTL formulas and parity automata (as in Proposition 2.1), one can construct a parity game $\mathcal{G}$ such that a play from initial vertex is winning in $\mathcal{G}$ if and only if its generated trace satisfies Φ.

Furthermore, we say a game $\mathcal{G}$ or game graph G is *total* w.r.t. $\mathtt{AP}' \subseteq \mathtt{AP}$ if for every trace γ' over $\mathtt{AP}'$, there exists a trace γ generated by a play from initial vertex in G such that $\gamma|_{\mathtt{AP}'} = \gamma'$. With this, we summarize the reduction from LTL formulas to parity games as follows.

Proposition 2.2 ([156, Section 4]). *Every LTL formula Φ over $\mathtt{AP} = \mathtt{AP}_0 \cup \mathtt{AP}_1$ can be translated into a parity game $\mathcal{G} = ((V, E, v_0, L), \mathit{Parity}(\mathbb{P}))$ with $v_0 \in V_1$ and $L := V_p \to 2^{\mathtt{AP}_{1-p}}$ such that $\mathcal{G}$ is total w.r.t. $\mathtt{AP}$. Moreover, a play from v_0 is winning in $\mathcal{G}$ iff its generated trace satisfies Φ.*

With Proposition 2.2, the problem of computing a logical controller which satisfies a given specification Φ (as in reactive synthesis setting without a plant) reduces to computing a winning strategy in a parity game $\mathcal{G}$. Furthermore, a standard product construction

(see standard books [93, 112] for details) with the plant model (as in Section 2.4) can be used to extend this reduction to the setting with a plant. Hence, the logical controller synthesis problem can be reduced to solving parity games.

2.5.3 Standard Results for Solving Two-Player Games

As the previous sections showed that Problems 2.1 and 2.2 reduce to solving parity games, we now recall standard results for solving two-player games with various ω-regular winning conditions (see standard books [93, 112] for details).

Proposition 2.3. *Given a game $\mathcal{G} = (G, \Phi)$ with n vertices and m edges, the winning regions $\langle\langle 0 \rangle\rangle \Phi$ and the cooperative winning regions $\langle\langle 0, 1 \rangle\rangle \Phi$ can be computed in time*

- $\mathcal{O}(m + n)$ *if $\Phi = \Box I$ is a safety condition;*
- $\mathcal{O}(m + n)$ *if $\Phi = \Diamond I$ is a reachability condition;*
- $\mathcal{O}(mn)$ *and $\mathcal{O}(m + n)$, respectively, if $\Phi = \Box\Diamond I$ is a Büchi condition;*
- $\mathcal{O}(mn)$ *and $\mathcal{O}(m + n)$, respectively, if $\Phi = \Diamond\Box I$ is a co-Büchi condition;*
- $n^{\mathcal{O}(\log n)}$ *and $\mathcal{O}((m + n) \log d)$, respectively, if $\Phi = Parity(\mathbb{P})$ is a parity condition with d priorities.*

We write $\textsc{solveSafety}(G, I)$, $\textsc{SolveReach}(G, I)$, $\textsc{solveBüchi}(G, I)$, $\textsc{solveCoBüchi}(G, I)$, $\textsc{solveParity}(G, \mathbb{P})$ for the corresponding procedures to compute winning regions $\langle\langle 0 \rangle\rangle \Phi$, and $\textsc{solveCoopSafety}(G, I)$, $\textsc{solveCoopReach}(G, I)$, $\textsc{solveCoopBüchi}(G, I)$, $\textsc{solveCoopCoBüchi}(G, I)$, $\textsc{solveCoopParity}(G, \mathbb{P})$ for the corresponding procedures to compute cooperative winning regions $\langle\langle 0, 1 \rangle\rangle \Phi$.

While recent results improve the time complexity of solving parity games, i.e., computing winning regions $\langle\langle 0 \rangle\rangle \Phi$, to quasi-polynomial time [52, 122, 171], the classical Zielonka's algorithm [225] is known to perform well in practice even though its worst-case time complexity is $\mathcal{O}(n^d)$. Throughout the thesis, we use both Zielonka's algorithm and the quasi-polynomial algorithms to solve parity games depending on the context.

2.5.4 Multi-Player Games

While two-player games are sufficient to model the interaction between a controller and its environment in *monolithic* systems, i.e., systems with a single controller interacting with its environment, they are not well-suited to model *distributed* systems, i.e., systems with multiple components that interact with each other. In such systems, each component has its own specification, and for each component, the other components acts as its environment. Hence, to model such systems, we need to consider multi-player games with multiple winning conditions, one for each player, as detailed below.

k-Player Game Graphs. A *k-player game graph* is a tuple $G = (V, E, v_0, L)$ played by k players in $\mathtt{P} = [0; k-1]$, where

- $V = \bigcup_{p \in \mathtt{P}} V_p$ is a finite set of *vertices* partitioned into k sets such that V_p is the set of vertices for Player p,
- E, v_0, and L are defined analogously to the two-player case.

For each $p \in \mathtt{P}$, we write E_p to denote the edges from Player p's vertices, i.e., $E_p = E \cap (V_p \times V)$. Further, we write $V_{\neg p}$ and $E_{\neg p}$ to denote the set $\bigcup_{q \neq p} V_p$ and $\bigcup_{q \neq p} E_q$, respectively. Again, we sometimes drop v_0 and/or L from the notation of a game graph if it is not relevant in the context.

Plays, play prefixes, winning conditions, and restrictions of multi-player game graphs are defined analogously to the two-player case.

k-player Games. A *k-player (multi-objective) game* is a pair $\mathcal{G} = (G, (\Phi_p)_{p \in \mathtt{P}})$ where G is a k-player game graph and each Φ_p is an *objective* for Player p over G. We call $\mathcal{G}$ a parity game if all involved winning conditions are parity objectives.

Note that a two-player game (G, Φ) can be seen as a special case of a k-player game $(G, (\Phi_p)_{p \in \mathtt{P}})$ with $k = 2$ and $\Phi_0 = \Phi$, $\Phi_1 = \neg\Phi$.

Strategies. A strategy of Player p, $p \in \mathtt{P}$, is a function $\pi_p \colon V^* V_p \to V$ such that for every $\rho v \in V^* V_p$, it holds that $(v, \pi_p(\rho v)) \in E$. A strategy profile for a set of players $\mathtt{P}' \subseteq \mathtt{P}$ is a tuple $(\pi_p)_{p \in \mathtt{P}'}$ of strategies, one for each player in $\mathtt{P}'$. To simplify notation, we write $\mathtt{P}_{\neg p}$ and $\pi_{\neg p}$ to denote the set $\mathtt{P} \setminus \{p\}$ and their strategy profile $(\pi_q)_{q \in \mathtt{P} \setminus \{p\}}$, respectively. Given a strategy profile $(\pi_p)_{p \in \mathtt{P}'}$, we say that a play $\rho = u_0 u_1 \dots$ is a *$(\pi_p)_{p \in \mathtt{P}'}$-play* if for every $p \in \mathtt{P}'$ and for all $\ell \geq 1$, it holds that $u_{\ell-1} \in V_p$ implies $u_\ell = \pi_p(u_0 \dots u_{\ell-1})$.

Winning. Given a k-player game graph G and an arbitrary objective φ, the following definitions capture the standard notions of winning for φ. Note that φ is not necessarily the objective of any player in this case.

- A play ρ is *winning* for φ if $\rho \in \mathcal{L}(\varphi)$.
- A strategy profile $(\pi_p)_{p \in \mathtt{P}'}$ for $\mathtt{P}' \subseteq \mathtt{P}$ is winning for φ from a vertex v, denoted by $(\pi_p)_{p \in \mathtt{P}'} \models_v \varphi$, if every $(\pi_p)_{p \in \mathtt{P}'}$-play from v is in $\mathcal{L}(\varphi)$.
- A strategy profile $(\pi_p)_{p \in \mathtt{P}'}$ for $\mathtt{P}' \subseteq \mathtt{P}$ is winning for φ, denoted by $(\pi_p)_{p \in \mathtt{P}'} \models \varphi$, if it is winning from the initial vertex.

We collect all vertices from which there exists a strategy profile for players in $\mathtt{P}'$ that is winning for φ in the region $\langle\!\langle \mathtt{P}' \rangle\!\rangle \varphi$.

Furthermore, given a k-player game $(G, (\Phi_p)_{p \in \mathtt{P}})$, the following definition captures the notion of winning in a fully cooperative setting.

- A play ρ is *winning* if for every $p \in \mathtt{P}$, it holds that $\rho \in \mathcal{L}(\Phi_p)$.
- A strategy profile $(\pi_p)_{p\in\mathtt{P}}$ is *winning from* a vertex v, if $(\pi_p)_{p\in\mathtt{P}} \models_v \Phi_p$ for every $p \in \mathtt{P}$.
- A strategy profile $(\pi_p)_{p\in\mathtt{P}}$ is *winning*, if it is winning from the initial vertex.

We collect all vertices from which there exists a strategy profile that is winning in the *winning region* $\langle\langle \mathtt{P} \rangle\rangle \bigwedge_{p\in\mathtt{P}} \Phi_p$. Furthermore, a strategy profile is called *uniformly winning* if it is winning from every vertex in the winning region.

Part A

Assumptions in Distributed Logical Systems

Part A of this thesis focuses on the permissiveness of assumptions in distributed logical systems. Recall that such systems consist of multiple interacting components, each with their own specification, where each component treats the others as its environment. We are particularly interested in the problem of *distributed synthesis*—the automatic construction of controllers for each component such that the overall system satisfies all components' specifications. While distributed synthesis is known to be undecidable in general [176], a common and tractable variant assumes that the strategic interaction structure among components (i.e., the *plant model*) is known and represented as a game graph.

A common approach to solve such distributed synthesis problems to employ *assume-guarantee reasoning*, where each component is synthesized under the *assumption* that others behave in a certain way. This reduces the distributed synthesis problem to computing an *assume-guarantee contract* for each component. While assume-guarantee reasoning has proven useful in verification [28, 106, 105], where all component implementations are known, its application to synthesis typically encounters a chicken-and-egg problem: without component implementations, no contracts can be obtained, and without contracts, no compliant implementations can be synthesized. This leads existing methods to either require centralization [98], impose strong rationality assumptions [60, 101, 44, 77, 99], or restrict assumptions to safety properties [146]. Most approaches also restrict each component to a single strategy, leading to overly restrictive contracts.

This part of the thesis addresses these shortcomings by locally computing permissive assumptions that retain as many feasible ways of cooperation as possible, allowing distributed implementations to be eventually discovered through iterative and distributed refinement. In particular, our framework addresses not only safety—which naturally allows for maximal permissiveness—but also enables certified interactive progress, which is known to be very challenging even for verification.

To this end, Chapter 3 provides the foundation by formalizing permissive assumptions and developing algorithms for computing them in a monolithic setting, where a single controller interacts with its environment modeled as a two-player game.

Building on this foundation, Chapters 4 and 5 extend these ideas to distributed synthesis by modeling each component as a player in a multi-player game. In particular, we develop *negotiation frameworks*—that iteratively negotiate the guarantees the players provide to each other based on the assumptions they receive from each other—computing permissive assume-guarantee contracts in the form of *contracted specifications* for each player. Such contracted specifications enable each player to independently synthesize a strategy that satisfies its own specification while allowing for permissive interaction with the other players, i.e., without enforcing a single strategy profile for all players. While Chapter 4 focuses on the setting where components interact rationally, Chapter 5 focuses on the fully cooperative setting.

Finally, Chapter 6 applies these permissive assume-guarantee contracts to human-robot interaction, where a robot must satisfy its specification while interacting with a human in a shared environment. We demonstrate how the robot leverages its guarantees to dynamically adapt its strategy, cooperating with the human whenever possible and requesting cooperation online only when necessary to ensure assumptions are fulfilled.

Chapter 3

Permissive Assumptions in Two-Player Games

In this chapter, we address the problem of computing permissive assumptions in monolithic systems, i.e., system with a single controller interacting with its environment. As discussed in Section 2.5, the synthesis problem for such systems can be naturally modeled as a two-player game between the controller (Player 0) and its environment (Player 1). Therefore, we focus on the problem of computing permissive assumptions for two-player games. While existing work [57, 55] on computing assumptions in two-player games focuses on sufficiency (i.e., assumptions that allow the controller to satisfy its specification) and implementability (i.e., assumptions that can be enforced by the environment), they do not ensure permissiveness. In this chapter, we therefore introduce and study the notion of *adequately permissive assumptions* (APAs) that are sufficient, implementable, and permissive.

To this end, we begin by formalizing the notion of APAs for two-player games in Section 3.1. Thereafter, we introduce various *local templates* to capture these assumptions and present algorithms to compute APAs for different winning conditions in Section 3.2. We then discuss alternative formulations of permissive assumptions in Section 3.3. Finally, we report on experimental evaluations using a prototype implementation SImPA in Section 3.4 and review related work in Section 3.5.

3.1 Adequately Permissive Assumptions for Synthesis

Given a two-player game $\mathcal{G}$, the goal of this chapter is to compute assumptions on Player 1 (i.e., the environment), such that both players cooperate *just enough* to fulfill Φ while retaining all possible cooperative strategy choices. Towards a formalization of this intuition, we first define winning under assumptions.

Definition 3.1. Let $\mathcal{G} = ((V, E), \Phi)$ be a game and Ψ be an LTL formula over V. Then a Player 0 strategy π_0 is winning from a vertex v in $\mathcal{G}$ under assumption Ψ, if *for every* Player 1 strategy π_1 s.t. $\texttt{plays}(\pi_1) \subseteq \mathcal{L}(\Psi)$ it holds that the unique (π_0, π_1)-play from

v is winning in $\mathcal{G}$. We denote by $\langle\langle 0\rangle\rangle_\Psi \Phi$ the set of vertices from which such a Player 0 strategy exists.

Let us note that Definition 3.1 slightly differs from the typical linear-time setting, where winning under assumption would be naturally defined in terms of plays instead of strategies (as liner-time properties are defined by a set of traces as discussed in Section 2.2). However, the choice of our formulation of 'winning under assumption' is inspired by synthesis for distributed systems. Here, strategy of the environment (i.e., other components) might be unknown to the controller. Our definition allows us to naturally argue that the strategy π_0 of Player 0 (Controller) is winning for *any* strategy π_1 that Player 1 (Environment) may choose to satisfy the assumption. Furthermore, even though both notions (strategy-based and play-based) are not equivalent in general, they coincide for the class of assumptions we compute. We elaborate on this later in Section 3.3.

From the above discussion, we see that the assumption Ψ introduced in Definition 3.1 *weakens* the strategy choices of the environment player (Player 1). We call assumptions *sufficient* if this weakening is strong enough to allow Player 0 to win from every vertex in the cooperative winning region.

Definition 3.2. An assumption Ψ is *sufficient* for (G, Φ) if $\langle\langle 0\rangle\rangle_\Psi \Phi \supseteq \langle\langle 0,1\rangle\rangle \Phi$.

Unfortunately, sufficient assumptions can be abused to change the given synthesis problem in an unintended way. Consider for instance the game in Figure 3.1 (left) with $\Phi = \square\diamondsuit\{v_0\}$ and $\Psi = \square\diamondsuit e_1$. Here, there is no strategy π_1 for Player 1 such that $\texttt{plays}(\pi_1) \subseteq \mathcal{L}(\Psi)$ as the controller (i.e., Player 0) can always falsify the assumption by simply not choosing e_1 infinitely often from v_1. Therefore, any such Player 0 strategy is winning under assumption even if Φ is violated. The assumption Ψ, however, is trivially sufficient, as $\langle\langle 0\rangle\rangle_\Psi \Phi = V$. In order to prevent sufficient assumptions to be falsifiable and thereby enabling vacuous winning, we define the notion of *implementability*, which ensures that Ψ solely restricts Player 1 moves.

Definition 3.3. An assumption Ψ is *implementable* for (G, Φ) if $\langle\langle 1\rangle\rangle \Psi = V$.

An assumption which is sufficient and implementable ensures that the cooperative winning region of the original game coincides with the winning region under that assumption, i.e., $\langle\langle 0\rangle\rangle_\Psi \Phi = \langle\langle 0,1\rangle\rangle \Phi$. However, it does not yet ensure that all cooperative strategy choices of both players are retained, which is ensured by the notion of *permissiveness*.

Definition 3.4. An assumption Ψ is *permissive* for (G, Φ) if $\mathcal{L}(\Phi) \subseteq \mathcal{L}(\Psi)$.

As discussed before, this notion of permissiveness is motivated by the intended use of assumptions for compositional distributed synthesis. In the simplest scenario of two interacting components, two synthesis tasks—one for each process—are considered in parallel. Here, generated assumptions in one synthesis task are used as additional specifications in the other synthesis problem. Therefore, permissiveness is crucial to not "skip" over possible cooperative solutions—each synthesis task needs to keep all allowed strategy choices for both players intact to allow for compositional reasoning. This scenario is illustrated in the following example to motivate the considered class of assumptions.

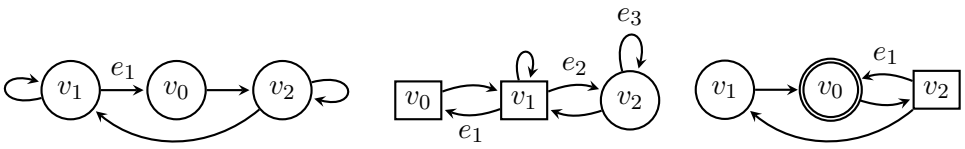


Figure 3.1: Two-player games with Player 1 (squares) and Player 0 (circles) vertices.

Example 3.1. Consider the (multi-objective) two-player game in Figure 3.1 (middle) with two different specifications for both players, namely $\Phi_0 = \Diamond\Box\{v_1, v_2\}$ and $\Phi_1 = \Diamond\Box\{v_1\}$. Now consider two candidate assumptions $\Psi_0 = \Diamond\Box\neg e_1$ and $\Psi_0' = (\Box\Diamond v_1 \Rightarrow \Box\Diamond e_2)$ on Player 1. Notice that both assumptions are sufficient and implementable for (G, Φ_0). However, Ψ_0' does not allow the play $\{v_1\}^\omega$ and hence is not permissive whereas Ψ_0 is permissive for (G, Φ_0). As a consequence, there is no way Player 1 can satisfy both her objective Φ_1 and the assumption Ψ_0' even if Player 0 cooperates, since $\mathcal{L}(\Phi_1) \cap \mathcal{L}(\Psi_0') = \emptyset$. However, under the assumption Ψ_0 on Player 1 and assumption $\Psi_1 = \Diamond\Box\neg e_3$ on Player 0 (which is sufficient and implementable for (G, Φ_1) if we interchange the vertices of the players), they can satisfy both their own objectives and the assumptions on themselves. Therefore, they can collectively satisfy both their objectives. ⌟

In summary, we are interested in assumptions that are sufficient, implementable, and permissive, which we formalize in the following definition.

Definition 3.5. An assumption Ψ is called *adequately permissive* (an APA for short) for (G, Φ) if it is sufficient, implementable and permissive.

3.1.1 Discussion on Definition 3.1

We first note some simple but interesting consequences of Definition 3.1. First, we have *anti-monotonicity*, i.e, if assumption Ψ_1 is stronger than assumption Ψ_2 (in terms of play inclusion), and π_0 is winning under Ψ_2, then it is also winning under Ψ_1. As a direct consequence of this observation, we also have *conjunctivity*, i.e., if π_0 is winning under Ψ_1 and π_0 is winning under Ψ_2, then π_0 is winning under $\Psi_1 \wedge \Psi_2$. Interestingly, however, Definition 3.1 does not allow for *disjunctivity*, i.e., if π_0 is winning under Ψ_1 and π_0 is winning under Ψ_2, then it need *not* be winning under $\Psi_1 \vee \Psi_2$. This last observation is illustrated by the following example.

Example 3.2. Consider the game graph in Figure 3.2 with the specification $\Phi = \Diamond\Box\{a\}$ (which requires the play to eventually only see vertex a). Then consider the assumptions $\Psi_1 = \neg e_0 \,\mathbf{U}\, \bigcirc e_1$ (when edge e_0 is taken for the first time, the next edge should be e_1) and $\Psi_2 = \neg e_0 \,\mathbf{U}\, \bigcirc e_2$ (when edge e_0 is taken for the first time, the next edge should be e_2). Notice that there is only one Player 1 strategy π_1, i.e., the one that never uses edge e_0, satisfying either assumption. So, any π_1-play eventually only visits vertex a, and hence, is winning. Therefore, any Player 0 strategy is winning under either assumption. In particular, consider the strategy π_0 that only uses edge e_1. Then π_0 is winning under

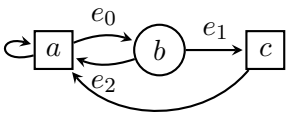


Figure 3.2: Example game graph with Player 0 (circle) and Player 1 (square) vertices illustrating non-disjunctivity of winning under assumption, as explained in Example 3.2.

assumption Ψ_i for each i. However, π_0 is not winning under $\Psi := \Psi_1 \vee \Psi_2 \equiv \texttt{true}$. To see this, note that assumption Ψ can be satisfied by any Player 1 strategy, in particular, the strategy $\widetilde{\pi}_1$ that always uses e_0 from state a. It is easy to see that the combination of $\widetilde{\pi}_1$ with π_0 yields the play $(abc)^\omega$ that satisfies Ψ but not Φ. Hence, π_0 is not winning under assumption $\Psi = \Psi_1 \vee \Psi_2$. ⌟

3.2 Computing Adequately Permissive Assumptions

In this section, we present our algorithm to compute *adequately permissive assumptions* (APA) for *parity games*, which are canonical representations of ω-regular games (as discussed in Section 2.5.2). For a gradual exposition of the topic, we first present algorithms for simpler winning conditions, namely safety (Section 3.2.1), Büchi (Section 3.2.2), and co-Büchi (Section 3.2.3), which are used as building blocks while presenting the algorithm for parity games (Section 3.2.4).

3.2.1 Unsafe edges for Safety Games

Recall that a safety game is a game $\mathcal{G} = (G, \Phi)$ with $\Phi := \square I$ for some $I \subseteq V$, and a play is winning for Φ if it never leaves I. Hence, it is not hard to see that APAs for safety games only need to disallow every Player 1 move that leaves the cooperative winning region in $\mathcal{G}$. This is formalized by *safety templates* in the following theorem that uses the cooperative winning region computed by the algorithm SOLVECOOPSAFETY from Section 2.5.3.

Theorem 3.1. *Let $\mathcal{G} = (G, \square I)$ be a safety game and $\mathsf{Win} = \textsc{solveCoopSafety}(G, I)$. Let $S = \{(u, v) \in E \mid (u \in V_1 \cap \mathsf{Win}) \wedge (v \notin \mathsf{Win})\}$ be the set of* unsafe *edges, then the safety template defined by them as*

$$\Lambda_{\textsc{unsafe}}(S) := \square \textstyle\bigwedge_{e \in S} \neg e, \tag{3.1}$$

is an APA for the game $\mathcal{G}$. We denote by $\textsc{AssumpSafe}(G, I)$ the algorithm computing S as above, which runs in time $\mathcal{O}(m + n)$, where $n = |V|$ and $m = |E|$.

Proof. First, note that as the runtime of S is dominated by the runtime of the procedure SOLVECOOPSAFETY, which runs in time $\mathcal{O}(m + n)$, the algorithm $\textsc{AssumpSafe}(G, I)$ runs in time $\mathcal{O}(m + n)$.

Now, we show that $\Psi = \Lambda_{\text{UNSAFE}}(S)$ is an APA assumption for safety games by proving sufficiency, implementability and permissiveness below.

▶ **Implementability.** As Win is the cooperative winning region, from every Player 1 vertex $u \in \mathsf{Win}$, there is a way to stay inside Win, i.e., there is an edge $(u, v) \in E$ such that $v \in \mathsf{Win}$. Hence, for every Player 1 vertex u, there is an edge e which is not unsafe, i.e., $e \notin S$. Furthermore, as S only contains Player 1 edges, it is easy to see that $\Lambda_{\text{UNSAFE}}(S)$ is implementable by Player 1, if he does not take the edges in S ever.

▶ **Sufficiency.** For sufficiency of the assumption, consider the strategy π_0 for Player 0: at a vertex $v \in \mathsf{Win}$, plays the transition that keeps the play in Win, and for other vertices, plays arbitrarily. The strategy is well-defined, since if at $v \in \mathsf{Win}$, there is no such transition, v would not be in Win, by definition. We will show that π_0 is winning under Ψ from every vertex in $\mathsf{Win} = \langle\langle 0, 1 \rangle\rangle \Box I$.

Let $v_0 \in \mathsf{Win}$. Let π_1 be an arbitrary strategy of Player 1 such that $\texttt{plays}(\pi_1) \subseteq \mathcal{L}(\Lambda_{\text{UNSAFE}}(S))$ and let $\rho = v_0 v_1 \dots$ be the unique (π_0, π_1)-play from v_0. Then $\rho \in \mathcal{L}(\Lambda_{\text{UNSAFE}}(S))$. It remains to show that $\rho \in \mathcal{L}(\Box I)$.

Suppose $\rho \notin \mathcal{L}(\Box I)$, i.e. ρ visits a vertex in $V \setminus I$. As $\mathsf{Win} \subseteq I$, ρ must also visit a vertex in $V \setminus \mathsf{Win}$, i.e., $v_i \notin \mathsf{Win}$ for some i. W.l.o.g. assume that i is the smallest such index, that is, for all $j < i$, $v_j \in \mathsf{Win}$. Hence, $v_{i-1} \in \mathsf{Win}$ but $v_i \notin \mathsf{Win}$. As π_0 only chooses edges that keep the play in Win, it must be the case that v_{i-1} is a Player 1 vertex. Then by definition, $(v_{i-1}, v_i) \in S$, which is a contradiction to the assumption that $\rho \in \mathcal{L}(\Lambda_{\text{UNSAFE}}(S))$. Hence, $\rho \in \mathcal{L}(\Box I)$.

▶ **Permissiveness.** Now for the permissiveness, let $\rho \in \mathcal{L}(\Box I)$. Suppose that $\rho \notin \mathcal{L}(\Lambda_{\text{UNSAFE}}(S))$. Then some edge $(v, v') \in S$ is taken in ρ. Then after reaching v', ρ still satisfies the safety condition. Hence, by correctness of SOLVECOOPSAFETY, $v' \in \mathsf{Win}$, but then $(v, v') \notin S$, which is a contradiction. Hence, $\rho \in \mathcal{L}(\Lambda_{\text{UNSAFE}}(S))$. □

3.2.2 Live Groups for Büchi Games

Recall that a Büchi game is a game $\mathcal{G} = (G, \Phi)$ where $\Phi = \Box\Diamond I$ for some target region $I \subseteq V$, and a play is winning for Φ if it visits the target region I infinitely often. Similar to the safety games, an APA for Büchi games would also need to disallow every Player 1 move that leaves the cooperative winning region in $\mathcal{G}$. In addition, an APA for Büchi games should also ensure that Player 0 can reach I from every vertex in the cooperative winning region. More specifically, from a subset of vertices in the cooperative winning region, where Player 0 can not move towards I without Player 1's cooperation, APA should ensure that Player 1 always eventually takes the edges towards I in order to ensure progress. This can be formalized using so-called live group templates.

Definition 3.6. Let $G = (V, E)$ be a game graph. Then a live group $H = \{e_j\}_{j \geq 0}$ is a set of edges $e_j = (s_j, t_j)$ with source vertices $src(H) := \{s_j\}_{j \geq 0}$. Given a set of live groups $\mathcal{H} = \{H_i\}_{i \geq 0}$, we define a live group template as

$$\Lambda_{\text{LIVE}}(\mathcal{H}) \coloneqq \bigwedge_{i \geq 0} \Box\Diamond src(H_i) \implies \Box\Diamond H_i. \tag{3.2}$$

Algorithm 3.1 AssumpBüchi(G, I)

Input: $G = (V = V_0 \cup V_1, E)$, Büchi objective $\Phi = \square\diamond I$, for $I \subseteq V$
Output: Assumption Ψ on Player 1

```
1: Win ← solveCoopBüchi(G, I)
2: S ← AssumpSafe(G, Win)
3: G ← G|Win, I ← I ∩ Win                ▷ All vertices are cooperatively Büchi winning
4: H ←AssumpLive(G, I)
5: return (S, H)

6: procedure AssumpLive(G, I)
7:     U ← I; H ← ∅
8:     while U ≠ V do
9:         U ← SolveReach(G, U)
10:        H ← E ∩ ((V \ U) × U)
11:        H ← H ∪ {H}
12:        U ← U ∪ src(H)
13:    return H
```

The live group template says that if some vertex from the source of a live group is visited infinitely often, then some edge from this group should be taken infinitely often. We will use this template to give the assumptions for Büchi games.

Remark 3.1. *We note that a closely related work by Chatterjee et al. [57] used* live edges *in their environment assumptions. Live edges are singleton live groups and are thereby less expressive. In particular, there are instances of Büchi games, where there is no permissive live edge assumption, but there is a permissive live group assumption*[1]*, e.g., in Figure 1.2c, the live edge assumption* $\square\diamond e_1 \wedge \square\diamond e_2$ *is sufficient but not permissive, whereas the live group assumption* $\square\diamond src(H) \implies \square\diamond H$ *with* $H = \{e_1, e_2\}$ *is an APA.*

Computing APAs for Büchi games. Before presenting the algorithm, let us highlight that an APA should not use live group templates to ensure that Player 1 *always* tries to make progress towards the target region I. It is only needed whenever Player 0 can not make progress without Player 1's cooperation, as explained in the following example.

Example 3.3. Consider the game in Figure 3.1 (right). Here marking e_1 as a live group would indeed force Player 1 to make progress towards I, making it a sufficient assumption. However, it is not permissive as the play $(v_2v_1v_0)^\omega \in \mathcal{L}(\Phi)$ but not in the language of the assumption. Here the permissive assumption would be $\Psi = \mathsf{True}$, as Player 0 doesn't need any cooperation from Player 1 to reach I. ⌟

[1] i.e., assumptions that use live group templates.

Building on the preceding definitions and intuitions, we now present Algorithm 3.1, which computes an APA for Büchi games. As discussed, unsafe edges are needed to ensure the play remains within the cooperative winning region. So, the algorithm first computes the cooperative winning region Win in Algorithm 3.1 using the procedure $\textsc{solveCoopBüchi}(G, I)$ from Section 2.5.3. Then, the safe group S is computed via $\textsc{AssumpSafe}(G, \mathsf{Win})$, which identifies all unsafe edges of Player 1 that must be avoided to stay within the cooperative winning condition. With the game now restricted to the cooperative winning region Win, the algorithm finds ways to ensure progress towards the target region I. To achieve this, the procedure $\textsc{AssumpLive}$ is used to iteratively identify the live groups necessary for reaching the target region. $\textsc{AssumpLive}$ maintains a set U of vertices from which the target region can be reached using the current live groups. In each iteration, U is updated by adding vertices from which Player 0 can reach I without any cooperation from Player 1, by solving a reachability game $\textsc{SolveReach}(G, I)$ (in Algorithm 3.1) using the procedure from Section 2.5.3. For vertices outside U, the procedure identifies live groups as the (Player 1) edges leading into U (in Algorithm 3.1). The new live group H is then added to the set of live groups, and U is further updated by including the sources of this live group (in Algorithm 3.1). This process repeats until all vertices are included in U, ensuring that from every vertex in the cooperative winning region, Player 0 can reach the target region I with the help of the computed live groups. Let us illustrate the algorithm using an example before stating its correctness.

Example 3.4. Consider the Büchi game in Figure 3.3 with target region $I = \{v_3\}$. We begin by computing the cooperative winning region $\mathsf{Win} = \{v_0, v_1, v_2, v_3, v_4\}$ using $\textsc{solveCoopBüchi}(G, I)$ (in Algorithm 3.1), as there is no way to reach I from v_5 even with cooperation. So, we mark the edge e_4 going from v_4 to v_5 as unsafe. Next, we restrict the game to Win and compute the live groups using $\textsc{AssumpLive}$. Initially, we set $U = I = \{v_3\}$. In the first iteration, Player 0 cannot reach U from any vertex (other than v_3) without any cooperation from Player 1. Hence, we identify the first live group $H_1 = \{e_2, e_3\}$, which are the edges from Player 1 vertices v_2 and v_4 to U (in Algorithm 3.1). We then update U to $U = \{v_2, v_3, v_4\}$ (in Algorithm 3.1). In the second iteration, Player 0 can reach U from v_1 without any cooperation from Player 1, as Player 0 can move from v_1 to v_2 directly. Hence, we first update U to $U = \{v_1, v_2, v_3, v_4\}$ (in Algorithm 3.1). Next, we identify the second live group $H_2 = \{e_1\}$, which is the edge from Player 1 vertex v_0 to U (in Algorithm 3.1). We then update U to $U = \{v_0, v_1, v_2, v_3, v_4\}$ (in Algorithm 3.1). Since now $U = \mathsf{Win}$, the algorithm terminates. The resulting APA

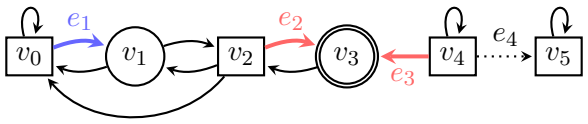


Figure 3.3: A Büchi game with Player 0 (circle), Player 1 (square) vertices, and target region $\{v_3\}$. The dotted edges are the unsafe edges and the similarly colored thick edges form live groups.

is $\Psi = \Lambda_{\text{UNSAFE}}(S) \wedge \Lambda_{\text{LIVE}}(\mathcal{H})$, where $S = \{e_4\}$, and $\mathcal{H} = \{H_1, H_2\}$ with $H_1 = \{e_2, e_3\}$ and $H_2 = \{e_1\}$. ⌟

We now state and prove the correctness of Algorithm 3.1.

Theorem 3.2. *Given a game $\mathcal{G} = (G, \Phi)$ with game graph $G = (V, E)$ and Büchi winning condition $\Phi = \square\diamond I$, Algorithm 3.1 terminates in time $\mathcal{O}(m + n)$, where $n = |V|$ and $m = |E|$. Furthermore, let $(S, \mathcal{H})$ be the output of the procedure AssumpBüchi(G, I), then $\Psi = \Lambda_{UNSAFE}(S) \wedge \Lambda_{LIVE}(\mathcal{H})$ is an APA for the game $\mathcal{G}$.*

Proof. We first show that the algorithm terminates. It is enough to show that the procedure AssumpLive terminates. Since in Algorithm 3.1, the game graph is restricted to cooperative winning region Win, we need to show that in the procedure, $U = V = \mathsf{Win}$ eventually holds. Let U_l be the value of U after the l-th iteration of AssumpLive(G, I), with $U_0 = I$. Since vertices are only added to U (and never removed) and there are only finitely many vertices, it must be the case that $U_0 \subset U_1 \subset \ldots \subset U_m = U_{m+1}$ for some $m \in \mathbb{N}$.

Since the $U_m \subseteq \mathsf{Win}$ holds trivially, we only need to show that $\mathsf{Win} \subseteq U_m$. Suppose this is not the case, i.e. $v \in \mathsf{Win} \setminus U_m$. Since $v \in \mathsf{Win}$, both players cooperatively can visit I from v. Then there is a finite path $\mathtt{h} = v_0 v_1 \cdots v_k$ with $v_k \in I$ and $v_0 = v$. But since $I = U_0 \subseteq U_m$ and $v \notin U_m$, $\mathtt{h}$ enters U_m eventually. Let l be the largest index such that $v_l \notin U_m$ but $v_{l+1} \in U_m$.

Then if $v_l \in V_0$, it would be added to U in Algorithm 3.1 of $(m+1)$-th iteration, in which case $U_m \neq U_{m+1}$. Else if $v_l \in V_1$, it would be added to U in Algorithm 3.1 of $(m+1)$-th iteration since $v_{l+1} \in U_m$, in which case $U_m \neq U_{m+1}$. In either case, we get a contradiction. Hence, $v \in U_m$, implying $\mathsf{Win} = U_m$. Hence, the procedure AssumpLive, and hence Algorithm 3.1, terminates.

Furthermore, as the procedure solveCoopBüchi takes $\mathcal{O}(m + n)$ time and the intermediate steps also take linear time, the procedure AssumpLive runs in $\mathcal{O}(m + n)$ time.

We now show that the assumption obtained is indeed an APA for the game $\mathcal{G}$, by proving implementability, sufficiency, and permissiveness.

▶ **Implementability.** As in proof of Theorem 3.1, for every Player 1 vertex, there will be edges that are not in S. Furthermore, as AssumpLive is only used on the game restricted to the cooperative winning region, the live groups $\mathcal{H}$ do not contain any unsafe edge. Hence, by choosing edges not in S, we can ensure that the $\Lambda_{\text{UNSAFE}}(S)$ is implementable by Player 1. Moreover, we note that the live groups H computed in Algorithm 3.1 contains only Player 1 edges, i.e., $src(H) \subseteq V_1$. That is because, if $v \in V_0 \cap src(H)$, then there is an edge from v to U, and hence $v \in U$ already by Algorithm 3.1. Hence, the assumption is easily implementable if Player 1 plays one of these live group edges infinitely often, when the sources are visited infinitely often.

▶ **Sufficiency.** Again, let U_l and m be as defined earlier. Define $X_l := U_l \setminus U_{l-1}$ for $1 \leq l \leq m$, and $X_0 = U_0 = I$. Then every vertex $v \in \mathsf{Win}$ is in X_l for some $l \in [0; m]$.

Consider the strategy π_0 for Player 0: at a vertex $v \in V_0 \cap X_l$, she plays the $\textsc{SolveReach}(G, U_{l-1})$ strategy to reach U_{l-1} (as these vertices are added in Algorithm 3.1), and for other vertices, she plays arbitrarily. We show that π_0 is winning under assumption Ψ from all vertices in the cooperative winning region $\mathsf{Win} = \langle\langle 0,1 \rangle\rangle \Box\Diamond I$.

Let $v_0 \in \mathsf{Win}$. Let π_1 be an arbitrary strategy of Player 1 such that $\texttt{plays}(\pi_1) \subseteq \mathcal{L}(\Psi)$, and $\rho = v_0 v_1 \dots$ be the unique (π_0, π_1)-play from v_0. Then $\rho \in \mathcal{L}(\Psi)$. It remains to show that $\rho \in \mathcal{L}(\Phi)$.

Suppose $\rho \notin \mathcal{L}(\Phi)$, i.e. $\inf(\rho) \cap I = \emptyset$. Note that ρ never leaves Win due to safety template. Let $0 \leq k \leq m$ be the least index such that $\inf(\rho) \cap X_k \neq \emptyset$. From the assumption, $k > 0$. Let $v \in \inf(\rho) \cap X_k$.

If $v \in V_0$, by the definition of π_0, every time ρ reaches v, it must reach U_{k-1}, contradicting the minimality of k. Else if $v \in V_1$, then by the definition of $\mathcal{H}$, infinitely often reaching v implies infinitely often reaching $\textsc{SolveReach}(G, U_{k-1})$. But again the play visits U_{k-1} by arguments above, giving a contradiction.

In either case, we get a contradiction, and hence, $\rho \in \mathcal{L}(\Phi)$, and $v_0 \in \langle\langle 0 \rangle\rangle_\Psi \Phi$.

▶ **Permissiveness.** Now for the permissiveness of the assumption, let $\rho \in \mathcal{L}(\Phi)$. Suppose that $\rho \notin \mathcal{L}(\Psi)$.

Case 1: If $\rho \notin \mathcal{L}(\Lambda_{\textsc{unsafe}}(S))$, then some edge $(v, v') \in S$ is taken in ρ. Then after reaching v', ρ still satisfies the Büchi condition. Hence, $v' \in \mathsf{Win} = \langle\langle 0,1 \rangle\rangle \Box\Diamond I$, but then $(v, v') \notin S$, which is a contradiction.

Case 2: If $\rho \notin \mathcal{L}(\Lambda_{\textsc{live}}(\mathcal{H}))$, then $\exists H \in \mathcal{H}$ computed in the l-th iteration such that ρ visits $src(H) \subseteq U_l \setminus \textsc{SolveReach}(G, U_{l-1})$ infinitely often, but no edge in H is taken infinitely often. Since $\rho \in \mathcal{L}(\Phi)$, it must visit $I \subseteq U_0 \subseteq U_{l-1} \subseteq \textsc{SolveReach}(G, U_{l-1})$ infinitely often. By construction, H contains all edges from $U_l \setminus \textsc{SolveReach}(G, U_{l-1})$ to $\textsc{SolveReach}(G, U_{l-1})$, and hence, ρ must take edges in H infinitely often. This is a contradiction and hence, $\rho \in \mathcal{L}(\Psi)$. □

3.2.3 Co-Live Edges for Co-Büchi Games

Recall that a co-Büchi game is a game $\mathcal{G} = (G, \Phi)$ where $\Phi = \Diamond\Box I$ for some $I \subseteq V$, and a play is winning for Φ if it eventually stays in the target region I. Hence, in addition to reaching the target region I, an APA for co-Büchi games should also ensure that Player 1 can eventually stay in I. This can be achieved by ensuring that Player 1 takes a certain set of edges only finitely often, which is formalized using so-called co-liveness templates.

Definition 3.7. Given a game graph $G = (V, E)$ and a set of *co-live* edges $D \subseteq E$, we define a *co-liveness template* as

$$\Lambda_{\textsc{colive}}(D) \coloneqq \Diamond\Box \bigwedge_{e \in D} \neg e. \tag{3.3}$$

The co-liveness template says that the co-live edges in D can be taken only finitely often. We will use this template, along with the safety templates, to construct an APA for co-Büchi games.

Algorithm 3.2 AssumpCoBüchi(G, I)

Input: $G = (V = V_0 \cup V_1, E), I \subseteq V$
Output: Assumption Ψ on Player 1

1: $\mathsf{Win} \leftarrow$ solveCoopCoBüchi(G, I)
2: $S \leftarrow$ AssumpSafe(G, Win)
3: $G \leftarrow G|_{\mathsf{Win}}, I \leftarrow I \cap \mathsf{Win}$ ▷ All vertices are cooperatively co-Büchi winning
4: $D \leftarrow$ AssumpCoLive(G, I)
5: **return** (S, D)

6: **procedure** AssumpCoLive(G, I)
7: $U \leftarrow$ solveCoopSafety(G, I) ▷ $U \subseteq I$
8: $D \leftarrow E_1 \cap (U \times V \backslash U)$
9: **while** $U \neq V$ **do**
10: $D \leftarrow D \cup (E_1 \cap (\mathsf{pre}(U) \times V \backslash U))$
11: $U \leftarrow U \cup \mathsf{pre}(U)$
12: **return** D

Computing APAs for co-Büchi games. The key idea is to compute the region U from which both players can guarantee that the play remains within the target region I. An APA is then constructed so that the safety template ensures the play stays in the cooperative winning region and the co-liveness templates ensure that the play does not go away from I infinitely often.

With this intuition, we present Algorithm 3.2, which computes an APA for co-Büchi games. Similar to Algorithm 3.1, the algorithm first computes the cooperative winning region Win in Algorithm 3.2 using the procedure solveCoopCoBüchi(G, I) from Section 2.5.3, and identifies the unsafe edges S. With the game graph restricted to Win, it then computes the co-live edges D via the procedure AssumpCoLive. As previously stated, AssumpCoLive first computes the region U from which both players can ensure the play stays in I (in Algorithm 3.2) using the procedure solveCoopSafety(G, I) from Section 2.5.3. Using this, it initializes co-live edges D with the Player 1 edges from U to outside U (in Algorithm 3.2). This ensures that Player 1 eventually prevents the play from leaving U. Then, it iteratively adds to D all Player 1 edges from the predecessors of U to outside U (in Algorithm 3.2), until no more predecessors can be added to U. These additional co-live edges ensure that Player 1 does take the play away from U infinitely often. Before proving the correctness of the algorithm, let us illustrate it using an example.

Example 3.5. Consider the co-Büchi game in Figure 3.4 with target region $I = \{v_2, v_3\}$. We begin by computing the cooperative winning region $\mathsf{Win} = \{v_0, v_1, v_2, v_3, v_4\}$ using solveCoopCoBüchi(G, I) (in Algorithm 3.2), as there is no way to reach I from v_5 even with cooperation. So, we mark the edge e_4 going from v_4 to v_5 as unsafe. Next, we restrict the game to Win and compute the co-live edges using AssumpCoLive. We first compute the region U from which both players can ensure that the play stays in I,

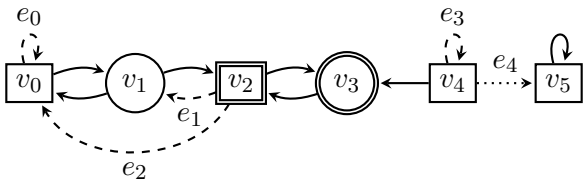


Figure 3.4: A co-Büchi game with Player 0 (circle), Player 1 (square) vertices, and target region $\{v_2, v_3\}$. The dotted edges are the unsafe edges and the dashed edges are the co-live edges.

using SOLVECOOPSAFETY(G, I) (in Algorithm 3.2). In this case, we have $U = \{v_2, v_3\}$. Next, we initialize the co-live edges $D = \{e_1, e_2\}$ with the Player 1 edges from U to outside U (in Algorithm 3.2). Then, in the first iteration of the while loop, we add to D the Player 1 edge e_3 from the predecessor v_4 of U to outside U (in Algorithm 3.2), and update U to $U = \{v_1, v_2, v_3, v_4\}$ (in Algorithm 3.2). In the second iteration, we add to D the Player 1 edge e_0 from the predecessor v_0 of U to outside U (in Algorithm 3.2), and update U to $U = \{v_0, v_1, v_2, v_3, v_4\}$ (in Algorithm 3.2). Since now $U = \mathsf{Win}$, the algorithm terminates. The resulting APA is $\Psi = \Lambda_{\text{UNSAFE}}(S) \wedge \Lambda_{\text{COLIVE}}(D)$, where $S = \{e_4\}$, and $D = \{e_0, e_1, e_2, e_3\}$. ⌟

We now state and prove the correctness of Algorithm 3.2.

Theorem 3.3. *Given a game $\mathcal{G} = (G, \Phi)$ with game graph $G = (V, E)$ and co-Büchi winning condition $\Phi = \Diamond\Box I$, Algorithm 3.2 terminates in time $\mathcal{O}(m+n)$, where $n = |V|$ and $m = |E|$. Furthermore, let (S, D) be the output of the procedure ASSUMPCOBÜCHI(G, I), then $\Psi = \Lambda_{UNSAFE}(S) \wedge \Lambda_{COLIVE}(D)$ is an APA for the game $\mathcal{G}$.*

Proof. We first show that the algorithm terminates. Since we restrict the graph to the cooperative winning region in Algorithm 3.2, We show that the procedure ASSUMPCOLIVE(G, U) terminates when all the vertices of the game graph are cooperatively winning for the co-Büchi objective $\Phi = \Diamond\Box I$. We claim that $U = V = \mathsf{Win}$, eventually.

Let U_l be the value of the variable U after l-th iteration of the while loop, with $U_0 =$ SOLVECOOPSAFETY(G, I). Since vertices are only added in U, $U_0 \subset U_1 \subset \ldots \subset U_m = U_{m+1}$ for some $m \in \mathbb{N}$. Suppose $V \not\subseteq U_m$, then there exists $v \in V \backslash U_m$. Since $v \in \mathsf{Win}$, there is a play $\rho = vv_1v_2\ldots$ from v to U_0 and stays there forever. Then consider the largest index l such that $v_l \notin U_m$, but $v_{l+1} \in U_m$. Note that this index exists because $U_0 \subseteq U_m$. But then v_l would be added to U in Algorithm 3.2 of $(m+1)$-th iteration, i.e. $U_m \neq U_{m+1}$, which is a contradiction. Hence, $v \in U_m$, implying $\mathsf{Win} = U_m$. Hence, the procedure ASSUMPCOLIVE, and hence Algorithm 3.2, terminates.

Furthermore, as the procedures SOLVECOOPCOBÜCHI takes time $\mathcal{O}(m+n)$ and the intermediate steps take linear time, the overall algorithm terminates in time $\mathcal{O}(m+n)$.

Now, we show that the assumption obtained is indeed an APA for the game $\mathcal{G}$, by proving implementability, sufficiency, and permissiveness. Again, let U_l and m be as defined earlier. Define $X_l := U_l \backslash U_{l-1}$ for $1 \leq l \leq m$, and $X_0 = U_0$. Then every vertex $v \in \mathsf{Win}$ is in X_l for some $l \in [0; m]$.

▶ **Implementability.** We again observe that the sources of the co-live edges in D are Player 1's vertices and by construction, each source has at least one alternative edge that is neither co-live nor unsafe. Hence, they can be easily implemented by Player 1, by taking those edges only finitely often, and then eventually only taking the alternative edges.

▶ **Sufficiency.** Consider the following strategy π_0 for Player 0: at a vertex $v \in X_0 \cap V_0$, she takes edge $(v, v') \in E$ such that $v' \in X_0$, at a vertex $v \in X_l \cap V_0$, for $l \in [1; m]$, she plays the $\textsc{SolveReach}(G, U_{l-1})$ strategy to reach U_{l-1}, and for all other vertices, she plays arbitrarily.

Let $v_0 \in \mathsf{Win}$. Let π_1 be an arbitrary strategy of Player 1 such that $\mathcal{L}(\pi_1) \subseteq \mathcal{L}(\Psi)$, and $\rho = v_0 v_1 \dots$ be the unique (π_0, π_1)-play from v_0. Then $\rho \in \mathcal{L}(\Psi)$. It remains to show that $\rho \in \mathcal{L}(\Phi)$.

Since $\rho \in \mathcal{L}(\Lambda_{\text{UNSAFE}}(S))$, ρ never leaves Win. Now suppose $\rho \notin \mathcal{L}(\Phi)$, i.e. $\inf(\rho) \cap (\mathsf{Win} \setminus I) \neq \emptyset$. Let $1 \leq k \leq m$ be the least index such that $\inf(\rho) \cap X_k \neq \emptyset$, and let $v \in \inf(\rho) \cap X_k$.

Consider the case $k > 1$. If $v \in V_0$, then by definition of π_0, every time v is visited, ρ must visit U_{k-1}, contradicting the minimality of k. If $v \in V_1$, by definition of co-liveness template D, Player 1 must eventually stop taking edges that does not lead to U_{k-1}, contradicting the fact that $v \in \inf(\rho)$.

Suppose $k = 1$. By previous argument, ρ also visits X_0 infinitely often. Hence, some edge from X_0 to outside X_0 must be taken infinitely often in ρ. Since π_0 never takes such edges, it must be that some edge in D (added in Algorithm 3.2) is taken infinitely often in ρ, contradicting the fact that $\rho \in \mathcal{L}(\Psi)$.

In both cases, we get a contradiction, and hence, $\rho \in \mathcal{L}(\Phi)$. So, $v_0 \in \langle\langle 0 \rangle\rangle_\Psi \Phi$.

▶ **Permissiveness.** Let $\rho = v_0 v_1 \dots$ such that $v_0 \in \mathsf{Win}$ and $\rho \in \mathcal{L}(\Phi)$. Suppose that $\rho \notin \mathcal{L}(\Psi)$.

Case 1: If $\rho \notin \Lambda_{\text{UNSAFE}}(S)$. Then the same argument as in the Büchi case gives a contradiction.

Case 2: If $\rho \notin \Lambda_{\text{COLIVE}}(D)$, that is $\exists (u, v) \in D$, such that ρ takes (u, v) infinitely often. By the definition of D, $v \in \mathsf{Win} \setminus U_0$. As U_0 is the region where both players can cooperatively stay inside I, it must be the case that ρ visits outside I infinitely often. Hence, $\rho \notin \mathcal{L}(\Phi)$, giving a contradiction. So $\rho \in \mathcal{L}(\Psi)$. □

3.2.4 Conditional Live Groups for Parity Games

Recall that a parity game $\mathcal{G} = (G, \Phi)$ is defined over a priority function $\mathbb{P} \colon V \to [0; k]$ such that $\mathbb{P}$ defines a partition $\{\mathbb{P}[i] \mid i \in [0; d]\}$ of the vertices V with $\mathbb{P}[i] = \{v \in V \mid \mathbb{P}(v) = i\}$ being the vertices with priority i. In this case, $\Phi = \mathit{Parity}(\mathbb{P})$ requires that the highest priority appearing infinitely often along a winning play to be even.

As seen in the previous sections, for games with simple winning conditions which require visiting a fixed set of edges infinitely often or only finitely often, a single assumption (conjoined with a simple safety assumption) suffices to characterize APAs, as there is just one way to win. However, in general parity games, there are usually multiple ways

Algorithm 3.3 AssumpParity$(G, \mathbb{P})$

Input: $G = (V, E)$, $\mathbb{P} : V \to \{0, 1, \ldots\}$
Output: Assumption Ψ on Player 1

```
1: Win ← solveCoopParity(G, ℙ)
2: S ← AssumpSafe(G, Win)
3: G ← G|_Win, ℙ ← ℙ|_Win
4: (ℍ, C) ← AssumpCondLive(G, ℙ)
5: D ← AssumpCoLive(G, V \ C)
6: return (S, D, ℍ)

7: procedure AssumpCondLive(G, ℙ)
8:     ℍ ← ∅; C ← ∅
9:     while G ≠ ∅ do
10:        d ← max{i | ℙ[i] ≠ ∅}
11:        if d is odd then
12:            W_¬d ← solveCoopParity(G|_{V\ℙ[d]}, ℙ)
13:            C ← C ∪ (V \ W_¬d)
14:        else
15:            W_d ← solveCoopBüchi(G, ℙ[d]), W_¬d ← V \ W_d
16:            for all i ∈_odd [0; d] do
17:                ℍ ← ℍ ∪ (W_d ∩ ℙ[i], AssumpLive(G|_{W_d}, ℙ[i+1] ∪ ℙ[i+3] ··· ∪ ℙ[d]))
18:        G ← G|_{W_¬d}, ℙ ← ℙ|_{W_¬d}
19:        ℙ[0] ← ℙ[0] ∪ ℙ[d], ℙ[d] ← ∅
20:    return (ℍ, C)
```

of winning: for example, in parity games with priorities $\{0, 1, 2\}$, a play will be winning if either (i) it only infinitely often sees vertices of priority 0, or (ii) it sees priority 1 infinitely often but also sees priority 2 infinitely often. Intuitively, winning option (i) requires the use of co-liveness assumptions as in Section 3.2.3. However, winning option (ii) actually requires the live group assumptions discussed in Section 3.2.2 to be *conditional* on whether certain states with priority 1 have actually been visited infinitely often. This is formalized by generalizing live group templates to *conditional live group templates*.

Definition 3.8. Let $G = (V, E)$ be a game graph. Then a *conditional live group* over G is a pair $(R, \mathcal{H})$, where $R \subseteq V$ and $\mathcal{H}$ is a set of live groups. Given a set of conditional live groups $\mathbb{H}$, we define a *conditional live group template* as the LTL formula

$$\Lambda_{\text{COND}}(\mathbb{H}) \coloneqq \bigwedge_{(R,\mathcal{H})\in\mathbb{H}} (\square\diamond R \implies \Lambda_{\text{LIVE}}(\mathcal{H})) \,. \tag{3.4}$$

The conditional live group template for $(R, \mathcal{H})$ says that if the condition set R is visited infinitely often, then the live group template for $\mathcal{H}$ must hold.

Computing APAs for parity games. With the generalization of live group assumptions to *conditional* live group assumptions, we actually have all the ingredients to define

an APA for parity games as a conjunction of safety, co-liveness, and conditional live group assumptions. Intuitively, we use (i) unsafe edges to prevent Player 1 to leave the cooperative winning region, (ii) co-live edges for each winning option that requires seeing a particular *odd* priority only finitely often, and (iii) a conditional live group for each winning option that requires seeing an *even* priority infinitely often if certain *odd* priority have been seen infinitely often.

With the above intuition, we present Algorithm 3.3, which computes an APA for parity games. As in Algorithms 3.1 and 3.2, the algorithm first computes the cooperative winning region Win in Algorithm 3.3 and the corresponding unsafe edges to allow Player 0 to remain within this region. Then, it uses the procedure AssumpCondLive on the game restricted to Win to compute the conditional live groups and the co-Büchi set, i.e., the region which should not be visited infinitely often in order to satisfy the parity objective.

The procedure AssumpCondLive iteratively considers the highest remaining priority d in the game. If this priority d is odd, it identifies regions $W_{\neg d}$ where cooperatively winning is possible without visiting vertices of this priority infinitely often (in Algorithm 3.3), and adds its complement to the co-Büchi set $\mathtt{C}$ (in Algorithm 3.3). If the highest priority d is even, the algorithm computes the region W_d where cooperatively winning is possible by visiting vertices of priority d infinitely often (in Algorithm 3.3). For this region, it computes conditional live group assumptions, which includes, every odd priority as the condition set with the corresponding live groups to visit higher even priorities. Then, the procedure restricts the game to the remaining region $W_{\neg d}$, and restarts from Algorithm 3.3.

Once AssumpCondLive is complete, the algorithm calls AssumpCoLive$(G, V \setminus \mathtt{C})$ in Algorithm 3.3 to compute the co-live edges to allow Player 0 to ensure not visiting $\mathtt{C}$ infinitely often. Finally, the algorithm combines the safety, co-liveness, and conditional live group assumptions to obtain an APA. Before we state the main theorem of this contribution, let us illustrate the algorithm with an example.

Example 3.6. Consider the example depicted in Figure 3.5. In Algorithm 3.3, we begin with computing the cooperative winning region Win of the entire game, to find that from vertex v_6, there is no way of satisfying the parity condition even with Player 1's cooperation, i.e., $\mathsf{Win} = \{v_0, \ldots, v_5\}$. So we mark the edge e_5 from v_5 to v_6 to be an unsafe edge, restrict the game to $G = G|_{\mathsf{Win}}$ and run AssumpCondLive on the new

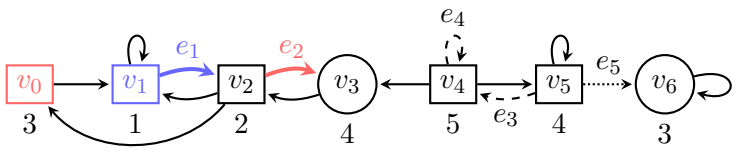


Figure 3.5: A parity game with Player 0 (circle) and Player 1 (square) vertices, where a vertex with priority i is labelled below with i. The dotted edges are the unsafe edges, the dashed edges are the co-live edges, and every colored vertices with thick similarly colored edges forms a conditional live group.

game.

In the new restricted game G, the highest priority is $d = 5$, which is odd, hence we execute Algorithm 3.3. Now a play would be winning only if eventually the play does not see v_4 any more. Hence, in Algorithm 3.3, we find the region $W_{\neg 5} = \{v_0, \ldots, v_3, v_5\}$ of the restricted graph $G|_{V \setminus \mathbb{P}[5]}$ (only containing nodes v_i with priority $\mathbb{P}(v_i) < 5$)) from where we can satisfy the parity condition without seeing v_4. We add $V \setminus W_{\neg 5} = \{v_4\}$ to the co-Büchi set $\mathtt{C}$ in Algorithm 3.3, which we want to ensure visiting only finitely many times. We compute the co-live edges for this set (in Algorithm 3.3) once we have completed the conditional live group computation.

Once we have taken care of v_4, we only need to focus on satisfying parity without visiting v_4 within $W_{\neg 5}$. This observation allows us to further restrict our computation to the game $\mathcal{G} = \mathcal{G}|_{W_{\neg 5}}$ in Algorithm 3.3 and update the priorities to only range from 0 to 4 in Algorithm 3.3. In our example, Algorithm 3.3 does not change anything. We then re-start from the beginning of the while loop in Algorithm 3.3.

In the restricted graph, the highest priority is 4 which is even, hence we execute Algorithm 3.3. One way of winning in this game is to visit $\mathbb{P}[4]$ infinitely often, so we compute the respective cooperative winning region W_4 in Algorithm 3.3. In our example we have $W_4 = W_{\neg 5} = \{v_0, \ldots, v_3, v_5\}$. Now, to ensure that from the vertices from which we can cooperatively visit $\mathbb{P}[4]$ and win, we have to make sure that every time a lower odd priority vertex is visited infinitely often, a higher priority is also visited. This can be ensured by conditional live group as computed in Algorithm 3.3. For every odd priority $i < 4$, (i.e, for $i = 1$ and $i = 3$) we have to make sure that either 2 or 4 (if $i = 1$) or 4 (if $i = 3$) is visited infinitely often. The resulting live groups $\mathbb{H}_i = (R_i, H_i^\ell)$ collect all vertices in W_4 with priority i in R_i and all live groups allowing to see even priorities j with $i < j \leq 4$ in H_i^ℓ, where the latter is computed using the procedure AssumpLive to compute live groups. The resulting live groups for $i = 1$ (blue) and $i = 3$ (red) are depicted in Figure 3.5 and given by $(\{v_1\}, \{\{e_1\}\})$ and $(\{v_0\}, \{\{e_2\}, \{e_1\}\})$, respectively.

At this point we have $W_{\neg 4} = \emptyset$. With this the game graph computed in Algorithm 3.3 becomes empty, and the procedure terminates with the current conditional live group set and the co-Büchi set $\mathtt{C} = \{v_4\}$. Finally, we compute the co-live edges needed to ensure visiting the co-Büchi set only finitely often. This is done by executing AssumpCoLive$(G, V \setminus \mathtt{C})$ in Algorithm 3.3, which computes co-live edges e_3 and e_4. In a different game graph, the reasoning done for priorities 5 and 4 above can also repeat for lower priorities if there are other parts of the game graph not contained in W_4, from where the game can be won by seeing priority 2 infinitely often. The main insight into the correctness of the outlined algorithm is that all computed assumptions can be conjoined to obtain an APA for the original parity game. ⌟

With Algorithm 3.3 in place, we can now state the main result of this contribution, i.e., the correctness of the procedure AssumpParity.

Theorem 3.4. *Given a game $\mathcal{G} = (G, \Phi)$ with game graph $G = (V, E)$ and parity winning condition $\Phi = Parity(\mathbb{P})$, Algorithm 3.3 terminates in time $\mathcal{O}(n^4)$, where $n = |V|$. Furthermore, let $(S, D, \mathbb{H})$ be the output of the procedure AssumpParity$(G, \mathbb{P})$, then $\Psi = \Lambda_{UNSAFE}(S) \wedge \Lambda_{COLIVE}(D) \wedge \Lambda_{COND}(\mathbb{H})$ is an APA for $\mathcal{G}$.*

Proof. We prove sufficiency, implementability and permissiveness below and then analyze the complexity of Algorithm 3.3. Let us write $\mathtt{C}$ to denote the set $\mathtt{C}$ obtained after the procedure AssumpCondLive is completed in Algorithm 3.3.

▶ **Implementability.** We note that by construction and by the implementability AssumpCoLive and AssumpSafe, from every vertex, there is an alternative edge that is neither unsafe nor co-live. Hence, $\Lambda_{\text{UNSAFE}}(S) \wedge \Lambda_{\text{COLIVE}}(D)$ is implementable. Again, by construction of the co-Büchi set $\mathtt{C}$, no edge in any conditional live group is unsafe or co-live. Hence, considering the live groups of the conditional live groups individually, by implementability of the live group assumptions, Player 1 can choose the live group edges (whenever the sources are visited), since they are controlled by Player 1. Therefore, the assumption Ψ is implementable.

▶ **Sufficiency.** Note that every vertex in the cooperative winning region Win either belongs to the co-Büchi set $\mathtt{C}$ or belongs to W_d in Algorithm 3.3 for some *unique* even priority d. Depending on where the play currently is, we define a Player 0 strategy π_0 as follows:

▷ For vertices in the co-Büchi set $\mathtt{C}$, by the implementability of the AssumpCoLive procedure, there is a winning strategy $\pi_0^{\mathtt{C}}$ for Player 0 under the assumption of AssumpCoLive$(G, V \setminus \mathtt{C})$ obtained in Algorithm 3.3 (in addition to the safety assumption $\Lambda_{\text{UNSAFE}}(S)$). If the play is in $\mathtt{C}$, then π_0 uses the winning strategy $\pi_0^{\mathtt{C}}$.

▷ Similarly, for vertices in W_d for some unique even priority d, we know that W_d is the cooperative winning region for $\square\diamondsuit\mathbb{P}[d]$ (in the restricted game) since $W_d =$ solveCoopBüchi$(G, \mathbb{P}[d])$ as in Algorithm 3.3. Hence, by implementability of AssumpLive, for every odd priority $i < d$, there is a winning strategy $\pi_0^{d,i}$ for Player 0 under the assumption of AssumpLive$(G|_{W_d}, \mathbb{P}[i+1] \cup \mathbb{P}[i+3] \cdots \cup \mathbb{P}[d])$ as obtained in Algorithm 3.3 (in addition to the safety assumption $\Lambda_{\text{UNSAFE}}(S)$). If the play is in W_d, then π_0 switches between the winning strategies $\pi_0^{d,i}$ for every odd priority $i < d$ in the order of increasing i, i.e., first uses $\pi_0^{d,1}$, then when the play is repeated at a vertex in W_d, it uses $\pi_0^{d,3}$, and so on.

▷ For every vertex in $V \setminus \mathsf{Win}$, π_0 plays arbitrarily.

Now, we will show that π_0 is winning under the assumption Ψ for all vertices in the cooperative winning region $\mathsf{Win} =$ solveCoopParity$(G, \mathbb{P})$. Let $v_0 \in \mathsf{Win}$. Let π_1 be an arbitrary strategy of Player 1 such that $\mathtt{plays}(\pi_1) \subseteq \mathcal{L}(\Psi)$, and let ρ be the unique (π_0, π_1)-play from v_0. It remains to show that $\rho \in \mathcal{L}(\Phi)$, i.e., the highest priority occurring infinitely often in ρ is even.

Since $\rho \in \mathcal{L}(\Lambda_{\text{UNSAFE}}(S))$, it must hold that ρ stays in the cooperative winning region Win. By the definition of $\pi_0^{\mathtt{C}}$ and since $\rho \in \mathcal{L}(\Lambda_{\text{COLIVE}}(D))$, ρ does not visit $\mathtt{C}$ infinitely often. Furthermore, by construction of the regions W_d, ρ cannot switch between different W_d regions infinitely often. Hence, ρ eventually stays in W_d for some unique even priority d. Then, by the definition of $\pi_0^{d,i}$ and since $\rho \in \mathcal{L}(\Lambda_{\text{COND}}(\mathbb{H}))$, if ρ visits a vertex with an odd priority $i < d$ infinitely often, then it also visits a vertex with an even priority $j \in \{i+1, i+2, \ldots, d\}$ infinitely often. Therefore, $\rho \in \mathcal{L}(\Phi)$.

▶ **Permissiveness.** Now for the permissiveness of the assumption, let $\rho \in \mathcal{L}(\Phi)$.

Using the similar arguments as in the proof of Theorem 3.2, we can show that $\rho \in \mathcal{L}(\Lambda_{\text{UNSAFE}}(S))$.

Furthermore, let us first show that ρ visits the co-Büchi set $\mathtt{C}$ only finitely many times, i.e., $\rho \in \mathcal{L}(\diamondsuit\square\neg\mathtt{C})$. Suppose not, then by construction, for some odd i, a vertex $v \in V \setminus W_{\neg i}$ is visited infinitely often. Note that ρ can not be winning by visiting an even priority $j > i$, since otherwise v would have been in $\text{SOLVECOOPBÜCHI}(G, \mathbb{P}[j])$ as from v we can infinitely often visit $\mathbb{P}[j]$, and hence would have been removed from $\mathcal{G}$ for the next recursive step. As ρ is winning, it does not visit $\mathbb{P}[i]$ infinitely often without visiting a higher even priority infinitely often, but then v would be in $W_{\neg d}$, which is a contradiction. Therefore, ρ visits $\mathtt{C}$ only finitely many times, and hence, is winning for the co-Büchi condition $\diamondsuit\square(V \setminus \mathtt{C})$. Then, by permissiveness of Algorithm 3.2, ρ must satisfy the assumption computed by $\text{ASSUMPCOLIVE}(G, V \setminus \mathtt{C})$. Hence, $\rho \in \mathcal{L}(\Lambda_{\text{COLIVE}}(D))$.

It only remains to show that $\rho \in \mathcal{L}(\Lambda_{\text{COND}}(\mathbb{H}))$. Suppose not, then for some even j and odd $i < j$, ρ visits $W_j \cap \mathbb{P}[i]$ infinitely often but does not satisfy the live group assumption $\text{ASSUMPLIVE}(G', \mathbb{P}[i+1] \cup \mathbb{P}[i+3] \cdots \cup \mathbb{P}[j])$, where $G' = G|_{W_j}$. Due to the construction of the set W_j, it is easy to see that once ρ visits W_j, it can never visit $V \setminus W_j$. Hence, eventually ρ stays in the game $\mathcal{G}'$ and visits $\mathbb{P}[i]$ infinitely often. Since $\rho \in \mathcal{L}(\Phi)$, it also visits some vertices of some even priority $> i$ infinitely often, and hence, it satisfies $\square\diamondsuit(\mathbb{P}[i+1] \cup \mathbb{P}[i+3] \cdots \cup \mathbb{P}[j])$ in G'. Since $\text{ASSUMPLIVE}(G', \mathbb{P}[i+1] \cup \mathbb{P}[i+3] \cdots \cup \mathbb{P}[j])$ is a permissive assumption for $(G', \square\diamondsuit(\mathbb{P}[i+1] \cup \mathbb{P}[i+3] \cdots \cup \mathbb{P}[j]))$, the play ρ must satisfy $\text{ASSUMPLIVE}(G', \mathbb{P}[i+1] \cup \mathbb{P}[i+3] \cdots \cup \mathbb{P}[j])$, which contradicts the assumption.

▶ **Complexity analysis.** The runtime of the algorithm is dominated by the runtime of ASSUMPCONDLIVE. In every iteration of the while loop in ASSUMPCONDLIVE, its runtime is dominated by the $\mathcal{O}(n)$ many calls to ASSUMPLIVE. Hence, every iteration of the while loop takes time $\mathcal{O}(n(m+n))$. As there can be at most n iterations of the while loop, the total running time of the algorithm is $\mathcal{O}(n^2(m+n)) = \mathcal{O}(n^4)$. □

3.3 Alternative Definition of Winning Under Assumptions

We continue our discussion on the alternative definition of winning under an assumption from Section 3.1.1. As mentioned earlier, both strategy-based and play-based definitions are equivalent for the class of assumptions we consider, even though they are not equivalent in general. To formalize this, we start by giving an alternative play-based formulation of winning under assumption from Definition 3.1.

Definition 3.9. Let $\mathcal{G} = ((V, E), \Phi)$ be a game and Ψ be an LTL formula over V. Then a Player 0 strategy π_0 is winning from v in $\mathcal{G}$ under assumption Ψ, if $\pi_0 \models_v (\Psi \Rightarrow \Phi)$, i.e., every play $\rho \in \mathtt{plays}_v(\pi_0)$ either fails to satisfy the assumption Ψ or satisfies the specification Φ.

It is easy to observe that a strategy π_0 that is winning under assumption by Definition 3.9 is also winning under assumption by Definition 3.1. However, the other direction is not true in general, as shown by the following example.

Example 3.7. Consider the same game as in Example 3.2, i.e., the game in Figure 3.2 with specification $\Phi = \Diamond\Box\{a\}$. Also, consider the assumption $\Psi_1 = \neg e_0 \,\mathbf{U}\, \bigcirc e_1$ as in Example 3.2. Then by the same arguments as before, the strategy π_0, which only uses edge e_1, is winning under Ψ_1 by Definition 3.1. However, note that the play $(abc)^\omega$ is a π_0-play and satisfies Ψ_1 but does not satisfy Φ. Hence, π_0 is not winning under Ψ_1 by Definition 3.9. ⌟

Interestingly, the class of assumptions we compute in this work does not allow for examples of the sort presented above. Intuitively, this is due to the fact that these assumptions are *implementable* by Player 1 and realizable by a combination of local templates. These assumptions can therefore be enforced by only restricting the moves of Player 1. This implies that for any π_0-play ρ that complies with the assumption, there does exist a strategy π_1 satisfying the assumption, which results in ρ. Hence, any strategy π_0 which is winning under assumption for by Definition 3.1 is also winning under assumption by Definition 3.9. This is formalized in the following proposition.

Proposition 3.1. *Given a parity game $\mathcal{G}$, let Ψ be an APA computed by Theorem 3.4. Then, a Player 0 strategy is winning from v under Ψ by Definition 3.1 if and only if it is winning from v under Ψ by Definition 3.9. Furthermore, this implies that for such assumptions Ψ, it holds that $\langle\!\langle 0 \rangle\!\rangle_\Psi \Phi = \langle\!\langle 0 \rangle\!\rangle(\Psi \Rightarrow \Phi)$.*

Proof. Suppose a Player 0 strategy π_0 is winning from v under assumption Ψ by Definition 3.9. Then every play $\rho \in \mathtt{plays}_v(\pi_0)$ either fails to satisfy the assumption Ψ or satisfies the specification Φ. Hence, $\mathtt{plays}_v(\pi_0) \subseteq \mathcal{L}(\Phi) \cup \mathcal{L}(\neg\Psi)$. Now, let π_1 be a Player 1 strategy s.t. $\mathtt{plays}(\pi_1) \subseteq \mathcal{L}(\Psi)$. Then, the unique (π_0, π_1)-play from v belongs to the set:

$$\begin{aligned}\mathtt{plays}_v(\pi_0) \cap \mathtt{plays}(\pi_1) &\subseteq \big(\mathcal{L}(\Phi) \cup \mathcal{L}(\neg\Psi)\big) \cap \mathcal{L}(\Psi) \\ &= \big(\mathcal{L}(\Phi) \cap \mathcal{L}(\Psi)\big) \cup \big(\mathcal{L}(\neg\Psi) \cap \mathcal{L}(\Psi)\big) \\ &= \mathcal{L}(\Phi) \cap \mathcal{L}(\Psi) \\ &\subseteq \mathcal{L}(\Phi).\end{aligned}$$

Hence, π_0 is also winning under assumption Ψ by Definition 3.1.

Now, for the other direction, suppose π_0 is winning from v under assumption Ψ by Definition 3.1. Let $\rho \in \mathtt{plays}_v(\pi_0)$. If $\rho \notin \mathcal{L}(\Psi)$, then we are done. Suppose $\rho \in \mathcal{L}(\Psi)$, then we have to show that $\rho \in \mathcal{L}(\Phi)$. We claim that there exists a Player 1 strategy π_1 such that $\mathtt{plays}(\pi_1) \subseteq \mathcal{L}(\Psi)$ and ρ is π_1-play. Then, by Definition 3.1, as ρ is the unique (π_0, π_1)-play from v, it is winning, i.e., $\rho \in \mathcal{L}(\Phi)$, and hence, we are done.

Now we only need to prove the claim. As Ψ is an implementable assumption, there exists a Player 1 strategy π_1^* such that $\mathtt{plays}(\pi_1^*) \subseteq \mathcal{L}(\Psi)$. Let $\rho = v_0 v_1 \cdots$. Now, let π_1 be another Player 1 strategy such that for every play prefix $\mathtt{h} \in V^* V_1$, it is defined as follows:

$$\pi_1(\mathtt{h}) = \begin{cases} v_k & \text{if } \mathtt{h} = v_0 v_1 \cdots v_{k-1} \\ \pi_1^*(\mathtt{h}) & \text{otherwise.} \end{cases}$$

Then, clearly, ρ is a π_1-play. Now, let $\rho' \in \texttt{plays}(\pi_1)$, then it is enough to show that $\rho' \in \mathcal{L}(\Psi)$. If $\rho' = \rho \in \mathcal{L}(\Psi)$, then we are done. Suppose not and let $\texttt{h}'$ be the maximal prefix of ρ' that is also a prefix of ρ (which can also be empty). By construction, the moves taken after the prefix $\texttt{h}'$ in ρ' are compliant with π_1^*. As the conditional live group templates and co-liveness templates are tail properties and are independent of prefixes, ρ' satisfies those templates of assumption Ψ. Furthermore, as $\texttt{h}'$ is a prefix of $\rho \in \mathcal{L}(\Psi)$ and the moves taken after $\texttt{h}'$ in ρ' are compliant with π_1^*, the play ρ' can not contain any unsafe edges marked by assumption Ψ. Therefore, $\rho' \in \mathcal{L}(\Psi)$. □

3.4 Experimental Evaluation

We have developed a C++-based prototype tool SIMPA [4] that computes **S**ufficient, **Im**plementable, and **P**ermissive **A**ssumptions for Büchi, co-Büchi, and parity games as presented in Section 3.2. We first compare SIMPA against the closest related tool GIST [57, 61] in Section 3.4.1. Subsequently, we demonstrate that SIMPA generates small and meaningful assumptions for the well-known 2-client arbiter synthesis problem given by Piterman et al. [174] in Section 3.4.2.

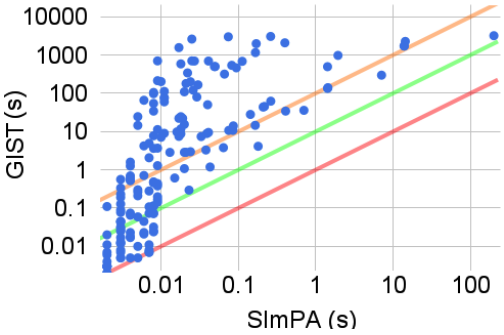


Figure 3.6: Running times of SIMPA vs GIST (in seconds, log-scale)

	SIMPA	GIST
Mean time	64.8s	1079.0s
Non-timeout mean time	64.8s	209.2s
Timeouts (1hr)	0 (0%)	59 (26%)
No assumption generated	0 (0%)	20 (9%)
Faster	230 (100%)	0 (0%)

Table 3.1: Summary of the experimental results

3.4.1 Performance Evaluation

We compare the effectiveness of our tool against a re-implementation of the closest related tool called GIST [61], which is no longer available from the authors[2]. The tool GIST originally implements the algorithm proposed by Chatterjee et al. [57] (as discussed in Remark 3.1) to compute sufficient and implementable assumptions (in the sense of Definitions 3.2 and 3.3) for parity games. The assumptions computed by GIST consist of *unsafe edges* and *live edges* (i.e., singleton live groups). Thus, the assumptions computed by GIST are not necessarily permissive. Furthermore, as live edges are not expressive enough to capture sufficient assumptions for all parity games (see Figure 1.2b for a simple

[2]The link provided in the paper is broken, and the authors informed us that the implementation is not available. Thus, we re-implemented the algorithm described in the original paper [57].

example where there is no sufficient assumption that can be expressed using live edges), GIST sometimes even fails to compute any sufficient assumption, as we will see later in this section.

We made a small modification to the termination condition of GIST for a fair comparison with our tool SImPA. The original termination condition of GIST computes assumptions only enabling a particular initial vertex to become winning. However, for the experiments, we run GIST until one of the cooperatively winning vertices is not winning anymore. Since GIST starts with a maximal assumption and keeps making it smaller until a fixed initial vertex is not winning anymore, our modification makes GIST faster as the modified termination condition is satisfied earlier. As our tool does not depend on any fixed initial vertex and the dependency on the initial vertex makes GIST slower, this modification allows a fair comparison.

We compared the performance and the quality of the assumptions computed by SImPA and GIST on a set of parity games collected from the SYNTCOMP benchmark suite [119]. For computing assumptions using both SImPA and GIST, we set a timeout of one hour. All the experiments were performed on a computer equipped with Intel (R) Core (TM) i5-10600T CPU @ 2.40GHz and 32 GiB RAM.

We provide a summary of the experimental results in Table 3.1. In addition, Figure 3.6 shows a scatter plot, where every instance of the benchmarks is depicted as a point, with the X and Y coordinates representing the running time for SImPA and GIST (in seconds), respectively. We see that SImPA is computationally much faster than GIST in every instance (all dots lie above the lower red line)—most times by one (above the middle green line) and many times even two (above the upper orange line) orders of magnitude. Moreover, GIST fails to compute any assumption in 20 instances (see the row labeled 'no assumption generated' in Table 3.1), while SImPA successfully computes APAs for all instances. Furthermore, we note that in all cases where the assumptions computed by GIST are actually APAs, SImPA computes the same assumptions orders of magnitude faster.

3.4.2 2-Client Arbiter Example

We consider the 2-client arbiter example given by Piterman et al. [174]. We show that the assumptions computed by our tool SImPA are similar but more permissive than the assumptions used by Piterman et al. [174] to render the synthesis problem realizable. Furthermore, we show that SImPA is significantly faster and computes more permissive assumptions than GIST on this example.

In this example, clients $i \in \{1, 2\}$ (Player 1) can request or free a shared resource by setting the input variables r_i to true or false, and the arbiter (Player 0) can set the output variables g_i to true or false to grant or withdraw the shared resource to/from client i. The game graph for this example is implicitly given as part of the specification (as this is a GR(1) synthesis problem, see [174] for details). We depict a relevant part of this game graph schematically in Figure 3.7. Here, rectangles and circles represent Player 1 and Player 0 vertices, respectively, and the double-lined vertices have priority 2 (are Büchi vertices), while all other vertices have priority 1. The labels of the Player 0 states indicate

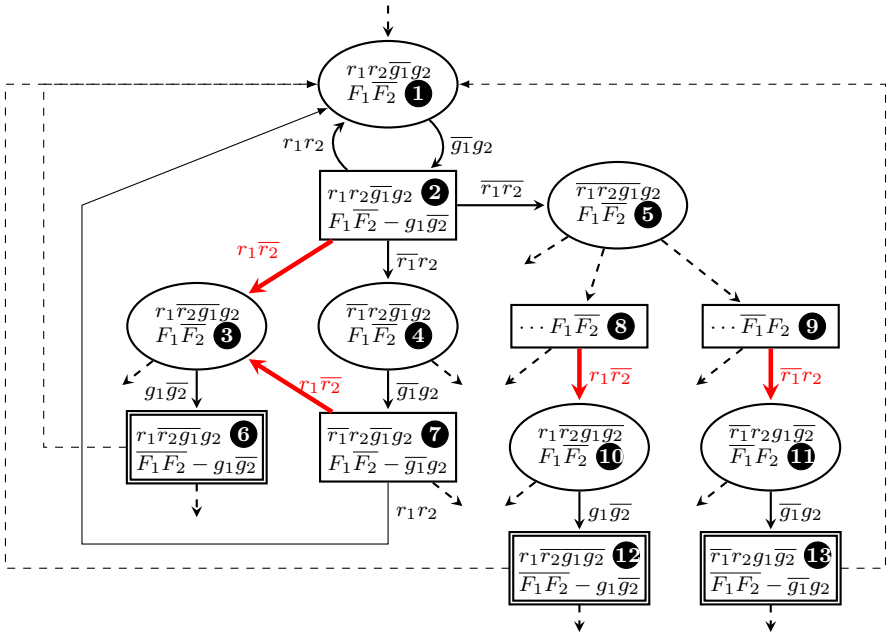


Figure 3.7: Illustration of a relevant part of the game graph for the 2-client arbiter.

the current status of the request and grant bits, and in addition, remember if a request is currently *pending*, i.e., $F_i = \overline{g_i}\ \mathsf{S}\ (r_i \wedge \overline{g_i})$, where S denotes the LTL operator "since". Labels of Player 1 vertices additionally remember the last move chosen by Player 0. We see that all vertices with no pending requests have priority 2. It is known that there does not exist a winning strategy in this game for Player 0 if the moves of the clients (Player 1) are unconstrained.

Running SIMPA on this example yields only live group assumptions (as this is a Büchi game and all vertices are cooperatively winning), which were computed in 0.01 seconds. The edges of one live group are indicated schematically by thick red arrows in Figure 3.7. We see that this live group ensures that the play eventually moves to vertices where Player 0 can force a visit to a Büchi vertex. The assumption used by Piterman et al. [174] to restrict the clients' behavior in order to render the synthesis problem realizable is given by

$$\tilde{\Psi} = \bigwedge_i (\overline{r_i} \wedge \square\big((r_i \neq g_i) \Rightarrow (r_i = \bigcirc r_i)\big) \wedge \square\big((r_i \wedge g_i) \Rightarrow \Diamond \overline{r_i}\big)\big).$$

We see that our live group assumptions are similar but more permissive. For example, when we persistently see states with label $r_1 \overline{g_1}$ and $r_2 g_2$ (e.g., cycling through states 1 and 2 in Figure 3.7), we enforce that *eventually* the edge labeled $r_1 \overline{r_2}$ needs to be taken. On the other hand, the second and third conditions in $\tilde{\Psi}$ (which are triggered by $r_1 \overline{g_1}$ and $r_2 g_2$, respectively) enforce that no other outgoing transition is allowed from state 2 except for the one labeled with $r_1 \overline{r_2}$, which is strictly more restrictive.

We have also run GIST on this example. It took 6.44 seconds to compute live edge

assumptions for unrestricted initial conditions, which is two orders of magnitude slower than SImPA. Further, in order to see ❻ infinitely often, GIST returns the live edges ❷–❸ and ❼–❶. This assumption is not permissive, as there exist winning plays that do not use either of these edges infinitely often. It turns out that an APA for this example will unavoidably require live groups—singleton live edges, as computed by GIST, will not suffice.

3.5 Related Work

To the best of our knowledge, we propose the first fully automated algorithm for computing *permissive* assumptions for general ω-regular games. We now discuss how related work addresses different aspects of this problem.

Environment Assumption Synthesis. The problem of automatically computing environment assumptions for synthesis was already addressed by Chatterjee et al. [57]. However, in general, their class of assumptions does not allow the assumptions to be *permissive* (or even *sufficient* in many cases) as discussed in Remark 3.1 and Section 3.4.1. Further, computing their assumptions is an NP-hard problem, while our algorithm computes APAs in $\mathcal{O}(n^4)$-time for a parity game with n vertices. The difference in the complexity arises because Chatterjee et al. require minimality of the assumptions. On the other hand, we trade minimality for permissiveness which allows us to utilize cooperative games, which are easier to solve.

Cooperative Multi-Objective Games. When considering cooperative solutions of multi-objective games (as in Section 2.5.4), related work either fix strategies for both players [60, 101], assume a particularly rational behavior of the environment [44] or restrict themselves to safety assumptions [145]. In contrast, we do not make any assumption on how the environment chooses its strategy.

Specification Repair. In the context of specification-repair in zerosum games, multiple automated methods for repairing environment models exist [194, 103, 104, 152, 57]. Unfortunately, all of these methods fail to provide permissive repairs. A recent work by Cavezza et al. [55] computes a minimally restrictive set of assumptions but only for GR(1) specifications, which are a strict subclass of the problem considered in our work.

Chapter 4

Negotiation Framework for Rational Players

While Chapter 3 introduced permissive assumptions for monolithic systems modeled as two-player games, we now apply these concepts to distributed systems comprising multiple components, each with their own specifications. As discussed in Section 2.5.4, such distributed systems can be modeled as k-player (multi-objective) games, where each player represents a component with its own objective. In this chapter, we consider settings where all players interact *rationally*, meaning that each player's primary goal is to satisfy their own objective and acts adversarially toward others only when doing so does not jeopardize their own objective. This notion of rational interaction in graph games was originally introduced by Chatterjee et al. [62] through the concept of *secure equilibria* (SE), a refinement of the classical Nash equilibria [161]. Intuitively, an SE is a Nash equilibrium with respect to lexicographically ordered objectives where each player first aims to satisfy their own specification and, only secondarily, seeks to falsify other players' specifications. Thus, an SE can be interpreted as a contract between the players that enforces cooperation: any unilateral selfish deviation by one player cannot disadvantage the other players if they adhere to the SE. While this property makes SE highly desirable, their main drawback is their restriction to a single strategy profile. To address this limitation, we generalize the idea of SE to *rationally contracted specifications* (RCS), a specification profile that allows each player to independently select a strategy that satisfies their specification while ensuring that any resulting strategy profile is indeed an SE.

The key insight we exploit here is that although these rationally interacting players are intrinsically selfish and prioritize achieving their own objectives, their selfish behavior may accidentally help others. We utilize APAs to over-approximate this selfish behavior of other players (as illustrated in Figure 1.3 (left)) and provide novel negotiation frameworks that iteratively refine the players' assumptions to yield an RCS, thereby offering a novel solution to distributed synthesis under rational interaction.

To this end, Section 4.1 introduces the notion of *rationally contracted specifications* (RCS) for k-player games, generalizing secure equilibria to permissive specification pro-

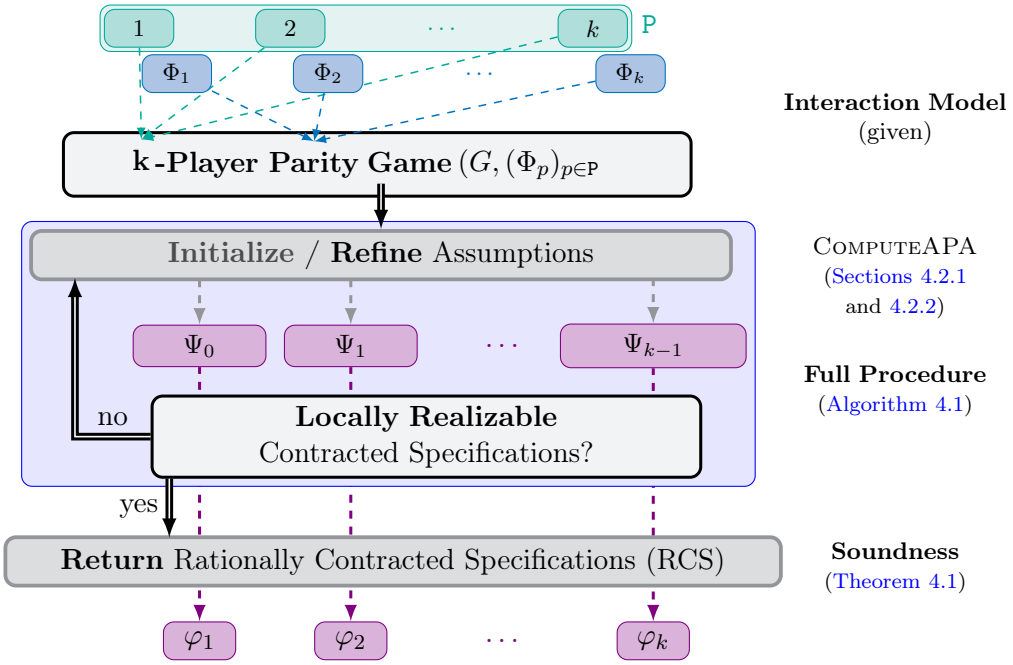


Figure 4.1: An overview of the negotiation framework for rational players.

files for rational players. Next, Section 4.2 presents a semi-algorithm (as schematically illustrated in Figure 4.1) for computing such RCSs in games with ω-regular objectives. We then propose optimized algorithms that yield a sound and efficient, though incomplete, synthesis procedure in Section 4.3. Finally, we discuss related work in Section 4.4.

4.1 Rationally Contracted Specifications (RCS)

To formalize the interaction of rational players, we use the concept of *secure equilibria* (SE) [62], which refines the classical notion of Nash equilibria [161]. Intuitively, a strategy profile is an SE if no player can improve their outcome (i.e., payoff defined with respect to the satisfaction of specifications) by unilaterally changing their strategy, where improvement is defined with respect to a preference order on outcomes. In particular, we consider the preference order where every player's primary objective is to satisfy their own specification and, as a secondary objective, to falsify the specifications of other players. We begin by formalizing secure equilibria below.

Secure Equilibria. Given a *k-player game* $\mathcal{G} = (G, (\Phi_p)_{p\in\mathtt{P}})$ and a strategy profile $(\pi_p)_{p\in\mathtt{P}}$, we define a payoff profile, denoted by $\mathtt{payoff}((\pi_p)_{p\in\mathtt{P}})$, as the tuple $(\mathtt{val}_p)_{p\in\mathtt{P}}$ such that $\mathtt{val}_p = 1$ if and only if $(\pi_p)_{p\in\mathtt{P}} \models \Phi_p$. With this, we can define a Player q

preference order $\prec_q$ on payoff profiles lexicographically, such that

$$
\begin{aligned}
&(\mathtt{val}_p)_{p\in\mathtt{P}} \prec_q (\mathtt{val}'_p)_{p\in\mathtt{P}} \text{ if and only if}\\
&- \mathtt{val}_q < \mathtt{val}'_q; \text{ or}\\
&- \mathtt{val}_q = \mathtt{val}'_q \text{ with } (\forall p \in \mathtt{P}.\mathtt{val}_p \geq \mathtt{val}'_p) \text{ and } (\exists p \in \mathtt{P}.\mathtt{val}_p > \mathtt{val}'_p).
\end{aligned}
$$

Intuitively, this preference order captures the idea that Player q prefers payoff profiles where they satisfy their own specification Φ_q over those where they do not. Among the payoff profiles where Player q satisfies their own specification, they prefer those where more of the other players' specifications are falsified.

Definition 4.1 ([62])**.** Given a *k-player game* $\mathcal{G} = (G, (\Phi_p)_{p\in\mathtt{P}})$, a strategy profile $(\pi_p)_{p\in\mathtt{P}}$ is a *secure equilibrium* (SE) if for all $p \in \mathtt{P}$, there does not exist a strategy π'_p of Player p such that $\mathtt{payoff}((\pi_p)_{p\in\mathtt{P}}) \prec_p \mathtt{payoff}(\pi'_p, \pi_{\neg p})$[1].

It is well known that every secure equilibrium is also a Nash equilibrium in the classical sense (see the original work on SE [62] for details). Within this work, we primarily consider *winning* secure equilibria (WSE), i.e., SE with the payoff profile $(\mathtt{val}_p = 1)_{p\in\mathtt{P}}$. Since WSEs have a trivial payoff profile, they can be characterized without referring to payoffs, as formalized next.

Definition 4.2. Given a k-player game $(G, (\Phi_p)_{p\in\mathtt{P}})$, a strategy profile $(\pi_p)_{p\in\mathtt{P}}$ is a *winning secure equilibrium* (WSE) if (i) $(\pi_p)_{p\in\mathtt{P}} \models \bigwedge_{p\in\mathtt{P}} \Phi_p$; and (ii) for every strategy π'_p of Player p, if $(\pi'_p, \pi_{\neg p}) \not\models \bigwedge_{q\neq p} \Phi_q$ holds, then $(\pi'_p, \pi_{\neg p}) \not\models \Phi_p$ holds.

Intuitively, (i) ensures that the strategy profile satisfies all player's objective, whereas (ii) ensures that no player can improve, i.e., falsify another player's objective without falsifying their own objective, by deviating from the prescribed strategy.

Rationally Contracted Specifications. As stated in the original work on SE [62, p.68], an SE (or a WSE) can thus be interpreted as a contract between the players which enforces cooperation: any unilateral selfish deviation by one player cannot put the other players at a disadvantage if they follow the SE. While this property makes SE very desirable, their main draw-back, as most prominently pointed out by Brenguier et al. [44], is their restriction to a *single* strategy profile. This, in combination with classical engines [100, 91] for reactive synthesis (as stated in Problem 2.2) typically preferring small and goal-oriented strategies, incentivizes "immediate punishment" of deviations from an SE strategy profile in the final implementation.

Example 4.1. To illustrate this effect, consider the game depicted in Figure 4.2, taken from the original paper on SE [62]. Here, an SE can be described as follows: if Player 1 always chooses $v_3 \to t_1$ (forming π_1) and Player 0 always chooses $v_0 \to t_0$ and $t_0 \to v_3$ (forming π_0), then both satisfy their specifications; if Player 1 deviates by choosing

[1] Recall that $\pi_{\neg p}$ represents the strategy profile $(\pi_q)_{q\in\mathtt{P}\setminus\{p\}}$, and hence, we use $(\pi'_p, \pi_{\neg p})$ to represent the strategy profile that extends $\pi_{\neg p}$ with a strategy π'_p for Player p.

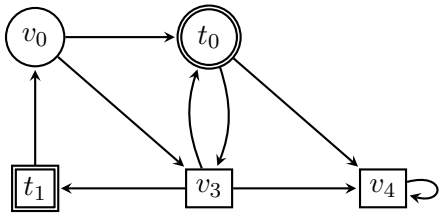


Figure 4.2: A two-player game with Player 1's vertices (squares), Player 0's vertices (circles) where Player p's specification $\Phi_p = \square\diamondsuit t_p$ is to visit t_p infinitely often.

$v_3 \to t_0$ (risking falsification of Φ_0), then Player 0 can retaliate by choosing $t_0 \to v_4$ (ensuring falsification of both specifications); similarly, if Player 0 deviates by choosing $v_0 \to v_3$ (risking falsification of Φ_1), then Player 1 can retaliate by choosing $v_3 \to v_4$ (ensuring falsification of both specifications). Clearly, the strategy profile (π_0, π_1) is an SE—in particular, a *winning* SE as both players satisfy their specifications when following it. However, since the outlined retaliating strategies (π'_0, π'_1) are also part of the final implementation generated from this SE, any play that deviates from (π_0, π_1) *only once* causes the game to end up in a loop at v_4, resulting in neither player satisfying their objectives. Intuitively, this way of implementing SE-based strategies makes components *act rationally* by ensuring that a deviation from the contract is *immediately punished*.

Having the interpretation of an SE as a contract in mind, it is however very appealing to think about the *realization* of this contract in the final implementation in a more *permissive* way. Intuitively, in the game depicted in Figure 4.2, both players can satisfy their specifications Φ_p without the help by the other player, as long as the play does not go to v_4. In particular, whenever both players independently choose a strategy π_p which ensures that they (i) never take their edge to v_4 and (ii) satisfy Φ_p for every strategy $\pi_{\neg p}$ of the other player that also never takes their edge to v_4, forms an SE strategy profile (π_0, π_1). These *minimal cooperation obligations* for an SE can be interpreted as a *specification profile* (φ_0, φ_1), s.t. $\varphi_0 := \Psi_0 \wedge (\Psi_1 \Rightarrow \Phi_0)$ and $\varphi_1 := \Psi_1 \wedge (\Psi_0 \Rightarrow \Phi_1)$, where $\Psi_0 = \Lambda_{\text{UNSAFE}}(e_{t_0 v_4})$[2] and $\Psi_1 = \Lambda_{\text{UNSAFE}}(e_{v_3 v_4})$ express the above discussed assumption that Player p does not move to v_4 from their vertex. Intuitively, Ψ_p captures the idea that Player p never takes the edge to v_4, and $(\Psi_{1-p} \Rightarrow \Phi_p)$ captures the idea that Player p satisfies their specification Φ_p assuming that the other player Player $1-p$ also never takes the edge to v_4. It turns out, that this new *specification profile* (φ_0, φ_1) has three nice properties: (i) it is *decentralized*, i.e., Player p has a strategy π_p that satisfies φ_p in a zero-sum sense, (i.e., no matter what the other player does), (ii) it is *sound*, i.e., every strategy profile (π_0, π_1), where Player p's strategy π_p satisfies φ_p (in a zero-sum sense) is a *winning* SE, and (iii) it is *maximal* meaning it does not lose any cooperative solution, i.e., $\Phi_0 \wedge \Phi_1 = \varphi_0 \wedge \varphi_1$. Conceptually, this allows us to move from *deviation-punishment* in SE-based implementations to a *distributed, assume-guarantee based realization* of SE. ⌟

Based on the insights gained from Example 4.1, we now generalize this idea of moving

[2]Recall that we use e_{uv} to denote the edge from u to v.

from a single SE strategy profile to a more permissive specification profile that allows each player to locally and fully independently pick a strategy that is winning for their specification. To formalize this, we define the notion of *rationally contracted specifications* (RCS) that collects potentially infinitely many secure equilibria into a specification profile as follows.

Definition 4.3. Given a game $(G, (\Phi_p)_{p\in\mathtt{P}})$, a specification profile $(\varphi_p)_{p\in\mathtt{P}}$ is said to be a *rationally contracted specification* (RCS) if it is both

(i) *decentralized*: it holds that $v_0 \in \langle\!\langle p \rangle\!\rangle \varphi_p$ for all $p \in \mathtt{P}$;

(ii) *sound*: any $(\pi_p)_{p\in\mathtt{P}}$ with $\pi_p \models \varphi_p$ is a WSE.

In addition, RCSs are called *maximal*[3] if $\mathcal{L}(\bigwedge_{p\in\mathtt{P}} \varphi_p) = \mathcal{L}(\bigwedge_{p\in\mathtt{P}} \Phi_p)$.

Intuitively, decentralization ensures that every player can independently realize their specification φ_p from the initial vertex v_0. Further, soundness ensures that any such local realization results in a WSE. Finally, maximality ensures that the transformation of the specifications $(\Phi_p)_{p\in\mathtt{P}}$ to an RCS $(\varphi_p)_{p\in\mathtt{P}}$ does not lose any winning play. Therefore, such RCSs allow all players to have more permissive behavior while still ensuring the rational interaction of players.

4.2 Computing RCS in Multi-Player Games

This section proposes an iterative semi-algorithm[4] of negotiation (as intuitively outlined before and illustrated in Figure 4.1) to compute RCS by utilizing the concept of adequately permissive assumptions (APA) from Chapter 3. The main idea is to over-approximate the way in which players (accidentally) help each other by letting each Player p compute an APA Ψ_p *on themself* (as illustrated in Figure 1.3 (left)). Recall that such an APA Ψ_p collects all restrictions on Player p strategies (i.e., moves they *choose themself*) s.t. the resulting play *can* be winning for Φ_p if others cooperate (somehow). It therefore over-approximates the set of all Player p strategies which could possibly form a WSE with the other players. As a consequence, the intersection $\Psi_{\neg p} = \bigwedge_{q\neq p} \Psi_q$ *under-approximates* the way in which other players (accidentally) help Player p. In order to refine this approximation, the next computation round can now use the APAs of other players when computing new local APAs.

In order to properly formalize this idea, we first need to adapt the definition of APAs for multi-player games in Section 4.2.1 and show how to refine them iteratively with the help of other players' APAs in Section 4.2.2. Finally, we present the semi-algorithm to compute RCS in Section 4.2.3.

4.2.1 APAs for Multi-Player Games

Adapting from Definitions 3.2 to 3.5, we define APAs on a player in a multi-player game as follows.

[3] We have not included maximality into the definition of RCS as we will later see in Chapter 5 that there exist contracted specifications which are not maximal.

[4] A semi-algorithm is an algorithm that is not guaranteed to halt on all inputs.

Definition 4.4. Given a k-player game graph $G = (V, E, v_0)$ and a specification Φ, we say that an assumption Ψ_p is an *adequately permissive (APA)* on Player p for Φ if it is

(i) *sufficient*: $\langle\langle \mathtt{P}_{\neg p} \rangle\rangle (\Psi_p \Rightarrow \Phi) = \langle\langle \mathtt{P} \rangle\rangle \Phi$

(ii) *implementable*: $\langle\langle p \rangle\rangle \Psi_p = V$; and

(iii) *permissive*: $\mathcal{L}(\Psi_p) \supseteq \mathcal{L}(\Phi)$.

Note that, due to equivalence between Definition 3.1 and Definition 3.9 as discussed in Section 3.3, the sufficiency condition here indeed captures the generalization of the original sufficiency definition in Definition 3.2 to multi-player games. Here, the intuition behind an APA is that even if a player can not realize a specification Φ, they should at least satisfy an APA on them as it will allow them to realize Φ if the other players are willing to help (sufficiency). Further, such a behavior by Player p does not prevent any WSE (permissiveness), and Player p can individually choose to satisfy an APA (implementability).

It is easy to see that computing such an APA on Player p can be reduced to computing APA on Player 1 in a 2-player game by considering Player p as Player 1 and all other players as Player 0. We can therefore use Theorem 3.4 from Chapter 3 to compute APAs for two-player parity games in polynomial time.

Corollary 4.1. *Given a k-player game graph $G = (V, E, v_0)$ and a parity specification $\Phi = \mathit{Parity}(\mathbb{P})$, an APA on Player p for Φ can be computed in time $\mathcal{O}(|V|^4)$. We write $\textsc{ComputeAPA}(G, \Phi, p)$ to denote the procedure that returns this APA.*

Remark 4.1. *We note that Corollary 4.1 also provides a method to compute APAs for games with LTL- or ω-regular specifications as such games can be converted into parity games (possibly with an extended game graph) by standard methods (as discussed in Section 2.5.2). Therefore, with a slight abuse of notation, we will also call the procedure $\textsc{ComputeAPA}(G, \Phi, p)$ even if Φ is not a parity specification, with the understanding that the game is always converted into a parity game first. This might incur an exponential blowup of the state space. As we call $\textsc{ComputeAPA}$ repeatedly to compute RCS, this blowup might cause non-termination (see Section 4.2.6 for details). In order to obtain a (non-optimal but) terminating algorithm for RCS computation, we will mitigate this blowup later in Section 4.3.*

4.2.2 Iterative Refinement of APAs

With the above results, we can use the algorithm ComputeAPA on a game $(G, (\Phi_p)_{p \in \mathtt{P}})$ to compute APAs for each player, i.e., $\Psi_p := \textsc{ComputeAPA}(G, \Phi_p, p)$. As outlined previously, we will iteratively refine these computed APAs (cf. Figure 4.1 (upper middle)) to finally compute the RCS. In order to do so, we want to condition the computation of the next-round APA Ψ'_p on the previous-round APAs of all other players $\Psi_{\neg p}$, as any secure strategy of players in $\mathtt{P}_{\neg p}$ is incentivized to comply with $\Psi_{\neg p}$. The most intuitive method to do this is to simply consider $\Psi_{\neg p} \Rightarrow \Phi_p$ as the specification for APA computation

in the next round. However, the way sufficiency is formulated for APAs prevents this approach, as the implication $\Psi_{\neg p} \Rightarrow \Phi_p$ is true if $\Psi_{\neg p}$ is false. As there obviously exists a strategy profile $\pi_{\neg p}$ which violates $\Psi_{\neg p}$, the sufficiency condition becomes meaningless for this specification.

However, as we know that $\Psi_{\neg p}$ are APAs, their implementability ensures that Player p can neither enforce nor falsify them. Therefore, a new specification $\Phi'_p := \Psi_{\neg p} \wedge \Phi_p$ still puts all the burden of satisfying $\Psi_{\neg p}$ to players in $\mathtt{P}_{\neg p}$ and hence, implicitly constrains the choices of $\mathtt{P}_{\neg p}$ to strategies complying with $\Psi_{\neg p}$ for sufficiency of the new APA. However, using $\Phi'_p := \Psi_{\neg p} \wedge \Phi_p$ indeed weakens the permissiveness requirement $\mathcal{L}(\Psi'_p) \supseteq \mathcal{L}(\Phi \wedge \Psi_{\neg p})$, i.e., the new APA Ψ'_p needs to be more general than the specification Φ, only when the assumption $\Psi_{\neg p}$ holds. With these refined conditions for sufficiency and permissiveness, it becomes evident that an APA for specification Φ under assumption $\Psi_{\neg p}$ is equivalent to an APA for the modified specification $\Psi_{\neg p} \wedge \Phi$, as formalized below.

Definition 4.5. Given a k-player game, a specification Φ_p and an assumption $\Psi_{\neg p}$, we say that the specification Ψ_p is an *APA on Player p for Φ_p under $\Psi_{\neg p}$* if it is an APA on Player p for specification $\Psi_{\neg p} \wedge \Phi$.

Following Remark 4.1, we denote by ComputeAPA$(G, \Psi_{\neg p} \wedge \Phi, p)$ the algorithm which computes *APAs* on Player p for Φ *under assumptions* $\Psi_{\neg p}$, even though $\Psi_{\neg p} \wedge \Phi$ is typically not a parity specification over G anymore.

4.2.3 Computing RCS

Using all the intuition discussed before, we now give a semi-algorithm of negotiation in Algorithm 4.1 (as schematically illustrated in Figure 4.1) to compute RCS for k-player games with ω-regular specifications for all players. The main idea is to iteratively compute assumptions $(\Psi_p)_{p \in \mathtt{P}}$ on every player and check if they are stable enough so that every player can satisfy their actual specification Φ_p under the assumption $\Psi_{\neg p}$. If not, then, in the next iteration, we compute new assumptions $(\Psi'_p)_{p \in \mathtt{P}}$ that are stricter than earlier ones, i.e., $\mathcal{L}(\Psi'_p) \subseteq \mathcal{L}(\Psi_p)$ but still more general than their specifications under the earlier assumption, i.e., $\mathcal{L}(\Psi'_p) \supseteq \mathcal{L}(\Psi_{\neg p} \wedge \Phi_p)$.

More specifically, we start with $\Psi_p = \mathsf{True}$ for each $p \in \mathtt{P}$ in the first iteration (as in Algorithm 4.1 and cf. Figure 4.1 (upper middle)), and then in every iteration, we want each player to satisfy $\varphi_p = \Psi_p \wedge (\Psi_{\neg p} \Rightarrow \Phi_p)$ (computed in Algorithm 4.1) *by themselves*, i.e., always satisfy their assumption Ψ_p and satisfy specification Φ_p whenever others satisfy their assumptions $\Psi_{\neg p}$. Note that, in this part of the algorithm it is correct to use this implication-style specification, as it is used for solving a *zero-sum 2-player game* between Player p and its opponent (i.e., the collection of all other players in $\mathtt{P}_{\neg p}$) for the specification φ_p. The winning regions $\langle\!\langle p \rangle\!\rangle \varphi_p$ for each such zero-sum 2-player game are then intersected in Algorithm 4.1 to obtain the winning region that is achievable by any strategy profile $(\pi_p)_{p \in \mathtt{P}}$ where π_p is a winning strategy of Player p w.r.t. φ_p (in a zero-sum sense). If this resulting winning region contains the initial vertex, then each player can realize their specification φ_p by themselves from the initial vertex (Figure 4.1

Algorithm 4.1 ComputeRCS($\mathcal{G}$)

Input: A k-player multi-objective parity game $\mathcal{G} = (G = (V, E, v_0), (\Phi_p)_{p\in\mathtt{P}})$.
Output: A rationally contracted specification $(\varphi_p)_{p\in\mathtt{P}}$.

1: $\Psi_p \leftarrow \mathsf{True}\ \forall p \in \mathtt{P}$
2: **return** RecursiveRCS$(\mathcal{G}, (\Psi_p)_{p\in\mathtt{P}})$

3: **procedure** RecursiveRCS$(\mathcal{G}, (\Psi_p)_{p\in\mathtt{P}})$
4: $\quad\Psi_{\neg p} \leftarrow \bigwedge_{q\neq p} \Psi_q\ \forall p \in \mathtt{P}$
5: $\quad\varphi_p \leftarrow \Psi_p \wedge (\Psi_{\neg p} \Rightarrow \Phi_p)\ \forall p \in \mathtt{P}$
6: $\quad$**if** $v_0 \in \bigcap_{p\in\mathtt{P}} \langle\!\langle p \rangle\!\rangle \varphi_p$ **then**
7: $\quad\quad$**return** $(\varphi_p)_{p\in\mathtt{P}}$
8: $\quad\Psi'_p \leftarrow \Psi_p \wedge$ ComputeAPA$(G, \Psi_{\neg p} \wedge \Phi_p, p)\ \forall p \in \mathtt{P}$
9: $\quad$**return** RecursiveRCS$(\mathcal{G}, (\Psi'_p)_{p\in\mathtt{P}})$

(lower middle)). Hence, we return the specification $(\varphi_p)_{p\in\mathtt{P}}$ (as in Algorithm 4.1 and cf. Figure 4.1 (bottom)), which is proven to indeed be a maximal RCS in Theorem 4.1.

If this is not the case, we keep on strengthening APAs, as discussed in Section 4.2.2, to make the above mentioned zero-sum 2-player games easier to solve (as they can rely on tighter assumptions now). Hence, we call ComputeAPA with the modified specifications $\Phi'_p := \Psi_{\neg p} \wedge \Phi_p$ for all players (Algorithm 4.1).

Example 4.2. Before proving the correctness of the (semi) Algorithm 4.1, let us first illustrate the steps using an example depicted in Figure 4.3. In Algorithm 4.1, we begin with $\Psi_0 = \Psi_1 = \mathsf{True}$ and run the recursive procedure RecursiveRCS in Algorithm 4.1.

Within the first iteration of RecursiveRCS, in Algorithm 4.1, we set $\varphi_p = \Phi_p$ as $\Psi_p = \mathsf{True}$ for all $p \in [0; 1]$. Then, in Algorithm 4.1, we check whether each player can satisfy $\varphi_p = \Phi_p$ without cooperation (i.e., in a zero-sum sense), from the initial vertex v_0. As no player can ensure that, we move to Algorithm 4.1. Here, as $\Psi_p = \mathsf{True}$ for $p \in [0; 1]$, the new assumptions Ψ'_p is an APA computed by ComputeAPA(G, Φ_p, p). This gives us $\Psi'_0 = \Diamond\Box\neg e_{00}$ and $\Psi'_1 = \Box\neg(e_{12} \wedge e_{34}) \wedge \Diamond\Box\neg e_{10}$. Intuitively, Ψ'_1 ensures that edges, i.e., $v_1 \to v_2$ and $v_3 \to v_4$, leading to the region from which it is not possible to satisfy Φ_1 are never taken; and the edge, i.e., $v_1 \to v_0$, restricting the play to progress

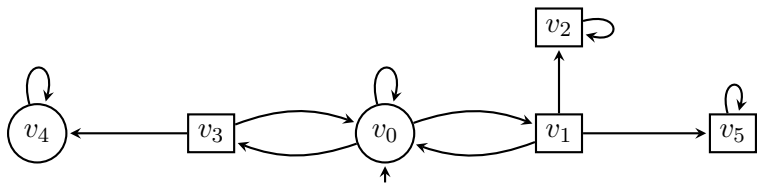


Figure 4.3: A two-player game with initial vertex v_0, Player 1's vertices (squares), Player 0's vertices (circles), edges $e_{ij} = (v_i, v_j)$, and specifications $\Phi_0 = \Diamond\Box\{v_4, v_5\}$ and $\Phi_1 = \Diamond\Box\{v_5\}$.

towards target vertex v_5 (as in Φ_1) is eventually not taken. Similarly, Ψ_0 ensures that the edge $v_0 \to v_0$ is eventually not taken that ensures progress towards Φ_0's target vertices $\{v_4, v_5\}$.

In the second iteration, we again compute the new potential RCS (φ_1, φ_0) with $\varphi_p = \Psi_p \wedge (\Psi_{\neg p} \Rightarrow \Phi_p)$ in Algorithm 4.1. In Algorithm 4.1, we find that $v_0 \notin \langle\langle 1 \rangle\rangle \varphi_1$. That is because Player 1 cannot ensure satisfying Φ_1 even when Player 0 satisfies Ψ_0 as Player 0 can always use edge $v_0 \to v_3$ leading to the play $(v_0 v_3)^\omega \not\models \Phi_1$. Hence, in Algorithm 4.1, the APA under Ψ_1 gives a more restricted assumption on Player 0: $\Psi'_0 = \Diamond\Box\neg(e_{00} \wedge e_{03})$. As the assumption Ψ_0 on Player 0 was very weak, the APA on Player 1 under Ψ_0 results in the same assumption as Ψ_1, and hence, $\Psi'_1 = \Psi_1$. Then, we move to the third iteration.

In this iteration, we find that both players can indeed satisfy their new specification φ_p from the initial vertex in Algorithm 4.1. Hence, we finally return an RCS (φ_0, φ_1) with $\varphi_p = \Psi_p \wedge (\Psi_{\neg p} \Rightarrow \Phi_p)$ where $\Psi_0 = \Diamond\Box\neg(e_{00} \wedge e_{03})$ and $\Psi_1 = \Box\neg(e_{12} \wedge e_{34}) \wedge \Diamond\Box\neg(e_{10})$. ⌟

Remark 4.2. *Let us remark that for the game depicted in Figure 4.3, the closely related approach of assume-admissible (AA) synthesis [44] has no solution. AA-synthesis utilizes a different, incomparable definition of rationality based on a dominance order. In their framework, a Player p strategy π_p is said to be* dominated *by π'_p if the set of strategy profiles that π' is winning against (i.e., satisfies Player p's specification) is strictly larger than that of π. A strategy not dominated by any other strategy is called* admissible. *In AA-synthesis, one needs to find an admissible strategy π_p for Player p such that for every admissible strategy $\pi'_{\neg p}$ for the other player, $(\pi_p, \pi'_{\neg p}) \models \Phi_p$. In this example, Player 1 has only one admissible strategy π_1 that always uses $v_1 \to v_5$ and $v_3 \to v_0$. However, with the admissible strategy π'_0 of Player 0 that always uses $v_0 \to v_3$, we have $(\pi_1, \pi'_0) \not\models \Phi_1$.*

The next theorem shows that Algorithm 4.1 indeed computes a maximal RCS if it terminates.

Theorem 4.1. *Given a k-player game $\mathcal{G}$ with game graph $G = (V, E, v_0)$ and parity specifications $(\Phi_p)_{p \in \mathtt{P}}$, if the algorithm* ComputeRCS*$(\mathcal{G})$ terminates, then it outputs a* maximal RCS $(\varphi_p)_{p \in \mathtt{P}}$.

Proof. First, let us claim that in every iteration of RecursiveRCS, for all $p \in \mathtt{P}$:

$$\text{(claim 1)} \quad \mathcal{L}(\Psi_p) \supseteq \mathcal{L}(\textstyle\bigwedge_{q \in \mathtt{P}} \Phi_q), \text{ and}$$
$$\text{(claim 2)} \quad \mathcal{L}(\Psi_p) \supseteq \mathcal{L}(\Psi_{\neg p} \wedge \Phi_p).$$

We will prove the claim using induction on the number of iterative calls to RecursiveRCS. For the base case, observe $\Psi_p = \mathsf{True}$ for all $p \in \mathtt{P}$, hence, the claim holds trivially. For the induction step, assume that claim 1+2 hold in the n-th iteration. Then, for all $p \in \mathtt{P}$, as Ψ'_p (computed in Algorithm 4.1) is Ψ in the next iteration, it suffices to show that $\mathcal{L}(\Psi'_p) \supseteq \mathcal{L}(\bigwedge_{q \in \mathtt{P}} \Phi_q)$ and $\mathcal{L}(\Psi'_p) \supseteq \mathcal{L}(\Psi'_{\neg p} \wedge \Phi_p)$.

By permissiveness of APA (as in Definition 4.4), for all $p \in \mathtt{P}$, we have

$$\mathcal{L}(\text{ComputeAPA}(G, \Psi_{\neg p} \wedge \Phi_p, p)) \supseteq \mathcal{L}(\Psi_{\neg p}) \cap \mathcal{L}(\Phi_p).$$

Hence, by Algorithm 4.1, for all $p \in \mathtt{P}$, we have $\mathcal{L}(\Psi'_p) \supseteq \mathcal{L}(\Psi_p) \cap \mathcal{L}(\Psi_{\neg p}) \cap \mathcal{L}(\Phi_p) = \left(\bigcap_{q \in \mathtt{P}} \mathcal{L}(\Psi_q)\right) \cap \mathcal{L}(\Phi_p)$, and hence, by claim 1, $\mathcal{L}(\Psi'_p) \supseteq \mathcal{L}(\bigwedge_{q \in \mathtt{P}} \Phi_q)$.

Similarly, for all $p \in \mathtt{P}$, as $\mathcal{L}(\Psi'_p) \supseteq \mathcal{L}(\Psi_p) \cap \mathcal{L}(\Psi_{\neg p}) \cap \mathcal{L}(\Phi_p)$, by claim 2, we also have $\mathcal{L}(\Psi'_p) \supseteq \mathcal{L}(\Psi_{\neg p}) \cap \mathcal{L}(\Phi_p)$. Furthermore, by Algorithm 4.1, for all $q \in \mathtt{P}$, we have $\mathcal{L}(\Psi_q) \supseteq \mathcal{L}(\Psi'_q)$, and hence, $\mathcal{L}(\Psi_{\neg p}) = \bigcap_{q \neq p} \mathcal{L}(\Psi_q) \supseteq \bigcap_{q \neq p} \mathcal{L}(\Psi'_q) = \mathcal{L}(\Psi'_{\neg p})$. Therefore, for all $p \in \mathtt{P}$, we have $\mathcal{L}(\Psi'_p) \supseteq \mathcal{L}(\Psi'_{\neg p}) \cap \mathcal{L}(\Phi_p) = \mathcal{L}(\Psi'_{\neg p} \wedge \Phi_p)$.

Now, we first show that the tuple $(\varphi^*_p)_{p \in \mathtt{P}}$ is indeed a maximal RCS by proving it is maximal, decentralized, and sound, as per Definition 4.3.

▶ **Maximality:** By construction, $\varphi^*_p = \Psi_p \wedge (\Psi_{\neg p} \Rightarrow \Phi_p)$ for the specifications $(\Psi_p)_{p \in \mathtt{P}}$ computed in last iteration. Hence, it holds that

$$\mathcal{L}\left(\bigwedge_{p \in \mathtt{P}} \varphi^*_p\right) = \bigcap_{p \in \mathtt{P}} \mathcal{L}(\Psi_p \wedge (\Psi_{\neg p} \Rightarrow \Phi_p)) = \bigcap_{p \in \mathtt{P}} \mathcal{L}(\Psi_p) \cap \bigcap_{p \in \mathtt{P}} \mathcal{L}(\Psi_{\neg p} \Rightarrow \Phi_p)$$

$$= \bigcap_{p \in \mathtt{P}} \mathcal{L}(\Psi_{\neg p}) \cap \bigcap_{p \in \mathtt{P}} \mathcal{L}(\Psi_{\neg p} \Rightarrow \Phi_p) = \bigcap_{p \in \mathtt{P}} \mathcal{L}(\Psi_{\neg p} \wedge (\Psi_{\neg p} \Rightarrow \Phi_p)) \subseteq \bigcap_{p \in \mathtt{P}} \mathcal{L}(\Phi_p) = \mathcal{L}\left(\bigwedge_{p \in \mathtt{P}} \Phi_p\right).$$

For the other direction, it holds that

$$\mathcal{L}(\varphi^*_p) = \mathcal{L}(\Psi_p) \wedge \mathcal{L}(\Psi_{\neg p} \Rightarrow \Phi_p) \supseteq \mathcal{L}(\Psi_p) \cap \mathcal{L}(\Phi_p) \tag{4.1}$$

Then, by claim 1, for all $p \in \mathtt{P}$, we have $\mathcal{L}(\varphi^*_p) \supseteq \mathcal{L}\left(\bigwedge_{p \in \mathtt{P}} \Phi_p\right)$, and hence, $\mathcal{L}\left(\bigwedge_{p \in \mathtt{P}} \varphi^*_p\right) \supseteq \mathcal{L}\left(\bigwedge_{p \in \mathtt{P}} \Phi_p\right)$. Therefore, $(\varphi^*_p)_{p \in \mathtt{P}}$ is maximal.

▶ **Decentralization:** Holds trivially by Algorithm 4.1 of Algorithm 4.1.

▶ **Soundness:** Let $(\pi_p)_{p \in \mathtt{P}}$ be a strategy profile with $\pi_p \models \varphi^*_p$. Then, every $(\pi_p)_{p \in \mathtt{P}}$-play from v_0 satisfies φ^*_p for all $p \in \mathtt{P}$, and hence, $(\pi_p)_{p \in \mathtt{P}} \models \bigwedge_{p \in \mathtt{P}} \varphi^*_p$. So, by maximality, we have $(\pi_p)_{p \in \mathtt{P}} \models \bigwedge_{p \in \mathtt{P}} \Phi_p$.

Now, to prove (ii) of Definition 4.2 (WSE), let π'_p be a strategy of Player p, and let ρ be the $(\pi'_p, \pi_{\neg p})$-play from v_0. As before, for all $q \in \mathtt{P}$, we have $\varphi^*_q = \Psi_q \wedge (\Psi_{\neg q} \Rightarrow \Phi_q)$. So, for every $q \neq p$, $\rho \in \mathcal{L}(\varphi^*_q) \subseteq \mathcal{L}(\Psi_q)$. Hence, we have $\rho \in \bigcap_{q \neq p} \mathcal{L}(\Psi_q) = \mathcal{L}(\Psi_{\neg p})$.

Now, if $\rho \in \mathcal{L}(\Phi_p)$, then $\rho \in \mathcal{L}(\Psi_{\neg p} \wedge \Phi_p)$. Then, by (4.1) and claim 2, we have $\rho \in \mathcal{L}(\varphi^*_p)$. Furthermore, as $\pi_{\neg p} \models \varphi^*_{\neg p}$, we have $\rho \in \mathcal{L}(\varphi^*_{\neg p})$. Therefore, $\rho \in \mathcal{L}(\varphi^*_p \wedge \varphi^*_{\neg p})$, and by maximality, $\rho \in \mathcal{L}(\Phi_p \wedge \bigwedge_{q \neq p} \Phi_q) \subseteq \mathcal{L}(\bigwedge_{q \neq p} \Phi_q)$. Then, by contraposition, (ii) of Definition 4.2 holds for $(\pi_p)_{p \in \mathtt{P}}$. Hence, $(\pi_p)_{p \in \mathtt{P}}$ is an SE, and hence, $(\varphi^*_p)_{p \in \mathtt{P}}$ is sound. □

4.2.4 Games with an Environment Player

Up to this point, we have only considered games played between k rational players, each representing a distinct component of the distributed system. However, in the context of reactive synthesis problems (as given in Problem 2.2), a different setup is often encountered. Here, the controller players play against an adversarial environment player. Consequently, the controller players must fulfill their objectives against all possible strategies employed by the environment player.

Interestingly, this framework can be seen as equivalent to a $(k+1)$-player game with the original k controller players $[0;k-1]$ and another k-th player, representing the environment. For this new player, the objective is simply $\Phi_k = \mathsf{True}$. Then, it is easy to see that an APA for such a specification Φ_k under any assumption is True. Hence, in each iteration of RecursiveRCS in Algorithm 4.1, the associated assumption Ψ_k is also True, and thus, $\varphi_k = \mathsf{True} \wedge ((\bigwedge_{p\in[0;k-1]} \Psi_p) \Rightarrow \mathsf{True}) \equiv \mathsf{True}$. Consequently, if ComputeRCS yields a maximal RCS $(\varphi_p^*)_{p\in[0;k]}$, the new objective of the environment player—$\varphi_k^* = \mathsf{True}$—doesn't impose any constraints on the environment's actions. Therefore, the tuple $(\varphi_p^*)_{p\in[0;k-1]}$ remains sound (as in Definition 4.3) for the k controller players because the environment player can never violate its new specification φ_k. In sum, games featuring an environment player can be effectively handled as a special case, as formally summarized below:

Corollary 4.2. *Let $G = (V, E)$ be a game graph with k controller players, i.e., $\mathtt{P} = [0;k-1]$, and an environment player $\mathtt{env}$ such that $V = \left(\bigcup_{p\in\mathtt{P}} V_p\right) \dot\cup\, V_{\mathtt{env}}$. Let $(\Phi_p)_{p\in\mathtt{P}}$ be the tuple of specifications, one for each controller player. Then, a tuple $(\varphi_p)_{p\in\mathtt{P}}$ is a maximal RCS for $(G, (\Phi_p)_{p\in\mathtt{P}})$ if and only if $(\varphi_p)_{p\in[0;k]}$ with $\varphi_k = \mathsf{True}$ is a maximal RCS for the $k+1$-player game $(G, (\Phi_p)_{p\in[0;k]})$ with $\Phi_k = \mathsf{True}$.*

Furthermore, in synthesis problems, the choices of the environment are sometimes restricted based on a certain assumption $\Phi_{\mathtt{env}}$. In such scenarios, a viable approach involves updating each controller player's specification Φ_p to $\Phi_{\mathtt{env}} \Rightarrow \Phi_p$ and subsequently utilizing Corollary 4.2 to compute a maximal RCS. An alternative approach is to consider a $(k+1)$-player game with specification $\Phi_k = \Phi_{\mathtt{env}}$ for the k-th player. With this approach, the solution becomes more meaningful, as any strategy profile for the controller players satisfying the resulting RCS allows the environment to satisfy its own assumptions $\Phi_{\mathtt{env}}$. This approach nicely complements existing work [65, 148] that aim to synthesize strategies for the controller player while allowing the environment to fulfill its own requirement.

4.2.5 Partially Winning RCS

In the preceding sections, we have presented a method for computing *winning* SE, i.e., equilibria where all players satisfy their objectives. However, it is worth noting that in certain scenarios, winning SE might not exist (see e.g. the paper on SE [62] for a detailed discussion). In such cases, a subset $\mathtt{P}'$ of players can still form a coalition, which serves their interests by enabling them to compute a maximal RCS for their coalition only, while treating the remaining players in $\mathtt{P} \setminus \mathtt{P}'$ as part of the adversarial environment. This can be accomplished by computing a maximal RCS with updated specifications denoted as $(\Phi'_p)_{p\in\mathtt{P}}$, wherein $\Phi'_p = \Phi_p$ for all $p \in \mathtt{P}'$ and $\Phi'_p = \mathsf{True}$ for all $p \notin \mathtt{P}'$. This scenario aligns with the concept of considering an adversarial environment from Section 4.2.4.

It is important to emphasize that for instances where no winning SE exists, there might not even exist a *unique maximal* outcome for which an SE is feasible, see [62, Sec. 5] for a simple example. As a result, there may be multiple coalitions that can offer

different advantages to individual players from the initial vertex. This scenario presents an intriguing, unexplored challenge for future research.

4.2.6 Computational Tractability and Termination

While Algorithm 4.1 has multiple desirable properties, additionally supported by the possible extensions discussed in Sections 4.2.4 and 4.2.5, its computational tractability and termination is questionable for the full class of ω-regular games.

As pointed out in Remark 4.1, the application of ComputeAPA might require changing the game graph if the input is not a parity specification (as in Section 2.5.2). While the *language* of the computed APA is guarantee to shrink in every iteration (see the proof of Theorem 4.1), this does not guarantee termination of Algorithm 4.1 as such a language still contains an infinite number of words. Due to the possibly repeated changes in the game graph for APA computation, the finiteness of the underlying model can also not be used as a termination argument.

In addition, the need to change game graphs induces a severe computational burden. While this might be not so obvious for the polynomial time algorithm ComputeAPA, this is actually quite critical in the case for the parity game solver (e.g., Zielonka's algorithm [225] as discussed in Section 2.5.3) which needs to be called in Algorithm 4.1 of Algorithm 4.1. As the specification for these games also keeps changing in each iteration, a new parity game needs to be constructed in each iteration, which might be increasingly harder to solve, depending on the nature of the added assumptions. In the following, we will see how these problems can be resolved by a suitable restriction of the considered assumption class.

4.3 Optimized Computation of RCS in Parity Games

As discussed in Section 4.2.6, the potential need to repeatedly change game graphs in the computations of Algorithm 4.1 in Algorithm 4.1 might incur increasing computational costs and prevents a termination guarantee. To circumvent these problems, this section proposes a different algorithm for RCS synthesis which over-approximates APA's by a simpler assumption class, called UCAs. The resulting algorithm is computationally more tractable and ensured to terminate. Nevertheless, unlike the semi-algorithm discussed in the previous section, this algorithm may not be able to compute an RCS in all scenarios where the semi-algorithm can.

4.3.1 From APAs to UCAs

One of the main features of APAs on Player p computed by ComputeAPA from Chapter 3, is the fact that they can be expressed by well structured templates using Player p's edges, namely *unsafe edges*, *co-live edges*, and *(conditional) live groups*. Unsafe and co-live edges are structurally very simple. Recall that, given a set of unsafe edges $S \subseteq E_p$ and co-live edges $D \subseteq E_p$ the respective assumption templates $\Lambda_{\text{UNSAFE}}(S) := \bigwedge_{e \in S} \Box \neg e$ and $\Lambda_{\text{COLIVE}}(D) := \bigwedge_{e \in D} \Diamond \Box \neg e$ simply assert that unsafe (resp. co-live) edges should never

(resp. only finitely often) be taken. We call an assumption which can be expressed by these two types of templates an **U**nsafe- and **C**olive-edge-template **A**ssumption (UCA), as defined next.

Definition 4.6. Given a k-player game graph $G = (V, E)$, a specification Ψ is called an unsafe- and co-live edge-template assumption (UCA) on Player p, if there exist sets $S, D \subseteq E_p$ s.t. $\Psi := \Lambda_{\text{UNSAFE}}(S) \wedge \Lambda_{\text{COLIVE}}(D)$. We write $\Psi \lhd (S, D)$ to denote such assumptions.

It was recently shown by Schmuck et al. [197] that two-player (zero-sum) parity games under UCA assumptions, i.e., games $(G, \Psi \Rightarrow \Phi)$ where Ψ is a UCA and Φ is a parity specification over G, can be directly solved over G without computational overhead, compared to the non-augmented version (G, Φ) of the same game. Interestingly, the synthesis problem under assumptions becomes provably harder if live group templates Λ_{LIVE} are needed to express an assumption, requiring a change of the game graph in most cases. Live group templates, are structurally more challenging tha UCAs, as they impose a Streett-type fairness conditions on edges in G (see the work by Schmuck et al. [197] for details).

Motivated by this result, we will restrict the assumption class used for RCS computation to UCAs in this section. Unfortunately, UCAs are typically not expressive enough to capture APAs for parity games (as in Theorem 3.4). We therefore need to over-approximate APAs by UCAs, by simply dropping the live groups, as formalized next.

Definition 4.7. Given the premises of Corollary 4.1, let $\Psi := \textsc{ComputeAPA}(G, \Phi, p) = \Lambda_{\text{UNSAFE}}(S) \wedge \Lambda_{\text{COLIVE}}(D) \wedge \Lambda_{\text{COND}}(\cdot)$. Then we denote by $\textsc{ApproxAPA}(G, \Phi, p)$ the algorithm that computes $\Psi' \lhd (S, D)$ by first executing $\textsc{ComputeAPA}(G, \Phi, p)$ and then dropping all Λ_{COND}-terms from the resulting APA.

It is easy to see that $\mathcal{L}(\Psi) \subseteq \mathcal{L}(\Psi')$. Therefore, it also follows that Ψ' is implementable and permissive (i.e., Definition 4.4 (ii) and (iii) holds). Unfortunately, Ψ' is in general no longer sufficient (i.e., Definition 4.4 (i) does not necessarily hold). As the proof of Theorem 4.1 only uses permissiveness of APA, even though sufficiency is lost for UCAs, replacing ComputeAPA by ApproxAPA in Algorithm 4.1 does not mitigate soundness, i.e., whenever ComputeRCS terminates in Algorithm 4.1 with a specification profile $(\varphi_p)_{p \in \mathtt{P}}$, this profile is indeed a maximal RCS, even if APAs are over-approximated by UCAs. This is formalized next.

Corollary 4.3. *Let AComputeRCS be the algorithm obtained by replacing procedure ComputeAPA by ApproxAPA in Algorithm 4.1. Then, given a k-player game $\mathcal{G}$ with parity specifications such that $(\varphi_p^*)_{p \in \mathtt{P}} = \textit{AComputeRCS}(\mathcal{G})$, the tuple $(\varphi_p^*)_{p \in \mathtt{P}}$ is a maximal RCS for $\mathcal{G}$.*

The rest of this section will now show how the restriction to UCAs allows to execute Algorithm 4.1 in Algorithm 4.1 efficiently and allows to prove termination of the resulting algorithm for RCS computation.

4.3.2 Iterative Computation of UCAs

We have seen in the previous section that UCAs can be computed by utilizing the procedure ComputeAPA and dropping all Λ_{LIVE} terms (called ApproxAPA). Of course, this can be done in every iteration of ComputeRCS. However, ComputeAPA expects a party game as an input, and from the second iteration of ComputeRCS onward the input to ComputeAPA is given by $(G, \Psi_{\neg p} \wedge \Phi_p, p)$, where $\Psi_{\neg p}$ is an assumption on players in $\mathtt{P}_{\neg p}$, which is not necessarily a parity game.

This section therefore provides a new algorithm, called ComputeUCA and given in Algorithm 4.2, which computes UCAs on Player p directly on the game graph G for games $(G, \Psi \wedge \Phi)$ where $\Psi \triangleleft (S, D)$ is a UCA on $\mathtt{P}_{\neg p}$ with unsafe edges $S \subseteq E_{\neg p}$ and co-live edges $D \subseteq E_{\neg p}$, and Φ is a parity specification, both over G. Intuitively, ComputeUCA first slightly modifies G to a new two-player game graph $\hat{G}$ (Algorithm 4.2) s.t. the specification $\Psi \wedge \Phi$ can be directly expressed as a parity specification $\hat{\Phi}$ on $\hat{G}$ (Algorithm 4.2). This allows to apply ApproxAPA to construct and return a UCA on Player 0 on $\hat{G}$ (Algorithm 4.2). As the resulting UCA is on Player p, the unsafe edge and co-live edge sets are subsets of E_p. Further, due to the mild modifications from G to $\hat{G}$, the edges of Player p are retained in $\hat{G}$ as E_0, hence, the resulting UCA is a well-defined UCA on Player p in G.

The soundness result of equivalence between the UCAs computed by ComputeUCA and ApproxAPA for UCA assumptions essentially relies on the observation that the parity specification $\hat{\Phi}$ in $\hat{G}$ expresses the language $\mathcal{L}(\Psi \wedge \Phi)$ when restricted to V, i.e, $\hat{\rho} \in \mathcal{L}(\hat{G}, \hat{\Phi})$ iff $\hat{\rho}|_V \in \mathcal{L}(G, \Phi \wedge \Psi)$, and the fact that every UCA on Player 0 in $\hat{G}$ is also a UCA on Player p in G. This is formalized in the following lemma.

Lemma 4.1. *Given a game graph G with parity specification Φ and a UCA $\Psi \triangleleft (S, D)$ on $\mathtt{P}_{\neg p}$, let $\hat{G}, \hat{\Phi}$ be the game graph and parity specification, respectively computed in Algorithm 4.2. Then, the following holds:*

(i) $\hat{\rho} \in \mathcal{L}(\hat{G}, \hat{\Phi})$ iff $\hat{\rho}|_V \in \mathcal{L}(G, \Phi \wedge \Psi)$;

(ii) Every UCA $\hat{\Psi} \triangleleft (\hat{S}, \hat{D})$ on Player 0 in $\hat{G}$ is a UCA on Player p in G.

Proof. It is easy to see that, for any play ρ in $\hat{G}$, the play $\rho|_V$, obtained by restricting the vertices of ρ only to V, is the corresponding play in G. Now, let us prove both items of the lemma.

▶ **(i)** Let ρ be play in $\mathcal{L}(G, \Phi \wedge \Psi)$. Then, $\rho \models \Lambda_{\text{UNSAFE}}(S)$, and hence, no unsafe edge of S appear in ρ. So, by construction as all other edges are retained from G to $\hat{G}$ (with additional middle vertex for co-live edges), there exists a corresponding play $\hat{\rho}$ in $\hat{G}$ such that $\hat{\rho}|_V = \rho$. Furthermore, as co-live edges appear only finitely often in ρ and ρ satisfies parity specification Φ, priority $2d + 1$ only appear finitely often in $\hat{\rho}$. Moreover, by the parity condition, the maximal priority in $[0; 2d]$ appearing infinitely in $\hat{\rho}$ is even. So, $\hat{\rho}$ also satisfies parity specification $\hat{\Phi}$, and hence, $\hat{\rho} \in \mathcal{L}(\hat{G}, \hat{\Phi})$. Analogously, one can show the other direction by proving for every play $\hat{\rho} \in \mathcal{L}(\hat{G}, \hat{\Phi})$, the corresponding play $\hat{\rho}|_V$ belongs to $\mathcal{L}(G, \Phi \wedge \Psi)$.

▶ **(ii)** As $\hat{\Psi}$ is a UCA on Player 0, it holds that $\hat{S}, \hat{D} \subseteq \hat{E}_0 = E_p$. Hence, $\hat{\Psi}$ is also an (well-defined) UCA on Player p in G. □

Algorithm 4.2 ComputeUCA$(G, \Psi \wedge \Phi, p)$ with $\Psi \lhd (S, D)$

Input: A k-player game graph $G = (V, E, v_0)$ and specification $\Psi \wedge \Phi$ with UCA $\Psi \lhd (S, D)$ on $\mathtt{P}_{\neg p}$, i.e., $S, D \subseteq E_{\neg p}$, and $\Phi = \mathit{Parity}(\mathbb{P})$ s.t. $\mathbb{P} : V \to [0; 2d+1]$.
Output: A UCA $\hat{\Psi}$ on Player p.

1: $\hat{V}_0 \leftarrow V_p$ and $\hat{V}_1 \leftarrow V_{\neg p} \uplus D$
2: $\hat{E}_0 \leftarrow E_p$ and $\hat{E}_1 \leftarrow E_{\neg p} \setminus (S \cup D) \cup \{(u, c), (c, v) \mid c = (u, v) \in D\}$
3: $\hat{\mathbb{P}}(v) = \begin{cases} \mathbb{P}(v) & \text{if } v \in V \\ 2d+1 & \text{otherwise.} \end{cases}$
4: $\hat{G} = (\hat{V}_0 \uplus \hat{V}_1, \hat{E}_0 \uplus \hat{E}_1, v_0)$; $\hat{\Phi} \leftarrow \mathit{Parity}(\hat{\mathbb{P}})$
5: **return** ApproxAPA$(\hat{G}, \hat{\Phi}, 0)$

With the above lemma, we have the following soundness result.

Proposition 4.1. *Given game graph $G = (V, E, v_0)$ with parity specification Φ and a UCA $\Psi \lhd (S, D)$ on $\mathtt{P}_{\neg p}$, let Ψ' and Ψ'' be the UCAs computed by ApproxAPA$(G, \Psi \wedge \Phi, p)$ and ComputeUCA$(G, \Psi \wedge \Phi, p)$, respectively, then $\mathcal{L}(\Psi') = \mathcal{L}(\Psi'')$. Furthermore, ComputeUCA terminates in time $\mathcal{O}((|V| + |E|)^4)$.*

Proof. By Lemma 4.1, $\hat{\Phi}$ expresses the same language as $\Psi \wedge \Phi$. Moreover, as $E_p = \hat{E}_0$, both APAs computed by ComputeAPA$(G, \Psi \wedge \Phi, p)$ and ComputeAPA$(\hat{G}, \hat{\Phi}, 0)$ uses the same edges of Player p in each template. Hence, approximating them with ApproxAPA results in same unsafe edge set and co-live edge set. Therefore, the UCAs ApproxAPA$(\hat{G}, \hat{\Phi}, 0)$ = ApproxAPA$(G, \Psi \wedge \Phi, p)$. As ComputeUCA returns the UCA ApproxAPA$(\hat{G}, \hat{\Phi}, 0)$, we have $\mathcal{L}(\Psi') = \mathcal{L}(\Psi'')$.

Furthermore, by construction, $\left|\hat{V}\right| \leq |V| + |E|$. Hence, by Corollary 4.1, the procedure ComputeUCA terminates in time $\mathcal{O}((|V| + |E|)^4)$. □

The usefulness of expressing the computed assumptions as unsafe and co-live edge sets S, D over the input game graph G is that there are only a finite number of edges in that graph. Therefore, there obviously also exists only a finite number of unsafe or co-live edge sets, which could all be enumerated in the worst case. Therefore, computing UCAs on the same game graph in every iteration, will ensure termination of the overall computation of RCS.

4.3.3 Solving Parity Games under UCAs

As the final step towards an optimized version of Algorithm 4.1, we now address the computations required in Algorithm 4.1 of Algorithm 4.1. Observe that this line requires to check $v_0 \in \bigcap_{p \in \mathtt{P}} \langle\!\langle p \rangle\!\rangle \varphi_p$ for $\varphi_p = \Psi_p \wedge (\Psi_{\neg p} \Rightarrow \Phi_p)$. If this check returns True the algorithm terminates, if it returns False new assumptions are computed. In both cases,

the game graph used to check this conditional will not have any effect on the future behavior of the algorithm.

Nevertheless, we utilize the recent result by Schmuck et al. [197] to compute $\langle\!\langle p \rangle\!\rangle \varphi_p$ more efficiently if Ψ_p and $\Psi_{\neg p}$ are UCAs on Player p and $\mathtt{P}_{\neg p}$, respectively. The construction uses the same idea as presented in Algorithm 4.2 to encode UCAs into a new, slightly modified two-player parity game $(\hat{G}, \hat{\Phi})$ which can then be solved by a standard parity game solver (as discussed in Section 2.5.3), which returns the winning region Win of Player 0 in this new game that corresponds to the winning region of Player p in G. The resulting procedure ComputeWin is given in Algorithm 4.3, which solves a game (G, φ) for Player p where $\varphi = \Psi_0 \wedge (\Psi_1 \Rightarrow \Phi)$ with UCA Ψ_0 on Player p and UCA Ψ_1 on $\mathtt{P}_{\neg p}$. Intuitively, as we only need to satisfy Ψ_1 or Φ, once an unsafe edge of Ψ_1 is used, we only need to satisfy Ψ_0. Hence, we have two copies of game graph: one copy to encode both assumptions and one identical to the game graph of Algorithm 4.2 only to ensure Ψ_0, both connected by vertices representing unsafe edges of Ψ_1.

Algorithm 4.3 ComputeWin$(G, \Psi_0 \wedge (\Psi_1 \Rightarrow \Phi), p)$

Input: A k-player game graph $G = (V, E, v_0)$, a specification $\varphi = \Psi_0 \wedge (\Psi_1 \Rightarrow \Phi)$ with UCA $\Psi_0 \lhd (S_0, D_0)$ on Player p and UCA $\Psi_1 \lhd (S_1, D_1)$ on $\mathtt{P}_{\neg p}$, and parity specification $\Phi = \mathit{Parity}(\mathbb{P})$ s.t. $\mathbb{P} : V \to [0; 2d]$.

Output: The set $\langle\!\langle p \rangle\!\rangle(G, \varphi)$.

1: $\hat{V}_0 \leftarrow V_p \cup \{\hat{v} \mid v \in V_p\}$ and $\hat{V}_1 \leftarrow V_{\neg p} \uplus (D_0 \cup D_1 \cup S_1) \cup \{\hat{v} \mid v \in V_{\neg p} \uplus D_0\}$

2: $\hat{E} \leftarrow E \setminus (S_0 \cup S_1 \cup D_0 \cup D_1) \cup \{(\hat{u}, \hat{v}) \mid (u, v) \in E \setminus (S_0 \cup D_0)\}$
$\cup \{(u, c), (c, v) \mid c = (u, v) \in D_0 \cup D_1\} \cup \{(\hat{u}, \hat{c}), (\hat{c}, \hat{v}) \mid c = (u, v) \in D_0\}$
$\cup \{(u, s), (s, \hat{v}) \mid s = (u, v) \in S_1\}$

3: $$\hat{\mathbb{P}} = \begin{cases} \mathbb{P}(v) & \text{if } v \in V \\ 2d & \text{if } v \in D_1 \\ 2d+1 & \text{if } v = u \text{ or } \hat{u} \text{ for some } u \in D_0 \\ 0 & \text{otherwise.} \end{cases}$$

4: $\hat{G} = (\hat{V}_0 \uplus \hat{V}_1, \hat{E}, v_0)$; $\hat{\Phi} \leftarrow \mathit{Parity}(\hat{\mathbb{P}})$

5: **return** $V \cap$ solveParity$(\hat{G}, \hat{\Phi})$

Similar to the construction presented in the previous section, due to the simplicity and locality of UCAs, the modifications required to go from (G, φ) to $(\hat{G}, \hat{\Phi})$ (Algorithm 4.3 in Algorithm 4.3) are mild enough that the check required in Algorithm 4.1 of Algorithm 4.1 can be directly performed over $\hat{G}$ without loosing correctness, as shown in Section 4.3.4.

Proposition 4.2. *Given a game graph $G = (V, E, v_0)$ with parity specification Φ, UCA $\Psi_0 \lhd (S_0, D_0)$ on Player p and UCA $\Psi_1 \lhd (S_1, D_1)$ on $\mathtt{P}_{\neg p}$, let $\mathsf{Win} =$ ComputeWin(G, φ, p) with $\varphi = \Psi_0 \wedge (\Psi_1 \Rightarrow \Phi)$. Then, it holds that $v_0 \in \langle\!\langle p \rangle\!\rangle \varphi$ if and only if $v_0 \in \mathsf{Win}$. Furthermore, ComputeWin for Φ with d priorities terminates in time $\mathcal{O}((2\,|V| + 2\,|E|)^{d+2})$.*

Proof. Let $v_0 \in \langle\!\langle p \rangle\!\rangle \varphi$. Then, Player p has a strategy π such that $\pi \models \varphi$. So, $\pi \vDash \Psi_0$, and hence, π never uses any unsafe edge of S_0. By construction, we have $E_p \setminus S_0 = \hat{E}_0$.

Hence, the following strategy $\hat{\pi}$ of Player 0 in $\hat{G}$ is well-defined: $\hat{\pi}(\hat{\rho}) = \pi(\hat{\rho}|_V)$ for each $\rho \in \hat{V}^*\hat{V}_0$. It is enough to show that $\hat{\pi} \models \hat{\Phi}$. Let $\hat{\rho}$ be a $\hat{\pi}$-play. Let $\rho = \hat{\rho}|_V$ be the corresponding π-play. As $\pi \vDash \Psi_0$, ρ uses edges of D_0 finitely many times, hence, $\hat{\rho}$ visits vertices in D_0, i.e., ones with priority $2d+1$, finitely often. Moreover, as $\pi \models \neg\Psi_1 \vee \Phi$, it holds that $\rho \models \neg\Psi_1$ or $\rho \models \Phi$. Let $\rho \models \neg\Psi_1$. If ρ uses an edge $s \in S_1$, then $\hat{\rho}$ move to the second copy, i.e., $\{\hat{v} \mid v \in V \cup D_1\}$, via s, and hence, only visits priority 0 infinitely often (as it does not visit priority $2d+1$ infinitely often). So, $\hat{\rho}$ satisfies $\hat{\Phi}$. If ρ uses an edge $c \in D_1$ infinitely often, then $\hat{\rho}$ visits c with priority $2d$ infinitely often, and hence, satisfies $\hat{\Phi}$. Finally, now, suppose $\rho \models \Phi$, then, the maximum priority ρ visits infinitely often in $[0; 2d]$ is even. As $\hat{\rho}$ does not visit priority $2d+1$ infinitely often, the maximum priority it visits is also even. Therefore, in any case, $\hat{\rho}$ satisfies $\hat{\Phi}$, and hence, $\hat{\pi} \models \hat{\Phi}$. Analogously, using similar arguments, one can show that if $v_0 \in \langle\langle 0 \rangle\rangle \hat{\Phi} = \textsc{solveParity}(\hat{G}, \hat{\Phi})$, then $v_0 \in \langle\langle p \rangle\rangle \varphi$.

Furthermore, by construction, $\left|\hat{V}\right| \leq 2(|V| + |E|)$ as there are two copies each with at most $(|V| + |E|)$ vertices. Then, the time complexity follows from Zielonka's algorithm's time complexity. □

4.3.4 Computation of RCS via UCAs

With the previously discussed algorithms in place, we are now in the position to propose an optimized, surely terminating algorithm to compute RCS, called OComputeRCS. Within ComputeRCS the recursive procedure RecursiveRCS is replaced by one which uses the algorithms ComputeUCA and ComputeWin for UCAs from Sections 4.3.2 and 4.3.3, as follows.

1: **procedure** RecursiveRCS($\mathcal{G}, (\Psi_p)_{p\in\mathtt{P}}$)
2: $\quad\Psi_{\neg p} \leftarrow \bigwedge_{q\neq p} \Psi_q \ \forall p \in \mathtt{P}$
3: $\quad\varphi_p \leftarrow \Psi_p \wedge (\Psi_{\neg p} \Rightarrow \Phi_p) \ \forall p \in \mathtt{P}$
4: $\quad\mathsf{Win}_p \leftarrow$ ComputeWin($\mathcal{G}, \varphi_p, p$)
5: $\quad$**if** $v_0 \in \bigcap_{p\in\mathtt{P}} \mathsf{Win}_p$ **then**
6: $\quad\quad$**return** $(\varphi_p)_{p\in\mathtt{P}}$
7: $\quad\Psi'_p \leftarrow \Psi_p \wedge$ ComputeUCA($G, \Psi_{\neg p} \wedge \Phi_p, p$) $\forall p \in \mathtt{P}$
8: $\quad$**if** $\Psi'_p = \Psi_p$ for all $p \in \mathtt{P}$ **then**
9: $\quad\quad$**return** False
10: $\quad$**return** RecursiveRCS($\mathcal{G}, (\Psi'_p)_{p\in\mathtt{P}}$)

We have the following main result of this section.

Theorem 4.2. *Let $\mathcal{G}$ be a k-player game with game graph $G = (V, E, v_0)$ and parity specifications $(\Phi_p)_{p\in\mathtt{P}}$ such that $(\varphi^*_p)_{p\in\mathtt{P}} = $ OComputeRCS$(\mathcal{G})$, then $(\varphi^*_p)_{p\in\mathtt{P}}$ is a maximal RCS for $\mathcal{G}$. Moreover, OComputeRCS terminates in time $\mathcal{O}(k^2\,|E| \cdot (2\,|V| + 2\,|E|)^{d+2})$, where d is the number of priorities used in the parity specifications.*

Proof. Combining results from Theorem 4.1 with Corollary 4.3 and Propositions 4.1 and 4.2 gives us that $(\varphi^*_p)_{p\in\mathtt{P}}$ is indeed a maximal RCS for $\mathcal{G}$. Furthermore, as Ψ_p

(for all $p \in \mathtt{P}$) in each iteration of the algorithm either remains the same or add more unsafe/co-live edges, it can only change $2\,|E|$ times. Hence, as there are k players, the algorithm OComputeRCS will terminate within $2k\,|E|$ iterations. Moreover, each iteration involves k calls to both ComputeWin and ComputeUCA. Using Zielonka's algorithm [225] for solving parity games, each iteration will take $\mathcal{O}((2\,|V| + 2\,|E|)^{d+2})$ time for d priorities (by Propositions 2.3, 4.1 and 4.2). In total, this gives us that OComputeRCS terminates in time $\mathcal{O}(k^2\,|E| \cdot (2\,|V| + 2\,|E|)^{d+2})$. □

We note that the time complexity is exponential as we use Zielonka's algorithm [225] to solve parity games. One can also use a quasi-polynomial algorithm [52] for solving parity games (as discussed in Section 2.5.3) to get a quasi-polynomial time complexity for OComputeRCS.

Remark 4.3. *As shown in Theorem 3.3, for games with co-Büchi specifications (i.e., $\Phi = \Diamond\Box T$ for some $T \subseteq V$), APAs are always expressible by UCAs. Hence, we note that ComputeAPA and ApproxAPA coincide for such games. This implies that no over approximation of assumptions is needed in this case and the optimizations discussed for ComputeUCA and ComputeWin can be directly applied for APAs.*

We further note that OComputeRCS also efficiently computes RCS for games with more expressive specifications. For instance, all games discussed in this chapter as well as the mutual exclusion protocol discussed in the original paper on SE [66] can be solved by OComputeRCS.

4.4 Related Work

After the introduction of *secure equilibria* (SE) by Chatterjee et al. [62], there has been several efforts on extending the notion to other classes of games, e.g., games with sup, inf, lim sup, lim inf, and mean-payoff measures [48], multi-player games with probabilistic transitions [78] or quantitative reachability games [47]. Furthermore, a variant of secure equilibria, called *Doomsday equilibria* was studied by Chatterjee et al. [58], where if any coalition of players deviates and violates one players' objective, then the objective of every player is violated. Moreover, the notion of secure equilibria has been applied effectively in the synthesis of mutual-exclusion protocols [66, 35] and fair-exchange protocols [130, 139].

Motivated by similar insights, other concepts of rationality have also been introduced in multi-player games, e.g. subgame perfect equilibria [216, 46, 207, 50, 45] or rational synthesis [101, 135, 94]. Similar to the implementations of SE, these works restrict implementations to a *single* strategy profile. In contrast, our work introduces a more flexible concept of rationality that is closely related to contract-based distributed synthesis [146, 98, 77, 14]. Here, an assume-guarantee contract is synthesized, such that every strategy realizing the guarantee is ensured to win whenever the other players satisfy the assumption. While this is conceptually similar to our synthesis of RCS, these works do not consider the players to be adversarial, and hence, there is no notion of *equilibria*.

To the best of our knowledge, the only other work that also combines permissiveness with equilibria is *assume-admissible (AA) synthesis* [44]. Their work utilizes a different, incomparable definition of rationality based on a dominance order. Both approaches are incomparable – there exist co-synthesis problems where our approach successfully synthesizes a RCS and no AA contract exists, and vice versa (see Example 4.2 for details). Conceptually, AA contracts still require *rational* behaviour of players within the contract, while our approach only uses rationality as a *concept* to synthesize meaningful local specifications which can then be implemented in an arbitrary (non-rational) manner. We believe that this is a superior strength of our approach compared to AA synthesis.

Chapter 5

Negotiation Framework for Cooperative Players

To allow for permissive interactions between components in a distributed system, Chapter 4 extended the concept of adequately permissive assumptions (APAs) in monolithic systems (Chapter 3) to distributed systems with rationally interacting components. In particular, as such distributed systems can be modeled as multi-player games (as in Section 2.5.4), Chapter 4 developed a negotiation framework to compute *rationally* contracted specifications (RCS) for k-player games. While such a rational interaction model is very robust against adversarial behavior of other components, it might not always be realistic. In particular, if interacting components are co-synthesized, the design of one component could take the needs of other components into account if these needs are known, resulting in an assume-guarantee synthesis for *cooperative* players.

Existing work on assume-guarantee synthesis for cooperative players (without imposing any rationality constraints) either do centralized synthesis [98] or restrict assumptions to safety properties [145] or consider only specific classes of specifications [76, 21]. Furthermore, all of these methods compute a single winning strategy profile for all players, which limits the freedom of each player in choosing their own strategy.

This chapter overcomes these limitations by developing a negotiation framework for cooperative players, which enables the synthesis of local strategies for each player such that any combination of these local strategies forms a globally winning strategy profile. As in the rational setting, the negotiation framework iteratively refines the assumptions players make on each other to eventually compute a new specification profile, called *cooperatively contracted specifications* (CCS), that can be realized fully locally by each player, leading to a permissive interaction between cooperative players.

While the overall structure of the negotiation framework for cooperative players is similar to the rational setting in Chapter 4, there are significant differences in the technical realization of the framework. One key difference to the rational setting lies in the fact that cooperative players can *rely* on each other to help in realizing their objectives. Hence, we can directly utilize the APA computation from Chapter 3 to compute the help a player needs from other players to realize their objective (as illustrated in Figure 1.3

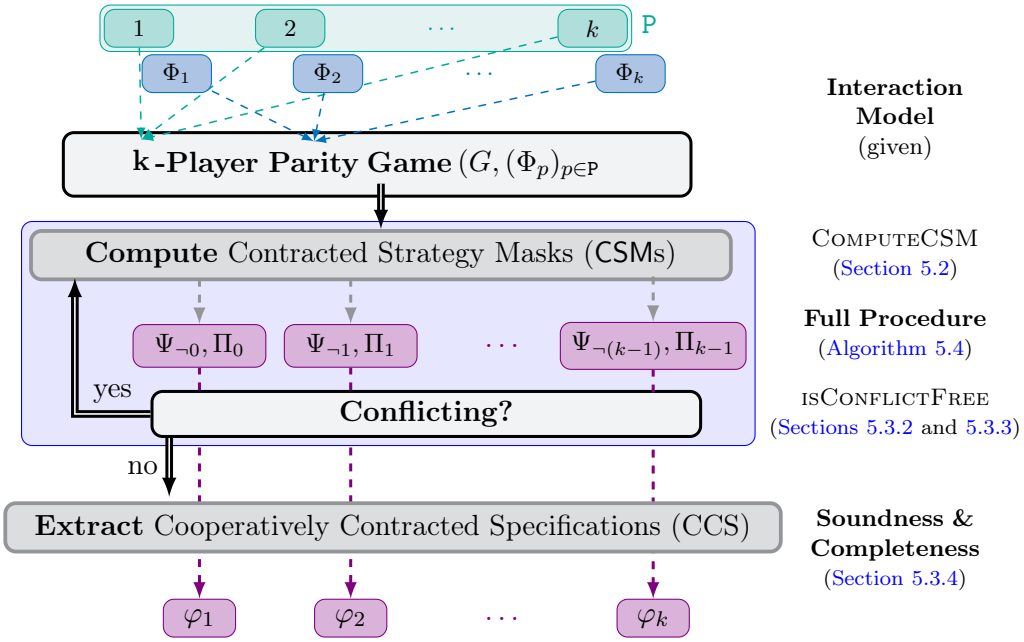


Figure 5.1: An overview of the negotiation framework for cooperative players.

(right)). Furthermore, due to the cooperative nature of the players, we can ensure termination of the negotiation framework (in contrast to the rational setting) by avoiding modifications of the game graph while checking if a player can locally satisfy their new objective (as discussed in Section 4.2.6). This is achieved by adapting the APA computation (as in Section 3.2) to extract not only the assumptions on other players, but also the strategic behavior of the current player that is winning under these assumptions, resulting in *contracted strategy masks* (CSM). By iteratively negotiating these CSMs between players, we obtain negotiation framework that computes CCSs in polynomial time.

The remainder of this chapter is organized as follows. Section 5.1 introduces *cooperatively contracted specifications* (CCS) that generalizes winning strategy profiles to permissive specification profiles for cooperative players. Thereafter, Section 5.2 introduces the notion of *contracted strategy masks* (CSM) that form the basis of our negotiation framework. Using CSMs, Section 5.3 then presents a polynomial-time negotiation framework (as schematically illustrated in Figure 5.1) to compute CCSs for k-player games with ω-regular objectives. We further discuss the application of CCSs in partial-observation settings using the well-known *knowledge-based abstraction* in Section 5.4. To showcase the practical applicability of our negotiation framework, Section 5.5 provides empirical evidence by comparing our prototype tool CoSMo to state-of-the-art solvers on a benchmark suite. Finally, we review related work in Section 5.6.

5.1 Cooperatively Contracted Specifications (CCS)

Similar to the rational setting, the cooperative negotiation framework for multi-player games also relies on the iterative computation of a new set of objectives, one for each player, such that each player can independently realize these new objectives. The main intuition behind the design of this – to be computed – new specification profile, is that it should (i) enable each player to realize this specification fully locally (decentralization), while (ii) ensure that any such local realization results in a global strategy profile that is winning (soundness), and (iii) not loosing any winning strategy profile on the way (maximality). This is formalized via cooperatively contracted specifications next.

Definition 5.1. Given a game $(G, (\Phi_p)_{p\in\mathtt{P}})$, a specification profile $(\varphi_p)_{p\in\mathtt{P}}$ is said to be a *cooperatively contracted specification* (CCS) if it is both

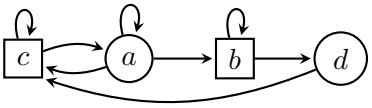
(i) *decentralized*: it holds that $v_0 \in \langle\!\langle p \rangle\!\rangle \varphi_p$ for all $p \in \mathtt{P}$;

(ii) *sound*: any $(\pi_p)_{p\in\mathtt{P}}$ with $\pi_p \models \varphi_p$ is winning, i.e., $(\pi_p)_{p\in\mathtt{P}} \models \bigwedge_{p\in\mathtt{P}} \Phi_p$.

In addition, CCSs are called *maximal* if $\mathcal{L}(\bigwedge_{p\in\mathtt{P}} \varphi_p) = \mathcal{L}(\bigwedge_{p\in\mathtt{P}} \Phi_p)$.

Similar to the algorithms for computing RCSs, we can also compute CCSs by utilizing permissive assumptions. Here, a player can *rely* on other players to help. However, in order to give every player as much freedom as possible in (locally) choosing their strategy, we want to reduce this help to the necessary minimum. In the iterative algorithm for computing CCSs every player therefore computes how *other* players *must help* her to make her own objective Φ_p realizable. Such an assumption $\Psi_{\neg p}$ on all other players $\mathtt{P}_{\neg p}$ can be computed by obtaining an APA for Φ_p on $\mathtt{P}_{\neg p}$ (as illustrated in Figure 1.3 (right)). While one can modify the assumption computation in Algorithm 4.1 to compute such a CCS, the procedure will still lack termination guarantees as in the rational setting. Recall from Section 4.2.6 that the rational negotiation framework in Algorithm 4.1 might not terminate as the game graph might be modified in every iteration while checking *realizability* of the new objectives, i.e., if a player can independently satisfy their new objective without any cooperation from other players. However, in the cooperative setting, we can avoid these modifications to the game graph. The key insight is that the local permissive templates used to express APAs (as introduced in Section 3.2) can also be used to express the strategic behavior of a player that is winning under these assumptions. In particular, during the APA computation for a player Player p w.r.t. their objective Φ_p, with slight modifications, we extract not only the assumption $\Psi_{\neg p}$ on other players $\mathtt{P}_{\neg p}$, but also the strategic behavior of Player p that is winning for Φ_p under the assumption $\Psi_{\neg p}$ in the form of these templates. This pair of permissive templates—the *assumption template* $\Psi_{\neg p}$ on other players and the *strategy template* Π_p for Player p—forms a *contracted strategy mask* (CSM) for player Player p. Let us briefly illustrate this idea via an example before going into details.

Example 5.1. In order to appreciate the simplicity templates bring to the computation of CCSs, let us consider the game graph in Figure 5.2. The winning condition Φ_0 for Player 0 requires vertex c to be seen infinitely often. Intuitively, every winning strategy for Player 0 w.r.t. Φ_0 needs to eventually take the edge e_{ac} if it sees vertex a infinitely



$$\begin{aligned}
\Phi_0 &= \square\diamondsuit\{c\} && \Rightarrow \Psi_1 = \Lambda_{\text{LIVE}}(\{e_{bd}\}),\ \Pi_0 = \Lambda_{\text{LIVE}}(\{e_{ac}\}) \\
\Phi_1 &= \diamondsuit\square\{a,c,d\} && \Rightarrow \Psi_0 = \Lambda_{\text{COLIVE}}(e_{ab}),\ \Pi_1 = \Lambda_{\text{COLIVE}}(e_{bb}) \\
\Phi_1' &= \diamondsuit\square\{a,b,c\} && \Rightarrow \Psi_0' = \texttt{true}\ , \qquad \Pi_1' = \Lambda_{\text{COLIVE}}(e_{bd})
\end{aligned}$$

Figure 5.2: A two-player game graph discussed in Example 5.1 with Player 1 (squares) and Player 0 (circles) vertices, different winning conditions Φ_p, and corresponding assumption templates $\Psi_{\neg p}$ and strategy templates Π_p for Player p.

often. Furthermore, Player 0 can only win from vertex b with the help of Player 1. In particular, Player 1 needs to ensure that whenever vertex b is seen infinitely often he takes the edge e_{bd} infinitely often. These two conditions can be concisely formulated via the strategy template $\Pi_0 = \Lambda_{\text{LIVE}}(\{e_{ac}\})$ on Player 0 and an assumption template $\Psi_1 = \Lambda_{\text{LIVE}}(\{e_{bd}\})$ on Player 1, both using our live group templates from Section 3.2.2. It is easy to see that every Player 0 strategy that follows Π_0 is winning for Φ_0 from all vertices under the assumption that Player 1 chooses a strategy that follows Ψ_1.

Now, consider the winning condition Φ_1 for Player 1 which requires the play to eventually stay in region $\{a, c, d\}$. This induces assumption Ψ_0 on Player 0 and strategy template Π_1 for Player 1 given in Figure 5.2 (right). Both are expressible via our colive templates from Section 3.2.3 – in particular, $\Psi_0 = \Lambda_{\text{COLIVE}}(e_{ab})$ ensures that edge e_{ab} is taken only finitely often, while $\Pi_1 = \Lambda_{\text{COLIVE}}(e_{bb})$ ensures that edge e_{bb} is taken only finitely often. Overall, these templates ensure that Player 1 can realize Φ_1 from all vertices under the assumption that Player 0 chooses a strategy following Ψ_0.

The tuples of assumption and strategy templates $(\Psi_{\neg p}, \Pi_p)$ we have constructed for both players in the above example form a contracted strategy mask (CSM) for each player. If the players now share the assumptions from their local CSMs, it is easy to see that in the above example both players can ensure the assumptions made by other player in addition to their own strategy templates, i.e., each Player p can realize $\Psi_p \wedge \Pi_p$ from all vertices. In this case, we call the CSMs $(\Psi_{\neg p}, \Pi_p)$ *realizable*. In such situations, the new specifications (φ_0, φ_1) with $\varphi_p = \Psi_p \wedge (\Psi_{\neg p} \Rightarrow \Phi_p)$ are directly computable from the given CSMs and indeed form a CCS as per Definition 5.1.

Unfortunately, locally computed CSMs are not always realizable. To see this, consider the slightly modified winning condition Φ_1' for Player 1 that induces strategy template Π_1' for Player 1. This template requires the edge e_{bd} to be taken only *finitely* often. Now, Player 1 cannot realize both Ψ_1 and Π_1' as the conditions given by both templates for edge e_{bd} are *conflicting* – the same edge cannot be taken infinitely often *and* finitely often. In this case one more round of negotiation is needed to ensure that both players eventually avoid vertex d by modifying the objectives to $\Phi_p' = \Phi_p \wedge \diamondsuit\square\neg d$. This will give us a new pair of CSMs that are indeed realizable, and a new pair of objectives (φ_0, φ_1) that are now again a CCS. ⌟

The overall negotiation framework for cooperative players is illustrated in Figure 5.1 and will be presented in detail in the following sections. We start by introducing the notion of contracted strategy masks in Section 5.2, followed by the negotiation framework

in Section 5.3 including the notion of conflicting templates in Section 5.3.2 and their resolution in Section 5.3.3. Finally, we prove soundness and completeness of the overall framework in Section 5.3.4.

5.2 Contracted Strategy Masks (CSMs)

Towards our goal of formalizing CCS via templates, this section defines *contracted strategy masks* (CSMs) (cf. Figure 5.1 (upper middle)), which contain two templates $\Psi_{\neg p}$ and Π_p, representing assumptions on $\mathtt{P}_{\neg p}$ and strategies for Player p respectively. Intuitively, this captures the assumption $\Psi_{\neg p}$ on player $\mathtt{P}_{\neg p}$ under which Player p can win the local game (G, Φ_p) with any strategy from Π_p. This is formalized below.

Definition 5.2. Given a k-player game graph $G = (V, E, v_0)$ and a specification Φ, we say that a pair $(\Psi_{\neg p}, \Pi_p)$ of specifications is a (adequately permissive) *contracted strategy mask (CSM)* for Player p if it is

(i) *sufficient*: any $(\pi_p, \pi_{\neg p})$ with $\mathtt{plays}(\pi_p) \subseteq \mathcal{L}(\Pi_p)$ and $\mathtt{plays}(\pi_{\neg p}) \subseteq \mathcal{L}(\Psi_{\neg p})$ is winning for Φ from all $v \in \langle\!\langle \mathtt{P} \rangle\!\rangle \Phi$;

(ii) *implementable*: $\langle\!\langle \mathtt{P}_{\neg p} \rangle\!\rangle \Psi_{\neg p} = V$ and $\langle\!\langle p \rangle\!\rangle \Pi_p = V$;

(iii) *permissive*: $\mathcal{L}(\Psi_{\neg p}) \supseteq \mathcal{L}(\Phi)$.

This definition captures the essence of APA into CSMs as follows. Sufficiency ensures that Player p can satisfy its objective Φ_p with any strategy following Π_p whenever the assumptions $\Psi_{\neg p}$ on $\mathtt{P}_{\neg p}$ are satisfied. Implementability ensures that $\mathtt{P}_{\neg p}$ can satisfy the assumptions $\Psi_{\neg p}$ and that Player p can satisfy the strategy template Π_p. Finally, permissiveness ensures that the assumptions $\Psi_{\neg p}$ do not prevent any cooperative solution of Φ_p.

Furthermore, considering a two-player game between Player p and $\mathtt{P}_{\neg p}$, we observe that the assumption template $\Psi_{\neg p}$ in a CSM for Player p is precisely the APA on $\mathtt{P}_{\neg p}$. Due to such close connection between APAs and CSMs, it turns out that the computation of CSMs can be done in very close analogy to the computation of APAs from Section 3.2. In particular, we inherit (i) the observation that conjunctions of safety, co-live and conditional live group templates are rich enough to express CSMs, and (ii) the existence of a polynomial time (i.e., very efficient) algorithm for their construction. To ease the presentation, we first introduce some notation for templates and then present the construction of CSMs for two-player games in detail, which can then be extended to multi-player games.

5.2.1 Expressing CSMs via Permissive Templates

As stated above, we will express CSMs using conjunctions of safety, co-live and conditional live group templates as in Section 3.2. To simplify notation, we write $\Lambda \lhd (S, D, \mathbb{H})$ to denote that Λ is the constraint given by the conjunction of a safety template $\Lambda_{\text{UNSAFE}}(S)$, a co-live template $\Lambda_{\text{COLIVE}}(D)$, and a conditional live group template

$\Lambda_{\text{COND}}(\mathbb{H})$. Similarly, we write $\Lambda \lhd (S, D, \mathcal{H})$ to denote that Λ is the constraint given by the conjunction of the corresponding safety and co-live templates, and a live group template. Finally, a strategy π is said to *follow* a template Λ, denoted $\pi \Vdash \Lambda$, if all π-plays satisfy Λ, i.e., $\texttt{plays}(\pi) \subseteq \mathcal{L}(\Lambda)$.

5.2.2 CSMs for Safety Games

Recall from Theorem 3.1 that in safety games $(G, \Box I)$, an APA (on Player 1 in a two-player game) simply disallows all moves of Player 1 that leave the cooperative winning region. With the same intuition, it is easy to see that a CSM for Player 0 in safety games consists of an assumption template that is an APA on Player 1 and a strategy template for Player 0 that disallows all Player 0 moves leaving the cooperative winning region. This is formalized in the following theorem.

Theorem 5.1. *For a safety game $\mathcal{G} = (G, \Box I)$, let $\mathsf{Win} = \textsc{solveCoopSafety}(G, I)$ and $S_p = \{(u, v) \in E \mid (u \in V_p \cap \mathsf{Win}) \wedge (v \notin \mathsf{Win})\}$ for $p \in \{0, 1\}$. Then the pair (Ψ_1, Π_0) defined by $\Psi_1 \lhd (S_1, \emptyset, \emptyset)$ and $\Pi_0 \lhd (S_0, \emptyset, \emptyset)$ is a CSM for Player 0. We denote by $\textsc{TempSafe}(G, I)$ the algorithm computing (S_0, S_1) as above, which runs in time $\mathcal{O}(m + n)$, where $n = |V|$ and $m = |E|$.*

Proof. The time complexity follows from the runtime of SOLVECOOPSAFETY. We show that the templates (Ψ_1, Π_0) form a CSM for Player 0 by proving its permissiveness, implementability, and sufficiency. Permissiveness follows from the proof of Theorem 3.1. Furthermore, observe that each set S_p contains only edges of Player p (i.e., $S_p \subseteq E_p$) that lead out of the cooperative winning region Win. Moreover, as every vertex v in Win is winning when both players cooperate, there exists an edge from v to a vertex in Win, which do not belong to S_p. Hence, both players can follow their respective templates by choosing such edges in their strategies, establishing implementability.

We only need to show sufficiency. Consider any strategies π_0 and π_1 for Player 0 and Player 1 respectively, such that $\pi_0 \Vdash \Pi_0$ and $\pi_1 \Vdash \Psi_1$. Let ρ be a (π_0, π_1)-play starting from a vertex $v \in \mathsf{Win}$. We need to show that ρ is winning, i.e., it always stays in I. Since both π_0 and π_1 follow their respective templates, no edge in S_0 or S_1 is taken along ρ. Hence, by construction of S_0 and S_1, all vertices along ρ belong to $\mathsf{Win} \subseteq I$. Therefore, ρ is winning, establishing sufficiency. □

5.2.3 CSMs for Büchi Games

Following the construction of APAs for Büchi games in Section 3.2.2, we now present an algorithm to compute CSMs for Büchi games in Algorithm 5.1. As in the case of APAs, we first compute the unsafe edges (in Algorithm 5.1) and then compute the live groups using procedure TEMPLIVE in the game restricted to the cooperative winning region.

The procedure TEMPLIVE also closely follows the procedure ASSUMPLIVE from Algorithm 3.1. The key difference is in Algorithms 5.1 to 5.2. Unlike in ASSUMPLIVE, we do not directly set $U \leftarrow \textsc{SolveReach}(G, U)$, but instead compute the edges from Player 0 vertices that make progress towards I and add these edges to the live group of

Algorithm 5.1 TEMPBÜCHI(G, I)

Input: $G = (V = V_0 \cup V_1, E)$, Büchi objective $\Phi = \square\diamondsuit I$, for $I \subseteq V$
Output: CSM (Ψ_1, Π_0) for Player 0

1: $\mathsf{Win} \leftarrow$ SOLVECOOPBÜCHI(G, I)
2: $(S_0, S_1) \leftarrow$ TEMPSAFE(G, Win)
3: $G \leftarrow G|_{\mathsf{Win}}, I \leftarrow I \cap \mathsf{Win}$ ▷ All vertices are cooperatively Büchi winning
4: $(\mathcal{H}_0, \mathcal{H}_1) \leftarrow$ TEMPLIVE(G, I)
5: $\Psi_1 \lhd (S_1, \emptyset, \mathcal{H}_1)$; $\Pi_0 \lhd (S_0, \emptyset, \mathcal{H}_0)$
6: **return** (Ψ_1, Π_0)

7: **procedure** TEMPLIVE(G, I)
8: $U \leftarrow I$; $\mathcal{H}_0 \leftarrow \emptyset$; $\mathcal{H}_1 \leftarrow \emptyset$
9: **while** $U \neq V$ **do**
10: **while** $U \neq$ SOLVEREACH(G, U) **do**
11: $H_0 \leftarrow E_0 \cap \big((V \setminus U) \times U\big)$
12: $\mathcal{H}_0 \leftarrow \mathcal{H}_0 \cup \{H_0\}$
13: $U \leftarrow U \cup \mathit{src}(H_0)$
14: $H_1 \leftarrow E \cap \big((V \setminus U) \times U\big)$
15: $\mathcal{H}_1 \leftarrow \mathcal{H}_1 \cup \{H_1\}$
16: $U \leftarrow U \cup \mathit{src}(H_1)$
17: **return** $(\mathcal{H}_0, \mathcal{H}_1)$

Player 0. This gives us a strategy template for Player 0 that ensures progress towards I under the assumption that Player 1 cooperates.

Theorem 5.2. *Given a game $\mathcal{G} = (G, \Phi)$ with game graph $G = (V, E)$ and Büchi winning condition $\Phi = \square\diamondsuit I$, Algorithm 5.1 terminates in time $\mathcal{O}(m + n)$, where $n = |V|$ and $m = |E|$. Furthermore, the output (Ψ_1, Π_0) of the procedure TEMPBÜCHI(G, I) is a CSM for Player 0.*

Proof. The time complexity follows from the runtime analysis of ASSUMPBÜCHI in the proof of Theorem 3.2. We show that the templates (Ψ_1, Π_0) form a CSM for Player 0 by proving its permissiveness, implementability, and sufficiency. Permissiveness follows from the proof of Theorem 3.2. For implementability, as live groups are computed in the game restricted to the cooperative winning region Win, any live group has no intersection with the unsafe edges. Furthermore, the live groups of Π_0 (respectively, Ψ_1) only contain edges of Player 0 (respectively, Player 1). Hence, the corresponding player can follow the template by choosing these edges infinitely often in its strategy, establishing implementability.

We only need to show sufficiency. Consider any strategies π_0 and π_1 for Player 0 and Player 1 respectively, such that $\pi_0 \Vdash \Pi_0$ and $\pi_1 \Vdash \Psi_1$. Let ρ be a (π_0, π_1)-play starting from a vertex $v_0 \in \mathsf{Win}$. We need to show that ρ is winning, i.e., it visits I infinitely often. By sufficiency of unsafe edges from Theorem 5.1, the play ρ never goes outside

Win.

Now, let us consider the game to be restricted to Win. Suppose $U_0, U_1, \ldots, U_m = V$ are the sets U at the end of each iteration of inner and outer while-loops in TempLive (Algorithm 5.1). As vertices are only added to U, we have $I = U_0 \subseteq U_1 \subseteq \ldots \subseteq U_m = V$. Suppose the play ρ does not visit I infinitely often. Let k be the least index such that ρ visits a vertex in U_k infinitely often, i.e., there exists a vertex $v \in U_k \cap \inf(\rho)$ but $\inf(\rho) \cap U_{k-1} = \emptyset$. If v is added to U_k in the inner while-loop (i.e., in Algorithm 5.1), then by construction of $\mathcal{H}_0$ in Algorithm 5.1, the corresponding live group in that iteration contains Player 0 edges from $U_k \setminus U_{k-1}$ to U_{k-1}. Since π_0 follows Π_0, the play ρ must take one such edge infinitely often, contradicting the assumption that $\inf(\rho) \cap U_{k-1} = \emptyset$. If v is added to U_k in the outer while-loop (i.e., in Algorithm 5.1), by using similar reasoning for Ψ_1 and π_1, we again reach a contradiction. Hence, the play ρ must visit I infinitely often, establishing sufficiency. □

5.2.4 CSMs for co-Büchi Games

Similar to the Büchi case, the construction of CSMs for co-Büchi games, presented in Algorithm 5.2, closely follows the construction of APAs for co-Büchi games from Section 3.2.3. The algorithm first computes the unsafe edges (in Algorithm 5.2) and then computes the co-live edges for both players using procedure TempCoLive in the game restricted to the cooperative winning region. The key difference in TempCoLive compared to AssumpCoLive from Algorithm 3.2 is that in both Algorithm 5.2, we add co-live edges for both players instead of only for Player 1. While the co-live edges for Player 1 form an APA as in Section 3.2.3, the co-live edges for Player 0 form a strategy template following which Player 0 can ensure to not leave the target region I infinitely often, under the assumption that Player 1 cooperates.

Theorem 5.3. *Given a game $\mathcal{G} = (G, \Phi)$ with game graph $G = (V, E)$ and co-Büchi winning condition $\Phi = \Diamond\Box I$, Algorithm 5.2 terminates in time $\mathcal{O}(m+n)$, where $n = |V|$ and $m = |E|$. Furthermore, the output (Ψ_1, Π_0) of the procedure TempCoBüchi(G, I) is a CSM for Player 0.*

Proof. The time complexity follows from the runtime analysis of AssumpCoBüchi in the proof of Theorem 3.3. We show that the templates (Ψ_1, Π_0) form a CSM for Player 0 by proving its permissiveness, implementability, and sufficiency. Permissiveness follows from the proof of Theorem 3.3. For implementability, we again observe that the template Ψ_1 (respectively, Π_0) only contains edges of Player 1 (respectively, Player 0) and for each source vertex of co-live edges, there exists an edge that is neither unsafe nor co-live. Hence, both players can follow their respective templates by choosing such edges in their strategies, establishing implementability.

We only need to show sufficiency. Consider any strategies π_0 and π_1 for Player 0 and Player 1 respectively, such that $\pi_0 \Vdash \Pi_0$ and $\pi_1 \Vdash \Psi_1$. Let ρ be a (π_0, π_1)-play starting from a vertex $v_0 \in$ Win. We need to show that ρ is winning, i.e., it eventually always stays in I. By sufficiency of unsafe edges from Theorem 5.1, the play ρ never go outside Win.

Algorithm 5.2 TEMPCOBÜCHI(G, I)

Input: $G = (V, E), I \subseteq V$
Output: CSM (Ψ_1, Π_0) for Player 0

```
1: Win ← SOLVECOOPCOBÜCHI(G, I)
2: (S_0, S_1) ← TEMPSAFE(G, Win)
3: G ← G|_Win, I ← I ∩ Win          ▷ All vertices are cooperatively co-Büchi winning
4: (D_0, D_1) ← TEMPCOLIVE(G, I)
5: Ψ_1 ◁ (S_1, D_1, ∅); Π_0 ◁ (S_0, D_0, ∅)
6: return (Ψ_1, Π_0)

7: procedure TEMPCOLIVE(G, I)
8:     U ← SOLVECOOPSAFETY(G, I)                                   ▷ U ⊆ I
9:     D_p ← E_p ∩ (U × V\U) for p ∈ {0, 1}
10:    while U ≠ V do
11:        D_p ← D_p ∪ (E_p ∩ (pre(U) × V\U)) for p ∈ {0, 1}
12:        U ← U ∪ pre(U)
13:    return (D_0, D_1)
```

Now, let us consider the game to be restricted to Win. Suppose $U_0, U_1, \ldots, U_m = V$ are the sets U at the end of each iteration of the while-loop in TEMPCOLIVE (Algorithm 5.2). As vertices are only added to U, we have $I = U_0 \subseteq U_1 \subseteq \ldots \subseteq U_m = V$. Suppose the play ρ does not eventually always stay in I. Let $k \geq 1$ be the least index such that ρ visits a vertex in $U_k \backslash I$ infinitely often, i.e., there exists a vertex $v \in (U_k \backslash I) \cap \inf(\rho)$. If $k = 1$, then by construction of D_p in Algorithm 5.2, some edge of D_p for $p \in \{0, 1\}$ is taken infinitely often along ρ, contradicting the assumption that both π_0 and π_1 follow their respective templates. If $k > 1$, then by construction of D_p in Algorithm 5.2, some edge of D_p for $p \in \{0, 1\}$ is taken infinitely often along ρ, again contradicting the assumption that both π_0 and π_1 follow their respective templates. Hence, the play ρ must eventually always stay in I, establishing sufficiency. □

5.2.5 CSMs for Parity Games

Finally, we present the construction of CSMs for parity games in Algorithm 5.3. The algorithm is very similar to the construction of APAs for parity games from Section 3.2.4, with the key difference being that we compute the live groups and co-live edges for both players instead of only for Player 1. In particular, Algorithm 5.3 uses the procedures TEMPLIVE, TEMPCOLIVE and TEMPSAFE defined in the previous sections to compute the live groups, co-live edges and unsafe edges respectively, instead of the corresponding APA procedures.

Theorem 5.4. *Given a game $\mathcal{G} = (G, \Phi)$ with game graph $G = (V, E)$ and parity winning condition $\Phi = Parity(\mathbb{P})$, Algorithm 5.3 terminates in time $\mathcal{O}(n^4)$, where $n = |V|$. Furthermore, the output (Ψ_1, Π_0) of the procedure TEMPPARITY$(G, \mathbb{P})$ is a CSM for*

Algorithm 5.3 TempParity$(G, \mathbb{P})$

Input: $G = (V, E)$, $\mathbb{P} : V \to \{0, 1, \ldots\}$
Output: CSM (Ψ_1, Π_0) for Player 0

$\mathsf{Win} \leftarrow$ solveCoopParity$(G, \mathbb{P})$
$(S_0, S_1) \leftarrow$ TempSafe(G, Win)
$G \leftarrow G|_{\mathsf{Win}}$, $\mathbb{P} \leftarrow \mathbb{P}|_{\mathsf{Win}}$
$(\mathbb{H}_0, \mathbb{H}_1, \mathtt{C}) \leftarrow$ TempCondLive$(G, \mathbb{P})$
$(D_0, D_1) \leftarrow$ TempCoLive$(G, V \setminus \mathtt{C})$
$\Psi_1 \lhd (S_1, D_1, \mathbb{H}_1)$; $\Pi_0 \lhd (S_0, D_0, \mathbb{H}_0)$
return $(\mathsf{Win}, \mathtt{C}, \Psi_1, \Pi_0)$

procedure TempCondLive$(G, \mathbb{P})$
    $\mathbb{H}_0 \leftarrow \emptyset$; $\mathbb{H}_1 \leftarrow \emptyset$; $\mathtt{C} \leftarrow \emptyset$
    while $G \neq \emptyset$ **do**
        $d \leftarrow \max\{i \mid \mathbb{P}[i] \neq \emptyset\}$
        if d is odd **then**
            $W_{\neg d} \leftarrow$ solveCoopParity$(G|_{V \setminus \mathbb{P}[d]}, \mathbb{P})$
            $\mathtt{C} \leftarrow \mathtt{C} \cup (V \setminus W_{\neg d})$
        else
            $W_d \leftarrow$ solveCoopBüchi$(G, \mathbb{P}[d])$, $W_{\neg d} \leftarrow V \setminus W_d$
            for all $i \in_{\text{odd}} [0; d]$ **do**
                $(\mathcal{H}_0, \mathcal{H}_1) \leftarrow$ TempLive$(G|_{W_d}, \mathbb{P}[i+1] \cup \mathbb{P}[i+3] \cdots \cup \mathbb{P}[d])$
                $\mathbb{H}_p \leftarrow \mathbb{H}_p \cup (W_d \cap \mathbb{P}[i], \mathcal{H}_p)$ for $p \in \{0, 1\}$
        $G \leftarrow G|_{W_{\neg d}}$, $\mathbb{P} \leftarrow \mathbb{P}|_{W_{\neg d}}$
        $\mathbb{P}[0] \leftarrow \mathbb{P}[0] \cup \mathbb{P}[d]$, $\mathbb{P}[d] \leftarrow \emptyset$
    return $(\mathbb{H}_0, \mathbb{H}_1, \mathtt{C})$

Player 0.

Proof. The time complexity follows from the runtime analysis of AssumpParity in the proof of Theorem 3.4. We show that the templates (Ψ_1, Π_0) form a CSM for Player 0 by proving its permissiveness, implementability, and sufficiency. Permissiveness follows from the proof of Theorem 3.4. For implementability, we again observe that the template Ψ_1 (respectively, Π_0) only contains edges of Player 1 (respectively, Player 0) and for each vertex, there exists an edge that is neither unsafe nor co-live. Moreover, for each source vertex of a conditional live group, there exists an edge that is neither unsafe nor co-live. Hence, both players can follow their respective templates by never choosing unsafe or co-live edges and by choosing edges of live groups infinitely often in their strategies, establishing implementability.

We only need to show sufficiency. Consider any strategies π_0 and π_1 for Player 0 and Player 1 respectively, such that $\pi_0 \Vdash \Pi_0$ and $\pi_1 \Vdash \Psi_1$. Let ρ be a (π_0, π_1)-play starting from a vertex $v_0 \in \mathsf{Win}$. We need to show that ρ is winning, i.e., the highest priority

Algorithm 5.4 ComputeCCS($\mathcal{G}$)

Input: A k-player game $\mathcal{G}$ with game graph $G = (V, E, v_0)$ and parity specifications $(\Phi_p)_{p\in\mathtt{P}}$.

Output: A cooperatively contracted specification $(\varphi_p)_{p\in\mathtt{P}}$ and templates $(\Lambda_p)_{p\in\mathtt{P}}$.

1: $\mathsf{Win}_p, \mathtt{C}_p, \Pi_p, \Psi_{\neg p} \leftarrow \textsc{ComputeCSM}(G, \Phi_p, p)$, $\forall p \in \mathtt{P}$
2: **if** $\textsc{isConflictFree}\left(G, \bigwedge_{p\in\mathtt{P}}(\Pi_p \wedge \Psi_{\neg p})\right)$ **then**
3: $\quad \Psi_p \leftarrow \bigwedge_{q\neq p} \Psi_{\neg q}|_{E_p}$ $\forall p \in \mathtt{P}$
4: $\quad \varphi_p \leftarrow \Psi_p \wedge (\Psi_{\neg p} \Rightarrow \Phi_p)$ $\forall p \in \mathtt{P}$
5: $\quad$ **return** $(\varphi_p)_{p\in\mathtt{P}}, (\Psi_p \wedge \Pi_p)_{p\in\mathtt{P}}$
6: **else**
7: $\quad \Phi_p \leftarrow \Phi_p \wedge \Box(\bigcap_{p\in\mathtt{P}} \mathsf{Win}_p) \wedge \Diamond\Box\neg(\bigcup_{p\in\mathtt{P}} \mathtt{C}_p)$, $\forall\ p \in \mathtt{P}$
8: $\quad$ **return** $\textsc{ComputeCCS}(G, \Phi'_0, \Phi'_1)$

visited infinitely often along ρ is even. By sufficiency of unsafe edges from Theorem 5.1, the play ρ never go outside Win.

Now, let us consider the game to be restricted to Win. By sufficiency of the procedure $\textsc{TempCoLive}(G, V \setminus \mathtt{C})$, we know that the play ρ does not visit vertices in $\mathtt{C}$ infinitely often. Furthermore, by construction of the regions W_d, ρ cannot switch between different W_d regions infinitely often. Hence, ρ eventually stays in W_d for some unique even priority d. Then, by sufficiency of $\textsc{TempLive}(G|_{W_d}, \mathbb{P}[i+1] \cup \mathbb{P}[i+3] \cdots \cup \mathbb{P}[d])$ for all odd $i < d$, we know that if ρ visits a vertex with an odd priority $i < d$ infinitely often, then it also visits a vertex with an even priority $j \in \{i+1, i+2, \ldots, d\}$ infinitely often. Hence, the highest priority visited infinitely often along ρ has to be even, establishing sufficiency. □

With the above construction of CSMs for two-player games, we can obtain a CSM for Player p in a k-player game by considering the two-player game between Player p and $\mathtt{P}_{\neg p}$ obtained by merging all players in $\mathtt{P}_{\neg p}$ into a single player. This leads to the following result.

Corollary 5.1. *Given a k-player game graph $G = (V, E, v_0)$ and a parity specification $\Phi = \mathit{Parity}(\mathbb{P})$, a CSM for Player p can be computed in time $\mathcal{O}(|V|^4)$. We write $\textsc{ComputeCSM}(G, \Phi, p)$ to denote the procedure that returns this CSM.*

5.3 Negotiation Framework

With the computation of CSMs at hand, we are now ready to present the negotiation framework for cooperative players (cf. Figure 5.1). As stated before, the key insight of the framework is to utilize CSMs is to efficiently check the realizability without changing the game graph. In particular, the idea is to compute a CSM for each player for their original specification, and then check if the composition of all the templates is realizable, i.e., if there exists a strategy profile that follows all the templates.

Formally, our algorithm, called ComputeCCS, is given in Algorithm 5.4 and schematically illustrated in Figure 5.1. It uses ComputeCSM, defined in Section 5.2.5, to compute CSMs $(\Psi_{\neg p}, \Pi_p)$ for each Player p in its corresponding game (G, Φ_p) locally (as in Algorithm 5.4 and cf. Figure 5.1 (upper middle)). We then check if the composition of these CSMs is realizable via the function isConflictFree formalized later in Section 5.3.2 (cf. Figure 5.1 (lower middle)). If their composition is realizable, they would form a CSS (as in Algorithm 5.4 and cf. Figure 5.1 (bottom)) and hence, ComputeCCS terminates. If the composition is not realizable, that means there exist conflicts between the templates which need to be resolved as formalized in Section 5.3.3. The required strengthening of all CSMs is again done locally by solving games with modified specifications again via ComputeCSM. As the resulting new CSMs might again be conflicting, this strengthening process repeats iteratively. We prove in Section 5.3.4 that there are always only a finite number of negotiation rounds.

5.3.1 Player Specific Templates.

Before going into the details of ComputeCCS, let us first formalize the notion of player-specific templates. As the assumption template $\Psi_{\neg p}$ in a CSM for Player p is defined over the edges of all players $\mathtt{P}_{\neg p}$, we need to separate the responsibility of each player in $\mathtt{P}_{\neg p}$ to encode their guarantees. In order to do so, we need to define and extract player specific templates from a given template as follows.

A template $\Lambda \lhd (S, D, \mathbb{H})$ over G is called a p-template for some $p \in \mathtt{P}$ if all edges used in the template are Player p's edges, i.e., $S \cup D \cup \overline{H} \subseteq E_p$, where $\overline{H} := \bigcup\{H \in \mathcal{H} \mid (\cdot, \mathcal{H}) \in \mathbb{H}\}$. Furthermore, given any template $\Lambda \lhd (S, D, \mathcal{H})$, one can extract a p-template $\Lambda|_{E_p}$ by restricting the edges used to E_p, i.e., $\Lambda|_{E_p} \lhd (S \cap E_p, D \cap E_p, \mathcal{H}|_{E_p})$ with $\mathcal{H}|_{E_p} = \{H \cap E_p \mid H \in \mathcal{H}\}$. Similarly, for $\Lambda \lhd (S, D, \mathbb{H})$, we define $\Lambda|_{E_p} \lhd (S \cap E_p, D \cap E_p, \mathbb{H}|_{E_p})$ with $\mathbb{H}|_{E_p} = \big\{(R, \mathcal{H}|_{E_p}) \mid (R, \mathcal{H}) \in \mathbb{H}\big\}$.

5.3.2 Checking Realizability of Composed Templates

Now, to compose CSMs for different players, we need to compose their templates. Intuitively, the composition of two templates is simply the conjunction of their constraints, i.e., unsafe edges are the union of unsafe edges in both templates, and similarly for co-live and (conditional) live group edges. This is formalized below.

Definition 5.3. Let $\Lambda \lhd (S, D, \mathbb{H})$ and $\Lambda' \lhd (S', D', \mathbb{H}')$ be two templates. We define their composition as the template $(\Lambda \wedge \Lambda') \lhd (S \cup S', D \cup D', \mathbb{H} \cup \mathbb{H}')$.

With this, in order to check if the composition of CSMs $(\Psi_{\neg p}, \Pi_p)$ for all $p \in \mathtt{P}$ is realizable, we need to check if there exists a strategy profile $(\pi_p)_{p \in \mathtt{P}}$ such that each π_p follows the corresponding p-template obtained from the composition of all templates, i.e., $\pi_p \Vdash \Pi_p \wedge \bigwedge_{q \neq p} \Psi_{\neg q}|_{E_p}$. Note that checking the existence of such a strategy profile is in generally expensive as the templates are basically LTL formulas. However, we can exploit the local nature of templates to give a sufficient condition for realizability, that

can be checked efficiently. To this end, we first define the notion of conflict-freeness of templates as follows.

Definition 5.4. A template $\Lambda \lhd (S, D, \mathbb{H})$ in G is said to be *conflict-free* if

(i) every vertex v has an outgoing edge that is neither co-live nor unsafe, i.e., $v \times E(v) \not\subseteq D \cup S$, and

(ii) in every live group $H \in \mathcal{H}$ s.t. $(\cdot, \mathcal{H}) \in \mathbb{H}$, every source vertex v has an outgoing edge in H that is neither co-live nor unsafe, i.e., $v \times H(v) \not\subseteq D \cup S$.

We denote the procedure of checking (i)-(ii) as isConflictFree(G, Λ), which returns True if (i)-(ii) holds, and False otherwise.

We note that checking (i)-(ii) can be done independently for every vertex, hence isConflictFree(G, Λ) runs in linear time $\mathcal{O}(n)$ for $n = |V|$. Intuitively, whenever the existentially quantified edge in (i) and (ii) of Definition 5.4 exists, a strategy that alternates between all these edges follows the given template. In addition, this strategy can also be extracted in linear time, as formalized next.

Proposition 5.1. *Given a game graph $G = (V, E)$ with conflict-free template $\Lambda \lhd (S, D, \mathbb{H})$ for a set of players $\mathtt{P}'$, a strategy profile $(\pi_p)_{p \in \mathtt{P}'}$ for $\mathtt{P}'$ that follows Λ can be extracted in time $\mathcal{O}(m)$, where m is the number of edges. This procedure is called* ExtractStrategy(G, Λ).

Proof. The proof is straightforward by constructing the strategy as follows. We first remove all unsafe and co-live edges from G and then construct strategies for players in $\mathtt{P}'$ that alternates between all remaining edges from every vertex. This strategy is well-defined as condition (i) in Definition 5.4 ensures that after removing all the unsafe and co-live edges a choice from every vertex remains. Moreover, if the vertex is a source of a live group edge, condition (ii) in Definition 5.4 ensures that there are outgoing edges satisfying every live group. Thereby, the constructed strategy indeed follows Λ. □

It is worth noting that ComputeCSM (and all its variants given in Section 5.2) always return conflict-free templates $\Psi_{\neg p}$ and Π_p by construction. Only when combining templates from all players conflicts may arise. However, conflict-freeness of such combined templates implies the existence of a strategy profile following them.

Corollary 5.2. *Given CSMs $(\Psi_{\neg p}, \Pi_p)_{p \in \mathtt{P}}$ in a game graph G, if the template $\bigwedge_{p \in \mathtt{P}} (\Psi_{\neg p} \wedge \Pi_p)$ is conflict-free, then there exists a strategy profile $(\pi_p)_{p \in \mathtt{P}}$ such that each π_p follows the corresponding p-template $\Psi_p \wedge \Pi_p$ with $\Psi_p = \bigwedge_{q \neq p} \Psi_{\neg q}|_{E_p}$.*

We note that the converse of Corollary 5.2 is not true, as there can be a strategy profile following $\Psi_p \wedge \Pi_p$ even when the corresponding templates are not conflict-free. However, this does not affect the completeness of our algorithm. Therefore, we focus our attention on ensuring conflict-freeness rather than the existence of such a strategy profile. Moreover, if such a strategy profile exists it will be retained by the conflict resolving mechanism of our negotiation framework, introduced next.

5.3.3 Resolving Conflicts

Given a conflict in $\Lambda = \bigwedge_{p \in \mathsf{P}}(\Pi_p \wedge \Psi_{\neg p}) \lhd (S, D, \mathbb{H})$ we now discuss how the modified specifications φ_p (as in Algorithm 5.4 of Algorithm 5.4) allows to resolve this conflict in the next iteration.

For this, first assume that $D = \emptyset$. In this case a conflict at a vertex exists because all of its available (live) edges are unsafe and should never be taken. Hence, an extracted strategy (via Proposition 5.1) is not well-defined (i.e., might get stuck in a vertex for which (i) of Definition 5.4 is false) or not ensured to be winning (i.e., will not be able to fulfill the liveness obligations in $\mathcal{H}$ if (ii) of Definition 5.4 is false).

In order to ensure strategies to be winning, templates need to be re-computed over a game graph where unsafe edges $e \in S$ in Λ are removed. Furthermore, by looking into the details of the computation of S within ComputeCSM, we see that unsafe edges always transition from the winning region $\mathsf{Win}_p = \langle\!\langle \mathsf{P} \rangle\!\rangle(\Phi_p)$ to its complement $\overline{\mathsf{Win}}_p = V \setminus \mathsf{Win}_p$, i.e., every (cooperatively winning) play should never visit states in $\overline{\mathsf{Win}}_p$. We therefore achieve the desired effect by adding the requirement $\Box\neg\mathsf{Win}$ to the specification, which obviously does not restrict the winning region, as ComputeCSM is ensured to not remove any cooperative solution (due to (i) in Definition 5.2).

This intuition generalizes to the case where $D \neq \emptyset$ as follows. Here, we need to resolve the game while ensuring that co-live edges $e \in D$ are only taken finitely often. Furthermore, co-live edges are computed by ComputeCSM s.t. they always transition to the set of vertices $\mathtt{C}_p$ that must only be seen finitely often along a winning play. In addition to Win_p, the set $\mathtt{C}_p$ is also be computed during the computation of D within ComputeCSM in Algorithm 5.4 of Algorithm 5.4. As for the unsafe-edge case, we can achieve the desired effect for recomputation by adding the requirement $\Diamond\Box\neg\mathtt{C}$ to the specification Φ_p (see Algorithm 5.4 of Algorithm 5.4). Again, this obviously does not alter the winning region of the game.

Remark 5.1. *We note that Algorithm 5.4 is slightly simplified, as the objective φ_p in Algorithm 5.4 of Algorithm 5.4 used as an input to ComputeCCS in latter iterations, is not a "plain" parity objective Parity($\mathbb{P}$). As ComputeCSM expects a parity game as an input, we need to convert φ_p into a parity objective by a simple reprocessing step. Luckily, both additional specifications can be dealt with using classical steps of Zielonka's algorithm [225], a well-known algorithm to solve parity games as discussed in Section 2.5.3, which is used as the basis for ComputeCSM. Concretely, we handle the $\Box\mathsf{Win}$ part of φ_p, by restricting the game graph G to Win and the $\Diamond\Box\neg\mathtt{C}$ part by assigning all vertices in $\mathtt{C}$ the highest odd priority $2d+1$. The correctness of these standard transformations follows from the same arguments as used to prove the correctness of similar steps of the ComputeCSM algorithm in Section 5.2.*

5.3.4 Properties of ComputeCCS

With this, we are finally ready to prove that (i) ComputeCCS always terminates in a finite number of steps, and (ii) upon termination, the computed CSMs indeed provide a CCS for the original game.

Termination. Intuitively, all local synthesis problems are performed over the same (possibly shrinking) game graph G. Therefore, there exists only a finite number of templates $\Lambda \lhd (S, D, \mathbb{H})$ over G, which, in the worst case, can all be enumerated in finite time.

Theorem 5.5. *Given a k-player game $\mathcal{G} = (G, (\Phi_p)_{p \in \mathtt{P}})$ with $\Phi_p = \mathit{Parity}(\mathbb{P}_p)$, Algorithm 5.4 always terminates in $\mathcal{O}(kn^6)$ time, where $n = |V|$.*

Proof. We prove termination via an induction on the lexicographically ordered sequence of pairs $(|\mathsf{Win}|, |\mathsf{Win} \setminus \mathtt{C}|)$. As the base case, observe that if $|\mathsf{Win}| = 0$ or $|\mathsf{Win} \setminus \mathtt{C}| = 0$, in the next iteration, the winning region for every controller player would be empty, implying $\Psi_{\neg p} \lhd (\emptyset, \emptyset, \emptyset)$ and $\Pi_p \lhd (\emptyset, \emptyset, \emptyset)$ for every $p \in \mathtt{P}$. Therefore, combination of all the templates is trivially conflict-free, and hence, isConflictFree returns True and ComputeCCS terminates.

Now for the induction step, suppose $|\mathsf{Win}| > 0$ and $|\mathsf{Win} \setminus \mathtt{C}| > 0$ in the *previous* iteration. If the combination of the templates is conflict-free, then ComputeCCS terminates. Suppose this is not the case. As G gets restricted to Win for this iteration (see Remark 5.1), unsafe edges can only occur if $\mathsf{Win}' \subset \mathsf{Win}$ (as they are by construction from Win' to $V \setminus \mathsf{Win}'$ s.t. the latter is a subset of Win), where Win' is the winning region computed in the *current* iteration. If $\mathsf{Win}' = \mathsf{Win}$ conflicts need to arise from co-live edges. As co-live edges are computed by ComputeCSM in a subgame that excludes all vertices with the highest odd priority (and therefore all vertices in $\mathtt{C}$ due to Remark 5.1), the existence of co-live edge conflicts implies the existence of new co-live edges, which implies that $|\mathsf{Win}' \setminus \mathtt{C}'| < |\mathsf{Win} \setminus \mathtt{C}|$. Therefore, $(|\mathsf{Win}|, |\mathsf{Win} \setminus \mathtt{C}|)$ always reduces (lexicographically) when conflicts occur. Hence, the algorithm terminates by induction hypothesis. Furthermore, as each iteration calls isConflictFree once and ComputeCSM k times which runs in $\mathcal{O}(|V|^4)$ time, Algorithm 5.4 terminates in $\mathcal{O}(k\,|V|^6)$ time. □

Soundness and Completeness. It remains to show that upon termination, the computed CSMs indeed provide a CCS for the original game. Furthermore, the local nature of templates ensures that the algorithm does not depend on the initial vertex, and hence, for every vertex v for which there exists a cooperatively winning strategy profile, i.e, for every $v \in \langle\!\langle \mathtt{P} \rangle\!\rangle \bigwedge_{p \in \mathtt{P}} \Phi_p$, the algorithm outputs a CCS even for the modified game graph (V, E, v). Moreover, if there are only two players, as we do not lose permissiveness by restricting the templates to player-specific edges, the resulting CCS is also maximal. This is formalized below.

Theorem 5.6. *Given a k-player game $\mathcal{G}$ with game graph $G = (V, E, v_0)$ and parity specifications $(\Phi_p)_{p \in \mathtt{P}}$, the specification profile $(\varphi_p)_{p \in \mathtt{P}}$ output by the procedure ComputeCCS($\mathcal{G}$) is CCS w.r.t. every game graph (V, E, v) with $v \in \langle\!\langle \mathtt{P} \rangle\!\rangle \bigwedge_{p \in \mathtt{P}} \Phi_p$. If $|\mathtt{P}| = 2$, $(\varphi_p)_{p \in \mathtt{P}}$ is also maximal.*

Proof. Let $\big((\varphi_p)_{p \in \mathtt{P}}, (\Lambda_p)_{p \in \mathtt{P}}\big)$ be the output of the procedure ComputeCCS($\mathcal{G}$). Note that in each iteration of ComputeCCS, the specifications $(\Phi_p)_{p \in \mathtt{P}}$ are modified in Algorithm 5.4 of Algorithm 5.4. Let Φ'_p the corresponding specification and let Ψ_p, $\Psi_{\neg p}$, and

Π_p be the corresponding templates used in the last iteration of ComputeCCS. Hence, $\varphi_p = \Psi_p \wedge (\Psi_{\neg p} \Rightarrow \Phi'_p)$ and $\Lambda_p = \Psi_p \wedge \Pi_p$ for each $p \in \mathtt{P}$. First let us show that the conjunction of the modified specifications Φ'_p still retain the same language as the original specifications Φ_p.

Claim 5.1. $\mathcal{L}(\bigwedge_{p\in\mathtt{P}} \Phi'_p) = \mathcal{L}(\bigwedge_{p\in\mathtt{P}} \Phi_p)$.

Proof. Observe that in every iteration, the procedure TempParity outputs the cooperative winning region Win_p and the co-Büchi region $\mathtt{C}_p$, i.e., the region that needs to be visited only finitely often along a winning play, for each player $p \in \mathtt{P}$. Hence, every play in the current Φ_p already satisfies the added terms $\Box\mathsf{Win}_p$ and $\Diamond\Box\neg\mathtt{C}_p$. Thereby, the addition of these terms to the specification does not exclude any cooperative winning play, hence, keeping $\bigwedge_{p\in\mathtt{P}} \Phi_p$ same in every iteration. Therefore, after the last iteration, we have $\mathcal{L}(\bigwedge_{p\in\mathtt{P}} \Phi'_p)$ equivalent to $\mathcal{L}(\bigwedge_{p\in\mathtt{P}} \Phi_p)$. ◁

Now, consider a game $\mathcal{G}'$ with game graph $G' = (V, E, v)$ with $v \in \langle\!\langle \mathtt{P} \rangle\!\rangle \bigwedge_{p\in\mathtt{P}} \Phi_p$ and specifications $(\Phi_p)_{p\in\mathtt{P}}$. We first show that the returned specification $(\varphi_p)_{p\in\mathtt{P}}$ forms a CCS for $\mathcal{G}'$, i.e., it is decentralized and sound (as in Definition 5.1), and then show maximality if $|\mathtt{P}| = 2$.

▶ **Decentralized.** We need to show that $v \in \langle\!\langle p \rangle\!\rangle \varphi_p$ for each $p \in \mathtt{P}$. By conflict-freeness of the combined template $\bigwedge_{p\in\mathtt{P}}(\Psi_{\neg p} \wedge \Pi_p)$ and by Corollary 5.2, there exists a strategy profile $(\pi_p)_{p\in\mathtt{P}}$ such that each π_p follows the corresponding p-template $\Psi_p \wedge \Pi_p$. Fix any $p \in \mathtt{P}$. Furthermore, by sufficiency (Definition 5.2(i)) of CSM $(\Psi_{\neg p}, \Pi_p)$ and due to the equivalence of the definitions as in Proposition 3.1, π_p is winning for $\Psi_{\neg p} \Rightarrow \Phi'_p$ from every vertex in $\langle\!\langle \mathtt{P} \rangle\!\rangle \Phi'$. As $v \in \langle\!\langle \mathtt{P} \rangle\!\rangle \bigwedge_{p\in\mathtt{P}} \Phi_p = \langle\!\langle \mathtt{P} \rangle\!\rangle \bigwedge_{p\in\mathtt{P}} \Phi'_p$ (by Claim 5.1), it holds that $v \in \langle\!\langle \mathtt{P} \rangle\!\rangle \Phi'_p$. Hence, $\pi_p \models_v (\Psi_{\neg p} \Rightarrow \Phi'_p)$ and $\pi_p \Vdash \Psi_p$. This means that $\pi_p \models_v \Psi_p \wedge (\Psi_{\neg p} \Rightarrow \Phi'_p)$, which is equivalent to $\pi_p \models_v \varphi_p$. Hence, $v \in \langle\!\langle p \rangle\!\rangle \varphi_p$.

▶ **Soundness.** Let $(\pi_p)_{p\in\mathtt{P}}$ be any strategy profile with $\pi_p \models_v \varphi_p$ for each $p \in \mathtt{P}$. We need to show that $(\pi_p)_{p\in\mathtt{P}} \models_v \bigwedge_{p\in\mathtt{P}}(\Phi_p)$. Observe that, by construction, we have $(\pi_p)_{p\in\mathtt{P}} \models_v \bigwedge_{p\in\mathtt{P}}(\varphi_p)$. Hence, it is enough to show that $\mathcal{L}(\bigwedge_{p\in\mathtt{P}} \varphi_p) \subseteq \mathcal{L}(\bigwedge_{p\in\mathtt{P}} \Phi_p)$.

By construction, we have $\mathcal{L}(\bigwedge_{p\in\mathtt{P}} \Psi_p) \subseteq \mathcal{L}(\bigwedge_{p\in\mathtt{P}} \Psi_{\neg p})$. Hence, it holds that

$$\begin{aligned}
\mathcal{L}(\textstyle\bigwedge_{p\in\mathtt{P}} \varphi_p) &= \mathcal{L}(\textstyle\bigwedge_{p\in\mathtt{P}} \Psi_p) \cap \mathcal{L}(\textstyle\bigwedge_{p\in\mathtt{P}} \Psi_{\neg p} \Rightarrow \Phi'_p) \\
&\subseteq \mathcal{L}(\textstyle\bigwedge_{p\in\mathtt{P}} \Psi_{\neg p}) \cap \mathcal{L}(\textstyle\bigwedge_{p\in\mathtt{P}} \Psi_{\neg p} \Rightarrow \Phi'_p) \\
&= \mathcal{L}(\textstyle\bigwedge_{p\in\mathtt{P}} \Psi_{\neg p} \wedge (\Psi_{\neg p} \Rightarrow \Phi'_p)) \\
&\subseteq \mathcal{L}(\textstyle\bigwedge_{p\in\mathtt{P}} \Phi'_p) \\
&= \mathcal{L}(\textstyle\bigwedge_{p\in\mathtt{P}} \Phi_p)
\end{aligned}$$

▶ **Maximality.** Now, suppose $|\mathtt{P}| = 2$, i.e., $\mathtt{P} = \{0, 1\}$. We already have shown that $\mathcal{L}(\bigwedge_{p\in\mathtt{P}} \varphi_p) \subseteq \mathcal{L}(\bigwedge_{p\in\mathtt{P}} \Phi_p)$. To show maximality, we need to show the other direction, i.e., $\mathcal{L}(\bigwedge_{p\in\mathtt{P}} \Phi_p) \subseteq \mathcal{L}(\bigwedge_{p\in\mathtt{P}} \varphi_p)$. As $\mathtt{P} = \{0, 1\}$, we have $\Psi_{\neg 0} = \Psi_1$ and $\Psi_{\neg 1} = \Psi_0$. Hence,

for each $p \in \mathtt{P}$, $\mathcal{L}(\varphi_p) = \mathcal{L}(\Psi_p) \cap \mathcal{L}(\Psi_{1-p} \Rightarrow \Phi_p)$. Furthermore, for each $p \in \mathtt{P}$, by permissiveness (Definition 5.2(iii)) of CSM $(\Psi_{\neg p}, \Pi_p)$, it holds that

$$\begin{aligned} & \mathcal{L}(\Phi_p) \subseteq \mathcal{L}(\Psi_{\neg p}) \\ \implies & \mathcal{L}(\Phi_p) \subseteq \mathcal{L}(\Psi_{\neg p}) \cap \mathcal{L}(\Psi_{\neg p} \Rightarrow \Phi_p) \\ & \qquad = \mathcal{L}(\Psi_{1-p}) \cap \mathcal{L}(\Psi_{1-p} \Rightarrow \Phi_p). \end{aligned}$$

Their conjunction gives,

$$\begin{aligned} \mathcal{L}(\Phi_0 \wedge \Phi_1) &\subseteq (\mathcal{L}(\Psi_1) \cap \mathcal{L}(\Psi_1 \Rightarrow \Phi_0)) \cap (\mathcal{L}(\Psi_0) \cap \mathcal{L}(\Psi_0 \Rightarrow \Phi_1)) \\ &= (\mathcal{L}(\Psi_0) \cap \mathcal{L}(\Psi_1 \Rightarrow \Phi_0)) \cap (\mathcal{L}(\Psi_1) \cap \mathcal{L}(\Psi_0 \Rightarrow \Phi_1)) \\ &= \mathcal{L}(\varphi_0) \cap \mathcal{L}(\varphi_1) = \mathcal{L}(\varphi_0 \wedge \varphi_1). \end{aligned}$$

Therefore, the CCS $(\varphi_p)_{p \in \mathtt{P}}$ is maximal. □

With this, by combining this with the properties of CCS from Corollary 5.2, the strategy choices for both players are fully decoupled. Moreover, since the ComputeCCS algorithm removes losing vertices in each iteration, if no solution exists, it terminates with an empty game graph and empty templates. This is summarized in the following *completeness* result.

Corollary 5.3. *In the context of Theorem 5.6, for any vertex v from which a winning strategy profile exists for the k-player parity game $(G, (\Phi_p)_{p \in \mathtt{P}})$, ComputeCCS also outputs a p-template $\Lambda_p = \Psi_p \wedge \Pi_p$ for each $p \in \mathtt{P}$ such that one can extract a winning strategy profile $(\pi_p)_{p \in \mathtt{P}}$ from v with each π_p following Λ_p. Conversely, if no such profile exists from any vertex, then all computed templates are empty, i.e., $\Lambda_p = \emptyset$ for all $p \in \mathtt{P}$.*

5.4 Utilizing Templates under Partial Observation

This section will *not* discuss the negotiation framework or computation of templates (CSMs) under partial observation. Instead, we assume that the interaction between players (i.e., the *plant model*), given as a game graph (as discussed in Section 2.5.2), is known during synthesis, but players have to actually *play* under partial observation which requires the extraction of a *partial observation strategy* from (full-observation) assumption and strategy templates (as obtained in Corollary 5.3). This complies with the concept of *output-feedback control* from engineering [147, 143, 201], where controller strategies are synthesized offline under full state information of the plant model, but need to operate based on measurements containing only partial state information.

To this end, we consider a partial observation setting where players cannot distinguish all vertices in the graph but still take turns when playing. We use the well known knowledge-based abstraction of games under partial observation [59] to build an abstraction over so-called *knowledge states* for each player, which collects all vertices of the game that a player can currently not distinguish (given the past history of the play). We then intersect all templates of vertices combined in these knowledge states. If the

resulting (knowledge-based) template is conflict-free, a strategy can be extracted in analogy to Proposition 5.1. The soundness of this approach follows directly from the known properties of knowledge-based abstractions and conflict-free templates.

5.4.1 Partial Observation Setting

We start by formalizing the partial observation setting we consider in this section. In this setting, every player $p \in \mathtt{P}$ has only *partial observation* over the vertex set V of a game graph, i.e., Player p cannot distinguish all vertices in V. In particular, for each player $p \in \mathtt{P}$, we consider to be given a partition of vertices $\Gamma_p \subseteq 2^V \setminus \emptyset$ which group all vertices that are indistinguishable for Player p. We formalize these partitions as follows.

Definition 5.5. An *observation partition* $\Gamma \subseteq 2^V \setminus \emptyset$ in a game graph $G = (V, E, v_0)$ is a collection of non-empty *observation sets* such that the following holds:

(i) $\bigcup_{\widehat{u} \in \Gamma} \widehat{u} = V$ and $\widehat{u} \cap \widehat{v} = \emptyset$ for all $\widehat{u} \neq \widehat{v} \in \Gamma$,

(ii) for all $\widehat{u} \in \Gamma$, there exists a player $p \in \mathtt{P}$ s.t. $\widehat{u} \subseteq V_p$.

Moreover, Γ induces a mapping $\Gamma^{\Uparrow} : V \to \Gamma$ s.t. $\Gamma^{\Uparrow}(v) := \widehat{v}$ iff $v \in \widehat{v}$.

The first condition in Definition 5.5 ensures that Γ is indeed a partition of V, while the second condition ensures that every observation set $\widehat{u} \in \Gamma$ contains vertices owned by the same player only. This means each player sees who is playing at each point in time, which is important to retain the turn-based nature of the game graph in the abstracted knowledge-based game graph defined later.

With the above definition of observation partitions, the synthesis problem for such k-player games under partial observation can now be formalized as follows.

Problem 5.1. *Given a k-player game $\mathcal{G} = (G, (\Phi_p)_{p \in \mathtt{P}})$ with an observation partition Γ_p for each player $p \in \mathtt{P}$, synthesize a winning strategy profile $(\pi_p)_{p \in \mathtt{P}}$ such that each π_p is* observation-preserving *under Γ_p, i.e., for all histories $\mathtt{h} = v_0 v_1 \dots v_n$ and $\mathtt{h}' = v'_0 v'_1 \dots v'_n$ with $\Gamma_p^{\Uparrow}(v_i) = \Gamma_p^{\Uparrow}(v'_i)$ for all $0 \leq i \leq n$, it holds that $\pi_p(\mathtt{h}) = \pi_p(\mathtt{h}')$.*

5.4.2 Knowledge-Based Abstractions

The key idea of knowledge-based abstractions is to build an abstract game graph where each state (so-called *knowledge state*) collects all vertices that a player cannot distinguish based on the history of the play so far. To this end, we need to capture two important aspects of partial observation settings. First, each player $p \in \mathtt{P}$ knows the full game graph structure, i.e., the interaction between players is common knowledge. Hence, based on the knowledge from the history of the play, Player p can deduce which vertices of an observation set might be currently active in the game. Second, Player p can only distinguish their own moves, i.e., labels on their own edges, but not the moves of other

players. In order to capture this, we define a new p-labelling function L_p based on the labelling function[1] L of the game graph that only distinguishes Player p's moves.

Definition 5.6. Given a game graph $G = (V, E, v_0, L)$ with labelling function $L : E \to \Sigma$ and $\bot \notin L(E)$, the *p-labelling function* $L_p : E \to \Sigma_p$ with $\Sigma_p := L(E_p) \cup \{\bot\}$ is defined as

$$L_p(e) := \begin{cases} L(e), & \text{if } e \in E_p \\ \bot. & \text{otherwise} \end{cases}$$

This p-labelling function L_p maps all edges owned by Player p to their original labels, while all other edges are mapped to a special label $\bot$ indicating that Player p cannot distinguish these moves. Using this p-labelling function and the given observation partition for Player p, we can now build the knowledge-based abstraction for Player p the usual way [59].

Definition 5.7. A (knowledge-based) p-abstraction of a game graph $G = (V, E, v_0, L)$ under the observation partition Γ is a tuple $\widehat{G}_p = (\widehat{\Gamma}, \widehat{v}_0, \Delta)$ such that

- $\widehat{\Gamma} = \{\widehat{u} \subseteq V \mid \emptyset \neq \widehat{u} \subseteq \widehat{u}' \text{ for some } \widehat{u}' \in \Gamma\}$;
- $\widehat{v}_0 := \Gamma^{\Uparrow}(v_0)$;
- $\Delta \subseteq \widehat{\Gamma} \times \Sigma_p \times \widehat{\Gamma}$ with $(\widehat{u}, \sigma, \widehat{v}) \in \Delta$ iff there exists $\widehat{v}' \in \Gamma$ s.t. $\widehat{v} = \widehat{v}' \cap \{v \in E(\widehat{u}) \mid L_p(u, v) = \sigma\}$.

We observe that the state space $\widehat{\Gamma}$ of $\widehat{G}$ is the collection of all non-empty subsets of observation sets in Γ. Furthermore, it induces a corresponding mapping $\widehat{\Gamma}^{\Uparrow} \colon V \to 2^{\widehat{\Gamma}}$ such that $\widehat{\Gamma}^{\Uparrow}(v) := \{\widehat{v} \in \widehat{\Gamma} \mid v \in \widehat{v}\}$ for all $v \in V$. We denote by $\widehat{\Gamma}_q$ the set of knowledge-states $\widehat{u}$ in $\widehat{G}_p$ which are owned by Player q, i.e., $\widehat{u} \subseteq V_q$.

5.4.3 Extracting Partial Observation Strategies

In this section, we show how a p-template in a game graph G can be abstracted into a p-template in a knowledge-based p-abstracted game graph $\widehat{G}_p$. In order to formalize this abstraction, we need to first extend templates to labelled game graphs.

Labelled Templates. Given a game graph $G = (V, E, v_0, L)$, we define the *enabled state-action pairs* as the set $A := \{(v, \sigma) \in V \times \Sigma \mid \exists (v, v') \in E.\ L(v, v') = \sigma\}$. With this, one can define a *labelled template* as a tuple $\Lambda \lhd (S, D, \mathbb{H})$ with $S \subseteq A$, $D \subseteq A$, and $\mathbb{H}$ containing pairs of the form $(R, \mathcal{H})$ for $R \subseteq V$ and $\mathcal{H} \subseteq A$. Furthermore, this naturally defines a labelled p-template if S, D and all $\mathcal{H}$ only contain state-action pairs (v, σ) with $v \in V_p$. Conflict-freeness and strategy extraction of labelled templates can also be defined in analogy to Definition 5.4 and Proposition 5.1.

[1] Note that the labels (as defined by the labelling function L of the game graph) are not needed and hence, not mentioned in the full observation settings, as moves are then uniquely determined by the edge successors.

We note that, due to the deterministic definition of transition labels in L, every template over G corresponds to a unique labelled template. Therefore, in this section, we will only consider labelled templates.

Knowledge-Based Templates. To define the knowledge-based abstraction of a labelled template, we first extend the induced mapping $\widehat{\Gamma}^{\Uparrow}$ to sets of state-action pairs $\alpha \subseteq A$ in the obvious way, i.e., $\widehat{\Gamma}^{\Uparrow}(\alpha) := \{(\widehat{v}, \sigma) \in A \mid (v, \sigma) \in \alpha, v \in \widehat{v}\}$. With this, a labelled template Λ in G naturally corresponds to an abstracted template $\widehat{\Lambda}$ in $\widehat{G}_p$ as follows.

Definition 5.8. Given a game graph $G = (V, E, v_0, L)$, its corresponding p-abstraction $\widehat{G}_p = (\widehat{\Gamma}, \widehat{v}_0, \Delta)$ and a labelled template $\Lambda \triangleleft (S, D, \mathbb{H})$ over G, we define the corresponding abstracted template $\widehat{\Lambda} \triangleleft (\widehat{S}, \widehat{D}, \widehat{\mathbb{H}})$ over $\widehat{G}_p$ as follows:

- $\widehat{S} := \widehat{\Gamma}^{\Uparrow}(S)$,
- $\widehat{D} := \widehat{\Gamma}^{\Uparrow}(D)$, and
- $\widehat{\mathbb{H}} := \{(\widehat{R}, \widehat{\mathcal{H}}) \mid (R, \mathcal{H}) \in \mathbb{H}\}$ with $\widehat{R} = \{\widehat{v} \in \widehat{\Gamma} \mid \widehat{v} \cap R \neq \emptyset\}$ and $\widehat{\mathcal{H}} = \{\widehat{\Gamma}^{\Uparrow}(H) \mid H \in \mathcal{H}\}$.

Given the above construction, the computed p-template Λ_p from Corollary 5.3 in the original game graph can be used to obtain an abstracted p-template $\widehat{\Lambda}_p$ in the p-abstracted game graph $\widehat{G}_p$. If $\widehat{\Lambda}_p$ is *conflict-free* in $\widehat{G}_p$, one can then extract a Player p strategy $\widehat{\pi}_p$ following $\widehat{\Lambda}_p$ in $\widehat{G}_p$ via Proposition 5.1. As knowledge-based abstractions ensure that every history of a play corresponds to a unique Player p state, $\widehat{\pi}_p$ is a well-defined observation-preserving strategy in G. Moreover, due to Corollary 5.3, a strategy profile $(\widehat{\pi}_p)_{p \in \mathtt{P}}$ containing strategies extracted this way for all players is a winning strategy profile in G. While this procedure does not guarantee completeness due to possible conflicts in the abstracted templates, it provides a sound method to solve Problem 5.1, as summarized below.

Corollary 5.4. *Given a k-player game $\mathcal{G} = (G, (\Phi_p)_{p \in \mathtt{P}})$ with an observation partition Γ_p for each player $p \in \mathtt{P}$, let Λ_p be the p-template computed as in Corollary 5.3 in G and $\widehat{\Lambda}_p$ its corresponding abstracted template in the knowledge-based p-abstraction $\widehat{G}_p$. If $\widehat{\Lambda}_p$ is conflict-free in $\widehat{G}_p$ for all $p \in \mathtt{P}$, then the strategy profile $(\widehat{\pi}_p)_{p \in \mathtt{P}}$ with $\widehat{\pi}_p \Vdash \widehat{\Lambda}_p$ is an observation-preserving winning strategy profile in G.*

5.5 Experimental Evaluation

To demonstrate the effectiveness of our approach, we conducted experiments using a prototype tool CoSMo (**Co**ntracted **S**trategy **M**ask Nego**t**iation) [2] that implements the negotiation framework (Algorithm 5.4) for computing CCS for mult-player parity games. All experiments were performed on a computer equipped with an Apple M1 Pro 8-core CPU and 16GB of RAM.

5.5.1 Factory Benchmark

We evaluate the performance of our tool CoSMo with a benchmark suite of a grid-like factory scenario involving two robots, R_1 and R_2, operating in a shared workspace. Both robots can move in four directions (up, down, left, right) and have individual objectives to navigate the workspace while avoiding collisions with each other and with the walls. We consider two types of objectives for the robots: (i) Büchi objectives requiring that robots R_1 and R_2 should visit the upper-right and upper-left corners, respectively, of the grid infinitely often, while ensuring that they never occupy the same location simultaneously and do not bump into a wall. (ii) Parity objectives requiring one robot to visit certain locations infinitely often whenever the other robot visits specific locations infinitely often, while still avoiding collisions and walls (e.g., if R_1 visits location A infinitely often, then R_2 must visit location B infinitely often).

Benchmark Generation. To generate problem instances with different computational difficulty, we have generated a comprehensive set of 2357 factory benchmark instances. Our benchmark generator takes four parameters to change the characteristics of the game graph: the number of columns x, the number of rows y, the number of walls w, and the maximum number of one-way corridors c. Given these parameters, the workspace of the robots is constructed as follows: first, w horizontal walls, i.e., walls between two adjacent rows, are generated randomly. We ensure that there is at least one passage from every row to its adjacent rows (if this is not possible with the given w, then w is set to the maximum possible number for the given x and y). Next, for each passage from one row to another, we randomly designate it to be a one-way corridor. For example, if a passage is an up-corridor, then the robots can only travel in the upward direction through this passage. We ensure that the grid has at most c one-way corridors. Given such a grid, we generate a game graph for two robots, denoted R_1 and R_2, that navigate the grid starting at the lower-left and lower-right corners of the grid, respectively. In this scenario the robots only interact explicitly via their shared workspace, i.e., possibly blocking each other's way to the target. We consider two types of objectives for the robots as mentioned before. An illustration of one such benchmark is depicted in Figure 5.3 (left).

Experimental Results. In a first set of experiments, we ran our tool on all the factory benchmarks instances and plot all average run-times per grid-size (but with varying parameters for c and w) in Figure 5.3 (right). We see that CoSMo takes significantly more time for parity objectives compared to Büchi objectives. That is because computing templates for Büchi games takes linear time in the size of the games whereas the same takes biquadratic time for parity games. Furthermore, the templates computed for Büchi objectives do not contain co-liveness templates, and hence, they do not raise conflicts in most cases. However, templates for parity objectives contain all types of templates and hence, typically need several rounds of negotiation.

In a second set of experiments, we compared the performance of CoSMo, with the related tool Agnes implementing the contract-based distributed synthesis method by Majumdar et al. [146]. Unfortunately, Agnes can only handle Büchi specifications and

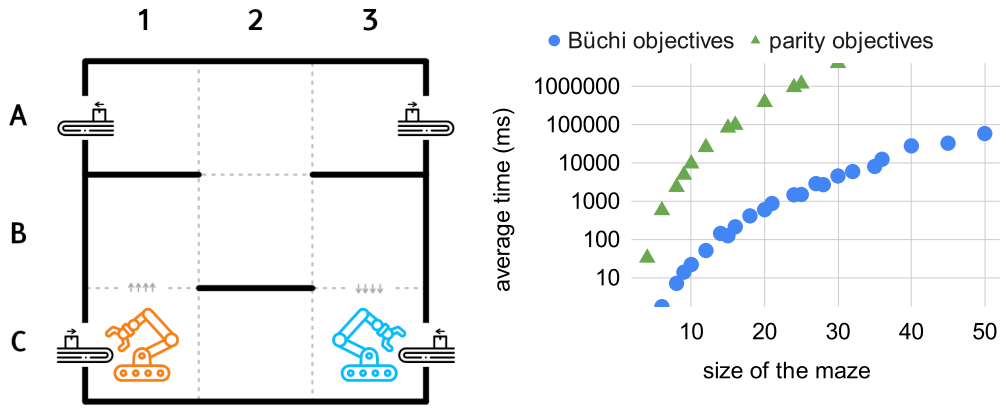


Figure 5.3: Left: Example of a factory benchmark with parameters $x = 3$, $y = 3$, $w = 3$, and $c = 2$. Solid lines denote walls, little up- and downward pointing arrows indicate one-way corridors. Right: Data points for factory benchmarks with Büchi objectives (blue circles) and parity objectives (green triangles) describing average execution time over all instances with the same grid size. The y-axis is given in log-scale.

resulted in segmentation faults for many benchmark instances. We have therefore only report computation times for all instances that have not resulted in segmentation faults.

The experimental results are summarized in Figures 5.4 and 5.5. As CoSMo implements a complete algorithm, it provably only reports that a given benchmark instance is unrealizable, if it truly is unrealizable, i.e., for 1.67% of the considered 120 instances. However, Agnes reports unrealizability in $36,67\%$ of its instances (see Figure 5.4 (left)), resulting in many false-negatives. Similarly, as CoSMo is ensured to always terminate, we see that all considered instances have terminated in the given time bound. While, Agnes typically computes a solution faster for a given instance (see Figure 5.5 (left)), it enters a non-terminating negotiation loop in $13,34\%$ of the instances (see Figure 5.4 (right)). This happens for almost all considered grid sizes, as visible from Figure 5.5 (right) where all non-terminating instances are included in the average after being mapped to $300s$, which was used as a time-out for the experiments.

While our experiments show that Agnes outperforms CoSMo in terms of computation times when it terminates on realizable instances (see Figure 5.5 (left)), it is unable to synthesize strategies either due to conservatism or non-termination in almost 50% of the considered instances (in addition to the ones which returned segmentation faults and which are therefore not included in the results). In addition to the fact that Agnes can only handle the small class of Büchi specifications while CoSMo can handle parity objectives, we conclude that the solution of CoSMo for the given synthesis task is much more satisfactory.

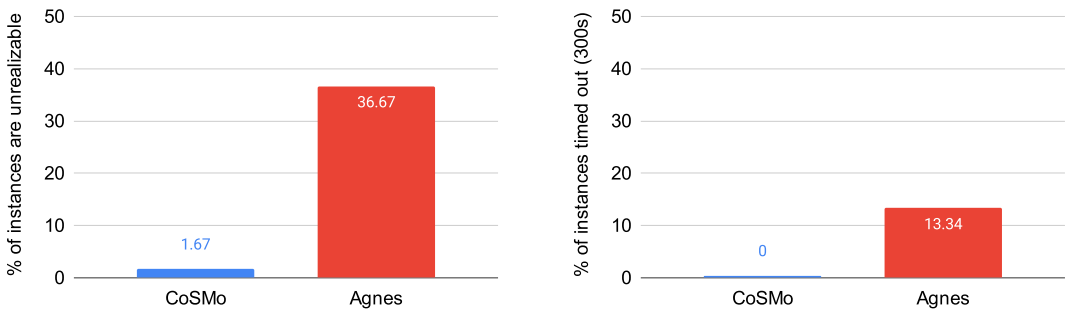


Figure 5.4: Left: Percentage of instances on which the respective tool reports unrealizability after termination. Right: Percentage in which the respective tool does not terminate. Both numbers are mutually exclusive.

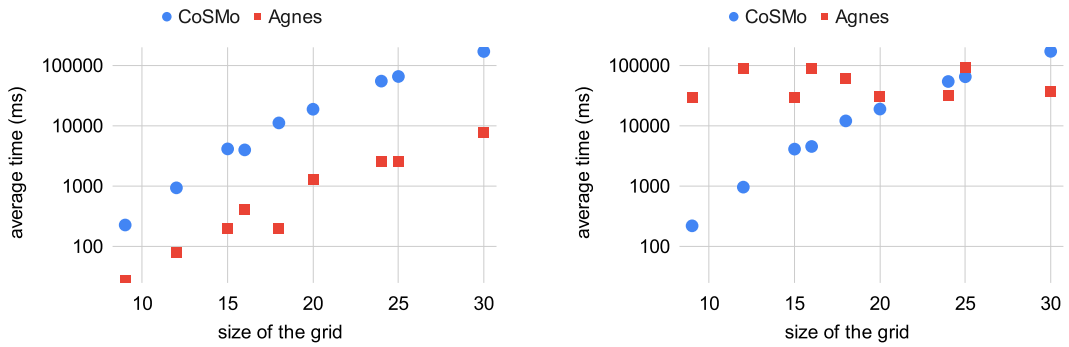


Figure 5.5: Average computation times over all instances with the same grid size for CoSMo (blue circles) and Agnes (red squares) without timed-out instances (left) and with timed-out instances mapped to the time-out of 300s (right). The y-axis is given in log-scale.

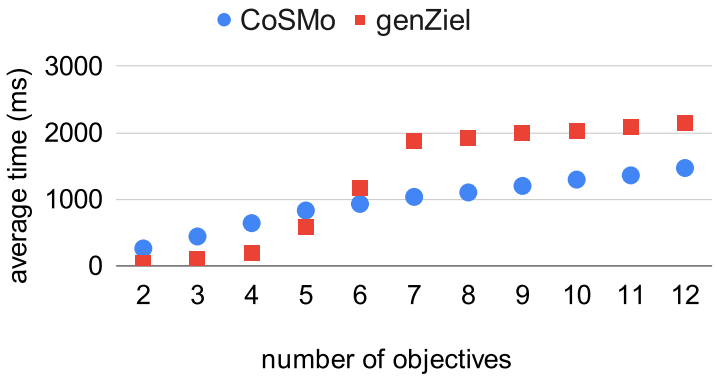


Figure 5.6: Experimental results over 2244 games when new parity objectives are added *incrementally one-by-one*. Data points give the average execution time (in ms) over all instances with the same number of parity objectives for CoSMo (blue circles) and genZiel [63] (red squares).

5.5.2 Incremental Synthesis and Negotiation

While the previous section evaluates our method for a single, static synthesis task, we want to now emphasize how the permissiveness of our approach allows for online adaptation of strategies. To this end, we assume that Algorithm 5.4 has terminated on the input (G, Φ_0, Φ_1) and realizable CSM's (Ψ_1, Π_0) and (Ψ_0, Π_1) have been obtained. Then a new parity objective Φ'_p over G arrives for component p, for which additional CSM $(\Psi'_{\neg p}, \Pi'_p) :=$ ComputeCSM(G, Φ'_p) can be computed. It is easy to observe that if $(\Psi'_{\neg p}, \Pi'_p)$ does not introduce new conflicts, no further negotiation needs to be done and the CSM of component p can simply be updated to $(\Psi_{\neg p} \wedge \Psi'_{\neg p}, \Pi_p \wedge \Pi'_p)$. Otherwise, we simply re-negotiate by running more iterations of Algorithm 5.4.

We note that, algorithmically, this variation of the problem requires solving a chain of *generalized* parity games, i.e., a parity game with a conjunction of a finite number of parity objectives. We therefore compare the performance of CoSMo on such synthesis problems to the best known solver for *generalized* parity games, i.e, genZiel by Chatterjee et al. [63] and implemented by Bruyère et al. [49]. Similar to our approach, genZiel is complete and based on Zielonka's algorithm. However, it solves one *centralized* cooperative game for the conjunction of all players objectives.

Synthesis Benchmark Generation. We have generated 2244 benchmarks from the games used for the Reactive Synthesis Competition (SYNTCOMP) [119] by adding randomly generated parity objectives to given parity games. The random generator takes two parameters: game graph G and maximum priority m. It selects 50% of the vertices in G randomly. These vertices are assigned priorities ranging from 0 to m (including 0 and m) such that $1/m$-th of those vertices are assigned priority 0 and $1/m$-th are assigned priority 1 and so on. The other vertices are assigned random priorities.

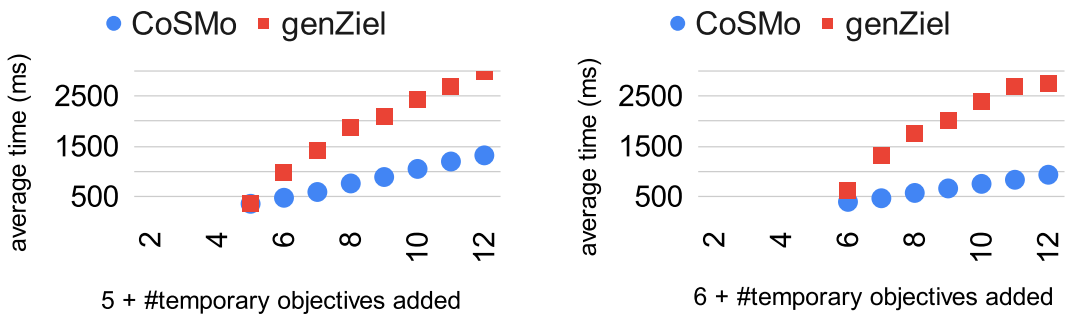


Figure 5.7: Variation of the experiment in Figure 5.6 with either 5 (left) or 6 (right) long-term objectives.

Comparative evaluation. Figure 5.6 shows the average computation time of GENZIEL and COSMO when objectives are added *incrementally one-by-one*, i.e., the game was solved with ℓ objectives, then one more objective was added, and the game was solved again. We see that for a low number of objectives, the negotiation of contracts in a distributed fashion by COSMO adds computational overhead, which reduces when more objectives are added. However, as more objectives are added the chance of the winning region to become empty increases. This gives GENZIEL an advantage, as it can detect an empty winning region very quickly (due to its centralized computation). In order to separate the effect of (i) the increased number of re-computations and (ii) the shrinking of the winning region induced by an increased number of incrementally added objectives, we conducted a separate experiment where added objectives are allowed to disappear again after some time. Here, we consider benchmarks with a fixed number of long-term objectives, and iteratively add just one *temporary* objective at a time. The results are summarized in Figure 5.7 when the number of long-term objectives are 5 (left) and 6 (right). We see that in this scenario COSMO clearly outperforms GENZIEL while performing computations in a distributed manner and returning strategy templates.

5.6 Related Work

Assume-guarantee contracts in multi-player games have been widely used to structure by encoding the assumptions each player makes about the others [60, 44, 10, 35, 98, 146, 31, 76]. While several frameworks assume some form of rationality among players [60, 101, 44, 77, 99], others employ explicit assume-guarantee reasoning [98, 146, 144] to compute formal contracts that specify allowed behaviors without constraining individual rationality. While Finkbeiner and Passing [98] employs a centralized bounded-synthesis formulation to find global contracts, other approaches use distributed negotiation frameworks [146, 144]. The work by Mainhardt and Schmuck [144] builds on concepts from supervisory control, while other frameworks, such as the work of Majumdar et al. [146], use environment assumptions following the works of Chatterjee et al. [57], but

their resulting contracts are not maximally cooperative. A detailed comparison between supervisory control and reactive synthesis can be found in related papers [84, 196, 150].

Overall, the existing literature provides rich foundations for distributed reactive synthesis under ω-regular objectives, but current frameworks either require centralization, assume rational player behavior, or compute non-maximal assume-guarantee contracts. Our work advances this state-of-the-art by developing a sound and complete distributed synthesis algorithm that constructs maximally cooperative assume-guarantee contracts without imposing rationality constraints on players.

Chapter 6

Permissive Human-Robot Interaction

While Chapter 3 formalized permissive assumptions for monolithic systems, and Chapters 4 and 5 extended this to permissive assume-guarantee based contracted specifications for distributed systems, this chapter focuses on utilizing the permissiveness of the developed theory in the context of human-robot interaction (HRI). In particular, we consider a human-robot interaction scenario where both the robot and the human, interacting in a shared environment, have their own independent tasks, which are not known to each other. In such a setting, we employ contracted strategy masks (CSMs), developed in Chapter 5, to capture all cooperative behaviors of both human and robot that enable a permissive interaction in which the robot can reliably satisfy its own task while effectively collaborating with the human.

Recall from Chapter 5 that CSMs consist of two components: (i) an *assumption template* on the environment (in this case, the human) that captures all cooperative behaviors of other agents that enable the controller (in this case, the robot) to satisfy its task, and (ii) a *strategy template* for the controller that prescribes how to act under the assumed cooperative behaviors. With this in mind, our framework leverages the permissiveness of CSMs to capture two key capabilities: (i) adaptation of the robot's strategy within its strategy template based on local observations of human behavior during interaction, and (ii) a tunable mechanism for providing feedback to the human when necessary, i.e., when the human's behavior consistently does not align with the assumption template. This balance minimizes interference with the human's autonomy while ensuring persistent robot task satisfaction even under conflicting human tasks.

The rest of this chapter is organized as follows. We first formalize the problem setup in Section 6.1, introducing the reactive-planning domain that models human-robot interaction, the temporal-logic tasks, and the overall problem statement. In Section 6.2, we present our proposed framework, detailing how CSMs are employed to formulate both the outlined strategy adaptation and feedback mechanisms. We then validate our approach in a real-world block-manipulation task using a Franka Emika Panda robotic arm and in the Overcooked-AI benchmarks in Section 6.3. Finally, we review related

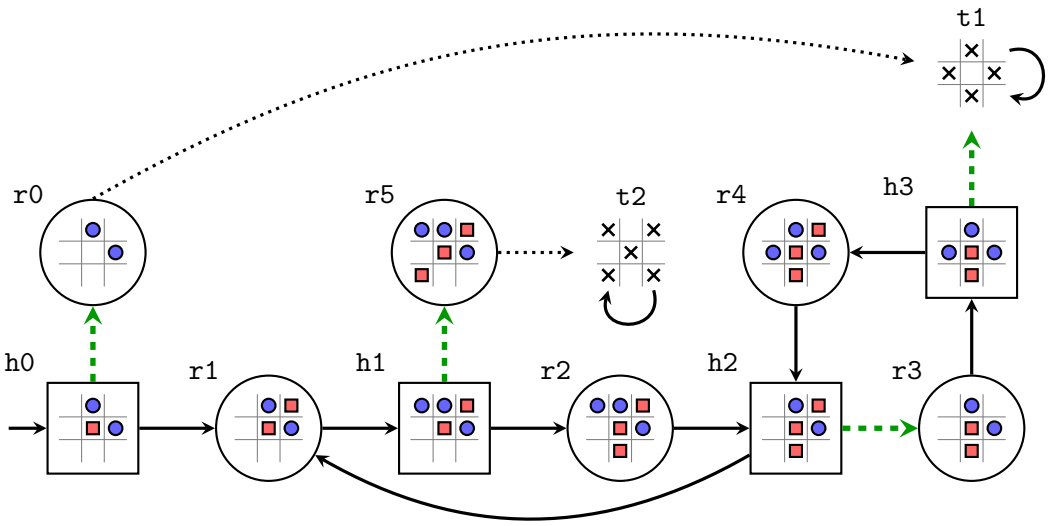


Figure 6.1: An example of a partial reactive planning domain for turn-based human-robot interaction in a grid world. The robot controls the circle states, while the human controls the square states. Each state contains a 3×3 grid showing the current positions of the human-placed objects (red squares) and the robot-placed objects (blue circles). Directed edges represent possible actions leading to successor states. The robot's objective is to repeatedly reach states with majority-occupied cells where the placed objects are non-adjacent (i.e., no two occupied cells are neighbors), as illustrated in states `t1` and `t2`. Green dashed edges denote *live* actions, which are the suggested actions by the strategy templates, while dotted edges denote sequences of actions that lead to the target states.

work in Section 6.4.

6.1 Problem Setup

We focus on a turn-based human-robot interaction scenario, where the robot and the human alternately act in a shared environment. Given a high-level temporal task for the robot, our goal is to develop a framework that enables the robot to adapt its strategy online to the human's observed behavior, and to provide feedback in a principled, tunable manner, so that the robot's task is reliably satisfied even when the human pursues an independent objective whose chosen strategy may conflict with the robot's progress. Let us first formalize this problem by providing all necessary components in the following subsections.

6.1.1 Reactive Planning Domain

We model the interaction between the robot and the human as a *reactive planning domain* represented by an alternating two-player game graph $G = (V, E, v_0, L)$ over `AP` (as

introduced in Section 2.5), where:

- $V = V_r \cup V_h$ is a set of states, partitioned into robot states V_r and human states V_h;
- $v_0 \in V$ is the initial state;
- $E = E_r \cup E_h$ is the set of actions (modeled as directed edges), partitioned into robot actions $E_r \subseteq V_r \times V_h$ and human actions $E_h \subseteq V_h \times V_r$;
- $\mathtt{AP}$ is the set of task-related propositions that can either hold or not hold in a given state.
- $L\colon V \to 2^{\mathtt{AP}}$ is a labeling function that labels each state with the set of propositions that hold in that state.

The planning domain can be specified using the planning domain description language (PDDL) [115], a standard language in the field of AI planning. In a PDDL description, a state captures relevant objects and their locations, while actions are defined in terms of preconditions and effects. A PDDL description can be automatically translated into a reactive planning domain (i.e., a two-player game graph) by enumerating all possible states and actions, as done in prior work [158, 159, 118, 199, 224].

A run $\rho = s_0 s_1 s_2 \ldots$ of the planning domain is an infinite sequence of states such that $s_0 = v_0$ and for all $i \geq 0$, $(s_i, s_{i+1}) \in E$, i.e., there is an action that takes the system from state s_i to state s_{i+1}. The run ρ induces a *trace* $L(\rho) = L(s_0)L(s_1)L(s_2)\ldots$, an infinite word over $2^{\mathtt{AP}}$, which is a sequence of labels corresponding to the states in the run. We assume that the robot and human take turns to act, i.e., if $s_i \in V_r$, then $s_{i+1} \in V_h$, and vice versa.

A robot *strategy* $\pi_r\colon V^* V_r \to E_r$ is a function that maps a sequence of states (representing the history of the interaction) ending in a robot state to the action that the robot should take. A run $\rho = s_0 s_1 s_2 \ldots$ is said to be a π_r-run if for all $i \geq 0$, whenever $s_i \in V_r$, then $s_{i+1} = \pi_r(s_0 s_1 \ldots s_i)$. A human strategy π_h and a π_h-run are defined similarly. Note that these notations are similar to those introduced in Section 2.5.

Example 6.1. Figure 6.1 shows a partial view of a turn-based human–robot interaction modeled as a reactive planning domain. Each circle (e.g, $\mathtt{r0}, \mathtt{r1}, \ldots$) corresponds to a robot state in V_r and each square (e.g., $\mathtt{h0}, \mathtt{h1}, \ldots$) corresponds to a human state in V_h. Inside each node, the 3×3 grid represents the environment: red squares denote human-placed objects, and blue circles denote robot-placed objects. Edges correspond to actions in E, alternating between human and robot moves according to the turn-based interaction. For example, the edge from state $\mathtt{r2}$ to state $\mathtt{h2}$ represents a robot action that removes a (blue circle) object from cell $(1, 1)$ (i.e., 1st row, 1st column). Figure 1.5 shows an image of the same action being executed by a Franka robotic platform in a real-world setting.

A labeling function L can be defined to capture task-related propositions in $\mathtt{AP}$. For instance, let us consider the propositions $\mathtt{AP} = \{\mathtt{adj}, \mathtt{diag}, \mathtt{major}\}$, where $\mathtt{adj}$ indicates

that no two objects are adjacent (horizontally or vertically), `diag` indicates that a diagonal is fully occupied, and `major` indicates that at least 4 out of 9 cells are occupied (majority-occupied). Hence, `adj` holds only in states `r0`, `t1`, `t2`; `diag` holds in states `r5`, `t2`; and `major` holds in every state except `h0` and `r0`. From the initial state `h0`, a possible run is $\rho = \texttt{h0}\,(\texttt{r1}\,\texttt{h1}\,\texttt{r2}\,\texttt{h2})^{\omega}$, which induces the trace $L(\rho) = \{\}\,\{\texttt{major}\}^{\omega}$. ⌟

6.1.2 Temporal Tasks as LTL formulas

To express robot/human tasks, we use LTL formulas (as introduced in Section 2.2.2) over the set of atomic propositions `AP`. We say a run ρ of a planning domain satisfies an LTL formula φ, denoted $\rho \models \varphi$, if the trace $L(\rho)$ satisfies the formula φ.

Example 6.2. Consider again the reactive planning domain in Example 6.1 with atomic propositions $\texttt{AP} = \{\texttt{adj}, \texttt{diag}, \texttt{major}\}$. Suppose the robot's task is to repeatedly reach states where no two objects are adjacent and at least four cells are occupied, which can be expressed by the LTL formula $\varphi_r = \square\diamondsuit(\texttt{adj} \wedge \texttt{major})$. A human's task could be to repeatedly reach a state where a diagonal is fully occupied, expressed by the LTL formula $\varphi_h = \square\diamondsuit\,\texttt{diag}$. A run that repeatedly visits states `t2` satisfies both tasks, while a run that eventually only loops in state `t1` only satisfies the robot's task. ⌟

6.1.3 Problem Statement

We are interested in the scenario of human-robot interaction where both the robot and the human have their own independent tasks, which are not known to each other. In such settings, each agent may follow a strategy that, even if unintentionally, can block or halt progress toward the other's task. Our goal is to develop a framework that enables the robot to adapt its strategy based on local observations and give feedback to the human when necessary, in order to persistently satisfy its own task over time.

Problem 6.1. *Given a human-robot interaction modeled as a reactive planning domain G with an LTL task φ for the robot in the presence of a human pursuing an unknown latent task, develop a framework that*

(a) ***captures*** *all cooperative behaviors of the human that enable the robot to satisfy its task φ;*

(b) ***adapts*** *the robot's strategy based on local observations of the human's strategic behavior during interaction;*

(c) ***incorporates*** *a tunable mechanism for providing feedback to the human when needed, thereby facilitating the robot's progress toward fulfilling its task φ.*

Example 6.3. Consider again the interaction in Example 6.1, where the robot's task is to repeatedly reach states with majority-occupied cells in which the placed objects are non-adjacent, as expressed by the LTL formula $\varphi_r = \square\diamondsuit(\texttt{adj} \wedge \texttt{major})$ in Example 6.2. Suppose the human, pursuing its own latent task, repeatedly places objects along a

diagonal in the grid. From the robot's perspective, this behavior can lead to a run satisfying φ_r, and thus this (unintentional) cooperation of the human should be exploited by the robot to fulfill φ_r.

However, in order to do so, the robot can not commit to a single strategy in advance to ensure that φ_r is satisfied regardless of the human's behavior. To see this, let us assume that the robot chooses to follow a fixed strategy which tries to satisfy φ_r by reaching a configuration as in `t1`. If the human still attempts to form a diagonal, their interaction will get stuck in a cycle where task φ_r will never be satisfied. Instead, the robot must *adapt* its strategy by recognizing, from local observations, that the human is systematically filling the diagonal. It should then autonomously choose actions that lead to states where φ_r is satisfied by reaching a configuration as in `t2`, which can accommodate the human's diagonal placements while still satisfying φ_r. This, however, is only possible because φ_r and the humans latent task can indeed be satisfied simultaneously.

If, however, both objectives are conflicting, it does not suffice for the robot to adapt to the human behavior. As an example, consider a scenario where the human persistently takes actions `h2` $\rightarrow$ `r3` and `h3` $\rightarrow$ `r4`. In such cases, the robot must recognize that the human's behavior can no longer be exploited for (unintended) cooperation. The robot then gives feedback to the human—e.g., requesting to remove the object at cell $(2, 2)$ (by taking action `h3` $\rightarrow$ `t1`). ⌟

6.2 Adaptability and Feedback Mechanisms via CSMs

To reason about the strategic behavior of the robot for the LTL task, we reduce the planning domain along with the LTL task to a two-player game between the robot and human, as commonly done in the literature [24]. We then leverage the *contracted strategy masks* (CSMs) from Section 5.2 to address the challenges outlined in Problem 6.1.

6.2.1 Planning Domain to Parity Games

As a first step, we state a well-known reduction from planning domains with LTL tasks to two-player parity games. As the planning domain G in Section 6.1.1 is already structured as an alternating two-player game graph between the robot and human, we only need to incorporate the LTL task φ into this game. With the results in Propositions 2.1 and 2.2, it is easy to generalize this reduction to our setting, giving us the following proposition.

Proposition 6.1. *Given a reactive planning domain G and an LTL formula φ over* `AP`*, we can construct a parity game $\mathcal{G} = (G', Parity(\mathbb{P}))$ such that there is a bijective correspondence between the runs of G and the plays of G', and a run ρ of G satisfies φ if and only if the corresponding play ρ' of G' is winning.*

6.2.2 Permissive Templates in CSMs

Let us recall that a CSM is defined via *permissive templates* (as formalized in Sections 3.2 and 5.2.1), which generalizes the notion of a strategy by succinctly representing an infinite

family of strategies through local constraints on the agent's actions. Formally, one can see a template Λ for agent p consists of the following types of constraints:

- *Unsafe actions* $S \subseteq E_p$: actions that the agent is prohibited from taking;
- *Co-live actions* $D \subseteq E_p$: actions that may only be taken finitely many times along any run;
- Conditional Live groups consisting of live groups $\mathcal{H} \subseteq 2^{E_p}$: sets of *live actions* such that, if the source state of some $H \in \mathcal{H}$ is visited infinitely often, the agent must take at least one action from the set infinitely often.

Recall that a run ρ is said to satisfy a template Λ, if it satisfies all the specified constraints. Further, a strategy π is said to follow a template Λ, denoted $\pi \Vdash \Lambda$, if all π-runs satisfy Λ.

As shown in Section 5.2, given a parity game between a robot and a human with an ω-regular specification (obtained from the LTL task φ), one can compute a CSM (Ψ_h, Π_r) for the robot consisting of an assumption template Ψ_h for the human and a strategy template Π_r capturing cooperative behaviors between the robot and human. This is formalized in the following proposition which directly follows from Definition 5.2 and Theorem 5.4.

Proposition 6.2. *Given a parity game* $\mathcal{G} = (G, Parity(\mathbb{P}))$*, a CSM* (Ψ_h, Π_r) *can be computed in polynomial time such that*

(i) every run $\rho \models \varphi$ *satisfying the task also satisfies* Ψ_h*; and*

(ii) every robot strategy $\pi_r \Vdash \Pi_r$ *ensures that all* π_r*-runs satisfying* Ψ_h *satisfy the task* φ*.*

In other words, the assumption template Ψ_h captures cooperative behaviors that the human can follow to enable the robot to satisfy its task, while the strategy template Π_r captures the robot's strategies that ensure task satisfaction as long as the human follows Ψ_h. Thus, the pair of templates (Ψ_h, Π_r) can effectively be used to address (a) of Problem 6.1. Let us illustrate this with an example.

Example 6.4. For the grid world in Figure 6.1 with the robot's task $\varphi_r = \square\diamondsuit(\texttt{adj} \wedge \texttt{major})$ as in Example 6.2, a parity game can be constructed as per Proposition 6.1 which has the same structure as the planning domain in Figure 6.1 but with an appropriate priority function $\mathbb{P}$ that assigns all states where $\texttt{adj} \wedge \texttt{major}$ holds the priority 2 (even) and all other states the priority 1 (odd). This captures the robot's objective of repeatedly reaching states with majority-occupied cells where the placed objects are non-adjacent. Using the synthesis procedure in Proposition 6.2, we can compute a pair of templates (Ψ_h, Π_r) for the robot and human, respectively, that capture cooperative behaviors. For instance, the human's template Ψ_h includes live groups $\{\texttt{h3} \rightarrow \texttt{t1}\}$ (in addition to other live groups as shown by green dashed edges in Figure 6.1), which ensure that the human does not consistently obstruct the robot's ability to make progress as discussed in Example 6.3. Similarly, the robot's template Π_r includes live groups that ensure the robot can always reach states satisfying $\texttt{adj} \wedge \texttt{major}$ as long as the human follows Ψ_h. ⌟

6.2.3 Adaptation and Feedback Mechanism

While the results in Proposition 6.2 provide a foundation for capturing cooperative behaviors, addressing (a) of Problem 6.1, they do not directly address the adaptation and feedback aspects outlined in (b) and (c) of Problem 6.1. To this end, we propose a framework that leverages the permissive nature of templates (Ψ_h, Π_r) to enable the robot to adapt its strategy based on its strategy template Π_r and provide feedback to the human based on the assumption template Ψ_h on human.

Adaptation: Since the template Π_r provides a set of possible actions at each state, the robot does not need to commit to a single strategy in advance. Instead, at runtime, the robot randomly selects an action from the set of enabled actions in Π_r at its current state. This approach allows the robot to adapt its choice of actions whenever it revisits a state. In particular, if an action taken by the robot does not lead to a desirable outcome and the run gets stuck in a loop (e.g., the human appears to be uncooperative), the run will eventually return to the same state, and by randomness, the robot can try different actions from the enabled set in Π_r. This adaptation mechanism effectively addresses (b) of Problem 6.1.

Feedback Mechanism: To facilitate a tunable feedback mechanism as outlined in (c) of Problem 6.1, we introduce a feedback threshold $\alpha \in [0, 1]$ that determines how often the robot provides feedback to the human. Note that, since unsafe actions in Ψ_h are actions that the human must avoid in order to satisfy the robot's task, the robot always communicates about unsafe actions whenever such an action is available at the current state. For other types of constraints in Ψ_h, i.e., co-live actions and live groups, the robot observes the human's actions and monitors how often the human violates these constraints (i.e., takes co-live actions or avoids actions from a live group). Whenever the frequency of such violations exceeds the feedback threshold α, the robot provides feedback to the human from the next time step onward until the frequency of violations drops below α. Due to Proposition 6.2, as long as the human follows Ψ_h, the robot's strategy will ensure that the task is satisfied. This feedback mechanism enables the robot to provide feedback in a tunable manner, only when human actions deviate significantly from the cooperative behaviors captured by Ψ_h, thus effectively addressing (c) of Problem 6.1.

Example 6.5. Continuing from Example 6.4, the robot can adapt its strategy at runtime by randomly selecting actions from the enabled set in its template Π_r. For instance, suppose the human consistently places objects along the diagonal, as discussed in Example 6.3. Consider the scenario in Figure 6.1 where the robot is at state `h0`, and currently it has made progress toward reaching a state with configuration as in `t1`. If the human continues to place objects along the diagonal, by taking actions $\texttt{h0} \to \texttt{r1}$ (which violates the live group $\{\texttt{h0} \to \texttt{r0}\}$ in Ψ_h), the robot will adapt its strategy that now may lead to states like `t2`, which can accommodate the human's diagonal placements while still satisfying φ_r. Now consider a scenario where the human, pursuing its own latent task,

consistently places objects in a manner that obstructs the robot's task and loops between states $\texttt{h2}, \texttt{r3}, \texttt{h3}, \texttt{r4}$ as discussed in Example 6.3. In this case, once the frequency of violations of the live groups exceeds the feedback threshold α, the robot will provide feedback to the human, requesting to take the live action $\texttt{h3} \rightarrow \texttt{t1}$, which will help the robot make progress toward satisfying its task φ_r. ⌟

6.3 Experiments

We evaluate the proposed framework on two experimental domains that illustrate complementary aspects of our novel HRℓI framework. To demonstrate the power of our approach on a physical robotic platform, the first domain is the simplified gridworld block-manipulation setting depicted in Figure 1.5 and described through our running example Examples 6.1 to 6.5. This example provides an interpretable test bed for visualizing how the robot adapts its strategy online in response to human actions and issues feedback in a tunable manner once the human blocks progress beyond a specified threshold.

The second domain is the Overcooked-AI environment [53], a standard benchmark for collaborative planning, which highlights the power of guidance and adaptability to enable the fully autonomous emergence of complex strategic human robot interactions. In particular, the natural ω-regular structure of the specifications allows us to investigate different levels of difficulty for such emergent interactions, ranging from complete alignment to partial or full misalignment between the human's and the robot's tasks.

Together, these experiments showcase both the transparency of our method in symbolic domains and its power to produce complex emergent strategic human robot interactions fully autonomously at runtime, which go far beyond the capabilities of existing approaches.

Remark 6.1. *Note that, Schuppe et al. [199] also considers a human-robot interaction 'Follow My Advice' (FMA) scenario where the robot provides advice to the human. However, their approach focuses on computing sufficient assumptions for the human to device a static feedback mechanism based on a pre-computed robot strategy to achieve a finite-horizon goal. Our framework, in contrast, emphasizes online adaptability of the robot's strategy to align with the human's behavior and employs a tunable feedback mechanism to handle persistent ω-regular objectives that we experimentally validate below. A faithful comparison would require re-formulating our settings to finite-horizon tasks, but this would undermine the core focus of our framework on the interplay between adaptability and feedback for persistent objectives.*

6.3.1 Gridworld Block-Manipulation

In this experiment, we implemented the simplified gridworld block-manipulation domain on a Franka Emika Panda robotic arm running ROS jazzy to demonstrate the feasibility of our framework beyond the abstract model. The setup consists of a robot hand

operating on a 3×3 workspace with tangible blocks that can be placed or removed, corresponding directly to the states illustrated in Example 6.1. The human interacts with the workspace by placing red blocks, while the robot places blue blocks. The system monitors the evolving configuration and evaluates whether the robot's task specifications (e.g., maintaining non-adjacent placements) are currently satisfied.

The reactive planning domain underlying this demonstration comprised approximately 7000 states and 18 propositions, encoding all possible placements of human and robot objects together with legal turn-taking moves. Our implementation required about 6 seconds to construct the parity game and synthesize the strategy templates, which was performed offline before execution. During execution, the robot follows its adaptive strategy: it updates its actions based on local observations of human moves, and when the human's behavior risks blocking task satisfaction (as in Example 6.3), the robot generates feedback through a display.

We include an image of the setup in Figure 1.5[1] to illustrate how the abstract domain is realized in practice. This experiment highlights how our framework scales from the formal model to a real-world setting, providing an interpretable test bed to assess both adaptability and human feedback.

6.3.2 Overcooked-AI

We further evaluate our framework in the *Overcooked-AI* environment, a widely used benchmark for collaborative planning with multiple agents. In this domain, the human and the robot repeatedly perform cooking tasks with the objective of persistently producing soups. Each participant is assigned an independent LTL task that encodes a recipe specification. As in the gridworld domain, these tasks are private and not known to the other.

We consider three classes of experimental scenarios, characterized by the relation between the human's and the robot's recipes. In the first class, the recipes are identical, such that both participants unknowingly pursue the same task. In the second class, the recipes are distinct but compatible, meaning that there exists at least one type of soup that simultaneously satisfies both specifications. In the third class, the recipes are incompatible, i.e., there is no soup that satisfies both specifications simultaneously. These classes capture increasing levels of misalignment between the tasks pursued by the human and the robot. Table 6.1 summarizes the recipe configurations considered in our experiments.

The Overcooked-AI environment provides a natural instantiation of persistent tasks, as both the human and the robot must repeatedly complete recipes over time. Each recipe specification corresponds to an ω-regular objective: an infinite run satisfies the task if the required recipe is produced infinitely often. This allows us to evaluate not only whether a single goal is reached, but also whether the tasks of the human and the robot can be persistently satisfied.

We model the Overcooked-AI environment as a reactive planning domain as per

[1] A full video of the experiment is available at https://youtu.be/61thSZDj5Ks.

Table 6.1: Recipe configurations in the Overcooked-AI experiments.

Scenario	**Robot recipe**	**Human recipe**
Identical	$= 3$ onions	$= 3$ onions
Incompatible	$= 3$ onions	$= 2$ onions
Compatible	≥ 2 onions	≤ 2 onions

Section 6.1.1, where the states encode the positions of the human and robot, the locations of ingredients, and the status of soups being cooked. The actions correspond to movement and interaction commands available to each participant. The ω-regular tasks for the human and the robot are specified as a parity condition over the states, which can be derived from the recipe specifications. The domain consists of approximately 68000 states with over 200 propositions encoding the relevant features of the environment. We implement the synthesis procedure in Proposition 6.2 to compute a pair of templates (Ψ_h, Π_r) for the robot and human, respectively, that capture cooperative behaviors, which took around 3 minutes to compute offline before execution. The robot then executes the adaptation and feedback mechanism as described in Section 6.2 while the human is simulated by a probabilistic strategy for its recipe.

We ran the three recipe scenarios (identical, compatible, incompatible) under different values of the feedback threshold α ranging from 0.00 to 0.10. For each scenario, the system was executed until 10 soups were delivered. Each run lasted up to 500 moves with an execution time of 1 minute and was repeated 10 times to account for the randomness in the human's and robot's action selection. For each run, we recorded the following metrics: (i) the percentage of soups delivered that satisfy the robot's recipe, (ii) the percentage that satisfy the human's recipe, and (iii) the percentage that satisfy both recipes simultaneously. Additionally, we tracked the frequency of feedback issued by the robot throughout the run. Figure 6.2 presents the temporal evolution of these metrics across all scenarios, illustrating how adaptability and feedback influence persistent task satisfaction.

Identical recipes. In the identical recipe setting, the human and the robot unknowingly pursue the same recipe specification. As shown in Figure 6.2a, both the human's and the robot's recipes are satisfied persistently across all runs. Crucially, no feedback is ever issued in this case. This illustrates the benefit of adaptability: even if the human and the robot start with different strategies for producing the same recipe, the robot is able to adjust online so that their behavior naturally aligns. In a system without adaptability, feedback would likely be needed to bring their strategies together, whereas our framework allows cooperation to emerge autonomously at runtime. Note that this also showcases the advantage of our framework over the static feedback mechanism in [199] (see Remark 6.1), which would issue feedback even when the human and robot have identical tasks, since it does not adapt the robot's strategy online.

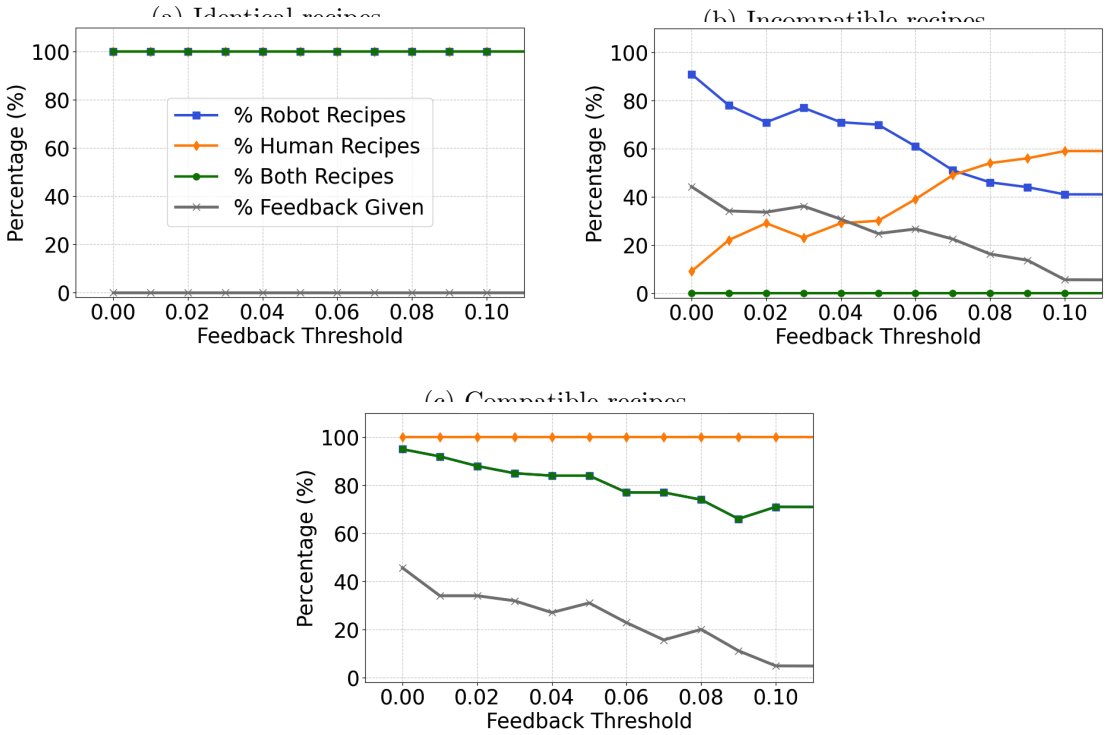


Figure 6.2: Satisfaction of human and robot recipe tasks over time in Overcooked-AI for three recipe scenarios. The plots show the proportion of runs satisfying each objective as well as the frequency of feedback given.

Incompatible recipes. In the incompatible recipe setting, the human and the robot follow recipe specifications such that no soup can satisfy both recipes simultaneously. Consequently, as shown in Figure 6.2b, the number of soups delivered that satisfy both recipes (green) remains zero. However, as objectives are formulated as ω-regular properties, it suffices to *always eventually* deliver a soup which satisfies the human's or robot's recipe, respectively. Hence, both agents can still cooperate by 'taking turns' in producing a soup via the human's (orange) and the robot's (blue) recipe. Figure 6.2b shows that this intuitive cooperative behavior indeed autonomously emerges and is influenced by the feedback threshold. As the feedback threshold α increases, the robot becomes more lenient to human non-cooperation, leading to a higher fraction of runs in which the human's recipe is delivered while the robot's satisfaction decreases. This effect highlights the role of feedback tuning: with an appropriately chosen threshold (in this case, $\alpha = 0.07$), both the human and the robot succeed in satisfying their respective recipes approximately 50% of the time persistently, demonstrating the importance of calibrated feedback sensitivity for the quality of the emerging cooperative behavior.

Compatible recipes. In the compatible recipe setting, the human and the robot pursue different recipe specifications, but there exists at least one soup that satisfies both. As shown in Figure 6.2c, in our experiments the human persistently satisfies their recipe in all runs, while the robot adapts its strategy and issues feedback to ensure its own recipe is also satisfied. Even with higher thresholds, the robot manages to satisfy its recipe in more than 70% of runs by adapting its strategy online towards the shared goal. When combined with more frequent feedback (i.e., lower thresholds), the robot can further increase the rate of joint satisfaction, achieving up to 95% of runs where both recipes are satisfied. This demonstrates the synergy between adaptability and feedback tuning, which together enable persistent satisfaction of both recipes.

6.4 Related Work

Due to the enormous relevance of reliable human-robot interaction (HRI) for trustworthy autonomy, there is an enormous body of work[2] in this research area. We therefore only focus on human-robot *logical* interactions (HRℓI) scenarios described by a game between the robot and the human. This research line is rooted in the seminal papers [131, 189] which are motivated by the observation that LTL is a powerful specification language to describe strategic objectives such as traffic rules for autonomous driving [154], or robot navigation [132].

Due to the natural formulation of HRℓI as a two-player game, classical approaches from graph games [37] can be used to compute a reactive strategy for the robot to fulfill its LTL objective. Unfortunately, classical solutions typically either over-constrain the robot or the human. In the first case, a robot strategy is computed which ensures the robot's objective against all human strategies. This, however, makes solutions overly restrictive and typically (as in the manipulation domain example above) fails to produce a robot strategy altogether. In the second case, full cooperation between the robot and the human are assumed, and a strategy is computed for both. Thereby, the human is forced into a very rigid, fully constrained behavior – ensuring reliability but at the cost of suppressing human autonomy.

A middle-ground is provided by various approaches which allow for increased human autonomy. When only *logical safety* is concerned – i.e., ensuring that no *bad* strategic interaction happens between the human and the robot – reactive shielding mechanisms [202, 185, 138, 113] which intervene with human behavior only to avoid such bad interactions, can be deployed. If, however, *logical liveness* objectives are present – i.e., requiring that something *good* (e.g., forming a diagonal in the above example) eventually happens – safety shielding is not sufficient to guarantee the satisfaction of the specification (as illustrated by the live-lock of human and robot trying to form a diagonal discussed before).

To mitigate these issues, many approaches explicitly model human behavior – either by predicting from trajectories [121] or by representing it as a Markov Decision Pro-

[2]Over two million results on Google Scholar searching for 'human robot interaction' on 15.9.25.

cess [92] – and integrate these models into the synthesis framework. Alternatively, HRℓI can be directly modelled as a stochastic two-player game [160]. While this introduces local viability of human strategies via stochasticity, it does not capture the need for strategic human autonomy.

To further improve human autonomy, admissibility-based methods [158, 159] can be used, which enable robots to adapt behaviors that remain robust against a broad range of human actions while still ensuring task satisfaction. Orthogonal to this approach, Schuppe et al. [199] focus on interactive advice, where the robot provides assume-guarantee style guidance to the human to support the satisfaction of a *shared* objective with minimal cooperation by the human. However, these approaches require the robot to commit to a *fixed pre-computed* strategy and rely either on stringent assumptions on human behavior, or on static, predefined forms of advice.

A different, but related approach to HRℓI only considers a (non-reactive) logical *planning* objective for the robot and ensures safety of humans in its workspace by implementing the resulting plan of the robot via control-barrier functions (CBF), which act as a safety filter on the resulting underlying continuous robot dynamics [203]. This approach was recently incorporated into cooperative HRℓI frameworks [219, 223, 206], where the (reactive) logical objectives of both the robot and the human are known, allowing an offline centralized game solution. Online human robot adaptations are only considered in the lower physical layer via CBFs but without any strategic autonomy. Similarly, recent dynamic game approaches to HRI [190, 172] are focusing on the immediate physical, rather than the long-term strategic adaptation and interaction of humans and robots.

In contrast to these approaches, we present an autonomy-driven approach to HRℓI that focuses on high-level strategic interactions. While prior work with this focus [158, 159, 118, 199, 224] result in pre-computed forms of cooperation or feedback and only consider fixed horizon objectives (as discussed before), our framework exploits the synergy between *online* adaptation and *tunable* feedback to generate complex *emergent* cooperation behavior for *finite and infinite* horizon LTL tasks.

The proposed framework is enabled by the flexibility of *local permissive templates* in graph games, which were recently introduced [11] and form the basis of the results presented in Chapter 3. While the initial formulation of these templates were used to represent permissive assumptions as in Chapter 3, a refined formulation was subsequently developed to represent permissive strategies [14]. The application of local permissive templates in distributed synthesis is studied in Chapters 4 and 5. Furthermore, these templates have been applied to continuous non-linear systems [162], which is presented in a later chapter of this thesis (Chapter 7). Beyond these, local permissive templates have also been applied in a range of other domains not covered in this thesis, including infinite-state systems [195], stochastic games [173], quantitative synthesis [12], and the shielding of learned policies [13]. To the best of our knowledge, this chapter presents the first application of local permissive templates to HRℓI.

Part B

Assumptions on the Plant Model

While Part A of this thesis focuses on the permissiveness of assumptions that restrict the behavior of the environment or other components in the logical distributed systems (as shown by ⓐ in Figure 1.1), Part B focuses on the permissiveness of assumptions on the *plant model* (as shown by ⓑ in Figure 1.1). Recall that the plant model is an abstracted discrete representation of the underlying dynamical system. In this part, we explore assumptions that are permissive in allowing richer behaviors of the dynamical system to be captured in the plant model. Exploiting this permissiveness, we develop novel interfaces and frameworks that address key challenges in logical controller synthesis.

In Chapter 7, we address the challenge of *seamless integration* of logical controllers with continuous dynamics. In many CPS, the logical context—such as goals, tasks, or environmental conditions—can change at any time, requiring the logical controller to adapt its strategic decisions accordingly. To enable such seamless reactivity, we introduce a novel synthesis framework based on a new class of permissive assumptions on the plant model, called *persistent live groups*. These assumptions capture rich liveness properties of the continuous dynamics, enabling the logical controller to dynamically adapt its behavior in response to context changes.

In Chapter 8, we address the challenge of *scalability* to large plant models. Standard approaches to construct game graphs from plant models suffer from state explosion, making them infeasible for large-scale systems. We address this by constructing a *universal controller* derived from the logical specification alone, whose decisions are conditioned by assumptions on the plant model, called *prophecies*. These prophecies are learned from representative plant models and expressed as branching-time temporal logic formulas, capturing possible future branching structures (e.g., the ability to reach certain goal locations). This enables generalization across different plant models by efficiently verifying learned prophecies at runtime.

Finally, in Chapter 9, we address the challenge of *robustness* under uncertainty or partial violations of assumptions on the plant model. While many assumptions on the plant model are branching-time in nature, existing robust semantics for temporal logics primarily focus on linear-time properties. To bridge this gap, we propose a new robust semantics for branching-time temporal logics, which allows formal reasoning about how logical controllers tolerate violations of branching-time assumptions.

Chapter 7

Seamless Reactivity of Hybrid Control

This chapter considers the setting of logical controller synthesis where the underlying system operates in continuous time. In contrast to previous chapters, where we directly worked on computing the logical controller for the logical game obtained from the specification and the plant model (as discussed in Section 2.5), here we consider a two-layer *hybrid* controller consisting of a high-level logical layer and a low-level continuous-time layer. The high-level logical layer makes strategic decisions based on the logical game, while the low-level continuous-time layer executes these decisions on the actual dynamical system. In such a setting, a key challenge arises when the logical context, imposed by an external environment, can change at any time. As a result, the high-level logical controller must be able to react to these context changes immediately, without being constrained by the ongoing dynamics of the low-level continuous-time controller. For instance, if a robot is currently moving towards a target location A and the environment changes the target to location B, the logical controller should be able to switch the low-level controller to move towards B right away, rather than waiting for the robot to reach A first.

Existing approaches for hybrid controller synthesis often struggle with this requirement. Many methods [30, 191, 120, 116, 221] do not even allow such logical context changes by the external environment. Those that do [210, 184] typically rely on discretization-based abstraction techniques that exhaustively discretize the continuous dynamics of the system, leading to significant conservatism and suffering from the curse of dimensionality.

This chapter addresses this challenge by developing a novel synthesis framework for hybrid controllers that can seamlessly react to context changes triggered by an external environment. Intuitively, our framework builds on two permissive interfaces between the high-level logical layer and the low-level continuous-time layer. First, we design a *top-down interface* that translates high-level strategic choices into low-level control problems that can be solved directly at the continuous-time level. This interface enables the high-level logical controller to make decisions that can be immediately implemented

by the low-level continuous-time controller. Second, we design a *bottom-up interface* that incorporates rich liveness properties of the low-level continuous-time controller into the (abstract) plant model used by the high-level logical game. This is done by annotating the plant model with a new class of permissive assumptions, called *persistent live groups*. Such annotations allow the high-level logical controller to reason about the capabilities of the low-level continuous-time controller in a non-conservative manner. Combining these two interfaces, we develop a hybrid controller synthesis framework that enables seamless reactivity to context changes while ensuring the satisfaction of the overall logical specification.

The rest of this chapter is organized as follows. We first present a motivating example in Section 7.1 to illustrate the challenges and the intuition behind our approach. We then introduce the necessary preliminaries on continuous-time dynamical systems and their connection to logical games in Section 7.2, followed by the formal problem statement in Section 7.3. Our proposed hybrid controller synthesis framework is described in Section 7.4, with detailed synthesis procedures for the higher and lower layers in Section 7.5 and Section 7.6, respectively. Finally, we demonstrate the effectiveness of the framework on the motivating example in Section 7.7 and discuss related work in Section 7.8.

7.1 Motivating Example

Throughout this chapter, we re-visit the following simple robot control example to outline the challenges and contributions of our new hybrid controller synthesis approach.

Example. We consider a simple moving robot $\mathbf{r}$, in a setting composed by two neighboring rooms, connected by a sliding door, as depicted in Figure 7.1. There are three target sets: $\mathcal{T}_1, \mathcal{T}_2$ in the left room and $\mathcal{T}_3$ in the right room. An external user (the logical *environment*), at each instant of time, chooses a mode among $\mathcal{M}_i$, $i \in \{1,2,3\}$ indicating the current desired target $\mathcal{T}_i$ for the robot. Moreover, the opening status of the door can be controlled by the robot – entering the target $\mathcal{T}_1$ or $\mathcal{T}_3$ opens the door (if it was previously closed) while entering the target $\mathcal{T}_2$ closes it (if it was previously open). This can be expressed by the LTL formula.

$$\begin{aligned}\Phi_A = &\square \bigwedge_{1\le i\le 3} \left(\mathcal{M}_i \Leftrightarrow \bigwedge_{1\le j\ne i\le 3} \neg \mathcal{M}_j \right) \\ &\wedge \square(\mathcal{T}_1 \vee \mathcal{T}_3 \Rightarrow \bigcirc \neg \mathcal{D}) \wedge \square(\mathcal{T}_2 \Rightarrow \bigcirc \mathcal{D}) \\ &\wedge \square(\mathcal{D} \Rightarrow \mathcal{D}\, \mathbf{W} (\mathcal{T}_1 \vee \mathcal{T}_3)) \wedge \square(\neg \mathcal{D} \Rightarrow \neg \mathcal{D}\, \mathbf{W}\, \mathcal{T}_2). \end{aligned} \tag{7.1a}$$

The goal is to design a controller that reacts to the external environment decisions $\mathcal{M}_i$, by moving to the chosen target $\mathcal{T}_i$ while adhering to additional safety-constraints, i.e. not hitting the walls $\mathcal{W}$ (including the door if it is closed). This can be expressed by the LTL formula

$$\Phi_G = \square \neg \mathcal{W} \bigwedge_{i=1,2,3} (\diamondsuit \square \mathcal{M}_i \Rightarrow \diamondsuit \square \mathcal{T}_i)\,. \tag{7.1b}$$

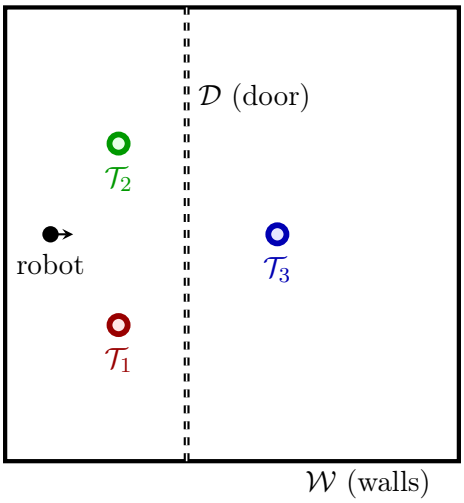


Figure 7.1: Motivating example: A robot must navigate to and remain at targets $\mathcal{T}_1$, $\mathcal{T}_2$ or $\mathcal{T}_3$ as directed by an external environment which imposes respective modes $\mathcal{M}_1$, $\mathcal{M}_2$, and $\mathcal{M}_3$, while avoiding any collision with the walls $\mathcal{W}$ and with the door $\mathcal{D}$ (if it is closed).

Summarizing formally, the overall specification for the robot is $\Phi_A \Rightarrow \Phi_G$, i.e, it needs to guarantee its goal Φ_G while assuming that Φ_A holds.

Challenges. This example showcases three main challenges that are tackled by our new controller synthesis approach.

First, the environment can change the mode *at any time*. Considering a real application where targets might be far away from each other, we would like the robot to immediately adapt its motion towards the new target, and not only after "completing" the previously assigned task of reaching another target. We achieve this *direct reactivity*, by autonomously switching the low-level continuous-time controller in reaction to a mode change.

Second, as the robot itself is controlling a part of the logical context (by being able to open and close the door), a hybrid controller cannot naively switch between low-level controllers for different targets based on the active mode. If, for example, the desired target is set to be equal to $\mathcal{T}_3$ and the robot is currently in the left room while the door is closed, the robot should automatically decide to first visit the target $\mathcal{T}_1$ to open the door. Scaling this to applications (e.g., in warehouses) where many logical requirements interact, requires a principled way to design a correct *strategy* for the robot to react to context changes such that a given formal specification, for instance $\Phi_A \Rightarrow \Phi_G$, is satisfied.

Third, it is important that the low-level controller does not simply implement what *should* be done (i.e., which target should be reached) but also what *should not* be done. For example, if the robot is in the left room moving towards $\mathcal{T}_3$ while the door is open, it must not pass over $\mathcal{T}_2$, as this would close the door. In addition, the door can be both

an obstacle and a target, depending on the current context.

To design a correct-by-construction hybrid controller tackling these last two challenges, one needs (i) a formally correct mechanism to translate strategic choices from the higher layer to control problems in the lower continuous-time layer (that can be solved using standard techniques from control theory) and (ii) to incorporate all necessary information about the workspace and the rich properties of the low-level continuous-time controller into the higher layer logical game. Furthermore, to allow both layers to have as much freedom as possible for their respective designs, it is crucial that both mechanisms (i) and (ii) are designed in a *permissive* manner, i.e., they do not introduce unnecessary restrictions on either layer. In some sense, this is similar to the permissiveness goals outlined in Part A for distributed logical systems, but now applied to different layers of a hybrid system rather than different components of a distributed system.

Our Solution. This chapter achieves these two goals by a new game-solving formalism for higher layer strategy synthesis, which (i) computes *strategy templates* instead of single strategies and (ii) allows for *persistent live group augmentations*. We show that (i) strategy templates provide a *certified top-down interface* by allowing a direct translation into *context-dependent reach-while-avoid* (RWA) controller synthesis problems (i.e., to reach a target set while avoiding unsafe regions), which, in turn, can be certifiably solved via standard techniques from control theory such as control Lyapunov functions [72]. This leads to provably correct low-level controllers implementing higher-layer strategy choices. Further, we show that (ii) persistent live group augmentations provide a *certified bottom-up interface* that enables a non-conservative and discretization-free incorporation of lower-layer rich properties into the higher-layer logical game.

7.2 Preliminaries

In this section we recall the main concepts and results from dynamical control systems theory and its interface with graph games, that will be used throughout this chapter. Note that due to the use of various concepts from different research fields, this chapter contains numerous notations, some of which may overlap with those used in other chapters. However, the notations used in this chapter are only local to this chapter and do not interfere with the notations used in other chapters. Furthermore, we use various mathematical notions and notations that are standard in the literature. For the sake of brevity, we do not introduce these standard notions and notations here, and we refer the interested reader to standard textbooks [72, 126] for further details.

7.2.1 Control Systems

Let us start by introducing the notion of continuous-time control systems considered in this chapter.

Definition 7.1. A (continuous-time) control system is defined by a triple $\mathcal{S} := (X, U, f)$ where:

- the open set $X \subseteq \mathbb{R}^{n_x}$ is the *state space*, of dimension $n_x \in \mathbb{N}$;
- the set $U \subseteq \mathbb{R}^{n_u}$ is the *input space*, of dimension $n_u \in \mathbb{N}$;
- the function $f \in \mathscr{C}^1(\mathbb{R}^{n_x} \times \mathbb{R}^{n_u}, \mathbb{R}^{n_x})$ describes the *system dynamics*, defined by

$$\dot{x} = f(x, u), \tag{7.2}$$

where $\mathscr{C}^1(Y, Z)$ denotes the set of continuously differentiable functions from Y to Z and $\dot{x}$ is the time-derivative of the state x.

Given a control system $\mathcal{S} := (X, U, f)$ and a measurable function $u : X \to U$, a *trajectory* of $\mathcal{S}$ for u starting at $x \in X$ is a function $\xi_{x,u} : [0, T) \to X$ (for some $T > 0$ and possibly $T = +\infty$) such that $\xi_{x,u}(0) = x$, $\xi_{x,u}(t) \in X$ for all $t \in [0, T)$ and $\dot{\xi}_{x,u}(t) = f\big(\xi_{x,u}(t), u(\xi_{x,u}(t))\big)$ for almost all $t \in [0, T)$.

7.2.2 Control Lyapunov Functions (CLF)

To cope with *reach-while-avoid* objectives, we must design low-level (continuous-time) controllers, driving the system to desired targets, possibly avoiding obstacles/staying in safe regions. Thus, we aim to design low-level controller, using the formalism of *control Lyapunov functions* (CLF). Let us recall in what follows the main definitions and concepts from the standard literature on CLFs [187, 204, 205], adapted to our specific context and notations. Given a function $w : X \to \mathbb{R}$ and any $c \in \mathbb{R}$, we denote by $X_w(c) := \{x \in X \mid w(x) \leq c\}$ the c-sublevel set of w.

Definition 7.2. Let us consider a compact set $X_T \subset X$ named the *target*. A function $w \in \mathscr{C}^1(X, \mathbb{R})$ is a *control Lyapunov function* (CLF) for system (7.2) with respect to X_T if there exist $0 < c < C$ and $\delta > 0$ such that

$$X_w(c) \subseteq X_T \quad \wedge \quad X_w(C) \subseteq X, \tag{7.3}$$

$$\inf_{u \in U} \langle \nabla w(x), f(x, u) \rangle \leq -\delta w(x), \ \forall x \in X_w(C) \setminus X_w(c), \tag{7.4}$$

where $\nabla w(x)$ is the gradient of w at x and $\langle \cdot, \cdot \rangle$ denotes the standard inner product in euclidean spaces. In this case, the set $X_w := X_w(C)$ is the *basin of attraction* of w. If $X = \mathbb{R}^{n_x}$, w is radially unbounded and inequality (7.4) holds in $\mathbb{R}^{n_x} \setminus X_w(c)$, then w is said to be a *global* CLF.

Intuitively, the condition (7.4) implies that, whenever $x \in X_w \setminus X_w(c)$, there exists a $u \in U$ for which the directional derivative of w along the vector $f(x, u)$ is strictly negative, and thus the value of the Lyapunov function is decreasing along trajectories of (7.2) following such direction. This observation motivates the following CLF-based result.

Lemma 7.1. *Consider a control system $\mathcal{S} := (X, U, f)$, a compact target set $X_T \subset X$, and suppose that $w \in \mathscr{C}^1(X, \mathbb{R})$ is a CLF in the sense of Definition 7.2. Consider a* low-level controller $u : X_w \to U$ *satisfying*

$$\langle \nabla w(x), f(x, u(x)) \rangle \leq -\delta w(x), \ \forall x \in X_w \setminus X_w(c), \tag{7.5}$$

then, for all $x \in X_w$*, it holds that* $\xi_{x,u}(t) \in X_w$ *for all* $t \in \mathbb{R}_+$ *and* $\exists\ T_x \geq 0$ *such that* $\xi_{x,u}(t) \in X_w(c)$, $\forall t \geq T_x$.

The proof follows from classic Lyapunov theory and the comparison argument, therefore, we refer to related literature [204, 73] for a detailed demonstration.

We note that Definition 7.2 considers basins of attraction X_w which are sublevel sets of CLFs. Hence, these sets are *safe by construction*, that is, all trajectories under a low-level controller u satisfying (7.5) will always stay inside X_w (in addition to eventually reaching $X_w(c)$). As such, the CLFs considered here allow to enforce *reach-while-avoid* objectives, by provably *avoiding* an unsafe region *while reaching* a target region within the state space. As the computation of such CLFs can introduce some conservatism, we note that more general approaches in control theory, such as control Lyapunov barrier functions [221, 120, 68] can similarly be used for the purpose of guaranteeing safety, depending on the specific application context.

7.2.3 Atomic Propositions for Control Systems

As introduced in Section 2.2.2, linear temporal logic (LTL) specifications are defined over a set of atomic propositions, i.e., boolean variables that can either be True or False. For control systems, these atomic propositions are used to symbolically represent important information about the low-level controllers and its environment to the high-level logical controller. In particular, we partition the set of atomic propositions into three categories $\mathtt{AP} = \mathtt{AP}_S \cup \mathtt{AP}_O \cup \mathtt{AP}_C$ as described below.

(i) *State propositions* $\mathtt{AP}_S$ (e.g., $\mathcal{T}_1$, $\mathcal{T}_2$, $\mathcal{T}_3$ in Figure 7.1) are associated with a *subset of the state space* s.t. $\mathcal{T}_i \in \mathtt{AP}_S$ is True at time t if the current state $x(t)$ of the underlying dynamical system is within this subset[1], i.e. $x(t) \in \mathcal{T}_i \subseteq X$.

(ii) *Observation propositions* $\mathtt{AP}_O$ denote all other aggregated information *observed* by the logical controller from the underlying continuous control system (e.g., $\mathcal{D}$ in Figure 7.1) and the external environment (e.g., $\mathcal{M}_1$, $\mathcal{M}_2$, and $\mathcal{M}_3$ in Figure 7.1).

(iii) *Control propositions* $\mathtt{AP}_C$ denote a finite set of low-level controllers that the high-level logical controller can choose (which will be introduced in Section 7.4.3).

Given a control system $\mathcal{S} = (X, U, f)$, the state propositions $\mathtt{AP}_S$ define a labelling function $\ell\colon X \to 2^{\mathtt{AP}_S}$ s.t. for all $\mathcal{X} \in \mathtt{AP}_S$ holds that $\mathcal{X} \in \ell(x) \Leftrightarrow x \in \mathcal{X}$. In addition, $\Upsilon\colon \mathbb{R}_+ \to 2^{\mathtt{AP}_O}$ denotes a piecewise-constant and right-continuous[2] *logical disturbance function* modelling the sequence of observation propositions acting on the system over time. We collect all logical disturbance functions acting on S in the set $\mathfrak{D}$.

[1] With a slight abuse of notation we denote the state subset associated with a state proposition by the same symbol.

[2] A function $\ell : \mathbb{R}_+ \to S$, with S a finite set, is piecewise-constant if it has a finite number of discontinuities in any bounded subinterval of $\mathbb{R}_+$; it is right-continuous if $\lim_{s \searrow t} \ell(s) = \ell(t)$ for all $t \in \mathbb{R}_+$.

7.2.4 Traces Generated by Control Systems

Given a control system $\mathcal{S}$ with labelling function ℓ, a trace $l_0 l_1 \dots$ over $\mathtt{AP}_S \cup \mathtt{AP}_O$ is said to be *generated* by a trajectory $\xi : \mathbb{R}_+ \to X$ (of the underlying dynamical system) under disturbance $\Upsilon \colon \mathbb{R}_+ \to 2^{\mathtt{AP}_O}$, if there exists an infinite sequence of time points $\tau_0, \tau_1, \dots$ for which it holds that:

- $\tau_0 = 0$, $\tau_i < \tau_{i+1}$, and τ_i goes to ∞ as i goes to ∞,
- for all $i \in \mathbb{N}$, $t \in [\tau_i, \tau_{i+1})$, $\ell(\xi(t)) \cup \Upsilon(t) = l_i$ holds.

We write $\mathit{Traces}_{\ell,\Upsilon}(\xi)$ to denote the set of all traces generated by ξ under ℓ and Υ.

7.2.5 Games and Strategy Templates

As presented in Section 2.5.2, the key idea to compute high-level logical controllers satisfying given LTL specifications is to reduce the problem to two-player games on graphs. Recall from Proposition 2.2 that every LTL formula can be translated into an equivalent alternating parity game, where Player 0 represents the controller player and Player 1 represents the environment player. Thus, the problem of computing a logical controller which satisfies a given specification Φ in interaction with an uncontrolled environment (i.e., independent of the plant model) reduces to computing a winning strategy (for Player 0) in a parity game $\mathcal{G}$.

Strategy templates. While it is well known how to compute a *single* winning strategy for a parity game $\mathcal{G}$ (as discussed in Section 2.5.3), it was recently shown that *strategy templates* [14], which characterize an infinite number of winning strategies in a succinct manner, are particularly useful in the context of controller synthesis for CPS. They are utilized within this chapter to obtain a novel translation of high-level logical controller decisions into low-level controllers.

Strategy templates are constructed from three types of local edge conditions, i.e., *safety*, *co-live* and *live group* templates, similar to the ones introduced in Section 3.2. Formally, given a game $\mathcal{G} = (G = (V, E, v_0, L), \mathit{Parity}(\mathbb{P}))$, a strategy template is a tuple $\Pi \lhd (S, D, \mathcal{H})$ consisting of a set of *unsafe* edges $S \subseteq E_0$, a set of *co-live* edges $D \subseteq E_0$, and a set of live groups $\mathcal{H} \subseteq 2^{E_0}$. This strategy template can also be represented by an LTL formula $\Pi = \Lambda_{\text{UNSAFE}}(S) \wedge \Lambda_{\text{COLIVE}}(D) \wedge \Lambda_{\text{LIVE}}(\mathcal{H})$, where

$$\Lambda_{\text{UNSAFE}}(S) \coloneqq \bigwedge_{e \in S} \Box \neg e,$$

$$\Lambda_{\text{COLIVE}}(D) \coloneqq \bigwedge_{e \in D} \Diamond \Box \neg e, \text{ and}$$

$$\Lambda_{\text{LIVE}}(\mathcal{H}) \coloneqq \bigwedge_{H \in \mathcal{H}} \Box \Diamond \mathit{src}(H) \Rightarrow \Box \Diamond H.$$

Note that these templates are exactly the same as the ones defined in Section 3.2, except that here they are used for characterizing Player 0's strategies instead of assumptions

on Player 1. Hence, sources of all the edges in these templates are Player 0's vertices. Furthermore, the notations we use here are consistent with the ones introduced for permissive templates in Section 5.2.

A Player 0's strategy π follows a strategy template Π if it is winning in the game (G, Π) from every vertex. Intuitively, Player 0's strategy π follows a strategy template $(S, D, \mathcal{H})$ if every π-play ρ satisfies the following:

(i) ρ never uses the unsafe edges in S;

(ii) eventually, ρ stops using the co-live edges in D; and

(iii) if ρ visits $src(H)$ infinitely many times, then it also uses the edges in H infinitely many times.

Moreover, a strategy template Π is *winning* if every strategy following Π is winning in the original game $\mathcal{G}$. The algorithm to compute a winning strategy template in a parity game lies in same time complexity class as the standard algorithm, i.e., Zielonka's algorithm [225], for solving parity games. This leads to the following result:

Lemma 7.2 ([14, Theorem 4])**.** *Given a parity game with game graph $G = (V, E, v_0, L)$ and priority function $\mathbb{P}\colon V \to [0, d]$, a winning strategy template can be computed in $\mathcal{O}\left(|V|^{d+\mathcal{O}(1)}\right)$ time.*

7.3 Problem Statement

This section gives a formal definition of the problem we are tackling in this chapter. Our goal is to automatically synthesize a hybrid controller that operates a control system based on external logical disturbances, while ensuring that the overall system behavior satisfies a given LTL specification. Towards a formal problem statement, we first define *a hybrid controller* which controls a system $\mathcal{S}$ while reacting to logical context switches induced by the sequence of observation propositions $\Upsilon \in \mathfrak{D}$ acting on $\mathcal{S}$ as logical disturbances.

Definition 7.3. Let $\mathcal{S} = (X, U, f)$ be a control system and $\Upsilon\colon \mathbb{R}_+ \to 2^{\mathtt{AP}_O}$ a disturbance function. A *hybrid controller* is a function $\mathtt{hc} : \mathbb{R}_+ \times X \times \mathfrak{D} \to U$. A *trajectory* of $\mathcal{S}$ for $\mathtt{hc}$ starting at $x \in X$ under Υ is a function $\xi_{x,\mathtt{hc},\Upsilon} : [0, T) \to X$ (for some $T > 0$ and possibly $T = +\infty$) such that $\xi_{x,\mathtt{hc},\Upsilon}(0) = x$, $\xi_{x,\mathtt{hc},\Upsilon}(t) \in X$ for all $t \in [0, T)$ and $\dot{\xi}_{x,\mathtt{hc},\Upsilon}(t) = f(\xi_{x,\mathtt{hc},\Upsilon}(t), \mathtt{hc}(t, \xi_{x,\mathtt{hc},\Upsilon}(t), \Upsilon(t)))$ for almost all $t \in [0, T)$.

This leads us to the following problem statement.

Problem 7.1. *Given a control system $\mathcal{S} = (X, U, f)$ with labelling function $\ell\colon X \to 2^{\mathtt{AP}_S}$ and an LTL specification Φ over the predicates $\mathtt{AP}_S \cup \mathtt{AP}_O$, find a set of* winning initial conditions *$X_{win} \subseteq X$ and hybrid controller $\mathtt{hc} : \mathbb{R}_+ \times X \times \mathfrak{D} \to U$ s.t. for all $x \in X_{win}$, all disturbance functions $\Upsilon \in \mathfrak{D}$ and all trajectories $\xi_{x,\mathtt{hc},\Upsilon}$, it holds that*

(i) $\xi_{x,\mathtt{hc},\Upsilon}(t) \in X_{win}$ *for all* $t \in \mathbb{R}_+$, *and*

(ii) *every trace* $\gamma \in \mathit{Traces}_{\ell,\Upsilon}(\xi_{x,\mathtt{hc},\Upsilon})$ *satisfies* Φ.

The remainder of this chapter illustrates our solution to Problem 7.1 by first providing an overview of the entire multi-step synthesis algorithm in Section 7.4, then highlighting additional details for selected steps in Section 7.5 and Section 7.6, and showing simulation results for the motivating example from Section 7.1 in Section 7.7.

7.4 Synthesis Overview

This section overviews our automated synthesis procedure which consists of five steps which are schematically depicted in Figure 7.2. First, in Section 7.4.1 (Figure 7.2, green) we solve a high-level logical game induced only by the specification. Then, in Section 7.4.2 (Figure 7.2, pink) we build a *top-down* interface which allows us to translate strategic choices from the logical level into certified low-level controllers. Afterwards, in Section 7.4.3 (Figure 7.2, cyan), we build a *bottom-up* interface to include relevant information about the low-level controllers into the logical game via *augmentations*. We then solve the resulting *augmented* game in Section 7.4.4 (Figure 7.2, violet). Finally, in Section 7.4.5 (Figure 7.2, orange), the obtained winning strategy is used to construct a hybrid controller which is proven to solve Problem 7.1.

7.4.1 High-Level Logical Synthesis

This initial step only considers the (high-level) reactive synthesis problem (as given in Problem 2.2) induced by the LTL specification Φ (realizing the green marked transitions in Figure 7.2). As formalized in Problem 7.1, the specification Φ only contains state and observation propositions, i.e., $\mathtt{AP} = \mathtt{AP}_S \cup \mathtt{AP}_O$. The definition of control propositions $\mathtt{AP}_C$ is part of our synthesis framework and will be discussed in Section 7.4.2.

In order to use Proposition 2.2 to construct the *initial parity game* $\mathcal{G}^I$ from Φ, we need to divide $\mathtt{AP}$ into controller (player 0) and environment (player 1) propositions. To do this, we optimistically assume that the controller can instantly activate/deactivate all state propositions in $\mathtt{AP}_S$, thus defining $\mathtt{AP}_0 := \mathtt{AP}_S$. This ignores the dynamics of $\mathcal{S}$ and how the state propositions are geometrically represented in the state-space. This is done on purpose to enable a *lazy* synthesis framework – our framework only adds aspects of both the dynamics and the geometric constraints which show to be *relevant* to the synthesis problem in a later step, discussed in Section 7.4.3.

As observation propositions are not under the control of the system or the controller, they are naturally interpreted as environment propositions, i.e., $\mathtt{AP}_1 := \mathtt{AP}_O$. Intuitively, the initial game $\mathcal{G}^I$ constructed from Φ via Proposition 2.2 reveals all logical dependencies of propositions relevant to the synthesis problem at hand. After constructing $\mathcal{G}^I$ from Φ (i.e., going from ① to ② in Figure 7.2), we can directly use Lemma 7.2 to synthesize a winning strategy template Π^I for $\mathcal{G}^I$ (i.e., going from ② to ③ in Figure 7.2) as discussed in Section 7.2.5.

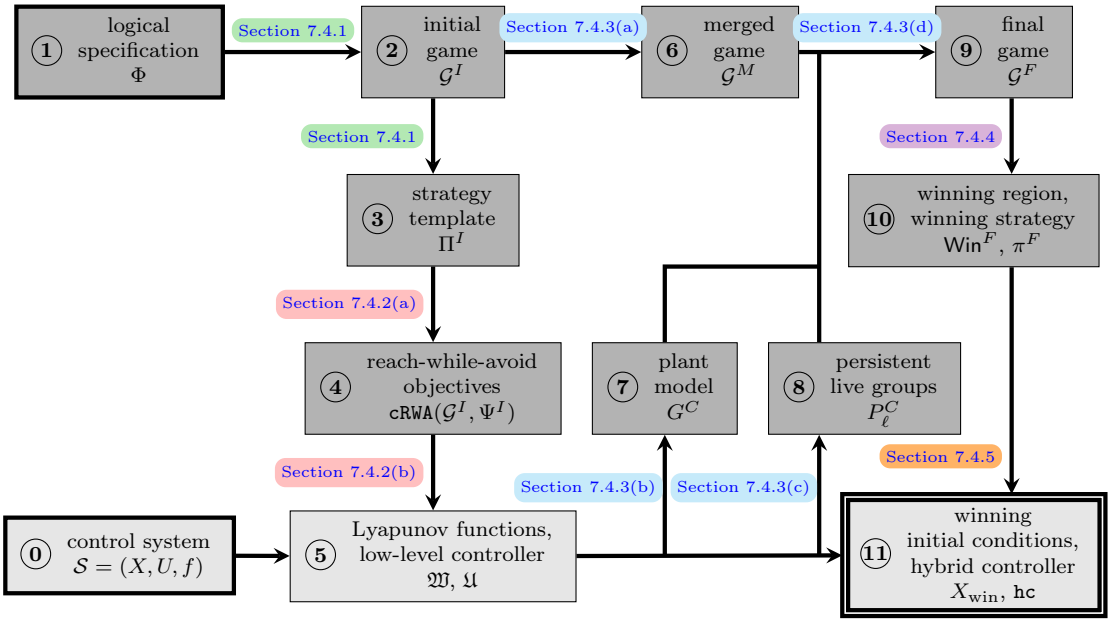


Figure 7.2: Flowchart illustrating the overall algorithm given in Section 7.4. Nodes (0), (1) are the inputs and node (11) is the output of our synthesis method. High-level and low-level synthesis steps are colored in dark and light grey, respectively, and discussed in the sections indicated at the arrows.

This gives the following result which is a direct consequence of Proposition 2.2 and the definition of strategy templates.

Proposition 7.1. *Given the LTL specification* Φ *over* $\mathtt{AP} = \mathtt{AP}_S \cup \mathtt{AP}_O$ *translated into an initial parity game* $\mathcal{G}^I$ *that is total w.r.t.* $\mathtt{AP}$ *via Proposition 2.2 and a winning strategy template* Π^I *for* $\mathcal{G}^I$*, the following holds: for every Player* 0 *strategy* π *that follows the strategy template* Π^I*, it holds that the trace generated by a* π*-play from initial vertex in the initial game* $\mathcal{G}^I$ *satisfies the specification* Φ*.*

Example 7.1. For the example from Section 7.1, the parity game $\mathcal{G}^I$ is constructed from the LTL specification $\Phi := \Phi_A \Rightarrow \Phi_G$ in (7.1) using Proposition 2.2 with $\mathtt{AP}_0 = \{\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3, \mathcal{W}\}$ and $\mathtt{AP}_1 = \{\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3, \mathcal{D}\}$. A part of the resulting parity game $\mathcal{G}^I$ is depicted in Figure 7.3.

A winning strategy template for the part of the parity game $\mathcal{G}^I$ depicted in Figure 7.3 is

$$\Pi^I = \Lambda_{\text{UNSAFE}}(e_{cf}, e_{df}) \wedge \Lambda_{\text{COLIVE}}(e_{cb}, e_{db}),$$

where $e_{vv'}$ denotes the edge from v to v'.

The strategy template Π^I forces the plays to never use the unsafe edges $\{e_{cf}, e_{df}\}$ (indicated schematically by dotted red arrows) as they lead to vertex f where proposition $\mathcal{W}$ is true signaling that the robot hits the wall. Furthermore, Π^I forces the plays to eventually stop using the co-live edges $\{e_{cb}, e_{db}\}$ (indicated schematically by dashed blue

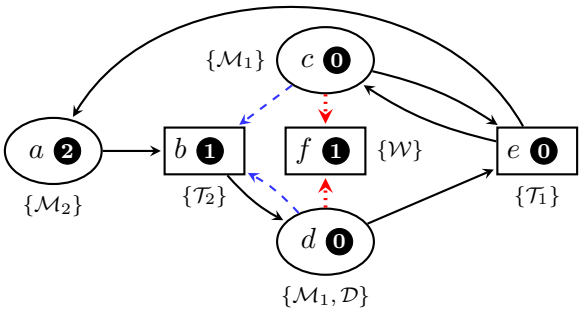


Figure 7.3: Illustration of a part of the *initial parity game* for the motivating example with Player 1 (squares) vertices and Player 0 (circles) vertices containing their priority in a black circle. A winning strategy template consists of unsafe edges indicated by red dotted arrows and co-live edges indicated by blue dashed arrows.

arrows). This is because if Player 0 (i.e., the controller) keeps using these edges, then Player 1 (i.e., the environment) can force a play to loop in one of the cycles $(cbde)^\omega$ or $(db)^\omega$ which does not lead to a winning play as the maximum priority seen infinitely often is odd (i.e., 1) in these cycles. ⌟

7.4.2 The Top-Down Interface

While Section 7.4.1 utilizes existing techniques from reactive synthesis, this section contains the first technical contribution of the work which is the translation of strategy templates into certified low-level controllers (realizing the pink marked transitions in Figure 7.2).

(a) Reach-While-Avoid Objectives

The strategy template Π^I computed in the last step defines, for all Player 0 vertices v, eventually required transitions (contained in $\mathcal{H}$) and (eventually) prohibited transitions (contained in S or D) for winning strategies in $\mathcal{G}^I$. While the game $\mathcal{G}^I$ assumes that these transitions can be instantaneously enabled (resp. disabled), they actually have to be enforced (resp. prevented) by a suitable actuation of the underlying dynamical system (e.g., the robot). The main observation that we exploit in this step is that the edge constraints for a Player 0 vertex v induced by a strategy template Π^I naturally translate into *context-dependent reach-while-avoid objectives* for the lower-layer continuous system.

Definition 7.4. A *context-dependent reach-while-avoid objective* (cRWA) is defined as a triple $\Omega := (\kappa, \mathcal{R}, \mathcal{A})$ where $\kappa \subseteq \mathtt{AP}_O$ is the *context*, $\mathcal{R} \in 2^{\mathtt{AP}_S}$ is the *target set* (to be reached) and $\mathcal{A} \in 2^{\mathtt{AP}_S}$ is the *obstacle set* (to be avoided). A control proposition $\mathcal{C} \in \mathtt{AP}_C$ is said to *implement* the *reach-while-avoid objective* Ω if the following constraint given by $\Phi_\mathcal{C}$ holds:

$$\Phi_\mathcal{C} := \square(\square(\mathcal{C} \ \wedge \ \kappa) \Rightarrow \diamondsuit\square\mathcal{R} \ \wedge \ \square\neg\mathcal{A}). \tag{7.6}$$

In practice, the translation of winning strategy templates into reach-while-avoid objectives (i.e., going from ③ to ④ in Figure 7.2) is done per vertex $v \in V_0$ (whose label defines the context) and reflects required and prohibited successors as targets and obstacles in the cRWA, respectively. In particular, as the final hybrid controller will make strategic decisions corresponding to exactly one transition, we compute cRWA's per required/allowed transition, while collecting all prohibited successors in the obstacles $\mathcal{A}$ of these cRWA's, as formalized next.

Definition 7.5. Let $\mathcal{G}$ be a parity game with game graph $G = (V, E, v_0, L)$ and winning strategy template $\Pi \lhd (S, D, \mathcal{H})$. For every $v \in V_0$ let $\mathrm{Suc}_{\mathcal{R}}(v) = \{v' \in E(v) \mid (v, v') \notin S \cup D\}$. Then, for each $v' \in \mathrm{Suc}_{\mathcal{R}}(v)$ we define $\Omega_a(v, v') := (L(v), L(v'), \mathcal{A}_a(v))$ and $\Omega_e(v, v') := (L(v), L(v'), \mathcal{A}_e(v))$ s.t.

- $\mathcal{A}_a(v) = \bigcup_{\{v'' \in V_1 \mid (v,v'') \in S\}} L(v'')$, and
- $\mathcal{A}_e(v) = \bigcup_{\{v'' \in V_1 \mid (v,v'') \in S \cup D\}} L(v'')$.

We collect all such cRWA's for the strategy template Π in the set $\texttt{cRWA}(\mathcal{G}, \Pi)$.

Intuitively, for such cRWA's, $\mathcal{A}_a$ consists of the propositions that need to be avoided "always", whereas $\mathcal{A}_e$ consists of the propositions that need to be avoided "eventually always". This definition is illustrated by the follwing example.

Example 7.2. Consider the winning strategy template Π^I computed in Example 7.1 for the parity game given in Figure 7.3. From vertex d, strategy template Π^I forces Player 0 to never use edge e_{df} and eventually stop using edge e_{db}. That means, Player 0 has to eventually only use edge e_{de} from vertex d. The labels of the vertices imply that whenever mode $\mathcal{M}_1$ is active and the door is closed, the system "always" has to reach $\mathcal{T}_1$ while avoiding walls $\mathcal{W}$ and "eventually always" has to reach $\mathcal{T}_1$ while avoiding both walls $\mathcal{W}$ and target $\mathcal{T}_2$. This leads to the cRWA's $\Omega_a(d, e) = (L(d), L(e), \mathcal{A}_a(d))$ and $\Omega_e(d, e) = (L(d), L(e), \mathcal{A}_e(d))$, where $L(d) = \{\mathcal{M}_1, \mathcal{D}\}$, $L(e) = \{\mathcal{T}_1\}$, $\mathcal{A}_a(d) = \{\mathcal{W}\}$, and $\mathcal{A}_e(d) = \{\mathcal{W}, \mathcal{T}_2\}$. ⌟

(b) Low-Level Controllers

Within this step, we utilize existing techniques to synthesize a low-level controller $u : X \to U$ associated to cRWA problem $\Omega = (\kappa, \mathcal{R}, \mathcal{A})$ (i.e., going from ④ to ⑤ in Figure 7.2), s.t. all traces generated by trajectories of $\mathcal{S}$ for u satisfy (7.6), given that $\mathcal{C}$ and κ are true for all $t \in \mathbb{R}_+$, where $\mathcal{C} \in \texttt{AP}_C$ is a controller proposition that flags that the low-level controller u is currently applied to $\mathcal{S}$.

To this end, we utilize techniques based on control Lyapunov functions (CLF), as introduced in Section 7.2.1, to define u from an $\Omega = (\kappa, \mathcal{R}, \mathcal{A})$. This is achieved by constructing a CLF $w : X \to \mathbb{R}$ (recall Definition 7.2) w.r.t. to the target $\mathcal{R}$ and enforcing that the basin of attraction $X_w \subseteq X$ excludes $\mathcal{A}$, i.e. $\mathcal{A} \cap X_w = \emptyset$.

We thus have the following definition.

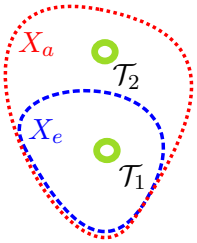


Figure 7.4: X_a (region enclosed by red dotted line) and X_e (region enclosed by blue dashed line) illustrate possible basins of attraction for the CLFs implementing the cRWAs $\Omega_a(d, e)$ (ensuring to reach $\mathcal{T}_1$ while avoiding only the walls) and $\Omega_e(d, e)$ (ensuring to reach $\mathcal{T}_1$ while avoiding walls and $\mathcal{T}_2$), respectively from Example 7.2.

Definition 7.6. Given the control system $\mathcal{S} = (X, U, f)$, consider a cRWA $\Omega = (\kappa, \mathcal{R}, \mathcal{A})$. We say that a CLF w (as in Definition 7.2) with basin of attraction X_w and the corresponding low-level controller $u_w : X_w \to U$ satisfying conditions in Lemma 7.1 are *associated to* Ω if $X_w \cap \mathcal{A} = \emptyset$ and $X_w(c) \subseteq \mathcal{R}$.

Section 7.6.1 will discuss a particular technique to synthesize X_w and u_w realizing a cRWA for particular classes of dynamical systems and state propositions. For any such realization of a cRWA we have the following guarantees on the resulting system under a constant context, i.e., w.r.t. a trivial disturbance function $\Upsilon := \kappa^\omega$, which are a direct consequence of Lemma 7.1 and Definition 7.6.

Proposition 7.2. *Given the control system $\mathcal{S} = (X, U, f)$ with labelling function ℓ, let $\Omega = (\kappa, \mathcal{R}, \mathcal{A})$ be a cRWA and let $u_w : X_w \to U$ be a low-level controller induced by a CLF w associated to Ω with basin of attraction X_w. Then, for all $x \in X_w$ and for all trajectories ξ_{x,u_w} of $\mathcal{S}$, it holds that*

(i) $\xi_{x,u_w}(t) \in X_w$ for all $t \in \mathbb{R}_+$,

(ii) every trace $\gamma \in Traces_{\ell,\Upsilon}(\xi_{x,u_w})$ satisfies $\phi_{\mathcal{C}_w}$ in (7.6), with $\mathcal{C}_w \in \mathtt{AP}_C$ being the control proposition associated to w and $\Upsilon := \kappa^\omega$ inducing a constant context.

Example 7.3. Consider the robot example given in Figure 7.1, the cRWAs $\Omega_a(d, e)$ and $\Omega_e(d, e)$ as given in Example 7.2. A possible set of corresponding CLFs w_a and w_e with basins of attraction X_a and X_e, respectively are depicted in Figure 7.4. ⌟

7.4.3 The Bottom-Up Interface

The synthesis procedure from Section 7.4.2 results in a finite set $\mathfrak{W}$ of CLFs with a finite set $\mathfrak{U}$ of low-level controllers, such that each low-level controller $u_w \in \mathfrak{U}$ (resulting from a CLF $w \in \mathfrak{W}$) is equipped with a basin of attraction $X_w \subseteq X$, associated to a given $\Omega \in \mathtt{cRWA}(\mathcal{G}^I, \Psi^I)$ resulting from a particular edge in the high-level logical game $\mathcal{G}^I$. This implies that whenever w is non-global, i.e., if $X_w \subsetneq X$, the low-level controller u_w cannot be applied anywhere.

Thinking back to the logical strategy computed in Section 7.4.1, low-level controller u_w must be used when its corresponding cRWA Ω for an edge e is "activated" by a logical controller, "choosing" the edge e in $\mathcal{G}^I$. By constructing the cRWA's for winning edges as defined in Definition 7.5, we essentially equip the resulting controller with a direct actuation capability of the underlying dynamical system — it must choose between available low-level controllers. To reflect this change of actuation capabilities in the higher-level game, we introduce a controller proposition $\mathcal{C}_w \in \mathtt{AP}_C$ for every available low-level controller u_w which flags that $u_w \in \mathfrak{U}$ should be used to actuate $\mathcal{S}$. Further, as every u_w is equipped with a basin of attraction X_w, the resulting hybrid controller is implementable only if the current continuous state x is in X_w. We therefore need to track this information in the logical game. For this purpose, we introduce a new state proposition $\mathcal{X}_w$ for every $u_w \in \mathfrak{U}$ that flags whether the state is in its basin of attraction, and we define $\mathtt{AP}_S^+ := \mathtt{AP}_S \cup \{\mathcal{X}_w \mid w \in \mathfrak{W}\}$ as the set of all state propositions including all additional state propositions $\mathcal{X}_w$'s.

The next four steps provide an algorithm that ensures that this information gets translated from the lower to the higher layer in a certified way (realizing the cyan marked transitions in Figure 7.2), such that the resulting higher-layer logical game allows to synthesize a hybrid controller that solves Problem 7.1.

(a) Changing Actuation Capabilities

As discussed before, in the initial game, the controller can activate/deactivate all state propositions in $\mathtt{AP}_S$. However, in order to prepare the high-layer initial game $\mathcal{G}^I$ from Section 7.4.1 for the incorporation of a refined system model, we need to incorporate the control propositions $\mathtt{AP}_C$ and make sure that these are the only propositions the controller can choose with its strategy, leading to the desired direct actuation of low-level controllers. In particular, first, we need to ensure that all state propositions and observation propositions can only be activated/deactivated by the environment player.

This is achieved by updating the initial game to a merged game $\mathcal{G}^M$ (i.e., going from ② to ⑥ in Figure 7.2) while preserving the parity condition and a one-to-one correspondence between the traces generated by plays in $\mathcal{G}^I$ and the ones generated by plays in $\mathcal{G}^M$.

Definition 7.7. Given an initial game $\mathcal{G}^I = (G^I, \mathit{Parity}(\mathbb{P}^I))$ with game graph $G^I = (V^I, E^I, v_0^I, L^I)$, the *merged game* $\mathcal{G}^M = (G^M, \mathit{Parity}(\mathbb{P}^M))$ with game graph $G^M = (V^M, E^M, v_0^M, L^M)$ is constructed as follows.

- The set of Player 1 vertices (including the initial vertex v_0^I) is retained with empty labelling, i.e., $V_1^M = V_1^I$ s.t. for each $v \in V_1^M$, $\mathbb{P}^M(v) = \mathbb{P}^I(v)$ and $L^M(v) = \emptyset$.
- The set of Player 0 vertices is constructed as follows. For every pair of Player 1 vertices $v_1, v_2 \in V_1^I$ connected via a Player 0 vertex $u \in V_0^I$, i.e.,$(v_1, u), (u, v_2) \in E^I$, we add:
  - a unique Player 0 vertex v to V_0^M with $L^M(v) = L^I(u) \cup L^I(v_2)$ and $\mathbb{P}^M(v) = \mathbb{P}^I(u)$,

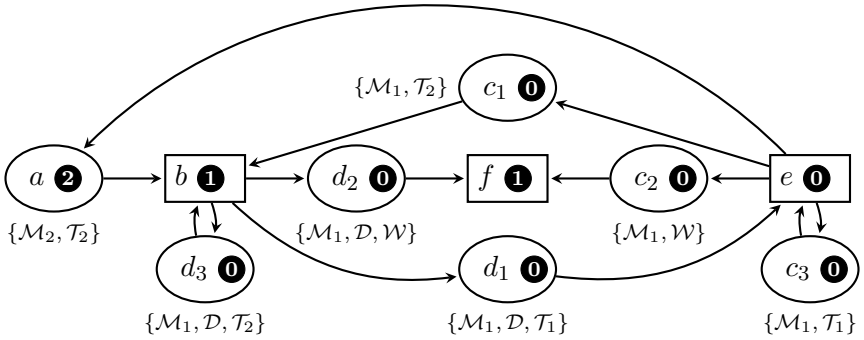


Figure 7.5: Corresponding merged game for the initial game given in Figure 7.3, where labels of Player 1 vertices are empty sets.

– new edges $(v_1, v), (v, v_2)$ to E^M.

This leads to the following lemma.

Lemma 7.3. *Let $\mathcal{G}^I$ be the parity game constructed from Φ over* AP *as in Proposition 7.1 and $\mathcal{G}^M$ its merged version constructed via Definition 7.7. Then $\mathcal{G}^M$ is total w.r.t.* AP*, and every play from v_0^M in $\mathcal{G}^M$ is winning iff its generated trace satisfies Φ.*

Proof. Let $\rho = v_0 v_1 \cdots$ be a winning play from $v_0 = v_0^M$ in $\mathcal{G}^M$ with $v_{2k} \in V_1^M$ for every $k \geq 0$, and let $\gamma = l_0 l_1 \cdots$ be the trace generated by the play ρ. Then by construction, vertices v_{2k} also belong to V_1^I with same priority, i.e., $\mathbb{P}^M(v_{2k}) = \mathbb{P}^I(v_{2k})$ for every $k \geq 0$. Furthermore, for every $v_{2k+1} \in V_0^M$, there exists a corresponding vertex $v'_{2k+1} \in V_0^I$ that connects the vertices v_{2k} and v_{2k+2} in the game $\mathcal{G}^I$ such that $\mathbb{P}^M(v_{2k+1}) = \mathbb{P}^I(v'_{2k+1})$ and $L^M(v_{2k+1}) = L^I(v'_{2k+1}) \cup L^I(v_{2k+2})$. Hence, the play $\rho' = v_0 v'_1 v_2 \cdots$ is a winning play in game $\mathcal{G}^I$ as maximum priority seen infinitely often in ρ' w.r.t. $\mathbb{P}^I$ is same as the maximum priority seen infinitely often in ρ w.r.t $\mathbb{P}^M$. Now, let $\gamma' = l'_0 l'_1 \cdots$ be the trace generated by ρ' in $\mathcal{G}^I$, then by construction of game $\mathcal{G}^I$, γ' satisfies the specification Φ. Moreover, since $L^M(v_{2k+2}) = \emptyset$ for every $k \geq 0$, we have, by definition, $l_k = L^M(v_{2k+1}) \cup L^M(v_{2k+2}) = L^M(v_{2k+1})$. Therefore, $l'_k = L^I(v_{2k+1}) \cup L^I(v_{2k+2}) = l_k$. So, $\gamma = \gamma'$, and hence, γ satisfies the specification Φ.

Conversely, for every play from v_0^M in $\mathcal{G}^M$ that is not winning, we can use similar arguments to show that its corresponding play in $\mathcal{G}^I$ is also not winning. Then, by construction of $\mathcal{G}^I$, its generated trace does not satisfy Φ.

Furthermore, using similar arguments, it can be shown that for every play in $\mathcal{G}^I$, there exists a corresponding play in $\mathcal{G}^M$ that generates the same trace. Hence, as $\mathcal{G}^I$ is total w.r.t. AP, so is $\mathcal{G}^M$. □

Example 7.4. Consider the initial game $\mathcal{G}^I$ given in Figure 7.3. Then the resulting merged game $\mathcal{G}^M$ is depicted in Figure 7.5. As shown in the figure, Player 1 vertices,

i.e., vertices b, e, f, are retained with same priorities but empty labels. For every pair of Player 1 vertices connected via a Player 0 vertex in $\mathcal{G}^I$, there is a new vertex with label containing all necessary propositions that connects the pair in $\mathcal{G}^M$, e.g., for vertex b and f connected via d in $\mathcal{G}^I$, the new vertex d_2 has labels of both d and f, and it connects vertex b and f in $\mathcal{G}^M$. ⌟

Note that we still have not explicitly incorporated the control propositions in the merged game. In the next steps, we will introduce the control propositions that are realizable by low-level controllers and incorporate them into the high-level game graph.

(b) Plant Model Construction

In this step we construct a game graph, representing the *plant model*, that captures the interplay of the environment and observation propositions contained in the context κ of a given cRWA (i.e., going from ⑤ to ⑦ in Figure 7.2) with the newly introduced control and state propositions $\mathcal{C}_w \in \mathtt{AP}_C$ and $\mathcal{X}_w \in \mathtt{AP}_S^+$. Intuitively, this graph captures which context changes an application of a particular low-level controller u_w for a CLF w (triggered by $\mathcal{C}_w$) might cause. When composed with the modifided game graph G^M from Section 7.4.3 this leads to the *lazy* refinement of the logical game discussed earlier, which only includes relevant information about the low-level controllers and their applicability.

Let us denote the cRWA's for which the CLF w was synthesized by $\Omega_w = (\kappa_w, \mathcal{R}_w, \mathcal{A}_w)$. Consider $\mathtt{AP}_S^+ \supseteq \mathtt{AP}_S$ the set of all state propositions including all additional state propositions $\mathcal{X}_w$'s as defined above, and $\ell^+ \colon X \to 2^{\mathtt{AP}_S^+}$ be an extended version of labelling function ℓ defined by $\ell^+(x) = \{\mathcal{X} \in \mathtt{AP}_S^+ \mid x \in \mathcal{X}\}$, (and thus, $\ell^+(x) \cap \mathtt{AP}_S = \ell(x)$ for all $x \in X$).

Definition 7.8. Given the control system $\mathcal{S} := (X, U, f)$ with labelling function ℓ^+ and the set $\mathfrak{W}$ of all CLFs computed as before, the *plant model* $G^C = (V^C, E^C, v_0^C, L^C)$ with $L^C \colon V \to 2^{\mathtt{AP}_S^+ \cup \mathtt{AP}_O}$ is constructed as follows.

1. We add an initial vertex v_0^C and a sink vertex $\mathtt{sink}^C$ to V_1^C, both with empty label, i.e., $L^C(v_0^C) = L^C(\mathtt{sink}^C) = \emptyset$.
2. For each CLF $w \in \mathfrak{W}$, we add two Player 1 vertices in V_1^C, a *transition* vertex and an *invariant* vertex, both with label $\{\mathcal{C}_w\}$.
3. For every subset of propositions $c \subseteq \mathtt{AP}_O \cup \mathtt{AP}_S^+$, we add a Player 0 vertex $v \in V_0^C$ with $L^C(v) = c$ iff there exists $x \in X$ such that $c \cap \mathtt{AP}_S^+ = \ell^+(x)$.
4. From the initial vertex v_0^C, we add an edge (v_0^C, v') to each $v' \in V_0^C$.
5. From each invariant vertex $v \in V_1^C$ of some CLF w, we add an edge (v, v') to $v' \in V_0^C$ iff $\mathcal{R}_w \subseteq L^C(v')$.
6. From each transition vertex $v \in V_1^C$ of some CLF w, we add an edge (v, v') to $v' \in V_0^C$ iff $\mathcal{X}_w \in L^C(v')$.

7. From each Player 0 vertex $v \in V_0^C$ with $\mathcal{X}_w \in L^C(v)$ and $\kappa_w = L^C(v) \cap \mathtt{AP}_O$ for some CLF w, if $\mathcal{R}_w \subseteq L^C(v)$, we add an edge to the invariant vertex of w, else we add an edge to the transition vertex of w.

8. For every dead-end vertex v, i.e., a vertex with no outgoing edges, we add an edge $(v, \mathtt{sink}^C)$, else we add an edge $(v, \mathtt{sink}^C)$. We add a self-loop $(\mathtt{sink}^C, \mathtt{sink}^C)$ on the sink vertex.

The construction of G^C via Definition 7.8 translates some characteristics of the low-level controllers captured by Proposition 7.2 into the high-level logical game. In addition, it ensures that a logical controller actuating a low-level controller u_w via control proposition $\mathcal{C}_w$ can only do so if context κ_w is true and the continuous system is in the basin of attraction X_w (signaled by the system proposition $\mathcal{X}_w$ being true). These translations can be formalized via LTL formulas which are ensured to hold true on every play over G^C as formalized in the next lemma.

Lemma 7.4. *Given the premises of Definition 7.8, it holds for every trace γ over G^C and every CLF $w \in \mathfrak{W}$ with basin of attraction $\mathcal{X}_w$, cRWA $\Omega_w := (\kappa_w, \mathcal{R}_w, \mathcal{A}_w)$ and associated controller $\mathcal{C}_w$, that*

$$\square(\mathcal{X}_w \Rightarrow \neg\mathcal{A}_w), \tag{7.7}$$
$$\square(\mathcal{C}_w \Rightarrow \mathcal{X}_w \wedge \kappa_w), \tag{7.8}$$
$$\square(\mathcal{R}_w \wedge \mathcal{C}_w \Rightarrow \bigcirc\mathcal{R}_w). \tag{7.9}$$
$$\square(\mathcal{X}_w \wedge \mathcal{C}_w \Rightarrow \bigcirc\mathcal{X}_w). \tag{7.10}$$

Proof. Let $\rho = v_0 v_1 \cdots$ be a play from v_0^C in G^C and $\gamma = l_0 l_1 \cdots$ be the trace generated by ρ. We need to show that γ satisfies (7.7)-(7.10). By Definition 7.6, for each $w \in \mathfrak{W}$, $X_w \cap \mathcal{A}_w = \emptyset$. Then, by step 3, for each $i \geq 0$, if $\mathcal{X}_w \in L^C(v_i)$ then $\mathcal{A}_w \cap L^C(v_i) = \emptyset$. Hence, γ satisfies (7.7). Next, by step 7, if $\mathcal{C}_w \in L^C(v_{i+1})$ for some $i \geq 0$, then $\mathcal{X}_w \in L^C(v_i)$ and $\kappa_w = L^C(v_i) \cap \mathtt{AP}_O$. Hence, γ also satisfies (7.8). Next, by step 5 and step 7, if $\mathcal{R}_w \subseteq L^C(v_i)$ and $\mathcal{C}_w \in L^C(v_{i+1})$ for some $i \geq 0$, then $\mathcal{R}_w \subseteq L^C(v_{i+2})$. Hence, γ also satisfies (7.9). Similarly, by step 6 and 7, if $\mathcal{X}_w \in L^C(v_i)$ and $\mathcal{C}_w \in L^C(v_{i+1})$ for some $i \geq 0$, then $\mathcal{X}_w \in L^C(v_{i+2})$. Hence, γ satisfies (7.10). □

Intuitively, given the premises of Lemma 7.4, equations (7.7)-(7.10) ensures the following low-level properties on the game graph level. First, (7.7) ensures that the basin of attaction $\mathcal{X}_w$ does not have an intersection with the avoid region $\mathcal{A}_w$. Next, (7.8) ensures that the controller $\mathcal{C}_w$ can only be applied if the system is within the corresponding basin of attaction $\mathcal{X}_w$ and the context κ_w holds. Note that this does not restrict the environment from changing the context right after the low-level controller associated with $\mathcal{C}_w$ was applied. Finally, (7.9)-(7.10) ensures that if the system is within the target region $\mathcal{R}_w$ (resp. the basin of attaction $\mathcal{X}_w$) and the controller $\mathcal{C}_w$ is applied, the system cannot leave $\mathcal{R}_w$ (resp. $\mathcal{X}_w$).

In total, the plant model G^C models all the state proposition sequences generated by a trajectory ξ triggered by the low-level controllers associated with $\mathfrak{W}$ as in Proposition 7.2.

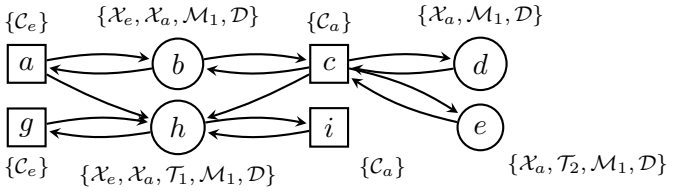


Figure 7.6: The corresponding plant model (without the initial vertex and the vertices going to sink vertices) for the basins of attraction in Figure 7.4.

Furthermore, it also models the logical disturbances received as inputs via the disturbance function $\Upsilon \in \mathfrak{D}$. This is formalized by the next lemma which directly follows by step 3 to 8 of Definition 7.8.

Lemma 7.5. *Given the premises of Definition 7.8, one of the following holds for every disturbance function* $\Upsilon \in \mathfrak{D}$

- *for some play from* v_0^C *in* G^C*, its generated trace* γ *satisfies that* $\gamma|_{\mathtt{AP}_O} = \Upsilon$*, or*
- *for some play* ρ *from* v_0^C *in* G^C *ending in the sink vertex, i.e.,* $\rho[k..] = (\mathtt{sink}^C)^\omega$*, its generated trace* γ *satisfies that* $\gamma[0; k-1]|_{\mathtt{AP}_O}$ *is a prefix of* Υ*.*

Example 7.5. For the CLFs w_a and w_e given in Example 7.3 with basins of attraction X_a and X_e as shown in Figure 7.4, the corresponding plant model is depicted in Figure 7.6. As in the figure, the transition vertices of w_e and w_a are vertices a and c, respectively, and the invariant vertices are vertices g and i, respectively. Note that both CLFs have context $\{\mathcal{M}_1, \mathcal{D}\}$. Hence, vertices with a label that contains $\mathcal{X}_a$ or $\mathcal{X}_e$ but not the propositions $\mathcal{M}_1$ or $\mathcal{D}$ are the vertices going to the sink vertices. For simplicity, those vertices are not shown in Figure 7.6. ⌟

While we could now take the product of G^C with $\mathcal{G}^M$ from the previous step in order to obtain the new, refined logical logical game, we note that this typically does not lead to a game that actually has a winning strategy. The reason for this lies in the fact that the modification of $\mathcal{G}^I$ to $\mathcal{G}^M$ gives the right to trigger state propositions to the environment, i.e., now the controller actuates $\mathtt{AP}_C$ and gets "notified" by the underlying dynamical systems via a triggering of $\mathtt{AP}_S$'s that the actuated low-level controller actually resulted in the (hopefully desired) state proposition change. From a two-player game perspective, the environment could now use its additional power to prevent the robot to reach the target, e.g., in Figure 7.6, starting from vertex b, if the controller keeps using the low-level controller for CLF w_e, then the environment can force the play to loop between vertex a and b instead of reaching target $\mathcal{T}_1$ represented by vertex h. This is because the resulting logical game still misses an essential information about the low-level dynamics under a given low-level controller. We thus incorporate, in what follows, the information captured by item (ii) of Proposition 7.2.

(c) Persistent Live-Groups

In order to capture item (ii) of Proposition 7.2 in the logical logical game, we construct so-called *persistent group liveness constraints* (i.e., going from ⑤ to ⑧ in Figure 7.2) to annotate the plant model G^C which are inspired by progress groups by Sun et al. [209].

Definition 7.9. Given a game graph $G = (V, E)$, a *persistent live group* is a tuple $(\mathtt{S}, \mathtt{C}, \mathtt{T})$ consisting of sets $\mathtt{S}, \mathtt{T} \subseteq V$ and $\mathtt{C} \subseteq E_0$ such that $\mathtt{T} \subseteq \mathtt{S}$. The constraints represented by such a persistent live group is expressed by the following LTL formula

$$\Lambda_{\text{PERS}}(\mathtt{S}, \mathtt{C}, \mathtt{T}) \coloneqq \Box\big(\Box(\mathtt{S} \wedge \Lambda_{\text{CONT}}(\mathtt{C})) \Rightarrow \Diamond \mathtt{T}\big), \tag{7.11}$$

where $\Lambda_{\text{CONT}}(\mathtt{C}) \coloneqq \mathit{src}(\mathtt{C}) \Rightarrow \mathtt{C}$. Moreover, the constraints represented by a set P_ℓ of persistent live groups is denoted by $\Lambda_{\text{PERS}}(P_\ell) \coloneqq \bigwedge_{(\mathtt{S},\mathtt{C},\mathtt{T}) \in P_\ell} \Lambda_{\text{PERS}}(\mathtt{S}, \mathtt{C}, \mathtt{T})$.

Intuitively, $\Lambda_{\text{CONT}}(\mathtt{C})$ ensures that edges in $\mathtt{C}$ are chosen when possible, as this is only possible for Player 0 vertices in $\mathtt{S}$. Furthermore, (7.11) ensures that persistently choosing the edges in $\mathtt{C}$ from the source vertices $\mathtt{S}$ will eventually lead us to a vertex in $\mathtt{T}$.

For a CLF $w \in \mathfrak{W}$, we construct a persistent live group $(\mathtt{S}_w, \mathtt{C}_w, \mathtt{T}_w)$ that captures Proposition 7.2 in the following way. Given the plant model G^C as defined before, and a CLF $w \in \mathfrak{W}$, first, the persistent activation of $\mathcal{C}_w$ is captured via the set $\mathtt{C}_w$ collecting all (Player 0) edges that end in vertices with labeled by $\mathcal{C}_w$, i.e.,

$$\mathtt{C}_w = E \cap \big(V \times \{v \in V \mid \mathcal{C}_w \in L^C(v)\}\big). \tag{7.12}$$

Always choosing an edge from $\mathtt{C}_w$ will force $\mathcal{X}_w$ to remain true within the same context κ_w, which is captured by the set $\mathtt{S}_w$ collecting all (Player 0) vertices labeled by $\mathcal{X}_w$ and propositions in κ_w, and all (Player 1) vertices labeled by $\mathcal{C}_w$, i.e.,

$$\mathtt{S}_w = \{v \in V \mid \mathcal{X}_w \in L^C(v), \kappa_w = L^C(v) \cap \mathtt{AP}_O\} \cup \{v \in V \mid \mathcal{C}_w \in L^C(v)\}. \tag{7.13}$$

Finally, we know that always choosing an edge from $\mathtt{C}_w$ will eventually lead us to a vertex where $\mathcal{R}_w$ is true, captured by the set $\mathtt{T}_w$ collecting all vertices labeled by $\mathcal{R}_w$, i.e.,

$$\mathtt{T}_w = \{v \in V \mid \mathcal{R}_w \subseteq L^C(v)\}. \tag{7.14}$$

Example 7.6. For example, consider the plant model shown in Figure 7.6 for Example 7.5. For CLF w_e of Ω_e, the corresponding persistent live group is $(\mathtt{S}, \mathtt{C}, \mathtt{T})$, where $\mathtt{S} = \{a, b, g, h\}$ corresponds to the region of basin of attraction for w_e with context $\{\mathcal{M}_1, \mathcal{D}\}$ being true, $\mathtt{C} = \{e_{ba}, e_{hg}\}$ corresponds to the edges that represent using the low-level controller associated with w_e, and $\mathtt{T} = \{h\}$ corresponds to the target region of Ω_e, i.e., vertices labeled by $\mathcal{T}_1$. ⌟

Given the set $\mathfrak{W}$ of all CLFs as given before, we collect all the corresponding persistent live groups for the CLFs in $\mathfrak{W}$ in the set P_ℓ^C. With the persistent live group assumptions P_ℓ^C, the plant model G^C also ensures that item (ii) of Proposition 7.2 holds at a higher level as formalized below.

Lemma 7.6. *Let G^C be a plant model as in Definition 7.8 and a set $\mathfrak{W}$ of CLFs with persistent live groups $(\mathtt{S}_w, \mathtt{C}_w, \mathtt{T}_w)$ for all $w \in \mathfrak{W}$ as in (7.12)-(7.14). Let ρ be a play from v_0^C in G^C and γ be the trace generated by ρ. Then ρ satisfies $\Lambda_{PERS}(\mathtt{S}_w, \mathtt{C}_w, \mathtt{T}_w)$ iff γ satisfies*

$$\square(\square(\mathcal{X}_w \wedge \kappa_w \wedge \mathcal{C}_w) \Rightarrow \diamondsuit \mathcal{R}_w). \tag{7.15}$$

Moreover, (7.15) along with (7.7)-(7.10) ensures that ρ satisfies $\Lambda_{PERS}(\mathtt{S}_w, \mathtt{C}_w, \mathtt{T}_w)$ iff γ satisfies (7.6).

Proof. By the definition of the persistent live groups as in (7.12)-(7.14), rewriting (7.11) in terms of propositions gives us that, ρ satisfies $\Lambda_{\text{PERS}}(\mathtt{S}_w, \mathtt{C}_w, \mathtt{T}_w)$ if and only if trace γ satisfies (7.15). Furthermore, by Lemma 7.4, the trace γ also satisfies (7.7)-(7.10).

Now, suppose γ satisfies (7.15), then we need to show that γ also satisfies $\Phi_{\mathcal{C}_w}$ in (7.6). Let $\gamma = l_0 l_1 \cdots$ be the trace. It suffices to show that for every $k \geq 0$, the trace $\gamma_k = l_k l_{k+1} \cdots$ satisfies the following:

$$\square(\mathcal{C}_w \ \wedge \ \kappa_w) \Rightarrow \diamondsuit\square \mathcal{R}_w \ \wedge \ \square\neg \mathcal{A}_w.$$

Suppose γ_k satisfies $\square(\mathcal{C}_w \wedge \kappa_w)$. Then, every $j \geq k$, l_j satisfies $\mathcal{C}_w$, which implies, by (7.8), l_j also satisfies $\mathcal{X}_w$. Moreover, by (7.7), l_j also satisfies $\neg\mathcal{A}_w$ for each $j \geq 0$. Therefore, trace γ_k satisfies both $\square(\mathcal{C}_w \wedge \kappa_w \wedge \mathcal{X}_w)$ and $\square\neg\mathcal{A}_w$, which then implies, by (7.15), γ_k also satisfies $\diamondsuit \mathcal{R}_w$. That means, there exists $m \geq k$ such that l_m satisfies $\mathcal{R}_w$. As l_m also satisfies $\mathcal{C}_w$, by (7.9), l_{m+1} satisfies $\mathcal{R}_w$. Using the same argument inductively, we can show that l_i satisfies $\mathcal{R}_w$ for all $i \geq m$. Therefore, γ_k satisfies both $\diamondsuit\square\mathcal{R}_w$ and $\square\neg\mathcal{A}_w$. Conversely, suppose γ satisfies (7.6), then we need to show that ρ satisfies $\Lambda_{\text{PERS}}(\mathtt{S}_w, \mathtt{C}_w, \mathtt{T}_w)$. It is enough to show that γ satisfies (7.15), which trivially follows from (7.6). □

(d) Final Augmented Parity Game

Given the three ingredients from the last steps, we are now ready to construct the final augmented (parity) game (i.e., going from ⑥, ⑦, ⑧ to ⑨ in Figure 7.2) which serves a new logical logical game for the final hybrid controller and is defined next.

Definition 7.10. An *augmented game* $\mathcal{G}$ is a tuple (G, Φ, P_ℓ) consisting of a game graph G, a set of persistent live groups P_ℓ over G and an LTL specification Φ. Moreover, an augmented game (G, Φ, P_ℓ) is equivalent to the game $(G, \Lambda_{\text{PERS}}(P_\ell) \Rightarrow \Phi)$.

Let us now describe how the final augmented parity game, i.e., an augmented game with parity specification, is constructed. Recall that V_p^M and V_p^C are the vertices of Player p in game graph G^M and G^C, respectively.

Definition 7.11. Given the merged game $\mathcal{G}^M$, plant model G^C, and persistent live groups P_ℓ^C as before, the *final augmented parity game* $\mathcal{G}^F = (G^F, Parity(\mathbb{P}^F), P_\ell^F)$ with $G^F = (V^F, E^F, v_0^F, L^F)$ is constructed by taking the product of the game $\mathcal{G}^M$ and the tuple (G^C, P_ℓ^C) as follows.

- For every $v^M \in V_p^M$ and $v^C \in V_p^C$ with $L^M(v^M) \cap (\mathtt{AP}_O \cup \mathtt{AP}_S) = L^C(v^C) \cap (\mathtt{AP}_O \cup \mathtt{AP}_S)$, add a vertex $v = (v^M, v^C)$ to V_p^F with label $L^F(v) = L^M(v^M) \cup L^C(v^C)$ and $\mathbb{P}(v) = \mathbb{P}^M(v^M)$.
- The initial vertex is $v_0^F = (v_0^M, v_0^C) \in V_1^F$.
- Add sink vertex $\mathtt{sink}^F$ to V_1^F with empty label and priority 1.
- Add an edge (v_1, v_2) to E^F from $v_1 = (v_1^M, v_1^C)$ to $v_2 = (v_2^M, v_2^C)$ if $(v_1^M, v_2^M) \in E^M$ and $(v_1^C, v_2^C) \in E^C$.
- For every $v = (v^M, v^C) \in V^F$, if $(v^C, \mathtt{sink}^C) \in E^C$, add an edge $(v, \mathtt{sink}^F)$ to E^F. Add a self-loop $(\mathtt{sink}^F, \mathtt{sink}^F)$ on the sink vertex.
- Add a persistent live group $(\mathtt{S}, \mathtt{C}, \mathtt{T})$ to P_ℓ^F if there exists a $(\mathtt{S}^C, \mathtt{C}^C, \mathtt{T}^C) \in P_\ell^C$ with:
  - $\mathtt{S} = V^F \cap (V^M \times \mathtt{S}^C)$,
  - $\mathtt{T} = V^F \cap (V^M \times \mathtt{T}^C)$,
  - for every edge $e = (v_1, v_2) \in E^F$ with $v_1 = (v_1^M, v_1^C)$ and $v_2 = (v_2^M, v_2^C)$, it holds $e \in \mathtt{C}$ if and only if $(v_1^C, v_2^C) \in \mathtt{C}^C$.

As the priority function $\mathbb{P}^F$ is defined by the priority function $\mathbb{P}^M$ of the merged game $\mathcal{G}^M$ and every winning play from v_0^F in $\mathcal{G}^F$ satisfying $\Lambda_{\text{PERS}}(P_\ell^F)$ needs to satisfy the parity condition $\mathit{Parity}(\mathbb{P}^F)$, the next proposition directly follows from Lemma 7.3.

Proposition 7.3. *Given the LTL specification Φ, initial game $\mathcal{G}^I$, and the final game $\mathcal{G}^F$ with persistent live groups P_ℓ^F as in Definition 7.11, suppose γ be a trace generated by a winning play from v_0^F in $\mathcal{G}^F$ satisfying $\Lambda_{\text{PERS}}(P_\ell^F)$, then γ satisfies the specification Φ.*

7.4.4 Solving the Final Augmented Game

As discussed in Section 7.4.1, the initial game $\mathcal{G}^I$ allowed the system to instantaneously activate or deactivate all state propositions in $\mathtt{AP}_S$. However, this was no longer possible in the merged game $\mathcal{G}^M$. But, in the final game $\mathcal{G}^F$, the persistent live groups, using the results described in Lemma 7.6, enable the system to activate or deactivate specific state propositions which are ensured to become *eventually* true (using the associated low-level controller) if no external context change is induced.

The next obvious step of our synthesis procedure is to solve the final augmented game $\mathcal{G}^F$, i.e., to compute a winning strategy in this game (realizing the violet marked transitions in Figure 7.2, i.e., going from ⑨ to ⑩). Based on the observation made in Definition 7.10 that an augmented game $(G, \mathit{Parity}(\mathbb{P}), P_\ell)$ is equivalent to the game $(G, \Lambda_{\text{PERS}}(P_\ell) \Rightarrow \mathit{Parity}(\mathbb{P}))$ one can use standard game solving techniques by reducing this to a classical parity game (as discussed in Section 2.5.2). This, however, usually results in computationally intractable problems. We will therefore provide a new algorithm for solving augmented parity games, in the subsequent Section 7.5, which has a

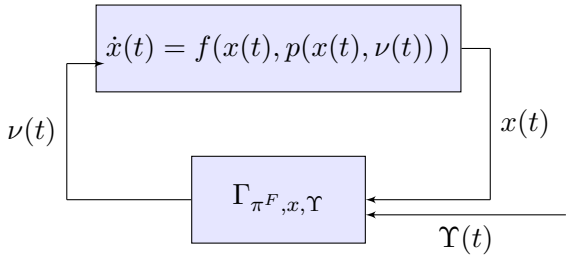


Figure 7.7: The interconnection between the control system and the hybrid system $\mathcal{H}_{\pi^F}$ defined in Definition 7.12

similar algorithmic structure and therefore also similar worst-case time complexity as the standard algorithm for solving classical (non-augmented parity) games and therefore allows for a computationally tractable solution.

For the time being, we assume that we have solved $\mathcal{G}^F$, i.e., we have computed a winning region $\mathsf{Win}^F \subseteq V^F$ and a memoryless strategy $\pi^F \colon V_0^F \to V_1^F$ that is uniformly winning, i.e., winning from every vertex in Win^F. Note that the initial vertex v_0^F of the final game $\mathcal{G}^F$ might not be winning as the initial state of the control system $\mathcal{S}$ is chosen by Player 1 (i.e., the environment) from v_0^F. However, we will see in the next section that the winning region Win^F of the final game $\mathcal{G}^F$ directly relates to the initial winning conditions of the control system $\mathcal{S}$ for the original specification Φ. This is because, given any Player 1 strategy that chooses a winning successor $v \in \mathsf{Win}^F$ of the initial vertex v_0^F, the (π^F, π_1)-play ρ from v_0^F is ensured to be winning in the final game $\mathcal{G}^F$. Furthermore, due to Proposition 7.3, we obtain that every such winning play ρ satisfying $\Lambda_{\text{PERS}}(P_\ell^F)$ in the final game $\mathcal{G}^F$ corresponds to a winning play in the initial game $\mathcal{G}^I$, and therefore, generates a trace satisfying the original specification Φ.

7.4.5 Constructing the Hybrid Controller

Given a winning region $\mathsf{Win}^F \subseteq V^F$ and a uniformly winning strategy $\pi^F : V_0^F \to V_1^F$, we now construct a set of *initial winning conditions* $X_{\text{win}} \subseteq X$ and a *hybrid controller* $\mathtt{hc} : \mathbb{R}_+ \times X \times \mathfrak{D} \to U$ (as in Definition 7.3) to solve Problem 7.1 (realizing the orange marked transitions in Figure 7.2, i.e., going from ⑩ to ⑪).

As discussed in Section 7.4.4, every winning successors $v \in \mathsf{Win}^F$ of the initial vertex v_0^F of the final game $\mathcal{G}^F$ corresponds to a possible initial winning condition of the control system $\mathcal{S}$ for the specification Φ. Therefore, we can define the set of initial winning conditions X_{win} via the labeling function ℓ^+ of the control system $\mathcal{S}$ as follows:

$$X_{\text{win}} := \{x \in X \mid \exists v \in \mathsf{Win}^F \cap E^F(v_0^F) \text{ s.t. } L^F(v) \cap \mathtt{AP}_S^+ = \ell^+(x)\}. \tag{7.16}$$

In order to translate the uniformly winning strategy $\pi^F : V_0^F \to V_1^F$ into a hybrid controller $\mathtt{hc}$ we take a two-step approach. We first construct a map Γ which uses π^F to translate *the history of* a continuous curve $\zeta : \mathbb{R}_+ \to X$ and a disturbance function $\Upsilon \colon \mathbb{R}_+ \to 2^{\mathtt{AP}_O}$ into a piece-wise constant function $\nu : \mathbb{R}_+ \to V_1^F$ of Player 1 vertices of

G^F. The hybrid controller $\mathtt{hc}$ then translates each vertex $\nu(t) \in V_1^F$ into the low-level controller $u_w : X \to U$ associated with its (unique) label $L^F(\nu(t)) = \{\mathcal{C}_w\}$ being a single control proposition in $\mathtt{AP}_C$ by construction of G^F. This low-level controller u_w is then applied to $\mathcal{S}$ via f. This is illustrated in Figure 7.7 and formalized in the following definition.

Definition 7.12. Let $\mathcal{S} = (X, U, f)$ be a control system with labelling function ℓ^+ and the set $\mathfrak{W}$ of all CLFs. Consider a uniformly winning strategy $\pi^F : V_0^F \to V_1^F$ over the final game $\mathcal{G}^F$, a continuous curve $\zeta : \mathbb{R}_+ \to X$ and a disturbance function $\Upsilon \colon \mathbb{R}_+ \to 2^{\mathtt{AP}_O}$. Then the map $\Gamma_{\pi^F,\zeta,\Upsilon}$ defines a piecewise constant function $\nu : \mathbb{R}_+ \to V_1^F$ such that:

1. $\nu(0) = \pi^F(v_1)$, where $v_1 \in E^F(v_0^F)$ s.t. $L^F(v_1) = \ell^+(\zeta(0)) \cup \Upsilon(0)$,
2. for any discontinuity point $\tau \in \mathbb{R}_+$ of $\ell^+(\zeta(\cdot)) \cup \Upsilon(\cdot)$, it holds that $\nu(\tau) := \pi^F(v)$ s.t. $(\nu(\tau^-), v) \in E^F$ and $L^F(v) = \ell^+(\zeta(\tau)) \cup \Upsilon(\tau)$, (where $\nu(\tau^-) := \lim_{s \nearrow \tau} \nu(s)$), and
3. the set of discontinuity points of $\nu(\cdot)$ is contained in the set of discontinuity points of $\ell^+(\zeta(\cdot)) \cup \Upsilon(\cdot)$.

Intuitively, Definition 7.12 models the fact that the high-level logical layer of the hybrid controller (modelled by the game) might actuate a change in the low-level controller only when the context changes. This context change can either be induced externally (when Υ has a discontinuity point, i.e., the observation proposition changes) or when $\ell^+(\zeta(t))$ changes, i.e., the underlying system dynamics causing state propositions to change. Both is detected by a discontinuity point in $\ell^+(\zeta(t)) \cup \Upsilon(t)$. At these triggering points (and only then), the map Γ_{π^F} mimics the move of the winning strategy π^F by moving to the environment vertex $\pi^F(v)$ selected by π^F in $\mathcal{G}^F$ while respecting the current context.

We emphasize that the definition of the map $\Gamma_{\pi^F,\zeta,\Upsilon}$ is actually causal. It only uses the information from the past of ζ and Υ up to time point t^- to compute $\nu(t)$. This implies that we can actually use it online to dynamically generate the signal ν from *the past observations* of a state trajectory ξ and the past logical disturbances Υ, as depicted in Figure 7.7. As, in this context, the state trajectory ξ is not known a priory, we slightly abuse notation and refer to $\Gamma_{\pi^F,\zeta,\Upsilon}$ as $\Gamma_{\pi^F,x,\Upsilon}$, where x is the starting point of ξ.

With this slight notation overload, we can define the *final closed loop system* as follows.

Definition 7.13. Given the premises of Definition 7.12, the *final closed loop system* is given by

$$\dot{x}(t) = f(x(t), \mathtt{hc}(x(t), \nu(t))\,), \tag{7.17}$$

where $\mathtt{hc}(x(t), \nu(t)) := u_w(x) \in U$ being the low-level controller associated with $L^F(\nu(t))$ and $\nu(t)$ is dynamically generated via $\Gamma_{\pi^F,x,\Upsilon}$ by interpreting (the past of) a trajectory $\xi_{x,\mathtt{hc},\Upsilon} : \mathbb{R}_+ \to X$ of (7.17) under $\mathtt{hc}$ and Υ, with starting point $x \in X$, as (the past of) ζ in Definition 7.12.

This leads to the main result of this section establishing the correctness of our synthesis procedure.

Theorem 7.1. *Consider a control system $\mathcal{S} = (X, U, f)$ with labelling function ℓ, an LTL specification Φ over the predicates $\mathtt{AP}_S \cup \mathtt{AP}_O$. Consider the final game $\mathcal{G}^F$, the set $\mathfrak{W}$ of all CLFs, the extended labelling function ℓ^+, the winning region Win^F, and a uniformly winning strategy $\pi^F : V_0^F \to V_1^F$. Then X_{win} as in (7.16) and $\mathtt{hc}$ as in Definition 7.13 solve Problem 7.1.*

The proof of Theorem 7.1 combines all correctness results established in Section 7.4.1-Section 7.4.4.

Proof. Since the plays ending in sink vertices are not winning in a game and π^F is a winning strategy in $\mathcal{G}^F$, no π^F-play from Win^F goes to sink vertex. Then, by Lemma 7.3 and Lemma 7.5, all possible changes in ℓ^+ (triggered by applying low-level controller associated with $\mathfrak{W}$) and Υ are captured by the game graph G^C. In particular, every trajectory $\xi_{x,\mathtt{hc},\Upsilon}$ with $x \in X_{\text{win}}$ corresponds to a play $\rho = v_0 v_1 \cdots$ from $v_0 = v_0^F$ in $\mathcal{G}^F$ such that every change in ℓ^+ and Υ corresponds to a move by Player 1 to a vertex with corresponding label in ρ. Furthermore, as $x \in X_{\text{win}}$, we have $v_1 \in \mathsf{Win}^F$. Moreover, by Definition 7.12, $\rho[1..]$ is a π^F-play starting from the winning region Win^F of game $\mathcal{G}^F$. So, $\rho[1..]$ and hence, ρ is a winning play (by prefix-independence of parity condition). Therefore, it always stays in Win^F. This implies, $\xi_{x,\mathtt{hc},\Upsilon}(t)$ also belongs to X_{win} for all $t \in \mathbb{R}_+$.

By the discussed correspondence between $\xi_{x,\mathtt{hc},\Upsilon}$ and play ρ, a trace γ generated by $\xi_{x,\mathtt{hc},\Upsilon}$ under ℓ is also the trace generated by the play ρ. Furthermore, every play in $\mathcal{G}^F$ corresponds to a play in the plant model G^C as in Definition 7.8. Moreover, by Proposition 7.2, γ satisfies (7.6). Then by Lemma 7.6 and Definition 7.10, γ is generated by a play in G^F satisfying $\Lambda_{\text{PERS}}(P_\ell^F)$. Hence, ρ satisfies $\Lambda_{\text{PERS}}(P_\ell^F)$. Moreover, as ρ is a winning play in $\mathcal{G}^F$, by Proposition 7.3, trace γ satisfies the specification Φ. □

7.5 Synthesis Details: High-Level

The previous section described our synthesis framework and established its ability to solve Problem 7.1 in Theorem 7.1. The main hypotheses in this statement are the existence of

1. a memoryless strategy that is uniformly winning for the final game $\mathcal{G}^F$, and

2. a CLF w for each $\mathtt{cRWA}$.

Within this section we give a novel algorithm to efficiently solving *augmented parity games* constructed in Section 7.4.3, thus tackling the first point. The second hypothesis is treated in subsequent Section 7.6, which presents the construction of low-level controllers implementing cRWAs via CLFs used in Section 7.4.2.

Algorithm 7.1 AttrPers(G, T, P_ℓ)

Input: An augmented game $\mathcal{G} = (G, \Phi, P_\ell)$ with $\Phi = \Diamond T$
Output: Winning region and uniformly winning strategy for $\mathcal{G}$

$\pi \leftarrow$ arbitrary memoryless Player 0 strategy
$(A, \pi^A) \leftarrow \text{Attr}^0(T)$
$\pi(v) \leftarrow \pi^A(v)$ for every $v \in A \setminus T$
for $(\mathtt{S}, \mathtt{C}, \mathtt{T}) \in P_\ell$ **do**
  if $(\mathtt{S} \setminus A) \cap \mathsf{pre}(A) \neq \emptyset$ **then**
    $(B, \pi^B) \leftarrow \text{solve}(G|_{\mathtt{C}}, \Phi_B)$ with $\Phi_B = \Diamond A \vee \Box(\mathtt{S} \setminus \mathtt{T})$
    if $B \not\subseteq A$ **then**
      $\pi(v) \leftarrow \pi^B(v)$ for every $v \in B \setminus A$
      $(C, \pi^C) \leftarrow \text{AttrPers}(G, A \cup B, P_\ell)$
      $\pi(v) \leftarrow \pi^C(v)$ for every $v \in C \setminus (A \cup B)$
      return (C, π)
return (A, π)

7.5.1 Augmented Reachability Games

In order to efficiently solve augmented games, we leverage the recent insight that local liveness constraints on the environment player typically fall into a class of synthesis problems that allow for an efficient direct synthesis procedure [209, 27]. The augmented games we consider are similar to the ones discussed by Sun et al. [209]. We, however, provide a novel algorithm that tackles the full class of parity games and thereby subsumes the restricted problem class considered by Sun et al. [209].

As discussed in Section 2.5.3, most efficient known algorithm to solve classical (non-augmented) parity games are Zielonka's algorithm [225] (for practical purposes) and quasi-polynomial time algorithms [52, 171, 137]. Such algorithms recursively solve reachability games for both players to compute a winning region and a uniformly winning strategy of the controller player in the original parity game. In order to mimic those algorithms for augmented games, we first discuss an algorithm to solve augmented *reachability games*.

An *augmented reachability game* is a tuple $\mathcal{G} = (G, \Phi, P_\ell)$ where the specification $\Phi = \Diamond T$ is to finally reach a set $T \subseteq V$ of target vertices. The new recursive algorithm that solves an augmented reachability game $\mathcal{G}$ is given in Algorithm 7.1. The main idea of the algorithm is to first compute the set of vertices A from which Player 0 can reach T even without the help of any persistent live group constraints (Algorithm 7.1) along with the corresponding strategy π for Player 0 (Algorithm 7.1). Afterwards, the algorithm computes the set of states B from which Player 0 has a strategy (i.e. π^B) to reach A with the help of a persistent live group (Algorithm 7.1). If this set B enlarges the winning state set A (Algorithm 7.1), we use recursion to solve another augmented reachability game with target $T := A \cup B$ (Algorithm 7.1).

Within Algorithm 7.1, we use the following notation. Given a game graph $G = (V, E)$

and a persistent live group $(\mathtt{S}, \mathtt{C}, \mathtt{T})$, we write $G|_{\mathtt{C}}$ to denote the restricted game graph[3] (V, E') such that $E' \subseteq E$ and for every edge $e = (u, v) \in E'$, either $e \in \mathtt{C}$ or $u \notin src(\mathtt{C})$. Furthermore, $\mathsf{pre}(T) \subseteq V$ is the set of vertices from which there is an edge to T.

For a set T of vertices, the attractor function $\textsc{Attr}^p(T)$ solves the (non-augmented) reachability game $(G, \Diamond T)$ for Player p and returns the attractor set, i.e., winning region $A := \mathsf{attr}^p(T) \subseteq V$, and a memoryless attractor strategy π^A of Player p, i.e., a winning strategy from A. Intuitively, A collects all vertices from which Player p has a strategy (i.e., π^A) to force every play starting in A to visit T in a finite number of steps. Moreover, the function $\textsc{Solve}(G, \Phi)$ returns the winning region and a memoryless uniformly winning strategy (for Player 0) in a game (G, Φ) with $\Phi = \Diamond A \vee \Box \neg T$ for some $A, T \subseteq V$. While ATTR can be implemented by reachability game algorithms as discussed in Section 2.5.3, SOLVE can be implemented using standard algorithms for solving safety games as discussed in the following remark.

Remark 7.1. *Given a game $\mathcal{G} = (G = (V, E), \Phi)$ where $\Phi = \Diamond A \vee \Box S$ for some $A, S \subseteq V$, one can reduce the game to a smaller safety game $(G', \Phi' = \Box S')$, where $S' = S \cup \{v_A\}$ and G' is the game graph obtained from G by merging all vertices in A to a single new sink vertex v_A, i.e., all incoming edges to A are retained but v_A has only one outgoing edge that is (v_A, v_A). In such a game, the winning region is $V \setminus \mathsf{attr}^1(V \setminus S')$ [22].*

With this, we can prove the correctness of Algorithm 7.1.

Theorem 7.2. *Given an augmented game $\mathcal{G} = (G, \Phi, P_\ell)$ with $\Phi = \Diamond T$, the algorithm $\textsc{AttrPers}(G, T, P_\ell)$ returns the winning region and a memoryless strategy that is uniformly winning in game $\mathcal{G}$. Moreover, the algorithm terminates in $\mathcal{O}(|P_\ell| \cdot |V| \cdot |E|)$ time.*

Proof. Suppose Win be the winning region in the augmented game $\mathcal{G}$. Using induction on the number of times $\textsc{AttrPers}(\cdot)$ is called, we show that the set returned by the algorithm is indeed Win, and the updated (memoryless) strategy π returned by the algorithm is a uniformly winning strategy in $\mathcal{G}$.

Base case: If $\textsc{AttrPers}(\cdot)$ is never called, i.e., the algorithm returned (A, π) in Algorithm 7.1. Hence, we need to show that $A = \mathsf{Win}$.

First, let us show that $A \subseteq \mathsf{Win}$. By the definition of attractor function $\textsc{Attr}^0(T)$, every π^A-play from A eventually visits T, and hence, satisfies Φ (which is stronger than $\Lambda_{\text{PERS}}(P_\ell) \Rightarrow \Phi$). Therefore, every vertex in A is trivially winning in $\mathcal{G}$, and hence, $A \subseteq \mathsf{Win}$.

Now, for the other direction, suppose v be a vertex such that $v \notin A$. It is enough to show that $v \notin \mathsf{Win}$. As $v \notin A = \mathsf{attr}^0(T)$, Player 0 can not force the plays to visit T. If $v \notin \mathtt{S}$ for every $(\mathtt{S}, \mathtt{C}, \mathtt{T}) \in P_\ell$, then the persistent group-liveness constraints are not relevant for vertex v. Now, suppose $v \in \mathtt{S}$ for some $(\mathtt{S}, \mathtt{C}, \mathtt{T}) \in P_\ell$. As the algorithm did not reach Algorithm 7.1, for every persistent live group, one of the conditional statements,

[3]Note that unlike the restriction of a game graph to a set of vertices, here we do *not* simply restrict the edge set to $\mathtt{C}$ only.

the one in Algorithm 7.1 or the one in Algorithm 7.1, is not satisfied. If the statement in Algorithm 7.1 is not satisfied, i.e., $(\mathtt{S} \setminus A) \cap \mathsf{pre}(A) = \emptyset$, then there is no edge from $\mathtt{S} \setminus A$ to A, and hence, this persistent live group constraint does not help in reaching A from $V \setminus A$ anyway.

Next, if the statement in Algorithm 7.1 is not satisfied, then it holds that $B \subseteq A$. Hence, $v \notin B$. As B is the winning region for game $(G|_{\mathtt{C}}, \Phi_B)$ and such a game is determined [22], Player 1 has a strategy π_1 such that every π_1-play in this game starting from v satisfies $\neg\Phi_B = \Box\neg A \wedge \Diamond(\mathtt{T} \cup V \setminus \mathtt{S})$. Therefore, every π_1-play trivially satisfies $\Lambda_{\text{PERS}}(\mathtt{S}, \mathtt{C}, \mathtt{T})$ without ever reaching A. Hence, if Player 1 sticks to strategy π_1, Player 0 can not make the plays from v visit $A \supseteq T$ using this constraint. Therefore, in any case, Player 0 has no strategy that can enforce a play from v to satisfy $\Lambda_{\text{PERS}}(P_\ell) \Rightarrow \Diamond T$. Hence, $v \notin \mathsf{Win}$.

Now, let us show that the returned strategy π is indeed a uniformly winning strategy in $\mathcal{G}$. As π^A is the attractor strategy to reach T, Algorithm 7.1, it is easy to verify that every π-play starting from $A \setminus T$ eventually visits T, and hence satisfies Φ. Therefore, every π-play from A is winning.

Induction case: Suppose the algorithm returned (C, π) in Algorithm 7.1 for some $(\mathtt{S}, \mathtt{C}, \mathtt{T}) \in P_\ell$. By induction hypothesis, C is the winning region and π^C is a uniformly winning strategy in the augmented game $\mathcal{G}_C = (G, \Phi_C, P_\ell)$ with $\Phi_C = \Diamond(A \cup B)$.

First, let us show that $\mathsf{Win} \subseteq C$. By the definition of attractor set $\mathsf{attr}^0(\cdot)$, it is easy to see that $T \subseteq A$. So, every play in G satisfies $\Diamond T \Rightarrow \Diamond(A \cup B)$. Therefore, a winning play in augmented game (G, T, P_ℓ) is also winning in augmented game $(G, A \cup B, P_\ell)$. Therefore, $\mathsf{Win} \subseteq C$.

Now, for the other direction, let us first show that $B \subseteq \mathsf{Win}$. As π^B is a uniformly winning strategy in game $\mathcal{G}_B$, every π^B-play ρ starting in B satisfies Φ_B. By definition of Φ_B, either ρ satisfies $\Diamond A$ or it satisfies $\Box(\mathtt{S} \setminus \mathtt{T})$. Suppose ρ satisfies $\Box(\mathtt{S} \setminus \mathtt{T})$. As ρ is a play in $G|_{\mathtt{C}}$, it satisfies $\Box(\mathtt{S} \wedge \Lambda_{\text{CONT}}(\mathtt{C}))$. Hence, if ρ satisfies $\Lambda_{\text{PERS}}(\mathtt{S}, \mathtt{C}, \mathtt{T})$, then it also satisfies $\Diamond\mathtt{T}$, which is a contradiction to the assumption that ρ satisfies $\Box(\mathtt{S} \setminus \mathtt{T})$. Therefore, ρ can not satisfy both $\Lambda_{\text{PERS}}(\mathtt{S}, \mathtt{C}, \mathtt{T})$ and $\Box(\mathtt{S} \setminus \mathtt{T})$. As a consequence, ρ satisfies $\Lambda_{\text{PERS}}(\mathtt{S}, \mathtt{C}, \mathtt{T}) \Rightarrow \Diamond A$. Furthermore, as we know, $A \subseteq \mathsf{Win}$. Therefore, ρ satisfies $\Diamond A \Rightarrow \Diamond\mathsf{Win}$, and hence, satisfies $\Lambda_{\text{PERS}}(\mathtt{S}, \mathtt{C}, \mathtt{T}) \Rightarrow \Diamond\mathsf{Win}$. So, every π^B-play starting in B satisfies $\Lambda_{\text{PERS}}(P_\ell) \Rightarrow \Diamond\mathsf{Win}$. Then, one can construct a Player 0 strategy π_0 (i.e., the one that uses π^B until the play reaches the winning region Win of game $\mathcal{G}$, and then switches to a uniformly winning strategy of game $\mathcal{G}$) such that every π_0-play starting in B satisfies the following

$$(\Lambda_{\text{PERS}}(P_\ell) \Rightarrow \Diamond\mathsf{Win}) \wedge \Box(\mathsf{Win} \wedge \Lambda_{\text{PERS}}(P_\ell) \Rightarrow \Diamond T), \tag{7.18}$$

and hence, satisfies $\Lambda_{\text{PERS}}(P_\ell) \Rightarrow \Diamond T$. Therefore, $B \subseteq \mathsf{Win}$.

Now, let us the other direction for induction case, i.e., $C \subseteq \mathsf{Win}$. As $B \subseteq \mathsf{Win}$ and $A \subseteq \mathsf{Win}$ as proven by the arguments given in base case, it holds that $A \cup B \subseteq \mathsf{Win}$. So, every play in G satisfies $\Diamond(A \cup B) \Rightarrow \Diamond\mathsf{Win}$. Furthermore, as π^C is a uniformly winning strategy in game $\mathcal{G}_C$, every π^C-play starting in C satisfies $\Lambda_{\text{PERS}}(P_\ell) \Rightarrow \Diamond(A \cup B)$, and

hence, satisfies $\Lambda_{\text{PERS}}(P_\ell) \Rightarrow \Diamond \mathsf{Win}$. Then, as in the last paragraph, one can construct a Player 0 strategy π_0 (i.e., the one that uses π^C until the play reaches the winning region Win of game $\mathcal{G}$, and then switches to a uniformly winning strategy of game $\mathcal{G}$) such that every π_0-play starting in C satisfies (7.18). Hence, every π_0-play starting in C satisfies $\Lambda_{\text{PERS}}(P_\ell) \Rightarrow \Diamond T$. Therefore, $C \subseteq \mathsf{Win}$.

Now, let us show that the returned strategy π in Algorithm 7.1 is also a uniformly winning strategy in game $\mathcal{G}$. As π is follows strategy π^C for vertices in $C \setminus (A \cup B)$, every π-play from $C \setminus (A \cup B)$ eventually visits $A \cup B$ when $\Lambda_{\text{PERS}}(P_\ell)$ holds. Now, let π^M be the updated strategy until Algorithm 7.1. Then, from Algorithm 7.1, it is easy to see that $\pi(v) = \pi^M(v)$ for every vertex v in $A \cup B$. As π^B is a uniformly winning strategy in game $\mathcal{G}_B$, using Algorithm 7.1 and the discussion above, every π-play from $B \setminus A$ eventually visits A when $\Lambda_{\text{PERS}}(P_\ell)$ holds. Then, using arguments of base case, every π-play from $A \setminus T$ eventually visits T. Therefore, in total, every π-play from C eventually visits T when $\Lambda_{\text{PERS}}(P_\ell)$ holds. Hence, π is indeed a uniformly winning strategy in game $\mathcal{G}$.

Time complexity: Let k be the number of times $\textsc{AttrPers}(\cdot)$ is called. If $T = V$, then $A = V$, and hence, $\mathtt{S} \setminus A = \emptyset$ for every $(\mathtt{S}, \mathtt{C}, \mathtt{T}) \in P_\ell$, and hence, $\textsc{AttrPers}(\cdot)$ will never be called. Furthermore, if $T \neq V$, then, by definition of $\mathsf{attr}^0(\cdot)$, it holds that $T \subseteq A$. So, in Algorithm 7.1, we keep adding at least one vertex to the target for the next call of $\textsc{AttrPers}(\cdot)$. Hence, k can be at most $|V|$. Moreover, in each iteration, we might need to solve game $(G|_{\mathtt{C}}, \Phi_B)$ for each $(\mathtt{S}, \mathtt{C}, \mathtt{T}) \in P_\ell$; and using Remark 7.1, solving such a game can be reduced to computing an attractor function $\mathsf{attr}^1(\cdot)$. As computing such an attractor function takes $\mathcal{O}(|E|)$ time [22], the algorithm takes $\mathcal{O}(|P_\ell| \cdot |V| \cdot |E|)$ time in total. □

7.5.2 Augmented Parity Games

To solve parity games augmented with persistent live groups, we use the fact that most of the algorithms to solve (non-augmented) parity games are based on the attractor functions $\textsc{Attr}^0(T)$. Furthermore, the only difference between the attractor function $\textsc{Attr}^0(T)$ and our new function $\textsc{AttrPers}(G, T, P_\ell)$ from Algorithm 7.1 is the utilization of augmented persistent live groups to solve reachability games. Hence, in many of these algorithms for parity games, one can simply replace every use of $\textsc{Attr}^0(T)$ with $\textsc{AttrPers}(G, T, P_\ell)$ to obtain an algorithm to solve parity games augmented with persistent live groups.

Let us first consider the classical recursive algorithm given by Zielonka [225] to solve parity games. The algorithm recursively solves smaller parity games obtained by restricting the game graph to a smaller set of vertices. To adapt this algorithm to solve parity games augmented with persistent live groups, we first need to define how the persistent live groups are transformed when we restrict the game graph in such a way. In addition, we also need to ensure that the transformed persistent live groups still express the same assumption w.r.t. the restricted game graph. Intuitively, one easy way to do this is by restricting all three sets $\mathtt{S}$, $\mathtt{C}$, $\mathtt{T}$ of a persistent live group to the set of vertices and edges

in the restricted game graph. However, if there is an edge $e = (v, w)$ in $\mathtt{C}$, then the constraint $\Lambda_{\text{CONT}}(\mathtt{C})$ enforces that edge e is taken from vertex v. If the restricted game graph contains v but not the edge e, then to satisfy the constraint $\Lambda_{\text{CONT}}(\mathtt{C})$, we need to ensure that vertex v is not visited. Hence, we need to remove such vertices from $\mathtt{S}$ in the transformed persistent live groups. This is formalized below.

Definition 7.14. Given a game graph $G = (V, E)$ augmented with a set P_ℓ of persistent live groups, and a set $U \subseteq V$, we define the set of persistent live groups restricted to U as $P_\ell|_U = \{(\mathtt{S}|_U, \mathtt{C}|_U, \mathtt{T}|_U) \mid (\mathtt{S}, \mathtt{C}, \mathtt{T}) \in P_\ell\}$ s.t.

$$\mathtt{T}|_U = \mathtt{T} \cap U, \quad \mathtt{C}|_U = \{(u, v) \in E \mid u, v \in U\}, \quad \mathtt{S}|_U = (\mathtt{S} \cap U) \setminus (\mathit{src}(\mathtt{C}) \setminus \mathit{src}(\mathtt{C}')).$$

One can show that $P_\ell|_U$ indeed captures the same assumption as P_ℓ w.r.t. the game restricted to U as formalized below.

Lemma 7.7. *Given an augmented parity game $\mathcal{G} = (G, \Phi, P_\ell)$ and a set $U \subseteq V$, let ρ be a play in $G|_U$. Then, ρ is winning in augmented game $\mathcal{G}|_U = (G|_U, \Phi|_U, P_\ell|_U)$ if and only if it is winning in augmented game $\mathcal{G}$.*

Proof. Suppose ρ is winning in $\mathcal{G}$, then either it satisfies the parity objective Φ or doesn't satisfy the persistent live group assumptions $\Lambda_{\text{PERS}}(P_\ell)$. If it satisfies Φ, then it also satisfies $\Phi|_U$ as it only visits vertices in U. Else if it doesn't satisfy persistent live group assumptions, then there exists a persistent live group $(\mathtt{S}, \mathtt{C}, \mathtt{T})$ such that ρ doesn't satisfy $\Lambda_{\text{PERS}}(\mathtt{S}, \mathtt{C}, \mathtt{T})$. Hence, there exists a suffix ρ' of ρ that satisfies $\Box(\mathtt{S} \wedge \Lambda_{\text{CONT}}(\mathtt{C})) \wedge \Box\neg\mathtt{T}$. As ρ' stays inside U and edges of $\mathtt{C}$ used in ρ' also belong to $\mathtt{C}|_U$, ρ' also satisfies $\Box(\mathtt{S}|_U \wedge \Lambda_{\text{CONT}}(\mathtt{C}|_U)) \wedge \Box\neg\mathtt{T}|_U$. Hence, ρ also doesn't satisfy the persistent live group assumptions $\Lambda_{\text{PERS}}(P_\ell|_U)$. Therefore, in all cases, ρ is also winning in $\mathcal{G}|_U$.

Now, for the other direction, let ρ is not winning in $\mathcal{G}$. Hence, ρ satisfies the persistent live group assumptions $\Lambda_{\text{PERS}}(P_\ell)$ but not parity objective Φ. So, ρ also doesn't satisfy parity objective $\Phi|_U$. We only need to show that ρ satisfies $\Lambda_{\text{PERS}}(P_\ell|_U)$. Consider a persistent live group $(\mathtt{S}, \mathtt{C}, \mathtt{T}) \in P_\ell$ and a suffix ρ' of ρ. It is enough to show that ρ' satisfies $\Lambda_{\text{PERS}}(\mathtt{S}|_U, \mathtt{C}|_U, \mathtt{T}|_U)$. Since the suffix ρ' satisfies $\Lambda_{\text{PERS}}(\mathtt{S}, \mathtt{C}, \mathtt{T})$, it satisfies one of the following: $\Diamond T$ or $\Diamond\neg\mathtt{S}$ or $\Diamond\neg\Lambda_{\text{CONT}}(\mathtt{C})$. If ρ' satisfies $\Diamond T$ or $\Diamond\neg\mathtt{S}$, as ρ' always stays inside U, it also satisfies $\Diamond T|_U$ or $\Diamond\neg\mathtt{S}|_U$, respectively and hence, it satisfies $\Lambda_{\text{PERS}}(\mathtt{S}|_U, \mathtt{C}|_U, \mathtt{T}|_U)$. Else if ρ' satisfies $\Diamond\neg\Lambda_{\text{CONT}}(\mathtt{C})$, then for some vertex $v \in \mathit{src}(\mathtt{C})$, ρ' visits v but doesn't take any edge in $\mathtt{C}$ from v. If $v \in \mathit{src}(\mathtt{C}|_U)$, then ρ' also satisfies $\Diamond\neg\Lambda_{\text{CONT}}(\mathtt{C}|_U)$. Otherwise, if $v \notin \mathit{src}(\mathtt{C}|_U)$, then by definition, $v \notin \mathtt{S}|_U$, and hence, ρ' satisfies $\Diamond\neg\mathtt{S}|_U$. Therefore, in all cases, ρ' satisfies $\Lambda_{\text{PERS}}(\mathtt{S}|_U, \mathtt{C}|_U, \mathtt{T}|_U)$. So, ρ is not winning in $\mathcal{G}|_U$. □

With this well-defined restrictions, one can replace every use of $\text{ATTR}^0(T)$ with $\text{ATTRPERS}(G, T, P_\ell)$ in the classical recursive algorithm given by Zielonka [225] to obtain an exponential-time algorithm for parity games augmented with persistent live groups. Furthermore, as Zielonka's algorithm can also be used to compute a memoryless uniformly winning strategy in addition to the winning region, the same holds for the adapted algorithm for augmented parity games. This is formalized below.

Corollary 7.1. *Given an augmented parity game $\mathcal{G} = (G, Parity(\mathbb{P}), P_\ell)$ with game graph $G = (V, E, L)$ and priority function $\mathbb{P}\colon V \to [0, d]$, one can compute the winning region and a memoryless uniformly winning strategy in $\mathcal{O}\left(|P_\ell| \cdot |V|^{d+\mathcal{O}(1)}\right)$ time.*

7.5.3 Quasi-Polynomial Algorithm

While the classical recursive algorithm given by Zielonka [225] runs in exponential time, it is well-known that parity games can be solved in quasi-polynomial time. We now show that one can also obtain a quasi-polynomial algorithm for augmented parity games by applying the same technique to the quasi-polynomial algorithm given by Lehtinen et al. [137] and Parys [171].

Algorithm

We present a quasi-polynomial time algorithm SolvePers for solving augmented parity games in Algorithm 7.2 (adapted from the one given by Parys [171, Algo. 2]). The algorithm uses two procedures $\textsc{Attr}^0_{\text{pers}}$ and $\textsc{Attr}^1_{\text{pers}}$ to compute the corresponding attractor sets and attractor strategies for players Player 0 and Player 1, respectively, in augmented reachability games. The procedure $\textsc{Attr}^0_{\text{pers}}(G, T, P_\ell)$ uses the procedure AttrPers to return the winning region $\mathsf{attr}^0_{\text{pers}}(G, T, P_\ell)$ and a memoryless uniformly winning strategy for the augmented reachability game $(G, \Diamond T, P_\ell)$. The procedure $\textsc{Attr}^1_{\text{pers}}(G, T, P_\ell)$ simply returns the attractor set $\mathsf{attr}^1(T)$ along with a memoryless attractor strategy for Player 1. Without loss of generality[4], Algorithm 7.2 assumes that there are no self-loops in the input game graph.

The main idea of the algorithm is similar to Zielonka's recursive algorithm for solving parity games [225], where each recursive call tries to remove some *dominions* of a player, i.e., sets of vertices from which a player can enforce winning while staying inside the set, until no vertices are left. Here, each call to RecQSolve tries to remove such winning vertices of Player $1-p$ from the game graph by the following steps: (i) compute the attractor set A for player $p \in \mathtt{P}$ to the set of vertices with the highest priority h (Algorithm 7.2), where h is even if $p = 0$ and odd otherwise; (ii) recursively call $\textsc{QSolve}_{1-p}$ on the subgame obtained by removing A from the game graph to compute some winning vertices W_{1-p} for Player $1-p$ (Algorithm 7.2), which are guaranteed to be winning for Player $1-p$ in $\mathcal{G}$ as the highest priority h has been removed; (iii) compute the attractor set B for Player $1-p$ to the winning vertices W_{1-p} (Algorithm 7.2), which are also winning for Player $1-p$ in $\mathcal{G}$.

The main difference from Zielonka's algorithm is the use of precision parameters η_0 and η_1 in the procedure $\textsc{QSolve}_p$ to limit the size the corresponding dominions that are removed in each recursive call. The key observation that is exploited here is that there can be only one dominion of size more than $|V|/2$. Hence, the procedure $\textsc{QSolve}_p$ utilizes this observation by doing three phases of removing dominions for Player $1-p$:

[4]Every self-loop on a vertex can be removed from the game graph by adding an edge back and forth to a new vertex with the same priority and ownership.

Algorithm 7.2 SolvePers$(G, Parity(\mathbb{P}), P_\ell)$

Input: An augmented parity game $\mathcal{G} = (G, Parity(\mathbb{P}), P_\ell)$ with game graph $G = (V, E)$ and priority function $\mathbb{P} : V \to [0; 2d]$
Output: Winning region for $\mathcal{G}$

```
Win ← QSolve_0(G, 2d, |V|, |V|)
return Win

procedure QSolve_p(G, h, η_p, η_{1−p})
    if G = ∅ ∨ η_p ≤ 1 then return ∅
    W_{1−p} ← V
    while W_{1−p} ≠ ∅ do (G, W_{1−p}) ← RecQSolve(G, h, ⌊η_{1−p}/2⌋, η_p)
    (G, W_{1−p}) ← RecQSolve(G, h, η_{1−p}, η_p)
    while W_{1−p} ≠ ∅ do (G, W_{1−p}) ← RecQSolve(G, h, ⌊η_{1−p}/2⌋, η_p)
    return V

procedure RecQSolve(G, h, η_{1−p}, η_p)
    N_h ← {v ∈ V | P(v) = h}
    A ← Attr^p_pers(G, N_h, P_ℓ)
    W_{1−p} ← QSolve_{1−p}(G|_{V\A}, h − 1, η_{1−p}, η_p)
    B ← Attr^{1−p}_pers(G, W_{1−p}, P_ℓ)
    return (G|_{V\B}, W_{1−p})
```

(i) first, it tries to remove all dominions of size at most $|V|/2$ by halving the precision parameter η_{1-p} in each iteration of the first while-loop (Algorithm 7.2); (ii) then, it tries once with the full precision parameter to remove any remaining dominion of size more than $|V|/2$ (Algorithm 7.2); (iii) finally, it again tries to remove all dominions of size at most $|V|/2$ by halving the precision parameter η_{1-p} in each iteration of the last while-loop (Algorithm 7.2). This idea was first introduced by Lehtinen et al. [171, 137] to obtain a quasi-polynomial time algorithm for solving standard parity games. We adapt this idea to augmented parity games with persistent live group assumptions by using the procedures $\text{Attr}^0_{\text{pers}}$ and $\text{Attr}^1_{\text{pers}}$ to compute the attractor sets in augmented reachability games.

Complexity Analysis of SolvePers

We begin by proving that Algorithm 7.2 runs in quasi-polynomial time.

Theorem 7.3. *Given an augmented parity game $\mathcal{G}$ with n vertices, SolvePers$(\mathcal{G})$ runs in time $n^{\mathcal{O}(\log n)}$.*

Proof. Given precision parameters η_0 and η_1 with $\lfloor \log \eta_0 \rfloor + \lfloor \log \eta_1 \rfloor = l$, let $R(h, l)$ denote the number of *non-trivial* executions of the procedures QSolve_0 and QSolve_1, i.e., those that do not terminate at the return statement in Algorithm 7.2, that occur within one

execution of $\text{QSolve}_p(\mathcal{G}, h, \eta_0, \eta_1)$. It is straightforward that $R(0,l) = R(h,0) = 0$. Furthermore, each execution of QSolve_{1-p} within QSolve_p (except for the final call of the while loops in Algorithm 7.2) removes at least one vertex from $\mathcal{G}$. Hence, there can be at most n such calls with halved precision and one call with full precision (in Algorithm 7.2). Moreover, in each call with halved precision, the value of $\lfloor \log \eta_{1-p} \rfloor$ (and thus l) decreases by one. Therefore, for $h, l \geq 1$, we have the following inequality:

$$R(h,l) \leq 1 + n \cdot R(h-1, l-1) + R(h-1, l)\,. \tag{7.19}$$

Let us prove the following claim using induction on the pair (h,l).

Claim 7.1. *For all $h, l \geq 0$, it holds that*

$$R(h,l) \leq n^l \cdot \binom{h+l}{l} - 1.$$

Proof. For base cases, if $h = 0$ or $l = 0$, then $R(h,l) = 0$ and the inequality holds trivially. Suppose the claim holds for $(h-1, l-1)$ and $(h-1, l)$, where $h, l \geq 1$. Then, by (7.19), we have

$$\begin{aligned} R(h,l) &\leq 1 + n \cdot R(h-1,l-1) + R(h-1,l) \\ &\leq 1 + n \cdot \left(n^{l-1} \cdot \binom{h+l-2}{l-1} - 1 \right) + n^l \cdot \binom{h+l-1}{l} - 1 \\ &\leq n^l \cdot \left(\binom{h+l-1}{l-1} + \binom{h+l-1}{l} \right) - 1 \\ &= n^l \cdot \binom{h+l}{l} - 1\,. \end{aligned}$$

This completes the proof of the claim. ◁

The above claim implies $R(h,l) \leq n^l \cdot (h+l)^l$. Since the procedure SolvePers begins with $l = 2\lfloor \log n \rfloor$ and $h = 2d$, this bound is

$$\begin{aligned} R(2d, 2\lfloor \log n \rfloor) &\leq n^{2\lfloor \log n \rfloor} \cdot (2d + 2\lfloor \log n \rfloor)^{2\lfloor \log n \rfloor} \\ &= n^{\mathcal{O}(\log n)} \cdot (d + \log n)^{\mathcal{O}(\log n)} = n^{\mathcal{O}(\log n)}\,. \end{aligned}$$

Furthermore, each execution of QSolve_p (excluding the running time of recursive calls) takes polynomial time, as the procedures $\text{Attr}^0_{\text{pers}}$ and $\text{Attr}^1_{\text{pers}}$ run in polynomial time by Theorem 7.2. Therefore, the overall running time of SolvePers is $n^{\mathcal{O}(\log n)}$. □

Correctness of SolvePers

We now prove the correctness of Algorithm 7.2, i.e., that it correctly computes the winning region in augmented parity games. As mentioned before, the algorithm is mainly based on properties of *dominions*. Hence, let us first formally define dominions in augmented games.

Definition 7.15. Given a game (G, Φ), a set $U \subseteq V$ is said to be a *dominion* for Player p if they can enforce every play from U to stay inside U in addition to being winning for them, i.e., there exists a Player p strategy π such that every π-play from U satisfies $\square U$ and is winning for Player p. Dominions for augmented games (G, Φ, P_ℓ) are defined w.r.t. the equivalent game $(G, \Lambda_{\text{PERS}}(P_\ell) \Rightarrow \Phi)$.

With this definition, it is easy to see that winning region is the largest dominion for Player 0. Before proving the correctness of the algorithm, let us prove some important properties of dominions that will be useful in the proof.

Lemma 7.8. *Given an augmented game $\mathcal{G} = (G = (V, E), \Phi, P_\ell)$ with a dominion S for Player p, the following holds for every set $X \subseteq V$:*

(a) *for $A = \mathsf{attr}^p_{pers}(G, X, P_\ell)$, the set $S \setminus A$ is a dominion for Player p in the augmented game $\mathcal{G}_A = \mathcal{G}|_{V \setminus A}$;*

(b) *if $S \cap X = \emptyset$, then for $B = \textsc{Attr}^{1-p}_{pers}(G, X, P_\ell)$, the set S is a dominion for Player p in the augmented game $\mathcal{G}_B = \mathcal{G}|_{V \setminus B}$.*

Proof. As S is a dominion for Player p in $\mathcal{G}$, there exists a Player p strategy π_p such that every π_p-play from S satisfies $\square S$ and is winning for Player p. Now, we prove each part of the lemma separately.

(a) By definition of attractor set, every Player p vertex $v \in V \setminus A$ has no successor in A and hence, π_p is well-defined in $\mathcal{G}_A$. Furthermore, since S is a dominion for Player p in $\mathcal{G}$, every π_p-play from $S \setminus A$ in $\mathcal{G}_A$ always stays in $S \setminus A$ and is winning for Player p w.r.t. $\mathcal{G}$ (and hence also w.r.t. $\mathcal{G}_A$ by Lemma 7.7). This implies that $S \setminus A$ is a dominion for Player p in $\mathcal{G}_A$.

(b) It is enough to show that $S \subseteq V \setminus B$, since this implies every π_p-play from S is also a π_p-play in $\mathcal{G}_B$ that stays inside S and is winning for Player p. Suppose there is some vertex $v \in S \cap B$. This means, every π_p-play from v stays inside S. Furthermore, by the definition of attractor set, there exists a Player $1-p$ strategy π_{1-p} such that every π_{1-p}-play from v reaches X. As $S \cap X = \emptyset$, this is a contradiction, and hence, $S \subseteq V \setminus B$. □

Using Lemma 7.8, we can now prove the correctness of Algorithm 7.2, i.e., the procedure SolvePers correctly computes the winning region for Player 0 in augmented parity games. This follows directly from the following theorem as Player 0 winning region is the largest dominion for Player 0.

Theorem 7.4. *Let $\mathcal{G}$ be an augmented parity game with at most h priorities. Suppose the procedure $QSolve_p(\mathcal{G}, h, \eta_p, \eta_{1-p})$ returns the set W_p, where $p = h \mod 2$, then we have the following:*

(a) *for every Player p dominion S with $|S| \leq \eta_p$, it holds that $S \subseteq W_p$; and*

(*b*) *for every Player* $1-p$ *dominion* S *with* $|S| \leq \eta_{1-p}$*, it holds that* $S \cap W_p = \emptyset$.

Proof. Let us prove using induction on h. For the base case, if $h = 0$, then $p = 0$ and in each execution of RecQSolve within QSolve_p, it holds that $N_h = V$ (computed in Algorithm 7.2), $A = V$ (computed in Algorithm 7.2), and $W_{1-p} = \emptyset$ (computed in Algorithm 7.2). Thus, $qsolve_p$ returns $W_p = V$. As there is no Player $1-p$ (i.e., Player 1) dominions in this case, both properties of the theorem hold trivially.

Now, for the inductive case, suppose the theorem holds for all augmented parity games with at most $h-1$ priorities, where $h \geq 1$. For each variable, let us denote its value within the i-th iteration of the procedure RecQSolve within QSolve_p by appending a superscript i to it, e.g., $\mathcal{G}^i$, W^i_{1-p}, etc. Suppose the procedure QSolve_p makes m such iterations, and let us denote the values after the final iteration by appending a superscript $m+1$ to them. We prove each property of the theorem separately.

(*a*) Let S be a Player p dominion with $|S| \leq \eta_p$. If $S = \emptyset$, then the first property holds trivially. Suppose $S \neq \emptyset$. As we assumed that there are no self-loops in the game graph, every dominion has at least two vertices. Therefore, $\eta_p \geq |S| \geq 2$, and hence, the procedure QSolve_p does not return at Algorithm 7.2. So, we need to show that $S \subseteq W_p = V^{m+1}$.

We show a stronger claim that for every $i \in [1; m+1]$, it holds that S is a dominion for Player p in the augmented game $\mathcal{G}^i = \mathcal{G}|_{V \setminus B^{i-1}}$, which implies $S \subseteq V^i$. Let us prove this claim using an internal induction on i. For the base case, if $i = 1$, then $\mathcal{G}^1 = \mathcal{G}$ and the claim holds trivially.

Now, for the inductive case, suppose the claim holds for i, i.e., S is a dominion for Player p in $\mathcal{G}^i$. Then, by Lemma 7.8(*a*), the set $S^i = S \setminus A^i$ is a dominion for Player p in the augmented game $\mathfrak{H}^i = \mathcal{G}^i|_{V^i \setminus A^i}$. Furthermore, since $\mathfrak{H}^i$ has at most $h-1$ priorities and $|S^i| \leq |S| \leq \eta_p$, the external inductive hypothesis on $\text{QSolve}_{1-p}(\mathfrak{H}^i, h-1, -, \eta_p)$ (called in Algorithm 7.2) implies that $S^i \cap W^i_{1-p} = \emptyset$. Then, $S \cap W^i_{1-p} = \emptyset$ as $W^i_{1-p} \subseteq V^i \setminus A^i$. Hence, by Lemma 7.8(*b*), the set S is a dominion for Player p in the augmented game $\mathcal{G}^{i+1} = \mathcal{G}^i|_{V^i \setminus B^i}$. This completes the proof of the claim and hence, the first property.

(*b*) Let S be a Player $1-p$ dominion with $|S| \leq \eta_{1-p}$. If $\eta_p \leq 1$, then the procedure QSolve_p returns $W_p = \emptyset$ at Algorithm 7.2 and the second property holds trivially. Suppose $\eta_p \geq 2$. Let $S^i = S \cap V^i$ for each $i \in [1; m+1]$, then we need to show that $S^{m+1} = S \cap W_p = \emptyset$.

We show this by proving several claims leading to it. Let us start by first showing the following claim by induction on i.

Claim 7.2. *For every* $i \in [1; m+1]$*, the set* S^i *is a dominion for Player* $1-p$ *in the augmented game* $\mathcal{G}^i$.

> *Proof.* For the base case, if $i = 1$, then $\mathcal{G}^1 = \mathcal{G}$ and the claim holds trivially. Now, for the inductive case, suppose the claim holds for i, i.e., S^i is a dominion for Player $1-p$ in $\mathcal{G}^i$. Then, by construction, $\mathcal{G}^{i+1} = \mathcal{G}^i|_{V^i \setminus B^i}$ and $S^{i+1} = S^i \setminus B^i$, where $B^i =$

$\textsc{Attr}^{1-p}_{\text{pers}}(G^i, W^i_{1-p}, P^i_\ell)$ (computed in Algorithm 7.2). Hence, by Lemma 7.8(a), the set S^{i+1} is a dominion for Player $1-p$ in $\mathcal{G}^{i+1}$. This completes the proof of the claim. $\lhd$

Now, for $i \in [1; m]$, let $Z^i \subseteq S^i \setminus N^i_h$ denote the set of vertices from which Player $1-p$ can win without seeing the highest priority h (which is even if $p = 0$ and odd otherwise), i.e., there exists a Player $1-p$ strategy π^i such that every π^i-play from Z^i satisfies $\Box(S^i \setminus N^i_h)$ and is winning for Player $1-p$ w.r.t. $\mathcal{G}^i$. Let us prove the following claim.

Claim 7.3. *If $S^i \neq \emptyset$, then $Z^i \neq \emptyset$.*

Proof. Suppose $S^i \neq \emptyset$ and $Z^i = \emptyset$. By Claim 7.2, there exists a Player $1-p$ strategy π such that every π-play from S^i satisfies $\Box S^i$ and is winning for Player $1-p$. As $Z^i = \emptyset$, π-plays from S^i cannot satisfy $\Box(S^i \setminus N^i_h)$. So, from every vertex in S^i, Player p has a strategy to eventually reach N^i_h against π. In particular, starting from some vertex $v_0 \in S^i$, Player p can force reaching N^i_h against π. Suppose the next vertex after visiting N^i_h is v_1. As $v_1 \notin Z^i$, Player p can again force reaching N^i_h against π from v_1. Continuing this way, Player p can force visiting N^i_h infinitely often against π starting from v_0. This implies that every π-play from v_0 visits the highest priority h infinitely often, and hence, is winning for Player p, which is a contradiction. Therefore, $Z^i \neq \emptyset$. $\lhd$

Let us also show that Z^i is a dominion for Player $1-p$ in $\mathfrak{H}^i$.

Claim 7.4. *For every $i \in [1; m]$, the set Z^i is a dominion for Player $1-p$ in the augmented game $\mathfrak{H}^i = \mathcal{G}^i|_{V^i \setminus A^i}$.*

Proof. By definition of Z^i, there exists a Player $1-p$ strategy π^i such that every π^i-play from Z^i satisfies $\Box(S^i \setminus N^i_h)$ and is winning for Player $1-p$ w.r.t. $\mathcal{G}^i$. This implies, if a π^i-play from Z^i reaches a vertex v, then $v \in Z^i$ as π^i-plays from v also satisfy $\Box(S^i \setminus N^i_h)$ and are winning for Player $1-p$. Hence, every π^i-play from Z^i stays inside Z^i. This means that Z^i is a dominion for Player $1-p$ in $\mathcal{G}^i$. Furthermore, as $Z^i \cap N^i_h = \emptyset$, by Lemma 7.8($b$), Z^i is also a dominion for Player $1-p$ in $\mathfrak{H}^i$. $\lhd$

Let the procedure RecQSolve is called $k-1$ times at Algorithm 7.2 before the k-th call to RecQSolve with full precision at Algorithm 7.2. We now prove the final claim leading to the second property.

Claim 7.5. $Z^m \subseteq W^m_{1-p}$.

Proof. First, recall that in each call to RecQSolve within $\textsc{QSolve}_p$, the set W^i_{1-p} is computed by calling $\textsc{QSolve}_{1-p}(\mathfrak{H}^i, h-1, \eta^i_{1-p}, \eta_p)$ (in Algorithm 7.2), where $\eta^k_{1-p} = \eta_{1-p}$ and $\eta^i_{1-p} = \lfloor \eta_{1-p}/2 \rfloor$ for $i \neq k$. Hence, as $Z^i \subseteq S^i \subseteq S$ and $|S| \leq \eta_{1-p}$, by the external inductive hypothesis on $\textsc{QSolve}_{1-p}(\mathfrak{H}^i, h-1, \eta^i_{1-p}, \eta_p)$ and by Claim 7.4, the following holds for every $i \in [1; m]$:

$$\text{if } \left|Z^i\right| \leq \eta^i_{1-p} \text{ or } i = k, \text{ then } Z^i \subseteq W^i_{1-p}. \tag{7.20}$$

Now, if $k = m$, then the claim holds directly from (7.20). Suppose $k < m$. By (7.20), we only need to show that $|Z^m| \leq \lfloor \eta_{1-p}/2 \rfloor$. As $Z^m \subseteq S^m \subseteq S^{k+1}$, it is enough to show that $\left|S^{k+1}\right| \leq \lfloor \eta_{1-p}/2 \rfloor$.

If $S^{k-1} = \emptyset$, then this holds trivially as $S^{k+1} \subseteq S^{k-1}$. Suppose $S^{k-1} \neq \emptyset$. Then, by Claim 7.3, $Z^{k-1} \neq \emptyset$. Furthermore, by the termination condition of the while-loop in Algorithm 7.2, we have $W^{k-1}_{1-p} = \emptyset$. Hence, by (7.20), it must be that $\left|Z^{k-1}\right| > \lfloor \eta_{1-p}/2 \rfloor$. As $W^{k-1}_{1-p} = \emptyset$, we have $\mathcal{G}^k = \mathcal{G}^{k-1}$, $S^k = S^{k-1}$, and $Z^k = Z^{k-1}$. As $Z^k \subseteq W^k_{1-p}$ by (7.20), we have $S^{k+1} = S^k \setminus B^k \subseteq S^k \setminus W^k_{1-p} \subseteq S^k \setminus Z^k$. Therefore, we have $\left|S^{k+1}\right| \leq \left|S^k\right| - \left|Z^k\right| \leq \eta_{1-p} - (\lfloor \eta_{1-p}/2 \rfloor + 1) = \lfloor \eta_{1-p}/2 \rfloor$. This completes the proof of the claim. ◁

By the termination condition of the while-loop in Algorithm 7.2, we have $W^m_{1-p} = \emptyset$. Hence, by Claim 7.5, we have $Z^m = \emptyset$. Then, by Claim 7.3, we have $S^m = \emptyset$, and hence, $S^{m+1} = \emptyset$ as $S^{m+1} \subseteq S^m$. This completes the proof of the second property. □

Combining Theorems 7.3 and 7.4, we obtain the main result of this section.

Corollary 7.2. *Augmented parity games can be solved in quasi-polynomial time in the number of vertices.*

7.6 Synthesis Details: Low-Level

This section illustrates an efficient and flexible numerical method to design CLFs which can then be used to design low-level controllers via Lemma 7.1.

7.6.1 Synthesis of Low-Level Controllers from cRWAs

It is well-known that the problem of synthesizing CLFs (in the sense of Section 7.4.2) for general nonlinear control systems (as in Definition 7.1) over a generic state space $X \subseteq \mathbb{R}^{n_x}$ solving a generic cRWA problem $\Omega = (\kappa, \mathcal{R}, \mathcal{A})$ is numerically intractable [39]. For this reason, particular characteristics of the system and its dynamics need to be exploited for tractability. In this section, we therefore restrict the discussion to systems with *affine dynamics*, as mature computational solutions exist for this systems class. In particular, we present a novel approach to controller synthesis for cRWA problems over affine dynamical systems, by means of semidefinite optimization, considering a class of quadratic control Lyapunov functions.

While this only gives a construction for the top-down interface in Section 7.4.2 for affine dynamical systems, we note that our overall hybrid controller synthesis approach discussed in Section 7.4 and summarized in Figure 7.2 can be applied to any dynamical system for which the generated cRWA problem can be solved. In particular, recent optimization-based approaches for enforcing logical constraints on more general nonlinear systems [221, 120, 68] can be utilized. We leave the integration of these methods into our synthesis framework for future work.

Assumption 7.1. The control system $\mathcal{S} = (X, U, f)$ has *affine dynamics* of the form

$$f(x, u) \coloneqq Ax + Bu + g, \tag{7.21}$$

for some $A \in \mathbb{R}^{n_x \times n_x}$, $B \in \mathbb{R}^{n_x \times n_u}$ and $g \in \mathbb{R}^{n_x}$. Moreover, we suppose that the input space is a convex polytope, i.e. $U = \mathbb{H}(y_U, H_U) \coloneqq \{x \in \mathbb{R}^{n_x} \ : \ H_U^\top (x - y_U) \leq \mathbf{1}\}$, for some y_U and H_U of appropriate dimensions.

In addition, we restrict the shape of the state-space regions linked to state propositions $\mathtt{AP}_S$.

Assumption 7.2. Given a state proposition $\mathcal{T} \in \mathtt{AP}_S$ its corresponding state-space region is either ellipsoidal of the type $\mathbb{E}(z, S) = \{x \in \mathbb{R}^{n_x} \ : \ (x - z)^\top S (x - z) \leq 1\}$ or a convex polytope $\mathbb{H}(y, H) = \{x \in \mathbb{R}^{n_x} \ : \ H^\top (x - y) \leq \mathbf{1}\}$, where $S \in \mathbb{R}^{n_x \times n_x}$ is a symmetric positive semidefinite matrix, $z, y \in \mathbb{R}^{n_x}$ are vectors and $H \in \mathbb{R}^{n_x \times m}$.

Under these assumptions, instead of searching for control Lyapunov functions all over the set of $\mathscr{C}^1$ functions, we restrict our search to *quadratic functions* of the form

$$w(x) = (x - x_c)^\top P (x - x_c), \tag{7.22}$$

where $x_c \in X$ is the *center of* w and $P \in \mathbb{R}^{n_x \times n_x}$, $P \succ 0$.

Inspired by the results by He et al. [116], we present a method to design a CLF $w(x)$ in the form of (7.22) *associated* with a cRWA problem $\Omega = (\kappa, \mathcal{R}, \mathcal{A})$ (as in Definition 7.6) in three steps:

(**A**) *Find* x_c such that $\mathcal{R} \subset \ell(x_c)$ and $\mathcal{A} \cap \ell(x_c) = \emptyset$.

(**B**) *Find a safe set* $\mathbb{S} \subseteq X$ such that $x_c \in \mathbb{S}$ and $\mathcal{A} \cap \ell(x) = \emptyset$ for all $x \in \mathbb{S}$.

(**C**) *Construct a CLF* w such that its basin of attraction is safe, i.e., $X_w \subseteq \mathbb{S}$.

These steps must be performed with awareness of the context κ and the changes that it causes in the continuous state space. First, (**A**) is a necessary condition for the existence of a CLF that generates a feasible controller for Ω. However, given that the set difference between the convex regions where $\mathcal{R}$ and $\mathcal{A}$ hold is potentially non-convex, checking whether such x_c exists is a very difficult problem. To avoid resorting to global optimization strategies such as branch-and-bound algorithms, we introduce another assumption.

Assumption 7.3. Given a cRWA problem $\Omega = (\kappa, \mathcal{R}, \mathcal{A})$, for all $x \in X$ such that $\mathcal{R} \subset \ell(x)$ we have $x \notin \mathcal{E}_\mathcal{A}$, where $\mathcal{E}_\mathcal{A} \subset 2^X$ is an ellipsoidal region associated with a proposition in $\mathcal{A}$.

Assumption 7.3 requires that any ellipsoidal set that is to be avoided in Ω does not intersect the region associated to $\mathcal{R}$, i.e. the region to be reached. In practice, if it is not the case, one can replace ellipsoidal obstacles by polytopic over-approximations.

Lemma 7.9. *A point x_c satisfying (A) exists if the following optimization problem is feasible:*

$$x_c \in X \subset \mathbb{R}^{n_x} \quad s.t. \tag{7.23}$$

$$\forall\ \mathbb{E}_i(z_r, S_r) \in \mathcal{E}_{\mathcal{R}}, \qquad \begin{bmatrix} 1 & \bullet \\ x_c - z_r & S_r^{-1} \end{bmatrix} \succ 0, \tag{7.24}$$

$$\forall\ \mathbb{H}_j(y_r, H_r) \in \mathcal{P}_{\mathcal{R}}, \qquad H_r^\top (x_c - y_r) < \mathbf{1}, \tag{7.25}$$

$$\forall\ \mathbb{H}_k(y_a, H_a) \in \mathcal{P}_{\mathcal{A}}. \qquad \|H_a^\top (x_c - y_a)\|_\infty > 1, \tag{7.26}$$

$$\exists u_c \in U \subseteq \mathbb{R}^{n_u} \qquad Ax_c + Bu_c + g = 0, \tag{7.27}$$

where $\mathcal{E}_{\mathcal{R}}$ and $\mathcal{P}_{\mathcal{R}}$ are respectively the set of ellipsoids and polytopes associated with propositions in $\mathcal{R}$ while $\mathcal{P}_{\mathcal{A}}$ is the set of polytopic sets associated with propositions in $\mathcal{A}$.

Proof. Applying the Schur Complement Lemma [42, p. 7], (7.24) becomes exactly the definition of an ellipsoid $\mathbb{E}(z_r, S_r)$. The condition (7.26) ensures that $\mathcal{A} \cap \ell(x_c) = \emptyset$. Finally, (7.27) enforces that x_c is a stationary point for the system under a constant input u_c. This last condition can be handled directly by semidefinite programs whenever U is also a polytope, i.e., $U = \mathbb{H}(y_U, H_U)$. □

To find a safe set $\mathbb{S}$ as required in (B), we shall search for the largest ellipsoid $\mathbb{E}(x_c, P_{\mathbb{S}})$ centered at x_c and shaped through $P_{\mathbb{S}} \in \mathbb{R}^{n_x \times n_x}$.

Lemma 7.10. *The ellipsoid $\mathbb{S} = \mathbb{E}(x_c, P_{\mathbb{S}})$ satisfies (B) if the following semidefinite program is feasible:*

$$\min_{P_{\mathbb{S}}, \beta_1, \beta_2, \ldots} \operatorname{tr}(P_{\mathbb{S}}) \qquad s.t. \tag{7.28}$$

$$\forall\ \mathbb{E}_i(z_a, P_a) \in \mathcal{E}_{\mathcal{A}}, \begin{bmatrix} P_{\mathbb{S}} + \beta_i P_a & -P_{\mathbb{S}} x_c - \beta_i P_a z_a \\ \bullet & \delta_i \end{bmatrix} \succ 0, \tag{7.29}$$

$$\forall\ \mathbb{H}_j(y_a, H_a) \in \mathcal{P}_{\mathcal{A}},\ \exists h \in \operatorname{cols}(H_a) \quad \alpha(h) P_{\mathbb{S}} \succ h h^\top, \tag{7.30}$$

where $\delta_i = x_c^\top P_{\mathbb{S}} x_c + \beta_i z_a^\top P_a z_a - 1 - \beta_i$ and $\alpha(h) = (1 + h^\top (y_a - x_c))^2$ and $\operatorname{cols}(H_a)$ denotes the set of column vectors of H_a.

Proof. Note that (7.29) is an application of the S-procedure [42, p. 23], ensuring that $x \notin \mathbb{E}(z_a, P_a)$ for all x such that $x \in \mathbb{E}(x_c, P_{\mathbb{S}})$. On the other hand, (7.30) ensures that all polytopes in $\mathcal{P}_{\mathcal{A}}$ have at least one hyperplane on their boundaries that separates them from the safe set $\mathbb{S}$. Indeed, we can prove the following statement:
For given polytope $\mathbb{H}(y, H)$ and ellipsoid $\mathbb{E}(z, S)$, if there is $h \in \operatorname{cols}(H)$ such that $(1 + h^\top (y - z))^2 S \succ h h^\top$, we have $\mathbb{H}(y, H) \cap \mathbb{E}(z, S) = \emptyset$.
Indeed, since $\mathbb{H}(y, H)$ and $\mathbb{E}(z, S)$ are convex sets, the intersection $\mathbb{H}(y, H) \cap \mathbb{E}(z, S)$ is empty if there exists one column $h \in \mathbb{R}^{n_x}$ of H such that

$$h^\top (x - y) > 1, \quad \forall\ x \in \mathbb{E}(z, S). \tag{7.31}$$

This inequality defines a separating hyperplane between $\mathbb{E}(z,S)$ and $\mathbb{H}(y,H)$, since $h^\top(x-y) \leq 1$ for all $x \in \mathbb{H}(y,H)$, by definition. Since $z \in \mathbb{E}(z,S)$ we have $h^\top(z-y) > 1$, and we can rewrite (7.31) as $(1+h^\top(y-z))^{-1}h^\top(x-z) < 1$, for all $x \in \mathbb{E}(z,S)$. Also, since $z \in \mathbb{R}^{n_x}$ is the center of $\mathbb{E}(z,S)$, this ellipsoid is contained also in the hyperplane defined by $(1+h^\top(y-z))^{-1}h^\top(x-z) > -1$, and thus we have $|(1+h^\top(y-z))^{-1}h^\top(x-z)| < 1$, for all $x \in \mathbb{E}(z,S)$. Thus (7.31) is equivalent to

$$(x-z)^\top(1+h^\top(y-z))^{-2}hh^\top(x-z) < 1$$

for all $x \in \mathbb{E}(z,S)$. This, by definition, holds if and only if $(1+h^\top(y-z))^2 S \succ hh^\top$, concluding the proof. □

Finally, having the safe set $\mathbb{S} = \mathbb{E}(x_c, P_\mathbb{S})$ fully determined, we can proceed with constructing the CLF and extracting a low-level controller from them, as required by **(C)**. We summarize our sufficient conditions in the following statement.

Lemma 7.11. *Suppose that the following semidefinite program, for a given decay rate $\delta > 0$, is feasible:*

$$\max_{Z,Y,\beta_1,\beta_2,\ldots} \operatorname{tr}(Z) \qquad s.t. \tag{7.32}$$

$$Z \prec P_\mathbb{S}^{-1} \tag{7.33}$$

$$AZ + ZA^\top + BY + Y^\top B^\top \prec -2\delta Z \tag{7.34}$$

$$\forall\ h_U \in \operatorname{cols}(H_U) \quad \begin{bmatrix} Z & Y^\top h_U \\ \bullet & (1+(y_U-u_0)^\top h_U)^2 \end{bmatrix} \succ 0. \tag{7.35}$$

Then, defining $P = Z^{-1}$ and $K = YP$, for the CLF defined by $w(x) := (x-x_c)^\top P(x-x_c)$ and the map $u(x) := K(x-x_c)+u_0$ it holds that

(i) $u(x) \in U$ for all $x \in X_w$,

(ii) $\langle \nabla w(x), f(x,u(x))\rangle \leq -\delta w(x)$, for all $x \in X_w$.

In particular, the function w satisfies conditions in (C).

Proof. First, (7.33) ensures safety as, inverting both sides of the inequality implies that $X_w(1) = \mathbb{E}(x_c, P) \subset \mathbb{S}$. Then (7.34) ensures the descent condition (7.4). Condition (7.35) implies that $u(x) \in U = \mathbb{H}(h_U, H_U)$ for all $x \in X_w(1)$. To show that, consider a $h_U \in \operatorname{cols}(H_U)$ and multiplying the first line and column of the matrix in (7.35) by P and apply the Schur Complement Lemma. The result is the equivalent matrix inequality $(1+h_U^\top(y_U-u_c))^2 P \succ K^\top h_U h_U^\top K$. Multiplying it to the right by $(x-x_c)$ and to the left by $(x-x_c)^\top$ while using the assumption that $x \in X_w(1) = \mathbb{E}(x_c,P)$ yields $(1+h_U^\top(y_U-u_c))^2 \succ (x-x_c)^\top K^\top h_U h_U^\top K(x-x_c)$, which can also be rewritten as $|h_U^\top(K(x-x_c)-y_U+u_c)| < 1$. By definition, this inequality being fulfilled for all $h_U \in \operatorname{cols}(H_U)$ is equivalent to $u(x) \in \mathbb{H}(y_U, H_U)$. □

Putting Lemmas 7.9 to 7.11 together, it can be seen that the map $u(x)$ constructed in Lemma 7.11 is a low-level controller satisfying Lemma 7.1, and hence also Proposition 7.2.

After providing all details on the synthesis of a hybrid controller solving Problem 7.1, we now discuss two additional issues in the correctness of this controller, which are not captured by Proposition 7.3.

7.6.2 Existence of Solutions

In our statement of Problem 7.1 and in the control technique formalized and summarized in Theorem 7.1 we state that *any* (trace of) trajectory of the closed loop system (7.17) satisfies the considered LTL specification. However, we did not provide a well-posedness result establishing existence of solutions for (7.17), for any initial condition and any external logical perturbation. Indeed, it is known that such systems with state-dependent piecewise-defined control input may exhibit pathological behaviors, such as chattering and sliding modes [74, 110, 111].

In what follows, we thus prove the existence of solutions, in the case studied in Section 7.6.1.

Proposition 7.4. *Consider a control system $\mathcal{S} = (X, U, f)$ with labelling function ℓ, an LTL specification Φ over the predicates $\mathtt{AP}_S \cup \mathtt{AP}_O$, the final game $\mathcal{G}^F$ and a winning strategy $\pi^F : V_0^F \to V_1^F$. Suppose that Assumptions 7.1 to 7.3 hold, and that the set of required CLFs $\mathfrak{W}$ is build following the procedure introduced in Section 7.6.1. For every $x \in X_{win}$, there exists a trajectory $\xi_{x,\mathtt{hc},\Upsilon} : \mathbb{R}_+ \to X$ to (7.17) starting at x, in the sense of Definition 7.3.*

Proof. First, we recall that by Assumptions 7.2 and 7.3 and by construction, any state proposition $\mathtt{AP}_S^+$ is associated to a compact (ellipsoidal or polyhedral) subset of X. The closed loop (7.17), under Assumption 7.1 can be compactly rewritten as

$$\dot{x} = G(t, x) = Ax + B\,\mathtt{hc}(x, \nu(t)\,) + g,$$

with $\mathtt{hc}(x, \{\mathcal{C}_w\}) = K_w(x - x_{cw}) + u_{0w}$, for all $x \in \mathbb{R}^n$ and all $\mathcal{C}_w \in \mathtt{AP}_C$, for some K_w, x_{cw} and u_{0w} of appropriate dimensions, recall Lemma 7.11. Thus, the time-varying vector field $G : \mathbb{R}_+ \times X \to \mathbb{R}^{n_x}$ is discontinuous in t, and recalling Definition 7.12, the discontinuity points are contained in the sequence of discontinuity points of $\ell^+(\xi_{x,\mathtt{hc},\Upsilon}(\cdot)) \cup \Upsilon(\cdot)$. We have to show that this sequence has no accumulation point, thus ruling out the so-called *Zeno phenomenon* [110]. Since $\Upsilon \in \mathfrak{D}$ by assumption is piece wise constant, we have to check the behavior of discontinuities of $\ell^+(\xi_{x,\mathtt{hc},\Upsilon}(\cdot))$, given a fixed context $\kappa \subseteq \mathtt{AP}_O$. By construction, these discontinuities can occur only if $\xi_{x,\mathtt{hc},\Upsilon}(\cdot)$ lies at the boundaries of the regions of attraction of the CLFs $w \in \mathfrak{W}$, with w associated to a $\mathtt{cRWA}$ with context κ, i.e. the CLFs that can be activated at that instant of time. For the boundaries of these region of attractions, the vector field G satisfies a tranversability condition

$$n(x)^\top G(t, x) < 0,$$

where $n(x)$ is the *normal vector* to the ellipsoid $\mathcal{X}_w$ in x, i.e. the vector field is "pointing inward" the set $\mathcal{X}_w$. This follows by (ii) in Lemma 7.11. This fact, also called *patchy vector field* property is a sufficient condition to ensure existence of solutions (in the sense of Definition 7.3), as proven by Ancona and Bressan [17, Proposition 3.1], to which we refer for the details. The completeness of solutions, i.e. the fact that any solution is well-defined on the whole positive real line $\mathbb{R}_+$, follows by the fact that, as proven in Theorem 7.1, by Definition 7.12, a winning play ρ always stays in Win^F. This implies, $\xi_{x,\mathtt{hc},\Upsilon}(t)$ also belongs to X_{win} for all $t \in \mathbb{R}_+$, concluding the proof. □

For a more detailed discussion regarding (properties of) solutions of discontinuous differential equations and hybrid systems, we refer to related papers [74, 110, 111].

7.6.3 Preventing Instability

As said, since the external environment can change at any instant of time, the closed loop system (7.17) exhibits *hybrid* behavior. This may lead to undesired phenomena on infinite horizons, as we highlight in the following simple example.

Example 7.7. Consider a control system of the form $\mathcal{S} := (\mathbb{R}^{n_x}, U, f)$, and two compact target sets $\mathcal{T}_1, \mathcal{T}_2 \subset \mathbb{R}^{n_x}$ such that $\mathcal{T}_1 \cap \mathcal{T}_2 = \emptyset$, and consider $\mathtt{AP}_S = \{\mathcal{T}_1, \mathcal{T}_2\}$. We consider the following desired *mode-target game specification* [26]:

$$\varphi := (\Diamond\Box\mathcal{M}_1 \implies \Diamond\Box\mathcal{T}_1) \wedge (\Diamond\Box\mathcal{M}_2 \implies \Diamond\Box\mathcal{T}_2) \tag{7.36}$$

where $\mathcal{M}_1, \mathcal{M}_2 \in \mathtt{AP}_O$ are the input atomic propositions representing the *modes* activated by the external environment. Suppose we have *global* CLFs $w_1, w_2 : \mathbb{R}^{n_x} \to \mathbb{R}$ with respect to the target $\mathcal{T}_1, \mathcal{T}_2$, in the sense of Definition 7.2, and consider low-level controller $u_i : \mathbb{R}^{n_x} \to \mathbb{R}^{n_u}$ satisfying (7.5) globally in $\mathbb{R}^{n_x} \setminus X_w(c)$, for any $i \in \{1, 2\}$. This provides a winning strategy for the game arising from (7.36): we activate the low-level controller u_i when the mode $\mathcal{M}_i$ is active. Now consider the disturbance function $\Upsilon : \mathbb{R}_+ \to \mathtt{AP}_O$ modeling the environment behavior. Then the resulting hybrid closed-loop system can be written as

$$\dot{x}(t) = g(x(t), \Upsilon(t)) \tag{7.37}$$

where $g(x, \mathcal{M}_i) := f(x, u_i(x))$ for $i \in \{1, 2\}$. Systems of the form (7.37) are known as *switched systems*, and have been intensively studied in recent years (see standard books [141, 110] for an overview). It is well-known that, even if the targets $\mathcal{T}_1$, $\mathcal{T}_2$ are asymptotically stable for the corresponding subsystems, the external disturbance $\Upsilon : \mathbb{R}_+ \to \mathtt{AP}_O$ can produce unbounded solutions for some initial condition $x \in \mathbb{R}^{n_x}$, which is undesired in many contexts [141, Chapter 1]. ⌟

There are many possible approaches to overcome the instability problem discussed in Example 7.7. Here, we informally highlight two of them.

First, consider a control system $\mathcal{S} = (X, U, f)$ and an LTL specification Φ over $\mathtt{AP}_S \cup \mathtt{AP}_O$. Suppose that the problem is global i.e., $X = \mathbb{R}^{n_x}$. Consider a large enough compact set $\mathcal{C} \subset \mathbb{R}^{n_x}$ such that $\mathcal{X} \subset \text{int}(\mathcal{C})$ for all $\mathcal{X} \in \mathtt{AP}_S$. Consider its boundary $\partial\mathcal{C}$,

add $\partial\mathcal{C} \in \mathtt{AP}_S$ (intuitively, a large enough "wall"), and consider a "new" specification Φ' defined by $\Phi' = \Phi \wedge \square\neg\partial\mathcal{C}$. Thus, paying the price of considering a more "convoluted" specification, we force, on the logical level, the trajectories of $\mathcal{S}$ to stay in the compact set $\mathcal{C}$.

Second, suppose that the environment, while being unpredictable, does satisfy some assumptions on the frequency of its decisions. More formally, suppose there exists a *dwell-time* $\tau > 0$, such that, if $t \in \mathbb{R}_+$ is a discontinuity point of the disturbance function Υ (i.e. an instant at which the external environment changes), we suppose that $\Upsilon(s) = \Upsilon(t)$, $\forall s \in [t, t+\tau)$. It is well-known that, if all the subsystems are asymptotically stable, a large enough dwell-time will ensure boundedness of solution of the switched system (7.37). The technical details are not reported here, we refer to standard books [141, Section 3.2].

While the above-mentioned approaches can provide a simple stability guarantee to the hybrid system arising from our design method, we point out that the formal study of stability/instability phenomena induced by LTL-based control is a largely open future research direction.

7.7 Experimental Results

In this section, we demonstrate the proposed techniques on an example. We consider the mode-target based example introduced in Section 7.1 in a 2-D space. The state space for the example is constrained to the box $[0, 10] \times [0, 10]$, and the three target regions $\mathcal{T}_1$, $\mathcal{T}_2$, and $\mathcal{T}_3$ are ellipsoidal balls of radius 0.2 located at co-ordinates $(3, 4)$, $(3, 6)$, and $(5, 5)$, respectively. The sliding door is a vertical line from $(4, 0)$ to $(4, 10)$. The considered dynamical model for the motion of the robot is of the form introduced in Assumption 7.1, with a 2-dimensional input space.

We used our proposed techniques to solve Problem 7.1 for this example. All computations were done on a MacBook Pro 2.5GHz with 16GB RAM. We started by constructing the initial game $\mathcal{G}^I$ from specification Φ, as given in Example 7.1. The initial game $\mathcal{G}^I$ has 51 vertices and 182 edges, which was constructed in 0.042 seconds. Next, we computed a strategy template for the initial game, and then, we translated this strategy template into several reach-while-avoid problems which took 0.007 seconds. Next, we constructed the plant model G^C with 159 vertices and 1704 edges in 6.13 seconds. Next, we constructed the final augmented game $\mathcal{G}^F$ with 826 vertices and 17604 edges in 0.652 seconds. Finally, we solved the final game to compute a winning strategy in 112.495 seconds which is used as a hybrid controller in the state space. In total, our algorithm took 120 seconds to solve Problem 7.1 for this example.

Furthermore, we also conducted a simulation[5] of this example that uses the hybrid controller computed by our algorithm. A screenshot from the simulation video at 16.30s is shown in Figure 7.8. The left part of the figures describes the continuous state-space, where we have three targets, i.e., $\mathcal{T}_1$ as an red colored dot (blurred), $\mathcal{T}_2$ as a green colored dot (blurred), and $\mathcal{T}_3$ as a blue colored dot, the robot as a black dot in motion, and

[5]Link: https://cloud.mpi-sws.org/index.php/s/Yrf2dDzspTkYm88

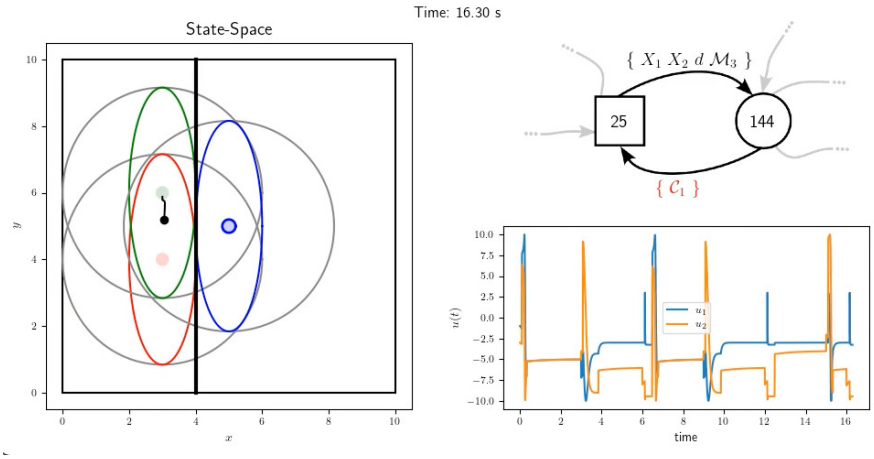


Figure 7.8: A screenshot from the simulation video

two basins of attraction per each target represented by the ellipsoids around the target. The smaller ellipsoids, i.e., green, red, blue colored ones around $\mathcal{T}_2$, $\mathcal{T}_1$, $\mathcal{T}_3$, respectively, are basins of attractions for the corresponding targets when the door is closed whereas the bigger gray ones are basins of attractions for the corresponding targets when the door is open. Moreover, this left part also describes the current state of the system. As we can see, the highlighted blue-colored target $\mathcal{T}_3$ indicates that currently mode $\mathcal{M}_3$ is active, the thick black line in the middle indicates that the door is closed, and the movement of the black dot from location of $\mathcal{T}_2$ towards $\mathcal{T}_1$ indicates that the robot is currently moving from target $\mathcal{T}_2$ to $\mathcal{T}_1$. Furthermore, the upper-right part of the figure describes the current state of the play in the final augmented game. Currently, the play in the game is looping between vertex 25 and vertex 144. The label of the edge from environment player's vertex (i.e., vertex 25) indicates that the robot is currently inside the intersection of the basins of attraction $\mathcal{X}_1$ and $\mathcal{X}_2$, and currently the door is closed and mode $\mathcal{M}_3$ is active. Furthermore, the label of the edge from controller player's vertex (i.e., vertex 144) indicates that currently low-level controller associated with $\mathcal{C}_1$ is being applied persistently. Intuitively, as mode $\mathcal{M}_3$ is active, the robot needs to reach target $\mathcal{T}_3$, and since the door is closed, the robot first need to visit target $\mathcal{T}_1$ in order to open the door. Specifically, in the video, the trajectory from 16.00s to 17.00s where the mode $\mathcal{M}_3$ remains consistently active can be described as follows: initially, at 16.00s, the robot was positioned at target $\mathcal{T}_2$ with the door closed. Subsequently, the robot moves towards target $\mathcal{T}_1$, as depicted in the screenshot shown in Figure 7.8. At 16.60s, the robot reaches $\mathcal{T}_1$, resulting in the door opening. Following that, the robot proceeds towards target $\mathcal{T}_3$ and successfully arrives at the target by 17.00s.

Returning to Figure 7.8, the lower-right part of the figure presents the time-responses of the two components of the control input, namely u_1 and u_2, which emerge from the hybrid controller defined in Section 7.4.5.

7.8 Related Work

Existing approaches tackling the outlined integrated controller synthesis problem, can roughly be divided into three different research lines. *First*, discretization-based abstraction techniques can be used to incorporate low-level dynamics into the high-level logical games (see e.g., standard literature [210, 184] for an overview and tool papers [188, 51, 117, 125, 140] for tool support). These approaches are able to handle the full problem class we tackle, but are known to suffer heavily from the curse of dimensionality and from conservatism introduced by the abstraction. *Second*, both the specification and the dynamics of the system can be mapped into a large optimization problem that searches for an optimal control law ensuring that both the logical specification and the dynamical constrains are satisfied (see e.g. survey paper by Belta and Sadra [30]). These methods, however, scale poorly with the number of logical constrains and cannot handle external environment inputs. *Third*, a constrained system can be generated, which searches for certificates on the lower level dynamical system to enforce a temporal specification (see e.g. the book by Sanfelice and Ricardo [193, Chapter 12] for an overview). This approach is usually restricted to particular classes of logical specifications and non-linear dynamics.

Within this work, we mainly follow the third approach utilizing certificates, in particular control Lyapunov functions, to realize reach-while-avoid objectives. What distinguishes our work from existing ones [191, 120, 116, 221] is the presence of logical inputs operated by the external environment. In the absence of these, the resulting synthesis problem reduces to a *temporal logic planning problem*, which does not require a reactive strategy on the higher layer, i.e., a single plan can be computed and executed in an *open-loop fashion*. Our approach produces *closed-loop* controllers in both layers instead.

While recent methods combining certificates with high-granularity abstractions [170] also produce closed-loop solutions, there, environment inputs can only be handled at transition points between abstract states. In our example, the robot would need to complete one motion (reaching a particular target) before it can receive a new objective, leading to an unsatisfying closed-loop behavior.

In addition, our new game solving formalism is also related to other work in the reactive synthesis community. While strategy templates have been very recently introduced [14, 11] forming the basis of the results presented in Chapter 3, progress group annotations appeared previously [209] for a restricted class of temporal specifications and only induced by uncontrolled dynamics. Further, the work by Rayna and Rupak [80] also tackles the problem of reactive control for dynamical systems via parity games, but only presents sufficient conditions for the existence of certificates and controllers, while our method is fully constructive.

Chapter 8

Universal Safety Controller with Learned Prophecies

While Chapter 7 focuses on the synthesis of logical controllers for continuous dynamical systems, where the plant models are constructed from the continuous dynamics, this chapter focuses on the controller synthesis problem at discrete level, where the plant model is already given. Specifically, we study the problem of synthesizing logical controllers from a specification and a plant model as given in Problem 2.1. As discussed previously, the standard approach to solving such problems involves constructing the product of the plant model and the logical game derived from the specification, then solving the resulting game to obtain a correct controller. However, in many practical scenarios, the plant model may be very large or partially unknown, causing traditional synthesis techniques that require complete state-space exploration to scale poorly or fail entirely. This chapter addresses this challenge by leveraging assumptions about plant behavior to synthesize controllers that generalize across diverse plants.

Our approach builds on the notion of *universal controllers* [96], which can be synthesized without explicit knowledge of the *given* plant. In such controllers, each action is paired with an assumption, called a *prophecy*, that characterizes the possible future behaviors of a plant. These prophecies enable the controller to adapt its behavior based on the actual plant with which it is composed at runtime. Since prophecies describe branching-time properties of the plant (e.g., the existence of certain paths contingent on the controller's actions that satisfy essential properties), existing work [96] represents them using tree automata [87]. However, tree automata are often complex to compute and computationally expensive to verify against plants during synthesis. To address this limitation, we propose a *learning-based* approach that approximates prophecies using computation tree logic (CTL) formulas learned from a small set of sample plants. As CTL formula are easier to understand and there exist efficient algorithms for their verification against plants [70], these approximations substantially reduce the computational complexity of universal controller synthesis while preserving generalizability across a broad range of plants.

To this end, we first revisit the notion of universal controllers and their synthesis

in Section 8.1. We then introduce the initial step of our learning-based framework in Section 8.2, defining under- and over-approximations of prophecies and their refinement from sample plants. Subsequently, we present our synthesis framework using standard CTL learning algorithms in Section 8.3. Finally, we demonstrate the generalizability and scalability of our approach through experimental results obtained with our prototype tool UCLearn in Section 8.4, followed by a discussion of related work in Section 8.5.

8.1 Universal Controller Synthesis

In this section, we revisit the notion of universal controllers and their synthesis from existing work [96]. In contrast to traditional controller synthesis, which requires explicit state-space exploration of a given plant, we consider a synthesis framework that abstracts away from explicit plants by reasoning over sets of possible plants, called *prophecies*. A prophecy captures assumptions about future plant behavior. Rather than solving the synthesis problem for each plant individually, we synthesize a controller that conditions its behavior on verified prophecies for the given plant. This leads to the notion of a prophecy-annotated controller[1] (*prophecy controller* for short), which conditions controller behavior on prophecies.

Definition 8.1. A *prophecy* $\theta \subseteq \texttt{Plants}$[2] is a set of plants. A *prophecy-annotated controller* $\mathcal{U}$ over an alphabet Σ and a set of prophecies $\mathbb{F}$ is a tuple (S, s_0, τ, κ), where S is a finite set of states, $s_0 \in S$ is the initial state, $\tau : S \times \Sigma \to S$ is the transition function, and $\kappa : S \times 2^{O_\mathtt{c}} \to \mathbb{F}$ is the prophecy annotation.

Note that, similar to ω-automata (as defined in Section 2.2.3), the transition function τ is defined over all possible alphabets; hence, $\mathcal{U} \times \mathcal{M}$ is well-defined for any implementation $\mathcal{M}$ as per Section 2.4. Furthermore, each controller output from a state is associated with a prophecy that must hold for that output to be valid. For an explicit plant, we compute a consistent controller by verifying the prophecies at each state.

Definition 8.2. Given a prophecy controller $\mathcal{U} = (S, s_0, \tau, \kappa)$ and a plant $\mathcal{M}_\mathtt{p}$, a controller $\mathcal{M}_\mathtt{c} = (S^\mathtt{c}, s_0^\mathtt{c}, \tau^\mathtt{c}, o^\mathtt{c})$ is said to be *consistent* with $\mathcal{U}$ w.r.t. $\mathcal{M}_\mathtt{p}$, denoted by $\mathcal{M}_\mathtt{c} \models \mathcal{U} \parallel \mathcal{M}_\mathtt{p}$, if for all $(s, s^\mathtt{p}, s^\mathtt{c}) \in \texttt{reachable}(\mathcal{U} \times (\mathcal{M}_\mathtt{c} \parallel \mathcal{M}_\mathtt{p}))$, it holds that $\mathcal{M}_\mathtt{p}(s^\mathtt{p}) \in \kappa(s, o^\mathtt{c}(s^\mathtt{c}))$.

We are interested in prophecy controllers that are correct when combined with a consistent controller for a plant.

Definition 8.3. Given a specification φ, a prophecy controller $\mathcal{U}$ is *correct* for a plant $\mathcal{M}_\mathtt{p}$, if it holds that $\mathcal{M}_\mathtt{c} \models \mathcal{U} \parallel \mathcal{M}_\mathtt{p}$ implies $\mathcal{M}_\mathtt{p} \parallel \mathcal{M}_\mathtt{c} \models \varphi$.

[1] Note that Finkbeiner et al. [96] refers to prophecy-annotated controllers as *universal controllers*. We call a prophecy-annotated controller *universal* if and only if it is correct and most permissive for all plants.

[2] Recall that `Plants` is the set of all plants.

It is possible that no controller $\mathcal{M}_\mathtt{c}$ exists that is consistent with the prophecy controller $\mathcal{U}$ for a given plant $\mathcal{M}_\mathtt{p}$. In that case, we say $\mathcal{U}$ is *incompatible* with $\mathcal{M}_\mathtt{p}$, denoted by $\mathcal{U} \parallel \mathcal{M}_\mathtt{p} = \emptyset$. If all correct controllers for an (admissible) plant are consistent with the universal controller, we call such a universal controller *most permissive*.

Definition 8.4. Given a specification φ, a prophecy controller $\mathcal{U}$ is *most permissive* for a plant $\mathcal{M}_\mathtt{p}$, if it holds that $\mathcal{M}_\mathtt{p} \parallel \mathcal{M}_\mathtt{c} \models \varphi$ implies $\mathcal{M}_\mathtt{c} \models \mathcal{U} \parallel \mathcal{M}_\mathtt{p}$.

8.1.1 Universal Controllers

The goal of universal controller synthesis is to construct a prophecy controller that is correct and most permissive for every plant. This ensures that it produces correct controllers for all admissible plants (i.e., plants for which a correct controller exists) and is incompatible with all inadmissible plants. However, a trivial solution to this problem would be a prophecy controller that includes all plants in the prophecies from safe states and excludes all plants in the prophecies from unsafe states. This occurs because every plant (including inadmissible ones) would satisfy the prophecies as long as the composition remains in the safe region, and would become incompatible only upon reaching an unsafe state. To avoid such trivial solutions, we require that if a plant satisfies a prophecy from a state, it must be compatible with the controller from that state. Formally, a prophecy controller $\mathcal{U} = (S, s_0, \tau, \kappa)$ is said to be *forward-complete* if for every plant $\mathcal{M}_\mathtt{p}$, whenever $\mathcal{M}_\mathtt{p} \in \kappa(s, \alpha)$ for some $s \in S$ and $\alpha \in 2^{O_\mathtt{c}}$, it holds that $\mathcal{U}(s) \parallel \mathcal{M}_\mathtt{p} \neq \emptyset$. With this definition, we can now define a universal controller.

Definition 8.5. A *universal controller* $\mathcal{U}$ for a specification φ is a forward-complete prophecy controller that is correct and most permissive for every plant.

Finkbeiner et al. [96] present a procedure to synthesize a universal safety controller (USC), i.e., a universal controller for safety LTL specifications. The procedure is based on the safety automaton of the specification and formulates the correct safe prophecies as tree automata [112]. Furthermore, it guarantees that the USC uses the same transition structure as its safety automaton [96].

Lemma 8.1 ([96, Theorem 3])**.** *Given a safety LTL specification with an equivalent automaton $\mathcal{A} = (Q, q_0, \Sigma, \delta, \Omega)$, there exists an USC $\mathcal{U} = (Q, q_0, \delta, \kappa)$ such that for all states $q \in Q$ and propositions $\alpha \subseteq O_\mathtt{c}$,*

$$\kappa(q, \alpha) = \{\mathcal{M}_\mathtt{p} \mid \exists \mathcal{M}_\mathtt{c} \in \mathtt{Contrs}.\ \mathit{out}(\mathcal{M}_\mathtt{c}, \epsilon) = \alpha \wedge \mathit{Runs}(\mathcal{A}, q, \mathcal{M}_\mathtt{p} \parallel \mathcal{M}_\mathtt{c}) \subseteq \Omega\},$$

i.e., the set of plants for which there exists a controller outputting α from state q such that all runs of $\mathcal{A}$ from q on the composition are accepting. Furthermore, every prophecy $\kappa(q, \alpha)$ can be recognized by a tree automaton.

With this lemma, USCs can be constructed by adding the prophecy annotations to the safety automaton. Hence, from now on, we refer to USCs (or any prophecy controller) as its automaton with prophecy annotations, i.e., $\mathcal{U} = (\mathcal{A}, \kappa)$. Furthermore,

as in existing work [96], this chapter only considers safety LTL specifications for logical controller synthesis. Formalizing universal controllers for more general specifications is left for future work.

8.2 Approximations of Universal Controllers

The synthesis procedure of a USC [96] computes the exact set of correct prophecies, ensuring correctness and most permissiveness for *all* plants. In practice, however, we typically work with a restricted class of plants, and building the (often complex) USC is not practical. In this work, we consider approximations of the USC that maintain both under- and over-approximations of the prophecies. These approximations can be used to generate positive and negative examples for learning the prophecies.

8.2.1 Approximations

Our goal is to construct prophecy controllers that provide both sound under- and over-approximations of the USC's prophecies. We formalize this notion below.

Definition 8.6. Given a USC $\mathcal{U} = (\mathcal{A}, \kappa)$ for a safety LTL specification, an *approximation* is a tuple $\mathcal{W} = (\mathcal{A}, \underline{\kappa}, \overline{\kappa})$, where $\underline{\kappa}$ and $\overline{\kappa}$ are prophecy annotations such that: $\underline{\kappa}(q, \alpha) \subseteq \kappa(q, \alpha) \subseteq \overline{\kappa}(q, \alpha)$ for all states q and propositions $\alpha \subseteq O_{\mathsf{c}}$.

Here, $\underline{\kappa}$ is an under-approximation and $\overline{\kappa}$ is an over-approximation of the prophecy annotations κ of the USC $\mathcal{U}$. We refer to under-approximation and over-approximation as $\underline{\mathcal{W}} = (\mathcal{A}, \underline{\kappa})$ and $\overline{\mathcal{W}} = (\mathcal{A}, \overline{\kappa})$, respectively. Note that the above definition does not require the approximations to be correct or most permissive for all plants. Nevertheless, it is straightforward to see that the under-approximation remains correct for every plant, while the over-approximation is most permissive for every plant.

Lemma 8.2. *For an approximation $\mathcal{W} = (\mathcal{A}, \underline{\kappa}, \overline{\kappa})$, its under-approximation $\underline{\mathcal{W}} = (\mathcal{A}, \underline{\kappa})$ is correct and its over-approximation $\overline{\mathcal{W}} = (\mathcal{A}, \overline{\kappa})$ is most permissive for all plants.*

8.2.2 Refinements

While Lemma 8.2 guarantees that any under-approximation is correct and any over-approximation is most permissive, such approximations are not immediately useful in practice. For example, a trivial approximation sets $\underline{\kappa}$ to the empty set and $\overline{\kappa}$ to the set of all plants, which yields no useful insight about the actual plants. In practical scenarios, approximations become valuable when the synthesis procedure targets a specific class of plants: For every new plant for which we synthesize a solution, we can refine the approximation $\mathcal{W}$. This incrementally increases the precision, targeting an approximation that is correct and permissive for this exact class of plants. In this scenario, we assume that the general structure of the synthesis problem remains consistent, while specific aspects, such as the arrangement of obstacles, vary between plants. For example, a refinement of the approximation in our introductory example (see Figure 1.8) could be that some

Algorithm 8.1 REFINE$(\mathcal{W}, \mathcal{M}_\mathtt{p})$

Input: An approximation $\mathcal{W} = (\mathcal{A}, \underline{\kappa}, \overline{\kappa})$ with $\mathcal{A} = (Q, q_0, \Sigma, \delta, \Omega)$, and a plant $\mathcal{M}_\mathtt{p} = (S^\mathtt{p}, s_0^\mathtt{p}, \tau^\mathtt{p}, o^\mathtt{p})$.
Output: A refined approximation $\mathcal{W}' = (\mathcal{A}, \underline{\kappa}', \overline{\kappa}')$.

1: $G \leftarrow (Q \times S^\mathtt{p}, (q_0, s_0^\mathtt{p}), \delta', \Omega') \leftarrow \mathcal{A} \times \mathcal{M}_\mathtt{p}$ ▷ Compose
2: $\mathit{Win} \leftarrow$ SOLVE(G) ▷ Winning states in the game
3: **for all** $q \in Q$, $s^\mathtt{p} \in S^\mathtt{p}$, $\alpha \subseteq O_\mathtt{c}$ **do** ▷ All outputs
4: **if** $\delta'((q, s^\mathtt{p}), \alpha \cup \beta) \in \mathit{Win}$ for every $\beta \in 2^{O_\mathtt{e}}$ **then**
5: $\underline{\kappa}(q, \alpha) \leftarrow \underline{\kappa}(q, \alpha) \cup \{\mathcal{M}_\mathtt{p}(s^\mathtt{p})\}$ ▷ Add plant
6: **else**
7: $\overline{\kappa}(q, \alpha) \leftarrow \overline{\kappa}(q, \alpha) \setminus \{\mathcal{M}_\mathtt{p}(s^\mathtt{p})\}$ ▷ Remove plant
8: **return** $(\mathcal{A}, \underline{\kappa}, \overline{\kappa})$

processor becomes unavailable under some circumstances. Once this plant is observed, the approximations are refined to check exactly for this behavior. While the controller strategies remain largely consistent across different plants, the associated prophecies may vary. To address this, we apply *refinements* to update an existing approximation so that it remains correct and most permissive for the current plant. To guarantee that each refinement increases the precision of the approximation, we formalize the following necessary conditions.

Definition 8.7. Given an approximation $\mathcal{W} = (\mathcal{A}, \underline{\kappa}, \overline{\kappa})$ and a plant $\mathcal{M}_\mathtt{p}$, a *refinement* is another approximation $\mathcal{W}' = (\mathcal{A}, \underline{\kappa}', \overline{\kappa}')$ such that:

(i) $\underline{\kappa}(q, \alpha) \subseteq \underline{\kappa}'(q, \alpha)$ and $\overline{\kappa}'(q, \alpha) \subseteq \overline{\kappa}(q, \alpha)$ for all states q and propositions $\alpha \subseteq O_\mathtt{c}$,

(ii) $\underline{\mathcal{W}'}$ and $\overline{\mathcal{W}'}$ are correct and most permissive for $\mathcal{M}_\mathtt{p}$.

The definition of refinement implies that the new approximation $\mathcal{W}'$ is correct or permissive for at least one more plant. We claim that, in practice, every refinement steps captures a multitude of new plants.

8.2.3 Computing Refinements

We continue by providing algorithms for the computation of refinements. The goal of the refinement is to incorporate the information of the plant $\mathcal{M}_\mathtt{p}$ into the prophecy approximations. We do so by extracting all the sub-plants of the given plant and updating each prophecy annotation by adding or removing them based on their correctness.

The overall algorithm for refining an approximation is presented in Algorithm 8.1. The algorithm takes an approximation $\mathcal{W} = (\mathcal{A}, \underline{\kappa}, \overline{\kappa})$ and a plant $\mathcal{M}_\mathtt{p}$ as input and returns a refined approximation $\mathcal{W}' = (\mathcal{A}, \underline{\kappa}', \overline{\kappa}')$. The first step is the construction of the game G that ranges over the composition of the specification automaton $\mathcal{A}$ and the plant $\mathcal{M}_\mathtt{p}$. We solve this game to obtain the winning region and the states of the winning region Win. Next, based on the winning states, it iterates over all states $q \in$

Q and propositions $\alpha \subseteq O_{\mathtt{c}}$ to update the prophecy annotations $\underline{\kappa}'$ and $\overline{\kappa}'$. If for a sub-plant $\mathcal{M}_{\mathtt{p}}(s^{\mathtt{P}})$, i.e., the implementation of the plant at state $s^{\mathtt{P}}$, the output α is a correct choice from state q, then the sub-plant is added to the under-approximation $\underline{\kappa}'(q, \alpha)$. Otherwise, if it is not a correct choice, the sub-plant is removed from the over-approximation $\overline{\kappa}'(q, \alpha)$. This ensures that the refined approximation $\mathcal{W}'$ is more precise than the original approximation $\mathcal{W}$, and it is correct and most permissive for the plant $\mathcal{M}_{\mathtt{p}}$.

Theorem 8.1. *Given an approximation $\mathcal{W}$ and a plant $\mathcal{M}_{\mathtt{p}}$, the procedure* REFINE$(\mathcal{W}, \mathcal{M}_{\mathtt{p}})$ *returns a refinement.*

Proof. First, let us recall the properties of the USC. Let $\mathcal{U} = (\mathcal{A} = (Q, q_0, \delta, \Omega), \kappa)$ be the USC returned by Lemma 8.1. By the definition of safe prophecies, we know that $\kappa(q, \alpha) = \{\mathcal{M}_{\mathtt{p}} \mid \exists \mathcal{M}_{\mathtt{c}} \in \mathtt{Contrs}.\mathit{out}(\mathcal{M}_{\mathtt{c}}, \epsilon) = \alpha \wedge \mathit{Runs}(\mathcal{A}, q, \mathcal{M}_{\mathtt{p}} \parallel \mathcal{M}_{\mathtt{c}}) \subseteq \Omega\}$ for all states $q \in Q$ and propositions $\alpha \subseteq O_{\mathtt{c}}$. This means that $\kappa(q, \alpha)$ contains all plant for which there exists a controller $\mathcal{M}_{\mathtt{c}}$ that outputs α initially from state q and ensures that the run of the composition of the plant and the controller is safe, i.e., it stays in the safe regions as in Ω.

Now, let $\mathcal{W} = (\mathcal{A}, \underline{\kappa}, \overline{\kappa})$ and $\mathcal{W}' = (\mathcal{A}, \underline{\kappa}', \overline{\kappa}')$ be the prophecy controllers before and after the refinement, respectively. By construction, we know that the winning states Win in the game G computed in Algorithm 8.1 of Algorithm 8.1 are the states for which there exists a controller that ensures the run in the composition game is safe. Therefore, $\kappa(q, \alpha)$ contains all plant for which there exists a controller that outputs α initially from state q and ensures that the run of their composition from q moves to a winning state. This means that $\kappa(q, \alpha)$ contains all plant for which α ensures going to a winning state from state q. Hence, Algorithm 8.1 of Algorithm 8.1 ensure that $\underline{\kappa}'(q, \alpha)$ includes such plants whereas $\overline{\kappa}'(q, \alpha)$ excludes plants that do not ensure going to a winning state. Thus, by construction, $\mathcal{W}'$ is an approximation of $\mathcal{U}$.

As Algorithm 8.1 of Algorithm 8.1 only add plants to $\underline{\kappa}'$ and remove plants from $\overline{\kappa}'$, we have that $\underline{\kappa}(q, \alpha) \subseteq \underline{\kappa}'(q, \alpha)$ and $\overline{\kappa}'(q, \alpha) \subseteq \overline{\kappa}(q, \alpha)$ as required by property (i) of refinements. Furthermore, as $\underline{\kappa}'(q, \alpha)$ contains all sub-plants of $\mathcal{M}_{\mathtt{p}}$ which ensures going to a winning state from state q for the output α, it is correct for the plant $\mathcal{M}_{\mathtt{p}}$. Similarly, as $\overline{\kappa}'(q, \alpha)$ contains all sub-plants of $\mathcal{M}_{\mathtt{p}}$ which does not ensure going to a winning state from state q for the output α, it is most permissive for the plant $\mathcal{M}_{\mathtt{p}}$. Thus, property (ii) of refinements holds, and hence, $\mathcal{W}'$ is a refinement of $\mathcal{W}$. □

One can start with a trivial approximation (e.g., $\underline{\kappa} = \emptyset$ and $\overline{\kappa} = \mathtt{Plants}$) and iteratively refine it with the information from the plants encountered in practice. Due to the monotonicity of the refinement, this also ensures that, at each step, both the under- and over-approximation preserve the solutions of previous computation steps.

8.3 Synthesis with Learned Prophecies

We are now ready to present our approach for approximating USCs by learning prophecies from plants. The overview of our approach is illustrated in Figure 8.1. The general idea

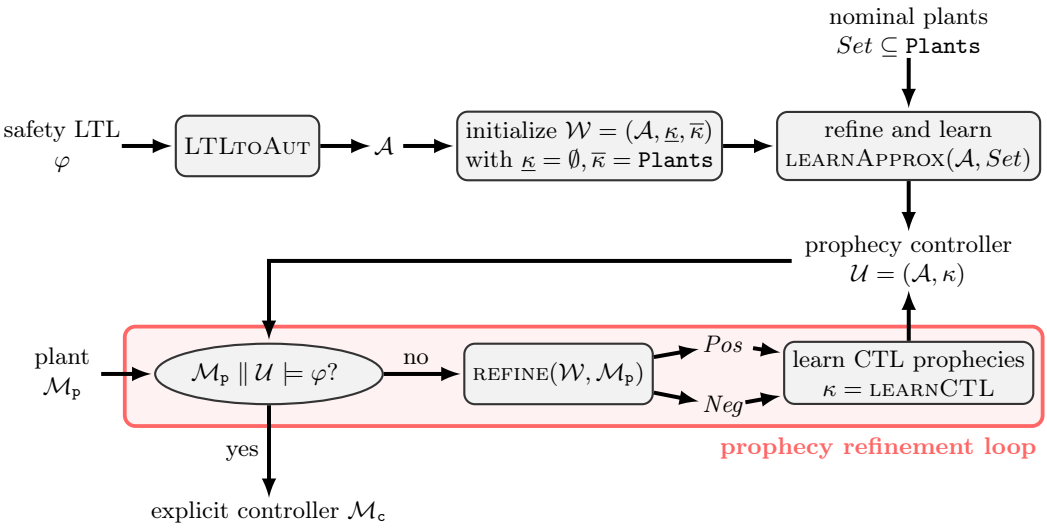


Figure 8.1: An overview of our approach for approximating USCs by learning prophecies.

is that an approximation is initially learned from the specification and a set of nominal plants, and then continuously refined based on new plants.

As shown in the upper part of Figure 8.1, we initialize the process by translating the safety LTL formula to an automaton, and setting $\underline{\kappa} = \emptyset$, i.e., the controller output is guaranteed to be correct for the empty set, and $\overline{\kappa} =$ Plants, i.e., the output is not guaranteed to be incorrect for all plants. Then, we learn an initial approximation by refining it with respect to a given set of nominal plants (shown on the top right of Figure 8.1) using the learning procedure learnApprox (detailed in Section 8.3.1). With this, we approximate USC with a prophecy controller $\mathcal{U}$.

The main refinement loop, framed in red in Figure 8.1, applies the current prophecy controller $\mathcal{U}$ to a new plant and uses the result to improve the approximation $\mathcal{W}$. We first concretize the prophecy controller to an explicit controller for the new plant by choosing the outputs on which the prophecy conditions are satisfied. We then verify if the explicit controller is correct for the plant: if yes, $\mathcal{W}$ already covers the plant and there is no need to refine, and hence, we output the explicit controller. If the controller is incorrect, we refine the approximation and obtain positive and negative samples based on the new under- and over-approximations. A CTL learning algorithm (learnCTL) then finds a formula that separates the new sets of positive/negative samples, which is used to update the prophecy annotations in the prophecy controller. This process repeats if we receive another plant.

In this section, we give details and intuition behind the main components of our approach: learning CTL formulas for prophecies in Section 8.3.1, obtaining explicit controllers from prophecy controllers in Section 8.3.2, and finally the synthesis procedure in Section 8.3.3.

Algorithm 8.2 LEARNAPPROX$(\mathcal{W}, \mathit{Set})$

Input: An approximation $\mathcal{W} = (\mathcal{A}, \underline{\kappa}, \overline{\kappa})$ with $\mathcal{A} = (Q, q_0, \Sigma, \delta, \Omega)$, and a set of plants $\mathit{Set} \subseteq \texttt{Plants}$.

Output: A prophecy controller $\mathcal{U}$.

1: **for all** $\mathcal{M}_\texttt{p} \in \mathit{Set}$ **do**
2: $\quad \mathcal{W} \leftarrow \text{REFINE}(\mathcal{W}, \mathcal{M}_\texttt{p})$ ▷ Refine the approximation
3: **for all** $q \in Q$, $\alpha \subseteq O_\texttt{c}$ **do**
4: $\quad \mathit{Pos} \leftarrow \underline{\kappa}(q, \alpha)$ ▷ Positive samples
5: $\quad \mathit{Neg} \leftarrow \texttt{Plants} \setminus \overline{\kappa}(q, \alpha)$ ▷ Negative samples
6: $\quad \phi \leftarrow \text{LEARNCTL}(\mathit{Pos}, \mathit{Neg})$ ▷ Learn a CTL formula
7: $\quad \kappa(q, \alpha) \leftarrow \mathcal{L}(\phi)$ ▷ Set prophecy annotation
8: **return** $(\mathcal{A}, \kappa)$

8.3.1 Learning CTL Formulas for Approximations

Prophecies of universal controllers, and their over- and under-approximation, are sets of plants. The procedure of Lemma 8.1, therefore, uses tree automata for prophecies that accept plants iff the considered output is correct for this plant. While tree automata represent prophecies precisely, and thereby preserve correctness and most permissiveness, they are computationally hard to compute, solve, and especially hard to understand. We choose a more lightweight formalism for categorizing trees: CTL formulas. While CTL formulas are not as expressive as tree automata [24], they are concise, human-readable, and easy to verify.

Our algorithmic solution to constructing CTL formulas for prophecies is learning. The key idea is to use the set of plants in the under-approximation as positive samples and the complement of the set of plants in the over-approximation as negative samples. From these samples, we use standard CTL learning techniques [40] to learn a CTL formula that accepts the positive samples and rejects the negative samples. This learned formula can then be used as the prophecy annotations in the resulting prophecy controller. The learning process is summarized in Algorithm 8.2.

As the algorithm returns a prophecy annotation that classifies the plants in the under-approximation as positive samples and the plants not in the over-approximation as negative samples, by Theorem 8.1 and property (ii) of refinements, the resulting prophecy controller is guaranteed to be correct and most permissive for every plant in the set. We formalize this result in the following.

Corollary 8.1. *Given an approximation $\mathcal{W} = (\mathcal{A}, \underline{\kappa}, \overline{\kappa})$ and a plant set $\mathit{Set} \subseteq \texttt{Plants}$, the procedure* LEARNAPPROX$(\mathcal{W}, \mathit{Set})$ *returns a prophecy controller $\mathcal{U} = (\mathcal{A}, \kappa)$ such that: $\underline{\kappa}(q, \alpha) \subseteq \kappa(q, \alpha) \subseteq \overline{\kappa}(q, \alpha)$ for all states q and propositions $\alpha \subseteq O_\texttt{c}$. Furthermore, $\mathcal{U}$ is correct and most permissive for every plant in Set.*

Algorithm 8.3 COMPOSE($\mathcal{U}, \mathcal{M}_\mathtt{p}$)

Input: A prophecy controller $\mathcal{U} = (\mathcal{A}, \kappa)$ with $\mathcal{A} = (Q, q_0, \Sigma, \delta, \Omega)$, and a plant $\mathcal{M}_\mathtt{p} = (S, s_0, \tau, o)$.
Output: An explicit controller $\mathcal{M}_\mathtt{c}$.

$s'_0 \leftarrow (q_0, s_0)$; $S' \leftarrow \{s'_0\}$; $\tau' \leftarrow [\,]$; $o' \leftarrow [\,]$ ▷ Initialize
$queue \leftarrow Queue(s'_0)$ ▷ Initialize the queue
while $(q, s) \leftarrow queue.pop()$ **do**
  for all $\alpha \subseteq O_\mathtt{c}$ **do**
    if $\mathcal{M}_\mathtt{p}(s) \in \kappa(q, \alpha)$ **then** ▷ If prophecy is satisfied
      $o'(q, s) \leftarrow \alpha$ ▷ Set the label
      for all $\beta \subseteq I_\mathtt{c}$ **do**
        $q'' \leftarrow \delta(q, \alpha \cup \beta)$
        $s'' \leftarrow \tau(s, (\alpha \cup \beta) \cap I_\mathtt{p})$
        $\tau'((q, s), \beta) \leftarrow (q'', s'')$
        if $(q'', s'') \notin S'$ **then**
          $S' \leftarrow S' \cup \{(q'', s'')\}$
          $queue.push((q'', s''))$
      break ▷ Found a valid output
return (S', s'_0, τ', o')

8.3.2 Composition with Prophecy Controller

The goal of Algorithm 8.2 is to obtain a prophecy controller that correctly approximates the USC restricted to the class of plants similar to the ones in the set *Set*, i.e., the prophecy controller that is correct and most permissive for this class of plants. Once the prophecy controller is learned, we can compose the learned prophecy controller with a concrete plant to obtain an explicit controller for the given plant, as long as the plant is contained in the learned prophecies. The composition algorithm is presented in Algorithm 8.3.

The algorithm implements an on-the-fly exploration of the state space of the composition of prophecy controller and plant, and only adds transitions that satisfy the prophecy annotations. For every visited state and every possible output, it checks whether the prophecy annotation is satisfied by the plant. If satisfied, the output is added as a label to the current state, and transitions to the next states are added for each possible input of the controller. Once all states have been explored, the algorithm returns the explicit controller.

Note that checking whether the prophecy is satisfied by a plant is done by model checking the corresponding CTL formula against the plant, which can be done in polynomial time [70]. This allows us to efficiently check the prophecies and compose the controller with the plant on-the-fly.

Algorithm 8.4 SYNTHESIZE$(\mathcal{W}, \mathcal{U}, \mathcal{M}_\mathtt{p})$

Input: An approximation $\mathcal{W}$, its learned prophecy controller $\mathcal{U} = (\mathcal{A}, \kappa)$ with $\mathcal{A} = (Q, q_0, \Sigma, \delta, \Omega)$, and a plant $\mathcal{M}_\mathtt{p}$.
Output: A correct controller $\mathcal{M}_\mathtt{c}$ for $\mathcal{M}_\mathtt{p}$.

1: $\mathcal{M}_\mathtt{c} \leftarrow$ COMPOSE$(\mathcal{U}, \mathcal{M}_\mathtt{p})$ ▷ Compose controller
2: **if** $\mathit{Runs}(\mathcal{A}, q_0, \mathcal{M}_\mathtt{p} \parallel \mathcal{M}_\mathtt{c}) \subseteq \Omega$ **then** ▷ Check correctness
3: **return** $\mathcal{M}_\mathtt{c}$ ▷ Return correct controller
4: **else**
5: $\mathcal{U} \leftarrow$ LEARNAPPROX$(\mathcal{W}, \{\mathcal{M}_\mathtt{p}\})$ ▷ Refine and re-learn
6: **return** COMPOSE$(\mathcal{U}, \mathcal{M}_\mathtt{p})$ ▷ Return new controller

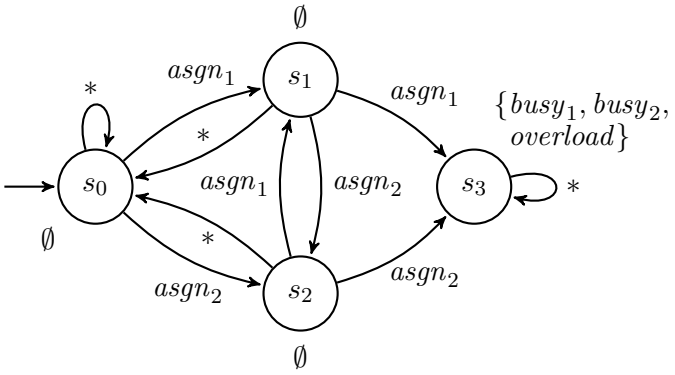


Figure 8.2: A plant for the load balancer example that only signals *busy* when processors are overloaded. We use $*$ to represent edges that are taken whenever no other edge guard is satisfied.

8.3.3 Synthesis via Learning and Refinement

Although Algorithm 8.3 can be used to synthesize an explicit controller for a given plant using a learned prophecy controller, its correctness is not immediately guaranteed: If the plant is not contained in the learned prophecies, the constructed controller might not be correct. To overcome this limitation, we add an additional model-checking step to verify whether the synthesized controller is correct for the given plant. If it is, we return the synthesized controller, otherwise, we refine the approximation using this plant and repeat the learning process to obtain a new prophecy controller that includes this plant. This overall synthesis procedure is summarized in Algorithm 8.4 and its correctness, as formalized below, follows from Theorem 8.1 and Corollary 8.1.

Corollary 8.2. *Given an approximation $\mathcal{W}$ and its learned prophecy controller $\mathcal{U}$ for a safety LTL specification φ, and a plant $\mathcal{M}_\mathtt{p}$, the procedure* SYNTHESIZE$(\mathcal{W}, \mathcal{U}, \mathcal{M}_\mathtt{p})$ *returns a controller $\mathcal{M}_\mathtt{c}$ such that $\mathcal{M}_\mathtt{p} \parallel \mathcal{M}_\mathtt{c} \vDash \varphi$.*

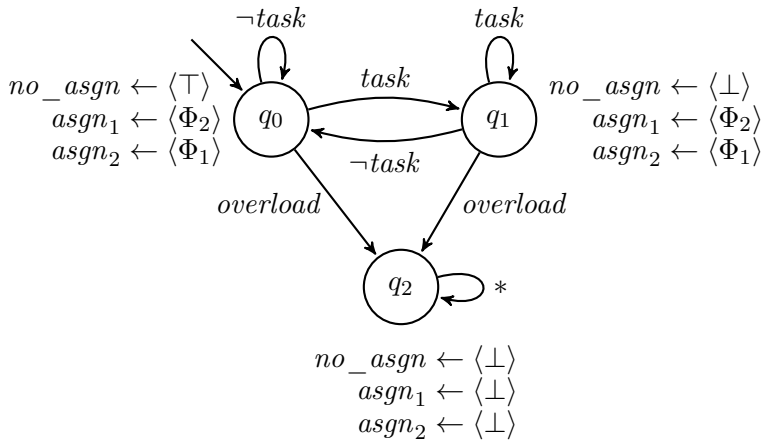


Figure 8.3: A refined prophecy controller for the load balancer example, where the CTL prophecy Φ_i is given by $\forall\bigcirc(\mathit{overload} \Rightarrow \mathit{asgn}_i)$ for $i \in \{1, 2\}$.

Example 8.1. To illustrate the overall synthesis procedure, we revisit the example in Section 1.2.5. Recall that the specification $\varphi := \square(\mathit{task} \rightarrow \bigcirc(\mathit{asgn}_1 \vee \mathit{asgn}_2)) \wedge \square\neg\mathit{overload}$ in (1.1) states that there should never be an *overload*, i.e., the controller should never assign a task to a processor that is already busy with another task. For this specification, we considered the nominal plant $\mathcal{M}_\mathtt{p}$ as in Figure 1.7. By using the procedure LEARNAPPROX$(\mathcal{W}, \{\mathcal{M}_\mathtt{p}\})$ on this plant with an initial (coarse) approximation, we obtain the prophecy controller $\mathcal{U}$ in Figure 1.8, where prophecies are represented as CTL formulas. As discussed in Section 1.2.5, this prophecy controller is not only correct for the plant $\mathcal{M}_\mathtt{p}$ but also generalizes to larger plants that have sufficient CPU availability and signal busy_i whenever cpu_i is busy.

Suppose we obtain a new plant that does not signal the *busy* status in the same way, e.g., a plant that only signals *busy* once the CPUs are overloaded. For instance, consider the plant $\mathcal{M}'_\mathtt{p}$ as in Figure 8.2. If we use the procedure COMPOSE$(\mathcal{U}, \mathcal{M}'_\mathtt{p})$ to synthesize a controller for this plant, we obtain an explicit controller that is not correct for this plant, as the prophecies are based on the assumption that the plant signals *busy* whenever a CPU is busy. Hence, we need to refine the approximation using this plant and re-learn the prophecy controller by calling LEARNAPPROX$(\mathcal{W}, \{\mathcal{M}'_\mathtt{p}\})$.

Figure 8.3 shows the refined prophecy controller after learning the new plant. Here, at states q_0 and q_1, the prophecy annotations for outputs asgn_1 and asgn_2 are represented by the CTL formulas $\Phi_i := \forall\bigcirc(\mathit{overload} \Rightarrow \mathit{asgn}_i)$, respectively, stating that if there is an overload in the next step, then the controller must have assigned the task to cpu_i. The controller can only assign a task to a CPU if it is guaranteed that the overload in the next step can only be caused by the other CPU. Therefore, the refined prophecy controller is correct for the new plant $\mathcal{M}'_\mathtt{p}$, and can be used to synthesize controllers for plants that implement similar behavior. ⌟

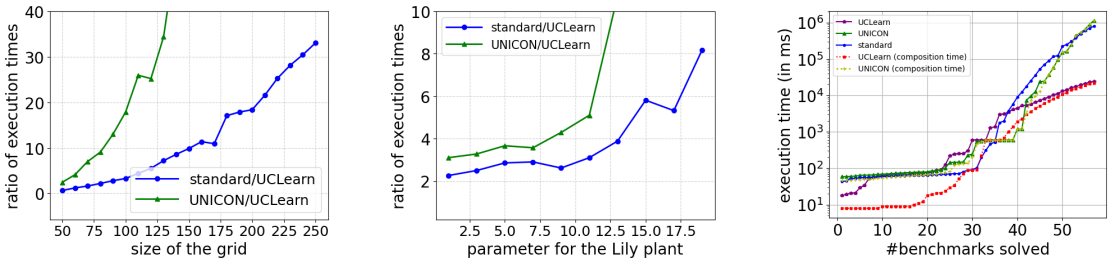


Figure 8.4: Experimental results showing scalability on the grid world (left) and lily (middle), and overall comparison (right).

8.4 Experimental Evaluation

We implemented the algorithms presented in Sections 8.2 and 8.3 in an F#-based prototype tool called UCLearn to assess the effectiveness of our method. The prototype utilizes Spot [156] for LTL and automata operations, Oink [218] for game solving, and the algorithm by Bordais et al. [40] for CTL learning. Additionally, we use Unicon [96] to compare our approach with the USC synthesis algorithm and the standard reactive synthesis algorithm.

As a baseline, we tested the running example from Example 8.1 using UCLearn, and the resulting prophecy controller with CTL prophecies are shown in Figures 1.8 and 8.3. This example illustrates that the learned prophecies are both simple and interpretable.

We further evaluated the prototype using a diverse set of benchmarks, including standard reactive synthesis benchmarks from SYNTCOMP [119] and a set of benchmarks based on a robot grids, which are detailed below. All experiments were performed on an Apple M1 Pro 8-core CPU and 16GB of RAM. The presented execution times are averages over three runs.

Scalability in Grid World. In this benchmark, the goal is to control a robot navigating an $n \times n$ grid. The plant encodes the grid structures with obstacles and the robot's position. The robot can move in four directions (up, down, left, right), and the controller's task is to ensure that the robot can move in the grid while avoiding obstacles, encoded as a simple LTL formula: $\square \neg \mathit{collision}$. We evaluate the performance of each approach on increasing grid sizes, and show the ratio of execution times between the existing approaches and UCLearn in Figure 8.4 (left). The results indicate that our approach is much more scalable and over an order of magnitude faster than the other approaches. This is not surprising as Unicon requires capturing all realizations of the plant, while UCLearn only needs to capture the realizations that are similar to a given set of plants. Furthermore, it is worth noting that the approximation was learned from a single plant with grid size 2 and consists of CTL formulas of size at most 2. This demonstrates that our approach is capable of learning a suitable approximation from a small benchmark and that the resulting formulas are human-readable.

Scalability in Lily. This benchmark, taken from SYNTCOMP [119], requires the controller to grant or cancel a request from a user within 3 time steps and ensure that no two requests are granted consecutively. This is encoded as the LTL formula: $\square(\mathit{request} \rightarrow \bigcirc(\mathit{grant} \vee \mathit{cancel} \vee \bigcirc(\mathit{grant} \vee \mathit{cancel} \vee \bigcirc(\mathit{grant} \vee \mathit{cancel})))) \wedge \square(\mathit{grant} \rightarrow \bigcirc \neg \mathit{grant})$. We evaluate all approaches on this benchmark with plants encoding increasing complexity of requests, and the results are shown in Figure 8.4 (middle). As with the grid world benchmark, the approximation is learned from a single plant with parameter 2, and the learning approach is up to 8 times faster than the standard approach.

Adaptability in SYNTCOMP. In addition to demonstrating the scalability of our approach, we also evaluate its adaptability to different plants. We use the above benchmarks and a set of safety benchmarks from SYNTCOMP, each consisting of an assumption and a guarantee. For such benchmarks, we obtain a plant satisfying the assumption and learn an approximation for the guarantee. This learned approximation is then used to obtain a controller for a plant of similar size satisfying the assumption. The results of this evaluation are shown in Figure 8.4 (right). The plot also compares the time required to compose the (already learned/constructed) prophecy controller with the plant, labeled as the composition time, for both UCLearn and unicon. These results indicate that UCLearn adapts to changes in the plant model much faster than both the standard approach and unicon. This is due to the fact that the CTL formulas learned from the plant are concise and small (with a size of at most 4), making them easily reusable for similar plants.

8.5 Related Work

The generalization and adaptability of the universal controller stem from its *permissiveness*—its ability to represent a set of controller strategies rather than a single one. Permissiveness has been widely studied in supervisory control [54] and reactive synthesis [32, 41, 102, 128, 14], though typically under a fixed plant model with limited adaptability. Other works consider strategies correct for sets of plants—e.g., dominant [77, 99] and admissible [33, 29] strategies, or restricted plant classes [11, 44]—but they yield a single strategy correct for all plants. In contrast, our method synthesizes a controller that adapts its strategy to the given plant model.

Temporal logic specification learning has been well studied [167, 182, 217, 180], but these works focus on inferring specifications from samples, not using them for control. Learning has been applied to LTL synthesis without plant models [133, 25], whereas we learn CTL formulas that characterize plant behavior. We also leverage prophecy variables [6, 34] to anticipate future plant actions during universal controller synthesis, following the framework introduced by Finkbeiner et al. [96].

Chapter 9

Robust Computation Tree Logic

Unlike Chapters 7 and 8, which focus on synthesizing logical controllers by constructing or utilizing assumptions about the plant model, this chapter investigates the robustness of logical controllers when these assumptions are violated, thereby enabling systems to operate reliably in more permissive environments. Specifically, we consider settings where these assumptions are explicitly encoded within the logical specifications. While the literature offers numerous approaches to robust control [36, 38, 186, 213], most focus on linear-time properties, preventing their applicability to branching-time properties that can express more complex plant behaviors (e.g., the ability to reach location A from B in one time step, as discussed in Section 1.2.6, or the CTL prophecies from Chapter 8). We overcome this limitation by introducing a novel robust semantics for branching-time temporal logics, specifically CTL and CTL*.

Our notion of robustness draws inspiration from the robust semantics for LTL introduced by Tabuada and Neider [213]. The central idea is to interpret temporal operators in a multivalued fashion, distinguishing between different levels of temporal property violations. This multivalued semantics enables the specifications to be more robust with respect to the assumptions: small violations in the assumptions lead only to small violations in the specifications. Moreover, the computational complexity of various problems remains within the same asymptotic bounds, yielding robustness at no additional cost.

The remainder of this chapter is organized as follows. We begin by reviewing the classical CTL semantics in Section 9.1 using a mapping-based formulation that facilitates our robust semantics definition. In Section 9.2, we introduce the robust extension of CTL, termed *robust CTL*, along with its multivalued semantics, expressiveness results, and complexity analysis for model-checking and synthesis problems. Next, we extend our semantics to CTL* by first revisiting the classical CTL* semantics in Section 9.3, then presenting its robust counterpart, *robust CTL**, in Section 9.4, together with expressiveness and complexity results. Finally, we discuss related work in Section 9.5.

9.1 Review of CTL Semantics

In this section, slightly deviating from the usual approach, we re-define the CTL semantics using a mapping V_{CTL} that maps a state/path and a CTL formula to a truth value in $\mathbb{B} = \{0, 1\}$. Also, some of our definitions are non-standard in order to be closer to the robust semantics introduced later. However, let us stress that the definition below is equivalent to the usual semantics of CTL as defined in Section 2.3.2, i.e., for a state s and a CTL state formula Φ, we have $V_{\mathrm{CTL}}(s, \Phi) = 1$ if and only if $s \models \Phi$ according to the usual semantics, and the same holds for paths and CTL path formulas.

Given a state s and state formulas Φ, Ψ, CTL semantics is defined as follows:

$$
\begin{aligned}
V_{\mathrm{CTL}}(s, p) &= \begin{cases} 0 & \text{if } p \notin L(s); \text{ and} \\ 1 & \text{if } p \in L(s), \end{cases} \\
V_{\mathrm{CTL}}(s, \Phi \vee \Psi) &= \max\{V_{\mathrm{CTL}}(s, \Phi), V_{\mathrm{CTL}}(s, \Psi)\}, \\
V_{\mathrm{CTL}}(s, \Phi \wedge \Psi) &= \min\{V_{\mathrm{CTL}}(s, \Phi), V_{\mathrm{CTL}}(s, \Psi)\}, \\
V_{\mathrm{CTL}}(s, \neg\Phi) &= 1 - V_{\mathrm{CTL}}(s, \Phi), \\
V_{\mathrm{CTL}}(s, \Phi \Rightarrow \Psi) &= \begin{cases} 1 & \text{if } V_{\mathrm{CTL}}(s, \Phi) \leq V_{\mathrm{CTL}}(s, \Psi); \text{and} \\ V_{\mathrm{CTL}}(s, \Psi) & \text{otherwise,} \end{cases} \\
V_{\mathrm{CTL}}(s, \exists\varphi) &= \max_{\rho \in \mathrm{paths}(s)} V_{\mathrm{CTL}}(\rho, \varphi), \\
V_{\mathrm{CTL}}(s, \forall\varphi) &= \min_{\rho \in \mathrm{paths}(s)} V_{\mathrm{CTL}}(\rho, \varphi).
\end{aligned}
$$

Similarly, for a path ρ, the CTL semantics of path formulas is defined as given below:

$$
\begin{aligned}
V_{\mathrm{CTL}}(\rho, \bigcirc \Phi) &= V_{\mathrm{CTL}}(\rho[1], \Phi), \\
V_{\mathrm{CTL}}(\rho, \Diamond \Phi) &= \max_{i \geq 0} V_{\mathrm{CTL}}(\rho[i], \Phi), \\
V_{\mathrm{CTL}}(\rho, \Box \Phi) &= \min_{i \geq 0} V_{\mathrm{CTL}}(\rho[i], \Phi), \\
V_{\mathrm{CTL}}(\rho, \Phi \,\mathbf{U}\, \Psi) &= \max_{j \geq 0} \min\{V_{\mathrm{CTL}}(\rho[j], \Psi), \min_{0 \leq i < j} V_{\mathrm{CTL}}(\rho[i], \Phi)\}, \\
V_{\mathrm{CTL}}(\rho, \Phi \,\mathbf{W}\, \Psi) &= \min_{j \geq 0} \max\{V_{\mathrm{CTL}}(\rho[j], \Phi), \max_{0 \leq i \leq j} V_{\mathrm{CTL}}(\rho[i], \Psi)\}.
\end{aligned}
$$

9.2 Robust Computation Tree Logic

In this section, we robustify CTL by generalizing the ideas underlying robust LTL [213] to CTL, obtaining the logic rCTL. We describe the syntax and semantics of rCTL and discuss the relation and differences between rCTL and other temporal logics.

As discussed in the robot example in the introduction, we want to capture the notion of robustness in CTL by ensuring that a small violation in environment assumptions leads to a small violation of system guarantees. To achieve that, we introduce a robust semantics for CTL. Following arguments given by Tabuada and Neider [213], we first

motivate the semantics of rCTL using an example. Consider the CTL path formula $\Box p$, where p is an atomic proposition. The formula can be satisfied in only one way, namely when p holds at every step, i.e., state, of the path. In contrast, the formula can be violated in several ways. Intuitively, $\Box p$ is violated in the worst manner when p fails to hold at every step. Then, we would prefer a case where p holds for finitely many steps. Even better would be the case when p holds at infinitely many steps. Finally, among all possible ways $\Box p$ can be violated, we would prefer the situation where p fails to hold for at most finitely many steps. Our robust semantics is designed to distinguish between satisfaction and these four different degrees of violation of $\Box p$. However, as convincing as this argument might be, a question persists: in which sense can we regard these five alternatives as canonical?

We answer this question by interpreting the satisfaction of $\Box p$ as a counting problem. Recall that the semantics of $\Box p$ for a path ρ is given by $V_{\mathrm{CTL}}(\rho, \Box p) = \min_{i \geq 0} V_{\mathrm{CTL}}(\rho[i], p)$. Now, observe that the truth value of the CTL formula $\Box p$ for a path ρ only depends on the number of occurrences of 0's and 1's in the infinite word $\alpha = V_{\mathrm{CTL}}(\rho[0], p) V_{\mathrm{CTL}}(\rho[1], p) \cdots \in \mathbb{B}^\omega$ but not on their order. From this perspective, $\Box p$ is violated in the worst manner when p fails to hold at every step, which corresponds to the number of occurrences of 1 in α being zero. The next degree of violation of $\Box p$ in which p holds at finitely many steps corresponds to having a finite number of 1's. Similarly, the next degree of violation corresponds to having an infinite number of 1's and an infinite number of 0's. Among all the ways in which $\Box p$ is violated, the most preferred way corresponds to having finitely many 0's. Finally, the satisfaction of $\Box p$ corresponds to having zero 0's. Note that the position where 0's and 1's occur is irrelevant for our argument. Furthermore, note that by successively applying permutations that swap position i with position $i+1$ and leave all the remaining elements of $\mathbb{N}$ unaltered, one can transform any $\alpha \in \mathbb{B}^\omega$ into words of one of the following five forms: $1^\omega, 0^k 1^\omega, (01)^\omega, 1^k 0^\omega, 0^\omega$. It is not hard to verify that the five cases of violations of $\Box p$ that we discussed above amount to the words of the five forms given above. Thus, we conclude the need for five truth values to describe five different ways of counting 0's and 1's that correspond to five different canonical forms of violations of $\Box p$.

According to our motivating example $\Box p$, the desired semantics should have one truth value corresponding to true and four truth values corresponding to the different shades of false. For notational convenience, we denote these truth values by $b = (b_1, b_2, b_3, b_4)$ with $b_i \in \mathbb{B}$. Intuitively, for the formula $\Box p$, b_1 captures whether p holds at every step, b_2 captures whether p fails to hold at most finitely many steps, b_3 captures whether p holds at infinitely many steps, and b_4 captures whether p holds at least once. Note that these cases are monotonic, i.e., $b_i = 1$ implies $b_{i+1} = 1$. Hence, we obtain the set $\mathbb{B}_4 = \{0000, 0001, 0011, 0111, 1111\}$ of truth values. The value 1111 corresponds to true, and the others correspond to different shades of false as explained above. The truth values are ordered naturally as $0000 < 0001 < 0011 < 0111 < 1111$.

It remains to explain how the semantics of Boolean connectives are defined for these truth values. The notion of a triangular-norm summarizes all the desirable properties of a many-valued conjunction (see P. Hájek [114] for details), and it is natural to model

conjunction and disjunction in $\mathbb{B}_4$ by min and max, respectively. Moreover, as in intuitionistic logic, we define the implication, denoted by $a \to b$ on the level of truth values, such that $c \leq a \to b$ if and only if $c \wedge a \leq b$ for every $c \in \mathbb{B}_4$. This leads to

$$a \to b = \begin{cases} 1111 & \text{if } a \leq b; \text{ and} \\ b & \text{otherwise.} \end{cases}$$

However, the negation, denoted by $\overline{a}$ on the level of truth values, defined by $a \to 0000$ as in intuitionistic logic, is not compatible with our interpretation that all elements in $\mathbb{B}_4 \setminus \{1111\}$ represent different shades of false and, thus, their negation should be 1111. To make this point clear, we present in Table 9.1 the intuitionistic negation in $\mathbb{B}_4$ and the desired negation compatible with the interpretation of the truth values in $\mathbb{B}_4$. What is then the algebraic structure on $\mathbb{B}_4$ that supports the desired negation,

Table 9.1: Desired negation vs. intuitionistic negation in $\mathbb{B}_4$.

Value	**Desired negation**	**Intuitionistic negation**
1111	0000	0000
0111	1111	0000
0011	1111	0000
0001	1111	0000
0000	1111	1111

dual to the intuitionistic negation? This very same problem was investigated by Priest and Graham [181], and the answer is *da Costa* algebras. Therefore, following the ideas introduced by rLTL, we use *da Costa algebras*[181] to define the negation:

$$\overline{a} = \begin{cases} 0000 & \text{if } a = 1111; \text{ and} \\ 1111 & \text{otherwise.} \end{cases}$$

In other words, "true" (1111) gets mapped to "false" (0000), while "shades of false" get mapped to "true".

It should be mentioned that working with a five-valued semantics has its price. As in intuitionistic logic, $\overline{\overline{a}}$ may not be equal to a as evidenced by taking $a = 0111$. Although it is still true that $\overline{\overline{a}} \to a$. Interestingly, we can think of double negation as quantization in the sense that true is mapped to true and all the shades of false are mapped to 0000 (false). Hence, double negation quantizes the five different truth values into two truth values (true and false) in a manner that is compatible with our interpretation of truth values.

Remark 9.1. *Although there are alternative ways to define negation that preserves its duality, i.e., $\overline{\overline{a}} = a$, our notion of negation (as in original rLTL work [213]) has been proven useful in many applications (see, e.g., Anevlavis et al. [20]).*

9.2.1 Syntax

The syntax of rCTL matches that of CTL, save for dotting temporal operators for visual distinction. Hence, formulas of rCTL are also classified into state and path formulas.

rCTL state formulas over $\mathtt{AP}$ are formed according to the grammar

$$\Phi ::= p \mid \Phi \vee \Phi \mid \Phi \wedge \Phi \mid \neg\Phi \mid \Phi \Rightarrow \Phi \mid \exists\varphi \mid \forall\varphi,$$

where $p \in \mathtt{AP}$ and φ is a path formula. rCTL path formulas are formed according to the grammar

$$\varphi ::= \odot\Phi \mid \diamondsuit\Phi \mid \boxdot\Phi \mid \Phi \,\dot{\mathbf{U}}\, \Phi \mid \Phi \,\dot{\mathbf{W}}\, \Phi.$$

Note that we include implication, conjunction, and weak until as part of the syntax, instead of derived operators. This is needed as we will see later, they can no longer be derived in rCTL. Furthermore, as illustrated before, it is instructive to include the operators eventually and always explicitly. Hence, we include them in the syntax as well.

The size of a formula is defined as the number of its syntactically distinct subformulas. Here, the set of subformulas of a state formula Φ is defined as for CTL (see Baier and Katoen [24] for details) and denoted by $\mathrm{Sub}(\Phi)$.

9.2.2 Semantics

Similar to the semantics of CTL, we define the semantics of rCTL by a mapping V, called *valuation*, that maps an rCTL formula and a state/path to an element of $\mathbb{B}_4$. For an atomic proposition $p \in \mathtt{AP}$, it is defined classically:

$$V(s,p) = \begin{cases} 0000 & \text{if } p \notin L(s); \text{ and} \\ 1111 & \text{if } p \in L(s). \end{cases}$$

Following the semantics of rLTL, we define the semantics for Boolean connectives in rCTL using da Costa algebras, as follows:

$$\begin{aligned} V(s, \Phi \vee \Psi) &= \max\{V(s,\Phi), V(s,\Psi)\}, \\ V(s, \Phi \wedge \Psi) &= \min\{V(s,\Phi), V(s,\Psi)\}, \\ V(s, \neg\Phi) &= \overline{V(s,\Phi)}, \\ V(s, \Phi \Rightarrow \Psi) &= V(s,\Phi) \to V(s,\Psi). \end{aligned}$$

For existential path quantification, we want $V(s, \exists\varphi) \geq b$ if there exists a path ρ starting in s such that $V(\rho,\varphi) \geq b$. Similarly, we want $V(s,\forall\varphi) \geq b$ if for all paths ρ starting in s it holds that $V(\rho,\varphi) \geq b$. This leads to

$$\begin{aligned} V(s, \exists\varphi) &= \max_{\rho \in \mathrm{paths}(s)} V(\rho,\varphi), \\ V(s, \forall\varphi) &= \min_{\rho \in \mathrm{paths}(s)} V(\rho,\varphi). \end{aligned}$$

For path formulas, we formalize the intuition above in the semantics of the temporal operators. For $1 \leq \ell \leq 4$, let V_ℓ denote the ℓ-th bit of the valuation V. Then, using the counting interpretation as discussed earlier, we define the semantics for $\boxdot$ by $V(\rho, \boxdot\Phi) = (b_1, b_2, b_3, b_4)$, where

$$\begin{aligned} b_1 &= \min_{i \geq 0} V_1(\rho[i], \varphi), \\ b_2 &= \max_{j \geq 0} \min_{i \geq j} V_2(\rho[i], \varphi), \\ b_3 &= \min_{j \geq 0} \max_{i \geq j} V_3(\rho[i], \varphi), \\ b_4 &= \max_{i \geq 0} V_4(\rho[i], \varphi). \end{aligned}$$

The semantics of $\diamondsuit\Phi$ mimics the classical semantics in that the truth value of $\diamondsuit\Phi$ on ρ is the maximal truth value of Φ that is assumed at any position of ρ. Analogously, the semantics for temporal operators $\bigcirc$ and $\mathbf{U}$ also mimics the classical semantics as follows:

$$\begin{aligned} V(\rho, \diamondsuit\Phi) &= \max_{i \geq 0} V(\rho[i], \Phi), \\ V(\rho, \bigcirc\Phi) &= V(\rho[1], \Phi), \\ V(\rho, \Phi\,\mathbf{U}\,\Psi) &= \max_{j \geq 0} \min\{V(\rho[j], \Psi), \min_{0 \leq i < j} V(\rho[i], \Phi)\}. \end{aligned}$$

Finally, using the counting interpretation as above, the semantics for $\mathbf{W}$ is defined by $V(\rho, \Phi\,\mathbf{W}\,\Psi) = (b_1, b_2, b_3, b_4)$, where

$$\begin{aligned} b_1 &= \min_{j \geq 0} \max\{V_1(\rho[j], \Phi), \max_{0 \leq i \leq j} V_1(\rho[i], \Psi)\}, \\ b_2 &= \max_{k \geq 0} \min_{j \geq k} \max\{V_2(\rho[j], \Phi), \max_{0 \leq i \leq j} V_2(\rho[i], \Psi)\}, \\ b_3 &= \min_{k \geq 0} \max_{j \geq k} \max\{V_3(\rho[j], \Phi), \max_{0 \leq i \leq j} V_3(\rho[i], \Psi)\}, \\ b_4 &= \max_{j \geq 0} \max\{V_4(\rho[j], \Phi), \max_{0 \leq i \leq j} V_4(\rho[i], \Psi)\}. \end{aligned}$$

Example 9.1. Having defined the rCTL semantics, let us recall the example of the specification for a robot given in Section 1.2.6: $\Phi_e \Rightarrow \Phi_s$. The environment assumption on the plant model is $\Phi_e = \forall\Box(A \Rightarrow \exists\bigcirc B)$, which states that from every reachable state, if the robot is in room A, then there exists a path where it can move to room B in one time step. The system guarantee is $\Phi_s = \forall\Box T$, which states that the robot should be able to perform task T in all reachable states at all times. The robust version of this formula is

$$\Phi := \forall\boxdot(A \Rightarrow \exists\bigcirc B) \Rightarrow \forall\boxdot T.$$

Let us demonstrate how this formula captures the robustness property as discussed in Section 1.2.6. Consider the case where Φ evaluates to 1111 in a given Kripke structure. Then the following hold:

- If the robot can always move from room A to room B in one time step, then in any path, $(A \Rightarrow \exists \bigcirc B)$ holds at every state. Hence, $\forall \boxdot (A \Rightarrow \exists \odot B)$ evaluates to 1111. Then by the semantics of $\Rightarrow$, the formula $\forall \boxdot T$ also must evaluate to 1111. That means, in any path, T also holds at every state. Therefore, from any state of a path, the robot can always perform task T. Hence, the desired behavior of the system is retained when the environment assumption holds with no violation.

- If the office workers obstruct the robot for a finite amount of time, in which case, for any path, $A \Rightarrow \exists \odot B$ holds eventually at every state. Hence, $\forall \boxdot (A \Rightarrow \exists \odot B)$ evaluates to 0111. Then, by the rCTL semantics, $\forall \boxdot T$ evaluates to 0111 or higher. Hence, in any path, T also needs to eventually hold at every state. That means, in any path, the robot would be able to perform task T after a finite amount of time.

- Similarly, if $A \Rightarrow \exists \odot B$ holds at infinitely many states (resp. some state) in every path, then T needs to hold at infinitely many states (resp. some state) in every path.

Hence, whenever the formula Φ evaluates to 1111, its semantics captures the intended robustness property by which a weakening of the assumption $\forall \boxdot (A \Rightarrow \exists \odot B)$ leads to a weakening of the guarantee $\forall \boxdot T$.

Now, a natural question arises: does the formula still provide useful information when its value is lower than 1111. It follows from the semantics of implication that Φ evaluates to $b < 1111$ only when $\forall \boxdot (A \Rightarrow \exists \odot B)$ evaluates to a higher value than b, whereas $\forall \boxdot T$ evaluates to b. So, the desired system guarantee is not satisfied. However, the value of Φ still describes which weakened guarantee follows from the environment assumption. This can be seen as another measure of robustness: despite $\forall \boxdot T$ not following from $\forall \boxdot (A \Rightarrow \exists \odot B)$, the system's behavior is not arbitrary, a value of b is still guaranteed for $\forall \boxdot T$. ⌟

9.2.3 Expressiveness of rCTL

In this section, we compare the expressiveness of rCTL with three other temporal logics: CTL, LTL, and rLTL. We show that the five truth values of rCTL make it more expressive than CTL. More precisely, there are properties that one can express in rCTL but not in CTL. However, the expressiveness of rCTL and LTL are incomparable, and the same also holds for rCTL and rLTL.

We compare the expressiveness of two classes of logics by comparing the expressiveness of their formulas. For logics $\mathcal{L}$ and $\mathcal{L}'$, we say $\mathcal{L}$ is as expressive as $\mathcal{L}'$ if for every formula in $\mathcal{L}'$ there is an equivalent formula in $\mathcal{L}$. Moreover, we say $\mathcal{L}$ is more expressive than $\mathcal{L}'$ if $\mathcal{L}$ is as expressive as $\mathcal{L}'$ but the converse is not true. Furthermore, we say $\mathcal{L}$ and $\mathcal{L}'$ have incomparable expressiveness if neither of $\mathcal{L}$ and $\mathcal{L}'$ is as expressive as the other one.

Now the question is what it means for two formulas to be equivalent. Intuitively speaking, equivalent means "express the same thing". Formally, we define the equivalence of two formulas using their satisfaction sets. For a given Kripke structure, and a state formula Φ, we define the satisfaction set $\mathrm{Sat}(\Phi, b)$ of an rCTL formula Φ and with value

$b \in \mathbb{B}_4$ to be the set of states s such that $V(s, \Phi) \geq b$. Since the satisfaction sets of an rCTL (state) formula are always associated with a truth value in $\mathbb{B}_4$, we always associate a truth value with an rCTL formula when comparing its expressiveness.

For two rCTL state formulas Φ_1, Φ_2 and two truth values $b_1, b_2 \in \mathbb{B}_4$, we say that Φ_1 with truth value b_1 is equivalent to Φ_2 with truth value b_2 if for every Kripke structure it holds that $\mathrm{Sat}(\Phi_1, b_1) = \mathrm{Sat}(\Phi_2, b_2)$. Similarly, an rCTL formula Φ_1 with truth value b_1 is equivalent to a CTL formula Φ_2 if for every Kripke structure it holds that $\mathrm{Sat}(\Phi_1, b_1) = \mathrm{Sat}_{\mathrm{CTL}}(\Phi_2)$, where $\mathrm{Sat}_{\mathrm{CTL}}(\cdot)$ denotes the satisfaction sets for CTL formulas.

For an LTL (or rLTL) formula φ (which is evaluated over paths), we define its satisfaction set to contain all states s such that ρ satisfies φ for every path $\rho \in \mathrm{paths}(s)$. Hence, an LTL or rLTL formula is equivalent to an rCTL formula, if they have the same satisfaction sets for all Kripke structures.

We begin by comparing the semantics of CTL and rCTL. First, we want to show that the CTL semantics is captured by the first bit of the rCTL semantics (recall that V_1 denotes the first bit of the rCTL valuation function). Due to the non-standard semantics of implication in robust logics, this does only work for CTL formulas without implications. This is of course not a restriction, as in classical semantics, implication can be derived from disjunction and negation.

Lemma 9.1. *For any CTL state formula Φ containing no implication, let Φ_r be the rCTL state formula obtained by dotting all temporal operators in Φ. Then for any state s, it holds that $V_{CTL}(s, \Phi) = V_1(s, \Phi_r)$. Consequently, it holds that* $\mathrm{Sat}_{CTL}(\Phi) = \mathrm{Sat}(\Phi_r, 1111)$.

Proof. Applying the definition of the rCTL semantics, we have the following:

$$
\begin{aligned}
V_1(s,p) &= \begin{cases} 0 & \text{if } p \notin L(s); \text{ and} \\ 1 & \text{if } p \in L(s), \end{cases} \\
V_1(s,\neg\Phi) &= \begin{cases} 0 & \text{if } V_1(s,\Phi) = 1; \text{ and} \\ 1 & \text{otherwise,} \end{cases} \\
V_1(s,\Phi \vee \Psi) &= \max\{V_1(s,\Phi), V_1(s,\Psi)\}, \\
V_1(s,\Phi \wedge \Psi) &= \min\{V_1(s,\Phi), V_1(s,\Psi)\}, \\
V_1(s,\exists\varphi) &= \max_{\rho\in\mathrm{paths}(s)} V_1(\rho,\varphi), \\
V_1(s,\forall\varphi) &= \min_{\rho\in\mathrm{paths}(s)} V_1(\rho,\varphi), \\
V_1(\rho,\odot\Phi) &= V_1(\rho[1],\Phi), \\
V_1(\rho,\diamondsuit\Phi) &= \max_{j\geq 0} V_1(\rho[j],\Phi) \\
V_1(\rho,\boxdot\Phi) &= \min_{j\geq 0} V_1(\rho[j],\Phi) \\
V_1(\rho,\Phi\,\mathbf{U}\,\Psi) &= \max_{j\geq 0} \min\{V_1(\rho[j],\Psi), \min_{0\leq i<j} V_1(\rho[i],\Phi)\}, \\
V_1(\rho,\Phi\,\mathbf{W}\,\Psi) &= \min_{j\geq 0} \max\{V_1(\rho[j],\Phi), \max_{0\leq i\leq j} V_1(\rho[i],\Psi)\}.
\end{aligned}
$$

Applying these equalities inductively proves that V_1 is indeed equal to the valuation V_{CTL}. □

Hence, rCTL is at least as expressive as CTL. However, the converse is not true, i.e., there exist rCTL formulas that have no equivalent CTL formula. For example, consider the rCTL formula $\Phi = \forall\boxdot p$ with truth value 0111. For a state s, we have $s \in \mathrm{Sat}(\Phi, 0111)$ if and only if for each $\rho \in \mathrm{paths}(s)$, there exists j such that $p \in L(\rho[i])$ for all $i \geq j$, which is equivalent to each path $\rho \in \mathrm{paths}(s)$ satisfying the LTL formula $\diamondsuit\Box p$. However, the formula $\diamondsuit\Box p$ can not be expressed in CTL (see Baier and Katoen [24] for details). Therefore, there is no CTL formula Ψ such that $\mathrm{Sat}(\Phi, 0111) = \mathrm{Sat}_{\mathrm{CTL}}(\Psi)$. In total, we obtain the following result.

Theorem 9.1. *rCTL is more expressive than CTL.*

It is known that the expressiveness of LTL and CTL is incomparable. For example, the CTL formula $\forall\diamondsuit\forall\Box p$ has no equivalent LTL formula, and the LTL formulas $\diamondsuit(p \wedge \bigcirc p)$ has no equivalent CTL formula (see Baier and Katoen [24] for details). The same holds for the expressiveness of LTL and rCTL. We just saw that the first bit of the rCTL semantics captures the CTL semantics (for a formula with no implication). Hence, it follows that for the rCTL formula $\forall\diamondsuit\forall\boxdot p$ (with value 1111), there is no equivalent LTL formula. Furthermore, one can see that the five-valued semantics does not help in expressing $\varphi = \diamondsuit(p \wedge \bigcirc p)$. Intuitively, a Kripke structure satisfies the formula φ if all paths contain a pair of consecutive states where p holds. Similarly to the proof of

inexpressibility of φ in CTL, it can be shown that this property is inexpressible in rCTL as well, as all path formulas are guarded with an existential or universal operator. One can express "all paths contain a state such that p holds at that state and at all (or some) of its successor" in rCTL, which is not the same as the property we want. Overall, we obtain the following result.

Theorem 9.2. *rCTL and LTL have incomparable expressiveness.*

In the work on rLTL [213], Tabuada and Neider showed that LTL and rLTL are equally expressive. Hence, a direct corollary of Theorem 9.2 is the following.

Corollary 9.1. *rCTL and rLTL have incomparable expressiveness.*

9.2.4 rCTL Model Checking

The classical CTL model checking problem asks whether the computation tree (the tree induced by all its executions) of a given system, satisfies a given CTL specification. However, in the context of rCTL, this question is more involved due to rCTL's many-valued semantics. A natural generalization is whether the computation tree satisfies a given property with at least a given value $b_0 \in \mathbb{B}_4$. As usual, we model systems by Kripke structures. So, the rCTL model checking problem is: for a given Kripke structure $\mathcal{K} = (S, I, R, L)$, an rCTL formula Φ and a truth value $b_0 \in \mathbb{B}_4$, does $V(s, \Phi) \geq b_0$ hold for all initial states $s \in I$?

Our rCTL model checking procedure is shown as pseudocode in Algorithm 9.1. It is similar to the standard CTL model checking algorithm in which it recursively computes the satisfaction sets $\mathrm{Sat}(\Psi, b)$ for each subformula $\Psi \in \mathrm{Sub}(\Phi)$ and each truth value $b \in \mathbb{B}_4$. To check whether the Kripke structure satisfies Φ, it is then enough to check whether all initial states belong to $\mathrm{Sat}(\Phi, b_0)$. Note that $\mathrm{Sat}(\Psi, 0000) = S$ since every state satisfies any rCTL formula Ψ with truth value 0000.

Algorithm 9.1 The rCTL model checking algorithm.

Input: Kripke structure $\mathcal{K}$, rCTL formula Φ, and a truth value $b_0 \in \mathbb{B}_4$

1: **for all** $\Psi \in \mathrm{Sub}(\Phi)$ in increasing size **do**
2: $\quad \mathrm{Sat}(\Psi, 0000) = S$
3: $\quad$ **for all** $b = 1111$ to 0001 **do**
4: $\quad\quad$ Compute $\mathrm{Sat}(\Psi, b)$ as characterized in Table 9.2
5: **return** $I \subseteq \mathrm{Sat}(\Phi, b_0)$

The key idea of Algorithm 9.1 is to recursively compute the satisfaction sets using a dynamic programming technique. More precisely, the satisfaction sets are computed by induction over the structure of Φ as characterized in Table 9.2. This characterization is explained in the next paragraphs and proven correct in Lemma 9.2. Since $\mathrm{Sat}(\Psi, 0000) = S$ for any rCTL formula Ψ, Table 9.2 only shows the cases for $b > 0000$.

To simplify the following presentation of the characterization, we split the discussion into three categories: atomic propositions, Boolean connectives, and temporal operators.

Table 9.2: Characterization of the satisfaction sets for rCTL formulas.

Symbol	$\mathrm{Sat}(\cdot,\cdot)$ for rCTL formulas Φ, Ψ and value $b \in \mathbb{B}_4 \setminus \{0000\}$
$p \in \mathtt{AP}$	$\mathrm{Sat}(p, b) = \{s \in S \mid p \in L(s)\}$
$\vee$	$\mathrm{Sat}(\Phi \vee \Psi, b) = \mathrm{Sat}(\Phi, b) \cup \mathrm{Sat}(\Psi, b)$
$\wedge$	$\mathrm{Sat}(\Phi \wedge \Psi, b) = \mathrm{Sat}(\Phi, b) \cap \mathrm{Sat}(\Psi, b)$
$\neg$	$\mathrm{Sat}(\neg\Phi, b) = S \setminus \mathrm{Sat}(\Phi, 1111)$
$\Rightarrow$	$\mathrm{Sat}(\Phi \Rightarrow \Psi, 1111) = \bigcap_b \mathrm{Sat}(\Psi, b) \cup (S \setminus \mathrm{Sat}(\Phi, b))$ $\mathrm{Sat}(\Phi \Rightarrow \Psi, b) = \mathrm{Sat}(\Phi \Rightarrow \Psi, 1111) \cup \mathrm{Sat}(\Psi, b)$ for any $b \leq 0111$
$\odot$	$\mathrm{Sat}(\exists \odot \Phi, b) = \{s \in S \mid \mathrm{post}(s) \cap \mathrm{Sat}(\Phi, b) \neq \emptyset\}$ $\mathrm{Sat}(\forall \odot \Phi, b) = \{s \in S \mid \mathrm{post}(s) \subseteq \mathrm{Sat}(\Phi, b)\}$
$\diamondsuit$	$\mathrm{Sat}(\exists \diamondsuit \Phi, b) = \mathsf{lfp}\, T.F^{\exists}\big(T, \mathrm{Sat}(\Phi, b), S\big)$ $\mathrm{Sat}(\forall \diamondsuit \Phi, b) = \mathsf{lfp}\, T.F^{\forall}\big(T, \mathrm{Sat}(\Phi, b), S\big)$
$\boxdot$	$\mathrm{Sat}(\exists \boxdot \Phi, 1111) = \mathsf{gfp}\, T.F^{\exists}\big(T, \emptyset, \mathrm{Sat}(\Phi, 1111)\big)$ $\mathrm{Sat}(\exists \boxdot \Phi, 0111) = \mathsf{lfp}\, T_1.\mathsf{gfp}\, T_2.G^{\exists}(T_1, T_2, \emptyset, \mathrm{Sat}(\Phi, 0111))$ $\mathrm{Sat}(\exists \boxdot \Phi, 0011) = \mathsf{gfp}\, T_2.\mathsf{lfp}\, T_1.G^{\exists}(T_1, T_2, \emptyset, \mathrm{Sat}(\Phi, 0011))$ $\mathrm{Sat}(\exists \boxdot \Phi, 0001) = \mathsf{lfp}\, T.F^{\exists}\big(T, \mathrm{Sat}(\Phi, 0001), S\big)$ $\mathrm{Sat}(\forall \boxdot \Phi, 1111) = \mathsf{gfp}\, T.F^{\forall}\big(T, \emptyset, \mathrm{Sat}(\Phi, 1111)\big)$ $\mathrm{Sat}(\forall \boxdot \Phi, 0111) = \mathsf{lfp}\, T_1.\mathsf{gfp}\, T_2.G^{\forall}(T_1, T_2, \emptyset, \mathrm{Sat}(\Phi, 0111))$ $\mathrm{Sat}(\forall \boxdot \Phi, 0011) = \mathsf{gfp}\, T_2.\mathsf{lfp}\, T_1.G^{\forall}(T_1, T_2, \emptyset, \mathrm{Sat}(\Phi, 0011))$ $\mathrm{Sat}(\forall \boxdot \Phi, 0001) = \mathsf{lfp}\, T.F^{\forall}\big(T, \mathrm{Sat}(\Phi, 0001), S\big)$
$\mathbf{U}$	$\mathrm{Sat}(\exists(\Phi\, \mathbf{U}\, \Psi), b) = \mathsf{lfp}\, T.F^{\exists}\big(T, \mathrm{Sat}(\Psi, b), \mathrm{Sat}(\Phi, b)\big)$ $\mathrm{Sat}(\forall(\Phi\, \mathbf{U}\, \Psi), b) = \mathsf{lfp}\, T.F^{\forall}\big(T, \mathrm{Sat}(\Psi, b), \mathrm{Sat}(\Phi, b)\big)$
$\mathbf{W}$	$\mathrm{Sat}(\exists(\Phi\, \mathbf{W}\, \Psi), 1111) = \mathsf{gfp}\, T.F^{\exists}\big(T, \mathrm{Sat}(\Psi, 1111), \mathrm{Sat}(\Phi, 1111)\big)$ $\mathrm{Sat}(\exists(\Phi\, \mathbf{W}\, \Psi), 0111) = \mathsf{lfp}\, T_1.\mathsf{gfp}\, T_2.G^{\exists}(T_1, T_2, \mathrm{Sat}(\Psi, 0111), \mathrm{Sat}(\Phi, 0111))$ $\mathrm{Sat}(\exists(\Phi\, \mathbf{W}\, \Psi), 0011) = \mathsf{gfp}\, T_2.\mathsf{lfp}\, T_1.G^{\exists}(T_1, T_2, \mathrm{Sat}(\Psi, 0011), \mathrm{Sat}(\Phi, 0011))$ $\mathrm{Sat}(\exists(\Phi\, \mathbf{W}\, \Psi), 0001) = \mathsf{lfp}\, T.F^{\exists}\big(T, \mathrm{Sat}(\Psi, 0001) \cup \mathrm{Sat}(\Phi, 0001), S\big)$ $\mathrm{Sat}(\forall(\Phi\, \mathbf{W}\, \Psi), 1111) = \mathsf{gfp}\, T.F^{\forall}\big(T, \mathrm{Sat}(\Psi, 1111), \mathrm{Sat}(\Phi, 1111)\big)$ $\mathrm{Sat}(\forall(\Phi\, \mathbf{W}\, \Psi), 0111) = \mathsf{lfp}\, T_1.\mathsf{gfp}\, T_2.G^{\forall}(T_1, T_2, \mathrm{Sat}(\Psi, 0111), \mathrm{Sat}(\Phi, 0111))$ $\mathrm{Sat}(\forall(\Phi\, \mathbf{W}\, \Psi), 0011) = \mathsf{gfp}\, T_2.\mathsf{lfp}\, T_1.G^{\forall}(T_1, T_2, \mathrm{Sat}(\Psi, 0011), \mathrm{Sat}(\Phi, 0011))$ $\mathrm{Sat}(\forall(\Phi\, \mathbf{W}\, \Psi), 0001) = \mathsf{lfp}\, T.F^{\forall}\big(T, \mathrm{Sat}(\Psi, 0001) \cup \mathrm{Sat}(\Phi, 0001), S\big)$

Atomic Propositions. The valuation for atomic propositions is defined classically, as in the case of CTL. Hence, the satisfaction set $\mathrm{Sat}(p, b)$ of an atomic proposition $p \in \mathtt{AP}$ with a value $b > 0000$ is the set of all states whose label contains p.

Boolean Connectives. The computation of the satisfaction sets for the Boolean connectives closely follows the semantic definition based on the da Costa algebra. Conjunction and disjunction are implemented using the usual intersection and union of sets, respectively. The set $\mathrm{Sat}(\neg\Phi, b)$ is the complement of all states on which Φ evaluates to 1111 (recall that we assume $b > 0000$). Finally, the implementation of the implication is more involved. By definition, the set $\mathrm{Sat}(\Phi \Rightarrow \Psi, 1111)$ is the set of states s for which $V(s, \Phi)$ is less than $V(s, \Psi)$; in set notation, this is expressed by the intersection of the sets $\mathrm{Sat}(\Psi, b) \cup (S \setminus \mathrm{Sat}(\Phi, b))$ for each $b \in \mathbb{B}_4$. For any other truth value $b \leq 0111$, $\mathrm{Sat}(\Phi \Rightarrow \Psi, b)$ consists of all states where the implication evaluates to 1111 or Ψ evaluates to at least b.

Temporal Operators. Now let us explain the characterization of the satisfaction sets for formulas with temporal operators. As the formulas can start with an existential or a universal operator, we discuss the satisfaction sets for them individually.

A state s satisfies the formula $\exists \odot \Phi$ with a value of at least b if one of its successors satisfies Φ with a value of at least b. Hence, the set $\mathrm{Sat}(\exists \odot \Phi, b)$ is the set of states s such that one of its successors is in $\mathrm{Sat}(\Phi, b)$. Dually, the set $\mathrm{Sat}(\forall \odot \Phi, b)$ is the set of states s such that all of its successors are in $\mathrm{Sat}(\Phi, b)$.

As for CTL, we use fixed point equations over sets of states to compute satisfaction sets for rCTL formulas with the remaining temporal operators. So, let us first briefly describe some notation and useful properties of fixed point equations over sets of states. A function F that maps a set of states to another set of states is monotonic if $T_1 \subseteq T_2$ implies $F(T_1) \subseteq F(T_2)$ for all sets T_1, T_2 of states. All monotonic functions have unique least and greatest fixed points [215]. Hence, given a monotonic function F (with variable T), we write $\mathsf{lfp}\, T.F(T)$ and $\mathsf{gfp}\, T.F(T)$ to denote the least fixed point and the greatest fixed point of F, respectively. All functions we consider in the following are monotonic.

We begin with formulas of the form $\exists \diamondsuit \Phi$. By definition, a state s satisfies $\exists \diamondsuit \Phi$ with a value of at least b if there exists a path from s containing a state that satisfies Φ with a value of at least b. Since we are now dealing with paths, we can apply the expansion laws of rLTL [213]. In this particular case, we obtain the following statement: a state s satisfies $\exists \diamondsuit \Phi$ with a value of at least b if and only if s satisfies Φ with a value of at least b or one of its immediate successors satisfies $\exists \diamondsuit \Phi$ with a value of at least b. Hence, as in CTL, $\mathrm{Sat}(\exists \diamondsuit \Phi, b)$ is the smallest subset T of S satisfying $\mathrm{Sat}(\Phi, b) \cup \{s \in S \mid \mathrm{post}(s) \cap T \neq \emptyset\} \subseteq T$. To capture this via fixed point operators, we define the function

$$F^{\exists}(T, S_1, S_2) = S_1 \cup \{s \in S_2 \mid \mathrm{post}(s) \cap T \neq \emptyset\}.$$

So, $F^{\exists}(T, S_1, S_2)$ contains all states in S_1 as well as all states in S_2 that have a successor in T. Note that this definition is more general than what we need it here, which will

be useful for other temporal operators. But by fixing $S_1 = \mathrm{Sat}(\Phi, b)$ and $S_2 = S$, we capture the expansion law of $\exists \diamondsuit$. Thus, consider the map

$$T \mapsto F^{\exists}(T, \mathrm{Sat}(\Phi, b), S)$$

mapping sets of states to sets of states. We will prove that its fixed point $\mathsf{lfp}\, T.F^{\exists}(T, \mathrm{Sat}(\Phi, b), S)$ is indeed the satisfaction set of $\exists \diamondsuit \Phi$.

Dually, a state s satisfies the formula $\forall \diamondsuit \Phi$ with a value of at least b if every path starting from s contains a state satisfying Φ with value at least b. Using analogous arguments, one can show that the set $\mathrm{Sat}(\forall \diamondsuit \Phi, b)$ is the least fixed point $\mathsf{lfp}\, T.F^{\forall}(T, \mathrm{Sat}(\Phi, b), S)$, where $F^{\forall}$ is defined as

$$F^{\forall}(T, S_1, S_2) = S_1 \cup \{s \in S_2 \mid \mathrm{post}(s) \subseteq T\}.$$

Next, we consider formulas of the form $\exists \boxdot \Phi$. The characterization of the set $\mathrm{Sat}(\exists \boxdot \Phi, b)$ is more complex, and we discuss each truth value separately. Firstly, a state s satisfies $\exists \boxdot \Phi$ with value 1111 if there exists a path from s on which every state satisfies Φ with value 1111. By again applying an expansion law similar to that of CTL, this statement is equivalent to s satisfying Φ with value 1111 and one of its successors satisfying $\exists \boxdot \Phi$ with value 1111. Hence, the set $\mathrm{Sat}(\exists \boxdot \Phi, 1111)$ equals the greatest fixed point $\mathsf{gfp}\, T.F^{\exists}(T, \emptyset, \mathrm{Sat}(\Phi, 1111))$.

Next, a state s satisfies $\exists \boxdot \Phi$ with a value of at least 0111 if there exists a path from s on which eventually every state satisfies Φ with a value of at least 0111. A set of states with such a property can be expressed using nested fixed points as usual (see Arnold and Niwinski [23] for details). We will prove that the set $\mathrm{Sat}(\exists \boxdot \Phi, 0111)$ is equal to the nested fixed point

$$\mathsf{lfp}\, T_1.\mathsf{gfp}\, T_2.G^{\exists}(T_1, T_2, \emptyset, \mathrm{Sat}(\Phi, 0111)),$$

where $G^{\exists}$ is defined as

$$G^{\exists}(T_1, T_2, S_1, S_2) = S_1 \cup \{s \in S \mid \mathrm{post}(s) \cap T_1 \neq \emptyset\} \cup \{s \in S_2 \mid \mathrm{post}(s) \cap T_2 \neq \emptyset\}.$$

Intuitively, the inner greatest fixed point in this nested fixed point represents the property of a path that all states on that path satisfy Φ with a value of at least 0111 (similar to the case of $\exists \boxdot \Phi$ and truth value 1111 just discussed). Then, the outer least fixed point ensures that there exists a path that has a suffix with that property (similar to the case of $\exists \diamondsuit \Phi$ discussed above).

Similarly, a state s satisfies $\exists \boxdot \Phi$ with a value of at least 0011 if there exists a path from s on which there exist infinitely many states satisfying Φ with a value of at least 0011. Note that the property that a path contains infinitely many states satisfying Φ (with a value b) is the dual of the property that a path contains finitely many states satisfying Φ (with a value b). Hence, similar to the last case, it holds that

$$\mathrm{Sat}(\exists \boxdot \Phi, 0011) = \mathsf{gfp}\, T_2.\mathsf{lfp}\, T_1.G^{\exists}(T_1, T_2, \emptyset, \mathrm{Sat}(\Phi, 0011)).$$

Finally, a state s satisfies $\exists \boxdot \Phi$ with a value of at least 0001 if there exists a path from s containing a state that satisfies Φ with a value of at least 0001, which is equivalent to

satisfying $\exists \diamondsuit \Phi$ with a value of at least 0001. Hence, $\mathrm{Sat}(\exists \boxdot \Phi, 0001)$ is the least fixed point $\mathsf{lfp}\, T.F^{\exists}(T, \mathrm{Sat}(\Phi, 0001), S)$, as in the case of $\exists \diamondsuit \Phi$.

Analogously, one can characterize $\forall \boxdot \Phi$ using the fixed points of the functions $F^{\forall}$ and $G^{\forall}$, where

$$G^{\forall}(T_1, T_2, S_1, S_2) = S_1 \cup \{s \in S \mid \mathrm{post}(s) \subseteq T_1\} \cup \{s \in S_2 \mid \mathrm{post}(s) \subseteq T_2\}.$$

As the semantics of $\mathbf{U}$ mimics the classical semantics, its characterization is generalized from that of $\diamondsuit$, as for CTL. Hence, its characterization can be obtained using the functions $F^{\exists}$ and $F^{\forall}$. We describe the case $\exists \Phi \,\mathbf{U}\, \Psi$, and the case $\forall \Phi \,\mathbf{U}\, \Psi$ is again similar. A state s satisfies $\exists \Phi \,\mathbf{U}\, \Psi$ with a value of at least b if there exists a path from s containing a state that satisfies Ψ with a value of at least b and every state before that in the path satisfies Φ with a value of at least b. By applying the expansion law of rLTL [213], this statement is equivalent to s satisfying Ψ with a value of at least b or it satisfying Φ with a value of at least b and one of its successors satisfying $\exists \Phi \,\mathbf{U}\, \Psi$ with a value of at least b. Hence, as in CTL, $\mathrm{Sat}(\exists \Phi \,\mathbf{U}\, \Psi, b)$ is the smallest subset T of S satisfying $\mathrm{Sat}(\Psi, b) \cup \{s \in \mathrm{Sat}(\Phi, b) \mid \mathrm{post}(s) \cap T \neq \emptyset\} \subseteq T$. This is captured by the map

$$T \mapsto F^{\exists}(T, \mathrm{Sat}(\Psi, b), \mathrm{Sat}(\Phi, b)).$$

Therefore, the least fixed point

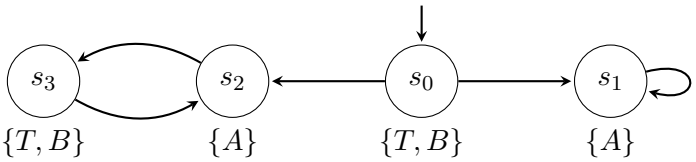
$$\mathsf{lfp}\, T.F^{\exists}(T, \mathrm{Sat}(\Psi, b), \mathrm{Sat}(\Phi, b))$$

of the map is the satisfaction set of $\exists \Phi \diamondsuit \Psi$.

Finally, the semantics of $\Phi \,\mathbf{W}\, \Psi$ is also defined using the counting interpretation described in Section 9.2, similarly to the semantics of $\boxdot \Phi$. However, note that the satisfaction sets for $\boxdot \Phi$ are characterized only using the satisfaction sets for Φ, whereas the satisfaction sets for $\Phi \,\mathbf{W}\, \Psi$ must be characterized using the satisfaction sets of both formulas Φ and Ψ. Hence, the characterization for $\mathbf{W}$ can be obtained using the similar fixed points as for $\boxdot$ but using the satisfaction sets of both formulas Φ and Ψ.

Example 9.2. Before proving that the characterization in Table 9.2 is correct, let us illustrate it on a simple Kripke structure, depicted in Figure 9.1. Continuing the example presented in Example 9.1, this Kripke structure demonstrates a scenario, where the robot can only perform its task in room B. Initially, the robot is in room B (and hence, it can perform its task), captured by the initial state s_0 (with label $\{T, B\}$) of the Kripke structure. From there, the robot can move towards room A by transitioning to state s_1 or state s_2 (both with label $\{A\}$). In one case, the robot can never return to room B, captured by self-loop in state s_1, and in the other case, it can move between both rooms, captured by transitions between states s_2 and s_3.

For this Kripke structure, we now compute, for each state, the maximal truth values with which the state satisfies the subformulas of $\Phi = \forall \boxdot (A \Rightarrow \exists \odot B) \Rightarrow \forall \boxdot T$. If this value is b for some state s and some subformula Ψ, then we have $s \in \mathrm{Sat}(\Psi, b')$ for all $b' \leq b$ and $s \notin \mathrm{Sat}(\Psi, b')$ for all $b' > b$. Furthermore, as $\mathrm{Sat}(\Psi, 0000) = S$ for any formula Ψ, the maximal valuation is 0000 in a state s whenever $s \notin \mathrm{Sat}(\Psi, b)$ for any $b > 0000$.



T:	1111	0000	1111	0000
A:	0000	1111	0000	1111
B:	1111	0000	1111	0000
$\exists \odot B$:	0000	1111	0000	0000
$A \Rightarrow \exists \odot B$:	1111	1111	1111	0000
$\forall \boxdot (A \Rightarrow \exists \odot B)$:	1111	1111	0001	0000
$\forall \boxdot T$:	0011	0011	0001	0000
$\forall \boxdot (A \Rightarrow \exists \odot B) \Rightarrow \forall \boxdot T$:	0011	0011	1111	1111

Figure 9.1: A Kripke structure that tracks a possible interaction between a robot and office workers. Within each state, we mention its identifier. Below each state, we mention its label and below that, we mention the maximal valuation that holds in the state for each subformula of $\Phi = \forall \boxdot (A \Rightarrow \exists \odot B) \Rightarrow \forall \boxdot T$.

These maximal valuations are indicated below the corresponding state in Figure 9.1.

In Figure 9.1, observe that T holds with a value of 1111 in states s_0 and s_3, and with a value of 0000 in states s_1 and s_2. This is because only the labels of states s_0 and s_3 contain T. This is consistent with our characterization of satisfiable set for atomic propositions, presented in Table 9.2:

$$\mathrm{Sat}(T, b) = \{s_0, s_3\} \text{ for } b > 0000.$$

Similarly, A holds with a value of 1111 only in states s_1 and s_2; and B holds with a value of 1111 only in states s_0 and s_3, as reflected by the characterization sets:

$$\mathrm{Sat}(A, b) = \{s_1, s_2\} \text{ and } \mathrm{Sat}(B, b) = \{s_0, s_3\} \text{ for } b > 0000.$$

Next, observe the formula $\exists \odot B$ holds with a value of 1111 in state s_2 and with a value of 0000 in all other states. This is because, $\mathrm{post}(s_2)$ contains a state where B holds with a value of 1111, namely, the state s_3. In contrast, $\mathrm{post}(s_0)$, $\mathrm{post}(s_1)$, and $\mathrm{post}(s_3)$ do not contain any states where B holds with a value larger than 0000. This is reflected in our characterization of the next operator: for any $b > 0000$, we have

$$\mathrm{Sat}(\exists \odot B, b) = \{s \in S \mid \mathrm{post}(s) \cap \mathrm{Sat}(B, b) \neq \emptyset\} = \{s \in S \mid \mathrm{post}(s) \cap \{s_0, s_3\} \neq \emptyset\} = \{s_2\}.$$

Furthermore, the formula $A \Rightarrow \exists \odot B$ holds with a value of 1111 in states s_0, s_2, and s_3, and with a value of 0000 in state s_1. This is because, in states s_0, s_2, and s_3, the maximal valuation of A is equal to that of $\exists \odot B$; and in state s_1, the maximal valuation of $\exists \odot B$ is 0000 which is less than that of A. This is captured in our characterization of implication:

$$\mathrm{Sat}(A \Rightarrow \exists \odot B, 1111) = \{s \in S \mid \textstyle\bigcap_b (\mathrm{Sat}(\exists \odot B, b) \cup S \setminus \mathrm{Sat}(A, b))\} = \{s_0, s_2, s_3\}, \text{ and}$$
$$\mathrm{Sat}(A \Rightarrow \exists \odot B, b) = \mathrm{Sat}(A \Rightarrow \exists \odot B, 1111) \cup \mathrm{Sat}(\exists \odot B, b) = \{s_0, s_2, s_3\} \text{ for } 0000 < b < 1111.$$

Now, the formula $\forall\boxdot(A \Rightarrow \exists\odot B)$ holds in states s_2 and s_3. This is because the only path from s_2 or s_3 is the one that loops between s_2 and s_3, and in both of these states, the formula $A \Rightarrow \exists\odot B$ holds with a value of 1111. This is exactly captured in our characterization for the always operator:

$$\begin{aligned}\mathrm{Sat}(\forall\boxdot(A \Rightarrow \exists\odot B), 1111) &= \mathsf{gfp}\, T.F^{\forall}\big(T, \emptyset, \mathrm{Sat}(A \Rightarrow \exists\odot B, 1111)\big)\\ &= \mathsf{gfp}\, T.F^{\forall}(T, \emptyset, \{s_0, s_2, s_3\}) = \{s_2, s_3\}.\end{aligned}$$

Furthermore, the formula $\forall\boxdot(A \Rightarrow \exists\odot B)$ holds with a value of 0001 in state s_0, and with a value of 0000 in state s_1. This is because, from state s_0, there is a path, i.e., $s_0(s_1)^{\omega}$, that eventually always visits a state where $A \Rightarrow \exists\odot B$ holds with a value of 0000. Similarly, from state s_1, there is only one path, i.e., $(s_1)^{\omega}$, that only visits states where $A \Rightarrow \exists\odot B$ holds with a value of 0000. Again, this is captured in our characterization for the always operator:

$$\begin{aligned}\mathrm{Sat}(\forall\boxdot(A \Rightarrow \exists\odot B), 0011) &= \mathsf{gfp}\, T_2.\mathsf{lfp}\, T_1.G^{\forall}\big(T_1, T_2, \emptyset, \mathrm{Sat}(A \Rightarrow \exists\odot B, 0011)\big)\\ &= \mathsf{gfp}\, T_2.\mathsf{lfp}\, T_1.G^{\forall}(T_1, T_2, \emptyset, \{s_0, s_2, s_3\}) = \{s_2, s_3\};\\ \mathrm{Sat}(\forall\boxdot(A \Rightarrow \exists\odot B), 0001) &= \mathsf{lfp}\, T.F^{\forall}\big(T, \mathrm{Sat}(A \Rightarrow \exists\odot B, 0001), S\big)\\ &= \mathsf{lfp}\, T.F^{\forall}\big(T, \{s_0, s_2, s_3\}, S\big) = \{s_0, s_2, s_3\}.\end{aligned}$$

Analogously, the formula $\forall\boxdot T$ holds with a value of 0011 in states s_2 and s_3 as the only path from s_2 or s_3 is the one that loops between s_2 and s_3 visiting a state where T holds with a value of 1111 infinitely often. Furthermore, the formula $\forall\boxdot T$ holds with a value of 0001 in state s_0 and with a value of 0000 in state s_1 as there exists a path from s_0 (resp. s_1) that eventually always (resp. always) visits a state where T holds with a value of 0000. This is captured in our characterization for the always operator:

$$\begin{aligned}\mathrm{Sat}(\forall\boxdot T, 0111) &= \mathsf{lfp}\, T_1.\mathsf{gfp}\, T_2.G^{\forall}\big(T_1, T_2, \emptyset, \mathrm{Sat}(T, 0011)\big)\\ &= \mathsf{lfp}\, T_1.\mathsf{gfp}\, T_2.G^{\forall}(T_1, T_2, \emptyset, \{s_0, s_3\}) = \emptyset;\\ \mathrm{Sat}(\forall\boxdot T, 0011) &= \mathsf{gfp}\, T_2.\mathsf{lfp}\, T_1.G^{\forall}\big(T_1, T_2, \emptyset, \mathrm{Sat}(T, 0011)\big)\\ &= \mathsf{gfp}\, T_2.\mathsf{lfp}\, T_1.G^{\forall}(T_1, T_2, \emptyset, \{s_0, s_3\}) = \{s_2, s_3\};\\ \mathrm{Sat}(\forall\boxdot T, 0001) &= \mathsf{lfp}\, T.F^{\forall}\big(T, \mathrm{Sat}(T, 0001), S\big)\\ &= \mathsf{lfp}\, T.F^{\forall}\big(T, \{s_0, s_3\}, S\big) = \{s_0, s_2, s_3\}.\end{aligned}$$

Finally, the formula $\Phi = \forall\boxdot(A \Rightarrow \exists\odot B) \Rightarrow \forall\boxdot T$ holds with a value of 1111 in states s_0 and s_1 as the maximal valuation $\forall\boxdot T$ is equal to that of $\forall\boxdot(A \Rightarrow \exists\odot B)$ in these states. In states s_2 and s_3, the formula Φ holds with a value of 0011 as the maximal valuation $\forall\boxdot T$ is 0011 which is less than that of $\forall\boxdot(A \Rightarrow \exists\odot B)$. This is captured in our characterization of implication:

$$\begin{aligned}&\mathrm{Sat}(\Phi, 1111) = \{s \in S \mid \textstyle\bigcap_b(\mathrm{Sat}(\forall\boxdot T, b) \cup S \setminus \mathrm{Sat}(\forall\boxdot(A \Rightarrow \exists\odot B), b))\} = \{s_0, s_1\};\\ &\mathrm{Sat}(\Phi, 0111) = \mathrm{Sat}(\Phi, 1111) \cup \mathrm{Sat}(\forall\boxdot T, 0111) = \{s_0, s_1\};\\ &\mathrm{Sat}(\Phi, 0011) = \mathrm{Sat}(\Phi, 1111) \cup \mathrm{Sat}(\forall\boxdot T, 0001) = \{s_0, s_1, s_2, s_3\}.\end{aligned}$$

We note that the intuition and the computation of the nested fixed points can be found in Arnold and Niwinski [23]. ⌟

Lemma 9.2. *The characterization of the satisfaction sets in Table 9.2 is correct.*

Proof. Let $\mathcal{K} = (S, I, R, L)$ be a given Kripke structure and $b \in \mathbb{B}_4 \setminus \{0000\}$. Now, we show that every equation in Table 9.2 using a case-by-case analysis.

- $$\begin{aligned} s \in \mathrm{Sat}(p, b) &\iff V(s,p) \geq b > 0000 \\ &\iff V(s,p) = 1111 \\ &\iff p \in L(s). \end{aligned}$$

- $$\begin{aligned} s \in \mathrm{Sat}(\Phi \vee \Psi, b) &\iff V(\Phi \vee \Psi) \geq b \\ &\iff \max\{V(s,\Phi), V(s,\Psi)\} \geq b \\ &\iff V(s,\Phi) \geq b \text{ or } V(s,\Psi) \geq b \\ &\iff s \in \mathrm{Sat}(\Phi, b) \text{ or } s \in \mathrm{Sat}(\Psi, b) \\ &\iff s \in \mathrm{Sat}(\Phi, b) \cup \mathrm{Sat}(\Psi, b). \end{aligned}$$

- $$\begin{aligned} s \in \mathrm{Sat}(\Phi \wedge \Psi, b) &\iff V(\Phi \wedge \Psi) \geq b \\ &\iff \min\{V(s,\Phi), V(s,\Psi)\} \geq b \\ &\iff V(s,\Phi) \geq b \text{ and } V(s,\Psi) \geq b \\ &\iff s \in \mathrm{Sat}(\Phi, b) \text{ and } s \in \mathrm{Sat}(\Psi, b) \\ &\iff s \in \mathrm{Sat}(\Phi, b) \cap \mathrm{Sat}(\Psi, b). \end{aligned}$$

- $$\begin{aligned} s \in \mathrm{Sat}(\neg\Phi, b) &\iff V(s, \neg\Phi) \geq b \geq 0001 \\ &\iff \overline{V(s,\Phi)} = 1111 \\ &\iff V(s,\Phi) \neq 1111 \\ &\iff s \in S \setminus \mathrm{Sat}(\Phi, 1111). \end{aligned}$$

- $$\begin{aligned} s \in \mathrm{Sat}(\Phi \Rightarrow \Psi, 1111) &\iff \big(V(s,\Phi) \to V(s,\Psi)\big) = 1111 \\ &\iff V(s,\Phi) \leq V(s,\Psi) \\ &\iff \forall b \in \mathbb{B}_4 \colon s \notin \mathrm{Sat}(\Phi, b) \setminus \mathrm{Sat}(\Psi, b) \\ &\iff \forall b \in \mathbb{B}_4 \colon s \in \big(S \setminus \mathrm{Sat}(\Phi, b)\big) \cup \mathrm{Sat}(\Psi, b) \\ &\iff s \in \bigcap_b \mathrm{Sat}(\Psi, b) \cup (S \setminus \mathrm{Sat}(\Phi, b)). \end{aligned}$$

  Similarly, for some $b \leq 0111$,

  $$\begin{aligned} s \in \mathrm{Sat}(\Phi \Rightarrow \Psi, b) &\iff \big(V(s,\Phi) \to V(s,\Psi)\big) \geq b \\ &\iff V(s,\Phi) \leq V(s,\Psi) \text{ or } V(s,\Psi) = b \\ &\iff s \in \mathrm{Sat}(\Phi \Rightarrow \Psi, 1111) \cup \mathrm{Sat}(\Psi, b). \end{aligned}$$

- $s \in \mathrm{Sat}(\exists \bigodot \Phi, b) \iff \exists \rho \in \mathrm{paths}(s) \colon V(\rho, \bigodot \Phi) \geq b$
  $\iff \exists \rho \in \mathrm{paths}(s) \colon V(\rho[1], \Phi) \geq b$
  $\iff \exists s' \in \mathrm{post}(s) \colon V(s', \Phi) \geq b$
  $\iff \mathrm{post}(s) \cap \mathrm{Sat}(\Phi, b) \neq \emptyset$

  and

  $s \in \mathrm{Sat}(\forall \bigodot \Phi, b) \iff \forall \rho \in \mathrm{paths}(s) \colon V(\rho, \bigodot \Phi) \geq b$
  $\iff \forall \rho \in \mathrm{paths}(s) \colon V(\rho[1], \Phi) \geq b$
  $\iff \forall s' \in \mathrm{post}(s) \colon V(s', \Phi) \geq b$
  $\iff \mathrm{post}(s) \subseteq \mathrm{Sat}(\Phi, b).$

It remains to consider the temporal operators eventually, always, until, and release. Here, we need to prove that the fixed-point characterizations presented are correct. For most cases, the technical core of the arguments are standard characterizations of safety (a state formula holds at every state of a given path), co-Büchi (a state formula holds almost always on a given path), Büchi (a state formula holds infinitely often on a given path), and reachability (a state formula holds in at least one state of a given path) conditions. We present several cases in detail and refer for the other ones to the book by Arnold and Niwinski [23] for more details.

- $s \in \mathrm{Sat}(\exists(\Phi \mathbin{\mathbf{U}} \Psi), b)$
  $\iff \exists \rho \in \mathrm{paths}(s) \colon V(\rho, \Phi \mathbin{\mathbf{U}} \Psi) \geq b$
  $\iff \exists \rho \in \mathrm{paths}(s), \exists j \geq 0, \forall i < j \colon V(\rho[j], \Psi) \geq b \wedge V(\rho[i], \Phi) \geq b$
  $\iff \big(V(s, \Psi) \geq b\big) \vee \big(V(s, \Phi) \geq b \wedge \exists s' \in \mathrm{post}(s) \colon V(s', \exists(\Phi \mathbin{\mathbf{U}} \Psi)) \geq b\big)$
  $\iff s \in \mathrm{Sat}(\Psi, b) \cup \{s' \in \mathrm{Sat}(\Phi, b) \mid \mathrm{post}(s') \cap \mathrm{Sat}(\exists(\Phi \mathbin{\mathbf{U}} \Psi, b)) \neq \emptyset\}.$

  Hence, $\mathrm{Sat}(\exists(\Phi \mathbin{\mathbf{U}} \Psi), b)$ is a fixed point of the function $T \mapsto F^{\exists}(T, \mathrm{Sat}(\Psi, b), \mathrm{Sat}(\Phi, b))$. Now, we only need to show that it is indeed the least fixed point. Suppose $T'$is another fixed point of that function. If $s_0 \in \mathrm{Sat}(\exists(\Phi \mathbin{\mathbf{U}} \Psi), b)$, then there exists a path $\rho = s_0 s_1 s_2 \cdots$ and some $j > 0$ such that $V(s_j, \Psi) \geq b$ and $V(s_i, \Phi) \geq b$ for all $0 \leq i < j$. Then:

  - $s_j \in \mathrm{Sat}(\Psi, b) \subseteq T'$;
  - $s_{j-1} \in T'$, since $s_j \in \mathrm{post}(s_{j-1}) \cap T'$ and $s_{j-1} \in \mathrm{Sat}(\Phi, b)$;
  - $s_{j-2} \in T'$, since $s_{j-1} \in \mathrm{post}(s_{j-2}) \cap T'$ and $s_{j-2} \in \mathrm{Sat}(\Phi, b)$;
  - Applying this argument repeatedly yields $s_0 \in T'$.

  So, $\mathrm{Sat}(\exists(\Phi \mathbin{\mathbf{U}} \Psi), b) \subseteq T'$. Therefore, $\mathrm{Sat}(\exists(\Phi \mathbin{\mathbf{U}} \Psi), b)$ is the least fixed point of $T \mapsto F^{\exists}(T, \mathrm{Sat}(\Psi, b), \mathrm{Sat}(\Phi, b))$. Similarly, it can be shown that $\mathrm{Sat}(\forall(\Phi \mathbin{\mathbf{U}} \Psi), b)$ is the least fixed point of $T \mapsto F^{\forall}(T, \mathrm{Sat}(\Psi, b), \mathrm{Sat}(\Phi, b))$.

- In the following, we use `true` as syntactic sugar for some tautology, e.g., $p \vee \neg p$. Then, $\diamondsuit \Phi$ is equivalent to $\mathtt{true} \mathbin{\mathbf{U}} \Phi$ and we have $\mathrm{Sat}(\mathtt{true}, b) = S$. Hence,

$$\begin{aligned}\mathrm{Sat}(\exists \diamondsuit\!\!\!\cdot\, \Phi, b) &= \mathrm{Sat}(\exists(\mathtt{true}\,\mathbf{U}\,\Phi)) \\ &= \mathsf{lfp}\, T.F^{\exists}\big(T, \mathrm{Sat}(\Phi, b), \mathrm{Sat}(\mathtt{true}, b)\big) \\ &= \mathsf{lfp}\, T.F^{\exists}\big(T, \mathrm{Sat}(\Phi, b), S\big).\end{aligned}$$

Similarly, we have $\mathrm{Sat}(\forall \diamondsuit\!\!\!\cdot\, \Phi, b) = \mathsf{lfp}\, T.F^{\forall}\big(T, \mathrm{Sat}(\Phi, b), S\big)$, as claimed.

- $s \in \mathrm{Sat}(\exists \boxdot \Phi, 1111)$
  $\iff \exists \rho \in \mathrm{paths}(s)\colon V(\rho, \boxdot \Phi) = 1111$
  $\iff \exists \rho \in \mathrm{paths}(s), \forall i \geq 0\colon V(\rho[i], \Phi) = 1111$
  $\iff s \in \mathrm{Sat}(\Phi, 1111) \wedge \big(\mathrm{post}(s) \cap \mathrm{Sat}(\exists \boxdot \Phi, 1111) \neq \emptyset\big)$
  $\iff s \in \mathrm{Sat}(\Phi, 1111) \cap \{s' \mid \mathrm{post}(s') \cap \mathrm{Sat}(\exists \boxdot \Phi, 1111) \neq \emptyset\}$.

  Hence, $\mathrm{Sat}(\exists \boxdot \Phi, 1111)$ is a fixed point of the function $T \mapsto F^{\exists}(T, \emptyset, \mathrm{Sat}(\Phi, 1111))$. Now, we only need to show that it is indeed the greatest fixed point. Now, suppose T' is another fixed point. If $s_0 \in T'$, then

  - since $s_0 \in T'$, there exists a state $s_1 \in \mathrm{post}(s_0) \cap T'$;
  - since $s_1 \in T'$, there exists a state $s_2 \in \mathrm{post}(s_0) \cap T'$;

  Applying this argument iteratively yields that there exists a path $s_0 s_1 \cdots$ starting from s such that $V(s_i, \Phi) = 1111$ for each $i \geq 0$. Hence, $s_0 \in \mathrm{Sat}(\exists \boxdot \Phi, 1111)$, which implies $T' \subseteq \mathrm{Sat}(\exists \boxdot \Phi, 1111)$. Therefore, $\mathrm{Sat}(\exists \boxdot \Phi, 1111)$ is the greatest fixed point of $T \mapsto F^{\exists}(T, \emptyset, \mathrm{Sat}(\Phi, 1111))$.

  Similarly, the following holds

  $s \in \mathrm{Sat}(\exists \boxdot \Phi, 0111)$
  $\iff \exists \rho \in \mathrm{paths}(s), V(\rho, \boxdot \Phi) \geq 0111$
  $\iff \exists \rho \in \mathrm{paths}(s), \exists j \geq 0, \forall i \geq j\colon V_2(\rho[i], \Phi) = 1$
  $\iff \rho$ visits $\mathrm{Sat}(\Phi, 0111)$ eventually always.

  Moreover, $s \in \mathrm{Sat}(\exists \boxdot \Phi, 0011)$
  $\iff \exists \rho \in \mathrm{paths}(s), V(\rho, \boxdot \Phi) \geq 0011$
  $\iff \exists \rho \in \mathrm{paths}(s), \forall j \geq 0, \exists i > j\colon V_3(\rho[i], \Phi) = 1$
  $\iff \rho$ visits $\mathrm{Sat}(\Phi, 0011)$ infinitely often.

  A path visiting a set eventually always or infinitely often can be written in terms of nested fixed points as claimed (see Arnold and Niwinski [23] for details).

  Finally, $s \in \mathrm{Sat}(\exists \boxdot \Phi, 0001)$
  $\iff \exists \rho \in \mathrm{paths}(s), V(\rho, \boxdot \Phi) \geq 0001.$
  $\iff \exists \rho \in \mathrm{paths}(s), \exists i \geq 0\colon V_4(\rho[i], \Phi) = 1$
  $\iff s \in \mathrm{Sat}(\exists \diamondsuit\!\!\!\cdot\, \Phi, 0001)$.

  Hence, $\mathrm{Sat}(\exists \boxdot \Phi, 0001) = \mathsf{lfp}\, T.F^{\exists}\big(T, \mathrm{Sat}(\Phi, 0001), S\big)$, as claimed.

  Analogously, one can show the claimed results for $\forall \boxdot \Phi$.

- For the operator $\mathbf{W}$, the claim can be shown using arguments similar to those for $\boxdot$. □

Algorithm 9.1 computes $5 \cdot |\mathrm{sub}(\Phi)|$ satisfaction sets following the subformula ordering. Using the standard fixed point iterations [75], the nested fixed points of depth two can be computed in time $\mathcal{O}(NK)$ on a Kripke structure with N vertices and K transitions [64]. So, we obtain the following.

Theorem 9.3. *Given an rCTL formula Φ and a Kripke structure with N states and K transitions, the rCTL model checking problem can be solved in time $\mathcal{O}(NK|\Phi|)$.*

Note that the CTL model checking algorithm also takes polynomial time in the size of the formula and the number of transitions of the Kripke structure [70]. Hence, both model checking problems are in PTIME. Moreover, a lower bound of the rCTL model checking problem can be derived from the PTIME lower bound of CTL model checking [198] and Lemma 9.1. In total, we obtain the following result, showing that the CTL and rCTL model checking problems have the same asymptotic complexity.

Corollary 9.2. *The model checking problem for rCTL is* PTIME*-complete.*

9.2.5 rCTL and the Modal μ-calculus

In the previous section, we have seen that one can solve the rCTL model checking problem by computing least and greatest fixed points. In this section, we show that every rCTL formula can be translated into an equivalent formula of the modal μ-calculus [129], i.e., modal logic with least and greatest fixed points. This is not necessarily surprising, as most temporal logics can be translated into the modal μ-calculus [71]. However, the result is very useful to settle the complexity of satisfiability and synthesis, which we achieve by reductions to satisfiability and synthesis for the modal μ-calculus.

We begin by reviewing the basic definitions of the modal μ-calculus. It consists of state formulas only, which are constructed from atomic propositions with Boolean connectives, the temporal operators $\exists \bigcirc$ and $\forall \bigcirc$, as well as the least (μ) and the greatest (ν) fixed point operator.

Formally, given a set $\mathtt{AP}$ of atomic propositions and a set $\mathtt{PV}$ of atomic proposition variables, μ-calculus formulas are given by the grammar

$$\Phi ::= p \mid y \mid \Phi \vee \Phi \mid \Phi \wedge \Phi \mid \neg\Phi \mid \Phi \Rightarrow \Phi \mid \exists \bigcirc \varphi \mid \forall \bigcirc \varphi \mid \mu y.\Phi \mid \nu y.\Phi,$$

where $p \in \mathtt{AP}$ and $y \in \mathtt{PV}$. As usual, we require that in subformulas of the form $\mu y.\Phi$ and $\nu y.\Phi$, every free occurrence of y in Φ is under the scope of an even number of negations. For further details, we refer the reader to standard literature on this topic, e.g., Grädel, Thomas, and Wilke [112].

Unlike temporal logics, the semantics of the μ-calculus is naturally defined using satisfaction sets. Given a Kripke structure $\mathcal{K} = (S, I, R, L)$, the satisfaction sets are defined with respect to a variable function $v \colon \mathtt{PV} \to 2^S$ that maps each atomic proposition

Table 9.3: Characterization of the satisfaction sets for μ-calculus formulas.

Symbol	$\mathrm{Sat}_\mu^v(\cdot)$ for μ-calculus formulas Φ, Ψ
$p \in \mathtt{AP}$	$\mathrm{Sat}_\mu^v(p) = \{s \in S \mid p \in L(s)\}$
$\vee$	$\mathrm{Sat}_\mu^v(\Phi \vee \Psi) = \mathrm{Sat}_\mu^v(\Phi) \cup \mathrm{Sat}_\mu^v(\Psi)$
$\wedge$	$\mathrm{Sat}_\mu^v(\Phi \wedge \Psi) = \mathrm{Sat}_\mu^v(\Phi) \cap \mathrm{Sat}_\mu^v(\Psi)$
$\neg$	$\mathrm{Sat}_\mu^v(\neg\Phi) = S \setminus \mathrm{Sat}_\mu^v(\Phi)$
$\Rightarrow$	$\mathrm{Sat}_\mu^v(\Phi \Rightarrow \Psi) = \mathrm{Sat}_\mu^v(\neg\Phi) \cup \mathrm{Sat}_\mu^v(\Psi)$
$\exists\bigcirc$	$\mathrm{Sat}_\mu^v(\exists\bigcirc\Phi) = \{s \in S \mid \mathrm{post}(s) \cap \mathrm{Sat}_\mu^v(\Phi) \neq \emptyset\}$
$\forall\bigcirc$	$\mathrm{Sat}_\mu^v(\forall\bigcirc\Phi) = \{s \in S \mid \mathrm{post}(s) \subseteq \mathrm{Sat}_\mu^v(\Phi)\}$
$y \in \mathtt{PV}$	$\mathrm{Sat}_\mu^v(y) = v(y)$
μy	$\mathrm{Sat}_\mu^v(\mu y.\Phi) = \mathsf{lfp}\, T.\mathrm{Sat}_\mu^{v[y \to T]}(\Phi)$
νy	$\mathrm{Sat}_\mu^v(\nu y.\Phi) = \mathsf{gfp}\, T.\mathrm{Sat}_\mu^{v[y \to T]}(\Phi)$

variable to a set of states. Moreover, for a subset $T \subseteq S$, let $v[y \to T]$ denote the variable function that maps y to T while preserving the value of v for every other input.

Given a variable function v and a μ-calculus formula Φ, let $\mathrm{Sat}_\mu^v(\Phi)$ denote the set of states satisfying Φ with respect to v. These sets are defined recursively using the least and greatest fixed points, as shown in Table 9.3. The functions the fixed point operators are applied to are monotonic since every occurrence of a fixed point variable is under an even number of negations. Hence, the fixed points all exist.

For μ-calculus sentences, i.e., formulas without free variables, the satisfaction sets Sat_μ^v are independent of v. Hence, we drop the parameter v from the notation whenever possible.

We now show that, like other temporal logics, rCTL can also be translated into the modal μ-calculus. As before, we say an rCTL formula Φ with a truth value $b \in \mathbb{B}_4$ is equivalent to a μ-calculus sentence Φ' if for every Kripke structure it holds that $\mathrm{Sat}(\Phi, b) = \mathrm{Sat}_\mu(\Phi')$. Then we have the following result.

Theorem 9.4. *For every rCTL formula and truth value, there is an equivalent μ-calculus sentence of linear size.*

Proof. We show that there exists a mapping t that assigns to every rCTL formula Φ and truth value $b \in \mathbb{B}_4$ an equivalent μ-calculus formula $t(\Phi, b)$. We define this mapping recursively, starting with the atomic rCTL formulas. In the following proof, we use `true` as syntactic sugar for an arbitrary tautology of the μ-calculus, e.g., $p \vee \neg p$.

First of all, for any rCTL formula Φ with truth value 0000, a trivial equivalent μ-calculus formula is $t(\Phi, 0000) = \mathtt{true}$. Furthermore, comparing the characterization of

the satisfaction sets of rCTL and the μ-calculus (Tables 9.2 and 9.3), one can see that for a Boolean combination of rCTL formulas Φ and Ψ with any truth value $b \in \mathbb{B}_4 \setminus \{0000\}$, the following recursive translations indeed results in an equivalent μ-calculus formula:

$$\begin{aligned} t(p, b) &= p \text{ for each } p \in \mathtt{AP}, \\ t(\Phi \vee \Psi, b) &= t(\Phi, b) \vee t(\Psi, b), \\ t(\Phi \wedge \Psi, b) &= t(\Phi, b) \wedge t(\Psi, b), \\ t(\neg\Phi, b) &= \neg t(\Phi, 1111). \end{aligned}$$

Moreover, we have

$$\begin{aligned} t(\Phi \Rightarrow \Psi, 1111) &= \bigwedge_b t(\Psi, b) \vee \neg t(\Phi, b), \\ t(\Phi \Rightarrow \Psi, b) &= t(\Phi \Rightarrow \Psi, 1111) \vee t(\Psi, b) \end{aligned}$$

for any $b \leq 0111$.

The rCTL formulas with the next operator are captured by applying the μ-calculus operators $\exists\bigcirc$ and $\forall\bigcirc$ as follows for $b \in \mathbb{B}_4 \setminus \{0000\}$:

$$\begin{aligned} t(\exists\odot\Phi, b) &= \exists\bigcirc t(\Phi, b), \\ t(\forall\odot\Phi, b) &= \forall\bigcirc t(\Phi, b). \end{aligned}$$

For rCTL formulas with other temporal operators, the satisfaction sets (in Table 9.2) are defined using fixed points of functions $F^\exists$, $F^\forall$, $G^\exists$, and $G^\forall$. Hence, we first give μ-calculus formulas that capture these functions. Note that if the sets T, S_1, S_2 are the satisfaction sets of the rCTL formulas Φ_t, Φ_1, Φ_2 with truth values b_t, b_1, b_2, respectively, then it holds that

$$\begin{aligned} F^\exists(T, S_1, S_2) &= S_1 \cup \{s \in S_2 \mid \mathrm{post}(s) \cap T \neq \emptyset\} \\ &= S_1 \cup (S_2 \cap \{s \in S \mid \mathrm{post}(s) \cap T \neq \emptyset\}) \\ &= \mathrm{Sat}(\Phi_1, b_1) \cup \big(\mathrm{Sat}(\Phi_2, b_2) \cap \mathrm{Sat}(\exists\odot\Phi_t, b_t)\big). \end{aligned}$$

Now, suppose that the μ-calculus formula $t(\Phi_1, b_1)$ is equivalent to the rCTL formula Φ_1 with truth value b_1, and the μ-calculus formula $t(\Phi_2, b_2)$ is equivalent to the rCTL formula Φ_2 with truth value b_2. Then, we have

$$F^\exists\big(T, \mathrm{Sat}(\Phi_1, b_1), \mathrm{Sat}(\Phi_2, b_2)\big) \quad = \quad \mathrm{Sat}_\mu^{v[y \to T]}\big(t(\Phi_1, b_1) \ \vee \ (t(\Phi_2, b_2) \ \wedge \ \exists\bigcirc y)\big).$$

Therefore, for μ-calculus formulas Φ_1' and Φ_2', the function $F^\exists$ can be represented by the following μ-calculus formula containing y as a free variable:

$$F_\mu^\exists(y, \Phi_1', \Phi_2') = \Phi_1' \vee (\Phi_2' \wedge \exists\bigcirc y).$$

Hence, using Table 9.2, one can see that an equivalent μ-calculus formula for the rCTL formula $\exists\diamondsuit\Phi$ with truth value $b \in \mathbb{B}_4 \setminus \{0000\}$ is the following:

$$t(\exists\diamondsuit\Phi, b) = \mu y.F_\mu^\exists(y, t(\Phi, b), \mathtt{true}).$$

Similarly, the functions $F^{\forall}$, $G^{\exists}$, and $G^{\forall}$ can be represented by the following μ-calculus formulas:

$$F^{\forall}_{\mu}(y, \Phi'_1, \Phi'_2) = \Phi'_1 \vee (\Phi'_2 \wedge \forall \bigcirc y),$$
$$G^{\exists}_{\mu}(y_1, y_2, \Phi'_1, \Phi'_2) = \exists \bigcirc y_1 \vee \Phi'_1 \vee (\Phi'_2 \wedge \exists \bigcirc y_2),$$
$$G^{\forall}_{\mu}(y_1, y_2, \Phi'_1, \Phi'_2) = \forall \bigcirc y_1 \vee \Phi'_1 \vee (\Phi'_2 \wedge \forall \bigcirc y_2).$$

Now, for an rCTL formula with temporal operators $\diamondsuit$, $\boxdot$, $\mathbf{U}$, and $\mathbf{W}$, we obtain an equivalent μ-calculus formula of linear size from the characterization of satisfaction sets given in Table 9.2 by replacing the functions and the satisfaction sets of subformulas with corresponding μ-calculus formulas. □

While it is true that every CTL formula can be transformed into an equivalent alternation-free (with alternation depth 1, as defined in [43]) μ-calculus formula, it's important to note that the constructed μ-calculus formulas for rCTL formulas typically have an alternation depth of at most 2. This limitation arises from the presence of two-depth alternation for some rCTL operators, such as $\exists \boxdot$ with value 0011 as illustrated in Table 9.2. Furthermore, as the model checking problem for μ-calculus formulas with alternation depth d can be solved in time $\mathcal{O}(n^{d+1})$ [87, 43], one can also solve rCTL model checking in cubic time by reducing it to μ-calculus model checking.

Let us conclude by mentioning that the converse of Theorem 9.4 does not hold: rCTL is strictly less expressive than the modal μ-calculus. This follows from a stronger result to be presented in Section 9.4.3.

9.2.6 rCTL Satisfiability

This section considers the satisfiability problem for rCTL, which is: given an rCTL formula Φ and a truth value $b_0 \in \mathbb{B}_4$, does there exist a Kripke structure $\mathcal{K} = (S, I, R, L)$ such that $I \subseteq \mathrm{Sat}(\Phi, b_0)$? The next theorem settles the complexity of the rCTL satisfiability problem.

Theorem 9.5. *The satisfiability problem for rCTL is* EXPTIME*-complete.*

Proof. The upper bound is obtained by translating a given rCTL formula and a given truth value into an equivalent μ-calculus formula of linear size (see Theorem 9.4) and then checking the resulting formula for satisfiability. Since the satisfiability problem for the μ-calculus (defined as expected) is EXPTIME-complete [88], rCTL satisfiability is in EXPTIME as well.

The matching lower bound already holds for CTL satisfiability (again defined as expected) [86], which, due to Lemma 9.1, reduces to rCTL satisfiability. □

Moreover, since every satisfiable formula of the μ-calculus has a model of exponential size [208], the same is true for rCTL.

Corollary 9.3. *Every satisfiable rCTL formula has a model of exponential size.*

There are satisfiable CTL formulas that have only models of at least exponential size [142].[1] Thus, the upper bound in Corollary 9.3 is tight.

Also, note that the asymptotic complexity of the rCTL satisfiability problem and the size of a model matches that of CTL.

9.2.7 rCTL Synthesis

We now turn to the problem of rCTL synthesis. As there is no explicit plant model involved (i.e., the assumption is explicitly encoded into the specification), we consider the reactive synthesis problem with rCTL specifications as given in Problem 2.2. Recall that in this setting, the set of atomic propositions $\mathtt{AP}$ is partitioned into input propositions I and output propositions O. An implementation (of the controller) is a Moore machine $\mathcal{M}$ over the inputs I and outputs O, which induces a Kripke structure $\mathcal{K}_\mathcal{M}$ as explained in Section 2.4.

We say that an implementation $\mathcal{M}$ realizes an rCTL formula Φ with at least value $b_0 \in \mathbb{B}_4$ if $V(s, \Phi) \geq b_0$ for all initial states s of $\mathcal{K}_\mathcal{M}$. Further, an rCTL formula is realizable with at least value b_0 if there is an implementation that realizes it with at least b_0. The rCTL synthesis problem is: given an rCTL formula Φ and a truth value $b_0 \in \mathbb{B}_4$, is Φ realizable with at least b_0? The next theorem settles its complexity.

Theorem 9.6. *The rCTL synthesis problem is* EXPTIME*-complete.*

Proof. The upper bound is obtained by translating a given rCTL formula and a given truth value into an equivalent μ-calculus formula of linear size (see Theorem 9.4) and then checking the resulting formula for realizability (which is defined as expected). Since the synthesis problem for the μ-calculus is EXPTIME-complete [136], rCTL synthesis is in EXPTIME as well.

The matching lower bound already holds for CTL synthesis [134], which, due to Lemma 9.1, reduces to rCTL synthesis. □

As every realizable formula of the μ-calculus is realized by an implementation of exponential size, which can be computed in exponential time [134], the same is true for rCTL.

Corollary 9.4. *If an rCTL-formula φ is realizable with at least b_0, then one can compute, in exponential time, an exponentially-sized implementation realizing φ with at least b_0.*

There are realizable CTL formulas that are only realized by implementations of exponential size: this follows from the exponential lower bound on the size of model (see the discussion below Corollary 9.3) and the fact that satisfiability can be reduced to synthesis [134]. Hence, the upper bound in Corollary 9.4 is tight.

Finally, note that Theorem 9.6 and Corollary 9.4 imply that the rCTL synthesis problem again has the same asymptotic complexity as the one for CTL.

[1] Note that the exponential lower bound is shown with respect to the *length* of the formula, i.e., the number of nodes of the syntax tree of the formula. In contrast, we measure the size of a formula by the number of distinct subformulas, i.e., the number of distinct subtrees of the syntax tree. Thus, our complexity measure might be smaller, which only strengthens the lower bound.

9.3 Review of CTL* Semantics

Similar to CTL semantics defined in Section 9.1, we re-define the semantics of CTL* using a valuation function V_{CTL^*}, which is equivalent to the one defined in Section 2.3.3.

Let $\mathcal{K}$ be a Kripke structure, Φ, Ψ two CTL* state formulas, and φ, ψ two CTL* path formulas. For a state s, the CTL* semantics $V_{\text{CTL}^*}(s, \Phi)$ is defined as the CTL semantics (see Section 9.1). For a path ρ, the semantics is analogous to the LTL semantics via a valuation function V_{CTL^*}:

$$
\begin{aligned}
V_{\text{CTL}^*}(\rho, \Phi) &= V_{\text{CTL}^*}(\rho[0], \Phi), \\
V_{\text{CTL}^*}(\rho, \varphi \vee \psi) &= \max\{V_{\text{CTL}^*}(\rho, \varphi), V_{\text{CTL}^*}(\rho, \psi)\}, \\
V_{\text{CTL}^*}(\rho, \varphi \wedge \psi) &= \min\{V_{\text{CTL}^*}(\rho, \varphi), V_{\text{CTL}^*}(\rho, \psi)\}. \\
V_{\text{CTL}^*}(\rho, \neg\varphi) &= 1 - V_{\text{CTL}^*}(\rho, \varphi). \\
V_{\text{CTL}^*}(\rho, \varphi \Rightarrow \psi) &= \begin{cases} 1 & \text{if } V_{\text{CTL}^*}(\rho, \varphi) \leq V_{\text{CTL}^*}(\rho, \psi); \text{and} \\ V_{\text{CTL}^*}(\rho, \psi) & \text{otherwise,} \end{cases} \\
V_{\text{CTL}^*}(\rho, \bigcirc \varphi) &= V_{\text{CTL}^*}(\rho[1..], \varphi). \\
V_{\text{CTL}^*}(\rho, \Diamond \varphi) &= \max_{i \geq 0} V_{\text{CTL}^*}(\rho[i..], \varphi). \\
V_{\text{CTL}^*}(\rho, \Box \varphi) &= \min_{i \geq 0} V_{\text{CTL}^*}(\rho[i..], \varphi), \\
V_{\text{CTL}^*}(\rho, \varphi \,\mathbf{U}\, \psi) &= \max_{j \geq 0} \min\{V_{\text{CTL}^*}(\rho[j..], \psi), \min_{0 \leq i < j} V_{\text{CTL}^*}(\rho[i..], \varphi)\}, \\
V_{\text{CTL}^*}(\rho, \varphi \,\mathbf{W}\, \psi) &= \min_{j \geq 0} \max\{V_{\text{CTL}^*}(\rho[j..], \varphi), \max_{0 \leq i \leq j} V_{\text{CTL}^*}(\rho[i..], \psi)\},
\end{aligned}
$$

9.4 Robust CTL*

In this section, we present the robust version of CTL*, named robust CTL*, which combines the features of rCTL and rLTL. We show that rCTL* is more expressive than both. In addition, we present an rCTL* model checking algorithm and address the rCTL* satisfiability and synthesis problems.

9.4.1 Syntax

Like CTL*, robust CTL* allows path quantifiers $\exists$ and $\forall$ to be arbitrarily nested with temporal operators. The syntax of rCTL* state formulas is the same as in rCTL and CTL*. Moreover, rCTL* path formulas are similar to rLTL formulas, with the only difference being the use of arbitrary rCTL* state formulas as atoms. Consequently, rCTL* state formulas over `AP` are formed according to the grammar

$$\Phi ::= p \mid \Phi \vee \Phi \mid \Phi \wedge \Phi \mid \neg\Phi \mid \Phi \Rightarrow \Phi \mid \exists\varphi \mid \forall\varphi,$$

where $p \in \mathtt{AP}$ and φ is a path formula. rCTL* path formulas are formed according to the grammar

$$\varphi \ ::= \ \Phi \ \mid \ \varphi \vee \psi \ \mid \ \varphi \wedge \psi \ \mid \ \neg\varphi \ \mid \ \varphi \ \Rightarrow \ \psi \ \mid \ \odot\varphi \ \mid \ \diamondsuit\varphi \ \mid \ \boxdot\varphi \ \mid \ \varphi \,\mathbf{U}\, \psi \ \mid \ \varphi \,\mathbf{W}\, \psi.$$

Again, the set of subformulas of a state formula Φ is denoted by $\mathrm{Sub}(\Phi)$ and the size of a formula is defined as the number of its syntactically different subformulas.

9.4.2 Semantics

As in CTL*, the semantics for rCTL* state and path formulas are analogous to the rCTL and rLTL semantics, respectively. In what follows, let $\mathcal{K}$ be a Kripke structure, Φ, Ψ two rCTL* state formulas, and φ, ψ two rCTL* path formulas.

For a state s, the rCTL* semantics $V(s, \Phi)$ is then the same as the rCTL semantics (see Section 9.2.2).

For a path ρ, the semantics is analogous to the rLTL semantics (cf. Tabuada and Neider [213]) via a valuation function V (note that, for notational convenience, we use the letter V both the rCTL and the rCTL* valuation function).

$$\begin{aligned}
V(\rho, \Phi) &= V(\rho[0], \Phi),\\
V(\rho, \varphi \vee \psi) &= \max\{V(\rho, \varphi), V(\rho, \psi)\},\\
V(\rho, \varphi \wedge \psi) &= \min\{V(\rho, \varphi), V(\rho, \psi)\}.\\
V(\rho, \neg\varphi) &= \overline{V(\rho, \varphi)}.\\
V(\rho, \varphi \Rightarrow \psi) &= V(\rho, \varphi) \to V(\rho, \psi).\\
V(\rho, \odot\varphi) &= V(\rho[1..], \varphi).\\
V(\rho, \diamondsuit\varphi) &= \max_{i \geq 0} V(\rho[i..], \varphi).\\
V(\rho, \boxdot\varphi) &= (b_1, b_2, b_3, b_4) \text{ where}\\
b_1 &= \min_{i \geq 0} V_1(\rho[i..], \varphi),\\
b_2 &= \max_{j \geq 0} \min_{i \geq j} V_2(\rho[i..], \varphi),\\
b_3 &= \min_{j \geq 0} \max_{i \geq j} V_3(\rho[i..], \varphi),\\
b_4 &= \max_{i \geq 0} V_4(\rho[i..], \varphi),\\
V(\rho, \varphi \,\mathbf{U}\, \psi) &= \max_{j \geq 0} \min\{V(\rho[j..], \psi), \min_{0 \leq i < j} V(\rho[i..], \varphi)\},\\
V(\rho, \varphi \,\mathbf{W}\, \psi) &= (b_1, b_2, b_3, b_4) \text{ where}\\
b_1 &= \min_{j \geq 0} \max\{V_1(\rho[j..], \varphi), \max_{0 \leq i \leq j} V_1(\rho[i..], \psi)\},\\
b_2 &= \max_{k \geq 0} \min_{j \geq k} \max\{V_2(\rho[j..], \varphi), \max_{0 \leq i \leq j} V_2(\rho[i..], \psi)\},\\
b_3 &= \min_{k \geq 0} \max_{j \geq k} \max\{V_3(\rho[j..], \varphi), \max_{0 \leq i \leq j} V_3(\rho[i..], \psi)\},\\
b_4 &= \max_{j \geq 0} \max\{V_4(\rho[j..], \varphi), \max_{0 \leq i \leq j} V_4(\rho[i..], \psi)\}.
\end{aligned}$$

Before studying the properties of rCTL*, let us illustrate the difference between rCTL and rCTL* using an example.

Example 9.3. Continuing our running example, we illustrate how the rCTL* formula $\forall(\boxdot(A \Rightarrow \exists\odot B) \Rightarrow \boxdot T)$ is different from the rCTL formula $\forall\boxdot(A \Rightarrow \exists\odot B) \Rightarrow \forall\boxdot T$ from Examples 9.1 and 9.2. Recall that $A \Rightarrow \exists\odot B$ states that the robot can reach from room A to room B in one time step, and T states that the robot can perform the task. Assume $\forall(\boxdot(A \Rightarrow \exists\odot B) \Rightarrow \boxdot T)$ evaluates to 1111. Then the formula $\boxdot(A \Rightarrow \exists\odot B) \Rightarrow \boxdot T$ must evaluate to 1111 for each path. Hence, the following holds:

- If $A \Rightarrow \exists\odot B$ holds at every state in a path ρ, then $V(\rho, \boxdot(A \Rightarrow \exists\odot B))$ evaluates to 1111. Hence, by the rCTL* semantics, $V(\rho, \boxdot T)$ must also evaluate to 1111. That means, T also holds at every state in ρ. Hence, in any path, if the robot can reach from room A to room B in one time step (i.e., never obstructed by office workers), then the robot can perform the task from every state in that path.
- Similarly, if $A \Rightarrow \exists\odot B$ holds eventually always for some path ρ, then $V(\rho, \boxdot(A \Rightarrow \exists\odot B))$ evaluates to 0111. Then, by the rCTL* semantics, $V(\rho, \boxdot T)$ evaluates to 0111 or higher. Hence, T also needs to hold eventually always in ρ. Therefore, in any path, if office workers obstruct the robot for a finite amount of time, then the robot can perform the task T after a finite amount of time.
- Similarly, if $A \Rightarrow \exists\odot B$ holds at infinitely (finitely) many states in some path ρ, then T needs to hold at infinitely (finitely) many states in ρ.

As we can see, the rCTL* formula $\forall(\boxdot(A \Rightarrow \exists\odot B) \Rightarrow \boxdot T)$ captures the robustness property for every path separately, whereas the rCTL formula $\forall\boxdot(A \Rightarrow \exists\odot B) \Rightarrow \forall\boxdot T$ captures the robustness property jointly for all paths starting from a state.

To understand the difference, let us consider the Kripke structure $\mathcal{K}$ with initial state s_0 as shown in Figure 9.1. Let us recall from Example 9.2 that the rCTL formula $\forall\boxdot(A \Rightarrow \exists\odot B) \Rightarrow \forall\boxdot T$ evaluates at state s_0 to 1111. Below, let us give a reasoning based on the valuation of the sub-formulas $A \Rightarrow \exists\odot B$ and T. Recall that the set of states satisfying the formula $A \Rightarrow \exists\odot B$ with value 1111 is $\{s_0, s_2, s_3\}$ and the set of states satisfying the formula T with value 1111 is $\{s_0, s_3\}$. As the remaining states satisfy the corresponding sub-formulas with a value of 0000, let us only focus on the states where the sub-formulas hold (with a value of 1111).

There are only two paths starting from s_0, i.e., $\rho_1 = s_0(s_2s_3)^\omega$ and $\rho_2 = s_0(s_1)^\omega$. Since $A \Rightarrow \exists\odot B$ holds at every state in the path ρ_1, we have $V(\rho_1, \boxdot(A \Rightarrow \exists\odot B)) = 1111$. Moreover, since $A \Rightarrow \exists\odot B$ holds only at the first state in the path ρ_2, we have $V(\rho_2, \boxdot(A \Rightarrow \exists\odot B)) = 0001$. Hence,

$$V(s_0, \forall\boxdot(A \Rightarrow \exists\odot B)) = \min_{i\in\{1,2\}} V(\rho_i, \boxdot(A \Rightarrow \exists\odot B)) = 0001.$$

Similarly, since T holds at infinitely many states of ρ_1 and only at the first state of ρ_2, we have $V(\rho_1, \boxdot T) = 0011$ and $V(\rho_2, \boxdot T) = 0001$. Hence, $V(s_0, \forall\boxdot T) = 0001$. Therefore, it holds that $V(s_0, \forall\boxdot(A \Rightarrow \exists\odot B) \Rightarrow \forall\boxdot T) = 1111$ according to the rCTL semantics.

Now, let us consider the rCTL* formula $\forall(\boxdot(A \Rightarrow \exists\odot B) \Rightarrow \boxdot T)$. As we have $V(\rho_1, \boxdot T) = 0011 < V(\rho_1, \boxdot(A \Rightarrow \exists\odot B))$, it holds that $V(\rho_1, \boxdot(A \Rightarrow \exists\odot B) \Rightarrow \boxdot T) = 0011$. Similarly, we have $V(\rho_2, \boxdot(A \Rightarrow \exists\odot B) \Rightarrow \boxdot T) = 1111$. Hence, according to the rCTL* semantics, we have

$$V(s_0, \forall(\boxdot(A \Rightarrow \exists\odot B) \Rightarrow \boxdot T)) = 0011.$$

Therefore, at state s_0, the rCTL formula $\forall\boxdot(A \Rightarrow \exists\odot B) \Rightarrow \forall\boxdot T$ evaluates to a different value (i.e. 1111) than the rCTL* formula $\forall(\boxdot(A \Rightarrow \exists\odot B) \Rightarrow \boxdot T)$. This is the case because both of the paths do not satisfy $\boxdot(A \Rightarrow \exists\odot B) \Rightarrow \boxdot T$ with value 1111 individually, but collectively, the state s_0 satisfies $\forall\boxdot(A \Rightarrow \exists\odot B) \Rightarrow \forall\boxdot T$. ⌟

9.4.3 Expressiveness of rCTL*

The satisfaction sets and the equivalence between two formulas in rCTL* are defined as for rCTL. As we can see, rCTL* is an extension of both rCTL and rLTL. Therefore, it subsumes both rCTL and rLTL (and hence, it also subsumes CTL and LTL). Furthermore, by Corollary 9.1, there exist rCTL formulas that are not expressible in rLTL and vice versa. In total, we obtain the following result.

Theorem 9.7. *rCTL* is more expressive than rLTL, rCTL, CTL, and LTL.*

Using the same idea as in Lemma 9.1, one can recover the CTL* semantics of a formula with no implication from the first component of the rCTL* semantics. Conversely, using the same arguments as for the analogous result for rLTL [213, Proposition 5], one can translate each rCTL* formula into four CTL* formulas that captures the four components of the rCTL* semantics. Hence, we obtain the following result.

Theorem 9.8. *CTL* and rCTL* are equally expressive.*

Proof. For any CTL* formula Φ containing no implication, let Φ_r be the rCTL* formula obtained by dotting all temporal operators in Φ. Then for any state s, it holds that $V_{\mathrm{CTL}}(s, \Phi) = V_1(s, \Phi_r)$, which is shown as the analogous result for CTL and rCTL (see the proof of Lemma 9.1). Consequently, it holds that $\mathrm{Sat}_{\mathrm{CTL}^*}(\Phi) = \mathrm{Sat}(\Phi_r, 1111)$. Furthermore, as $\Phi \Rightarrow \Psi$ is equivalent to $\Psi \vee \neg\Phi$ in CTL*, every CTL* formula can be rewritten as a formula containing no implication. Therefore, for every CTL* formula, there is an equivalent rCTL* formula with respect to the truth value 1111.

For the other direction, we define a mapping t that assigns to every rCTL* state formula Φ and truth value b an equivalent CTL* formula $t(\Phi, b)$. Furthermore, t maps every rCTL* path formula φ and a truth value b to $t(\varphi, b)$ such that for every path ρ,

$$V_{CTL^*}(\rho, t(\varphi, b)) = 1 \text{ if and only if } V(\rho, \varphi) \geq b.$$

Again, for $b = 0000$, we can define $t(\Phi, b) = \texttt{true}$ for every state formula Φ, where `true` is an arbitrary tautology of CTL*, e.g., $p \vee \neg p$.

In the following, we assume $b > 0000$. Then, for state formulas Φ and Ψ, the mapping t is defined inductively as follows:

$$\begin{aligned} t(p, b) &= p \text{ for any } p \in \mathtt{AP}, \\ t(\Phi \vee \Psi, b) &= t(\Phi, b) \vee t(\Psi, b), \\ t(\Phi \wedge \Psi, b) &= t(\Phi, b) \wedge t(\Psi, b), \\ t(\neg\Phi, b) &= \neg t(\Phi, 1111). \end{aligned}$$

Moreover, we define

$$t(\Phi \Rightarrow \Psi, 1111) = \bigwedge_{b>0000} t(\Psi, b) \vee \neg t(\Phi, b),$$

and

$$t(\Phi \Rightarrow \Psi, b) = t(\Phi \Rightarrow \Psi, 1111) \vee t(\Psi, b)$$

for each $b \leq 0111$.

For Boolean combinations of path formulas, the mapping t can be defined analogously as for state formulas. For path formulas φ and ψ with temporal operators, t is defined as follows:

$$\begin{aligned} t(\exists\varphi, b) &= \exists t(\varphi, b), \\ t(\forall\varphi, b) &= \forall t(\varphi, b), \\ t(\mathbin{\odot}\varphi, b) &= \mathbin{\bigcirc} t(\varphi, b), \\ t(\mathbin{\diamondsuit}\varphi, b) &= \mathbin{\diamondsuit} t(\psi, b), \\ t(\mathbin{\boxdot}\varphi, 1111) &= \mathbin{\Box} t(\varphi, 1111), \\ t(\mathbin{\boxdot}\varphi, 0111) &= \mathbin{\diamondsuit\Box} t(\varphi, 0111), \\ t(\mathbin{\boxdot}\varphi, 0011) &= \mathbin{\Box\diamondsuit} t(\varphi, 0011), \\ t(\mathbin{\boxdot}\varphi, 0001) &= \mathbin{\diamondsuit} t(\varphi, 0001), \\ t(\varphi \mathbin{\mathbf{U}} \psi, b) &= t(\varphi, b) \mathbin{\mathbf{U}} t(\psi, b), \\ t(\varphi \mathbin{\mathbf{W}} \psi, 1111) &= t(\varphi, 1111) \mathbin{\mathbf{W}} t(\psi, 1111), \\ t(\varphi \mathbin{\mathbf{W}} \psi, 0111) &= \mathbin{\diamondsuit\Box} t(\varphi, 0111) \vee \mathbin{\diamondsuit} t(\psi, 0111), \\ t(\varphi \mathbin{\mathbf{W}} \psi, 0011) &= \mathbin{\Box\diamondsuit} t(\varphi, 0011) \vee \mathbin{\diamondsuit} t(\psi, 0011), \\ t(\varphi \mathbin{\mathbf{W}} \psi, 0001) &= \mathbin{\diamondsuit} t(\varphi, 0001) \vee \mathbin{\diamondsuit} t(\psi, 0001). \end{aligned}$$

A structural induction shows that the translation t has the desired property. □

Although we do not require the result, let us just mention that Theorem 9.8 also gives a translation of rCTL* into the modal μ-calculus: rCTL* can be translated into CTL*, which can be translated into the modal μ-calculus [71]. Conversely, the modal μ-calculus is strictly more expressive than CTL* (see, e.g., Demri, Goranko, and Lange [79, Chapter 10]). Hence, it is also strictly more expressive than rCTL* (due to Theorem 9.8) and rCTL (which is a fragment of rCTL*).

9.4.4 rCTL* Model Checking

The model checking problem for rCTL* is analogous to that of rCTL: for a given Kripke structure $\mathcal{K} = (S, I, R, L)$, an rCTL* formula Φ and a truth value $b_0 \in \mathbb{B}_4$, does $V(s, \Phi) \geq b_0$ hold for all initial states $s \in I$? One way of solving the rCTL* model checking problem is by applying the translation from rCTL* into CTL* (see Theorem 9.8) and then apply a CTL* model checking algorithm.

Here, we will present an alternative approach via a combination of rCTL and rLTL model checking. This approach is analogous to the classical CTL* model checking algorithm, which is a combination of CTL and LTL model checking. In practice, the choice for one algorithm over the other depends on whether one wants to apply an CTL* model checker or an rLTL model checker.

As in rCTL, for the rCTL* model checking, we use the characterization of the satisfaction sets. $\mathrm{Sat}(\Phi, b)$ can be computed using Table 9.2 for every state formula Φ which is either an atomic proposition or can be expressed as a Boolean combination (conjunction, negation, etc.) of two subformulas. Otherwise, we use an rLTL model checking algorithm to compute $\mathrm{Sat}(\Phi, b)$ for a state formula starting with a path quantifier.

Let us first go through the basic concepts of rLTL and its model checking algorithm. As we have described earlier, rCTL* is an extension of rLTL. Both rCTL* path formulas and rLTL formulas are defined using almost the same syntax, with the only difference being the use of state formulas as atoms in rCTL*. Moreover, the valuation V for rLTL formulas is defined the same way as it is defined for rCTL* path formulas. The rLTL model checking problem is: given a Kripke structure $\mathcal{K}$, an rLTL formula φ, and a truth value $b_0 \in \mathbb{B}_4$, determine whether for all initial states s and all $\rho \in \mathrm{paths}(s)$, it holds that $V(\rho, \varphi) \geq b_0$.[2] To solve the rLTL model checking problem, Tabuada and Neider [213] have provided an algorithm to compute a generalized Büchi automaton recognizing all traces over the alphabet $2^{\mathtt{AP}}$ satisfying a given formula with a value $b \in B$ for a given set $B \subseteq \mathbb{B}_4$, as formalized below.

Lemma 9.3 (Tabuada and Neider [213])**.** *Given an rLTL formula φ, and a set of truth values $B \subseteq \mathbb{B}_4$, one can construct a generalized Büchi automaton $A_{\varphi,B}$ with $\mathcal{O}(5^{|\varphi|})$ states and $\mathcal{O}(|\varphi|)$ accepting sets that recognizes all paths ρ such that $V(\rho, \varphi) \in B$.*

One can now solve the rLTL model checking problem by first translating $\mathcal{K}$ into a Büchi automaton $A_{\mathcal{K}}$ accepting exactly the traces labeling the paths of $\mathcal{K}$ starting in an initial state. Then, one determines whether $L(A_{\mathcal{K}}) \cap L(A_{\varphi,\{b \in \mathbb{B}_4 \mid b < b_0\}})$ is empty.

Coming back to computing $\mathrm{Sat}(\Phi, b)$ for Φ starting with a path quantifier, let us consider $\Phi = \forall\varphi$. Observe that $s \in \mathrm{Sat}(\forall\varphi, b)$ if and only if $V(s, \forall\varphi) \geq b$. Further, $V(s, \forall\varphi) \geq b$ if and only if $V(\rho, \varphi) \geq b$ for all $\rho \in \mathrm{paths}(s)$. The basic idea is now to replace all maximal proper state subformulas Ψ of φ by fresh atomic propositions a_Ψ and use the rLTL model checking algorithm to compute all the states from which all paths satisfy the rLTL formula φ with value at least b. However, we need to make a minor modification in the construction of the Büchi automaton of Lemma 9.3 such

[2] Actually, the original definition by Tabuada and Neider is slightly more general [213].

that for each a_Ψ, it holds that $V(s, a_\Psi) \geq b$ whenever $s \in \mathrm{Sat}(\Psi, b)$ and $V(s, a_\Psi) < b$ whenever $s \notin \mathrm{Sat}(\Psi, b)$. This can be done by initializing these atomic propositions with the required truth value.

Similarly, we can compute $\mathrm{Sat}(\exists\varphi, b)$ by the rLTL model checking algorithm using the observation that $s \notin \mathrm{Sat}(\exists\varphi, b)$ if and only if $V(\rho, \varphi) < b$ for all $\rho \in \mathrm{paths}(s)$.

Now, one can solve the rCTL* model checking problem using Algorithm 9.1. However, the time complexity of the algorithm is not the same as in rCTL since the computation of Sat uses the rLTL model checking algorithm, which takes exponential time in the size of the formula and the Kripke structure (Tabuada and Neider [213]). Hence, the time complexity of the rCTL* model checking algorithm is dominated by the time complexity of the rLTL model checking algorithm.

Altogether, our algorithm runs in polynomial space (as rLTL model checking is in PSPACE [213]). A matching lower bound already holds for CTL* [89].

Theorem 9.9. *The rCTL* model checking problem is* PSPACE*-complete.*

The CTL* model checking problem is also PSPACE-complete [89]. Hence, both the CTL* and the rCTL* model checking problem have the same asymptotic complexity.

9.4.5 rCTL* Satisfiability

This section considers the satisfiability problem for rCTL*, which is: for a given rCTL* formula Φ and truth value $b_0 \in \mathbb{B}_4$, does there exist a Kripke structure $\mathcal{K} = (S, I, R, L)$ such that $I \subseteq \mathrm{Sat}(\Phi, b_0)$?

Theorem 9.10. *The satisfiability problem for rCTL* is* 2EXPTIME*-complete.*

Proof. Both the upper and the lower bound follow immediately from Theorem 9.8 and the fact that CTL* satisfiability (which is defined as expected) is 2EXPTIME-complete [88]. The linear translation from rCTL* to CTL* yields the upper bound while the linear translation from CTL* to rCTL* yields the lower bound. □

Furthermore, as every satisfiable CTL* formula has a model of doubly-exponential size (see, e.g., Demri, Goranko, and Lange [79, Chapter 15]), the same is true for rCTL*.

Corollary 9.5. *Every satisfiable rCTL* formula has a model of doubly-exponential size.*

There are satisfiable CTL* formulas that have only models of doubly-exponential size (see, e.g., Demri, Goranko, and Lange [79, Chapter 15]). Hence, the upper bound in Corollary 9.5 is tight.

Also, note again that the asymptotic complexity of the rCTL* satisfiability problem and the size of a model matches that of CTL*.

9.4.6 rCTL* Synthesis

The notions of an implementation realizing an rCTL formula with at last value b_0 and an rCTL formula being realizable can be generalized to rCTL*. Then, the rCTL* synthesis problem is defined analogously to the rCTL synthesis problem: given an rCTL* formula Φ and a truth value $b_0 \in \mathbb{B}_4$, is Φ realizable with value at least b_0?

Theorem 9.11. *The rCTL* synthesis problem is* 2EXPTIME*-complete.*

Proof. The lower bound again follows immediately from CTL* synthesis (again defined as expected) being 2EXPTIME-complete [134] and the fact that the CTL* semantics is a special case of rCTL* (see Theorem 9.8).

Here, the upper bound follows from the converse translation, i.e., given an rCTL* formula Φ and a truth value $b_0 \in \mathbb{B}_4$, Theorem 9.8 allows us to construct (in linear time) a CTL* formula $t(\Phi, b_0)$ that is equivalent to Φ with respect to b_0: an implementation realizes Φ with at least b_0 if and only if it realizes $t(\Phi, b_0)$. As CTL* synthesis is in 2EXPTIME [134], we obtain the desired upper bound. □

As every realizable CTL* formula is realized by an implementation of doubly-exponential size [134], which can be computed in doubly-exponential time, the same is true for rCTL*.

Corollary 9.6. *If an rCTL* formula Φ is realizable with a value at least b_0, then one can compute, in doubly-exponential time, a doubly-exponentially-sized implementation realizing φ with at least b_0.*

There are realizable LTL formulas (and therefore CTL* formulas, as LTL is a fragment of CTL*) that are only realized by implementations with at least doubly-exponentially many states [179]. Hence, the doubly-exponential upper bound in Corollary 9.6 is tight.

Perhaps unsurprisingly at this point, the asymptotic complexity of the rCTL* synthesis problem and the tight bound on the size of an implementation realizing an rCTL* specification match those of CTL*.

9.5 Related Work

Numerous efforts have sought to formalize robustness in cyber-physical systems using formal methods. This section summarizes the most relevant frameworks.

Bloem et al. [36] introduced a unified framework combining two quantitative notions of robustness: (i) safety robustness, which bounds the ratio between violated assumptions and guarantees by a parameter k (so-called k-robustness) based on designer-provided error functions, and (ii) liveness robustness, which compares the number of violated assumptions and guarantees in specifications of the form $\bigwedge_{i \in I} \Diamond\Box p_i \implies \bigwedge_{j \in J} \Diamond\Box q_j$. While our semantics distinguishes between different violation patterns, it does not differentiate between one or multiple violated assumptions, and we do not separate safety and liveness properties.

In follow-up work, Bloem et al. [38] proposed a distinct robust synthesis framework based on whether a system still satisfies guarantees when inputs are hidden, misread, or

when assumptions fail finitely or infinitely often. Our semantics similarly allows weaker guarantees under weaker assumptions but distinguishes only between zero, finite, and infinite violations rather than counting them, making direct comparison infeasible.

Rodionova et al. [186] connected LTL/MTL semantics to Linear Time-Invariant filtering, interpreting logical operations as max/min and using filtering kernels to express weaker or stronger semantics. Here, designers must manually select kernels to capture robustness, unlike our approach, which requires only the desired specification.

In software systems, Zhang et al. [222] define robustness as the largest set of environmental deviations under which a system still satisfies a property. While they focus on computing such sets, some temporal deviations could be captured by our semantics. Related works by Chaudhuri et al. [67] and Majumdar et al. [149] study continuity of program behaviors—robustness in the Turing model, not applicable to reactive systems as considered here.

Several works examine robustness for real-valued, continuous-time signals [90, 81, 8, 7, 155], defining "time robustness" as the time needed for a formula's truth value to change. While conceptually similar to our temporal interpretation, these notions rely on quantitative signal values, whereas our semantics operates over discrete time and Boolean signals. Hence, these works are orthogonal and complementary to ours.

Almagor et al. [9] introduced multi-valued extensions of LTL, called LTL$[\mathcal{F}]$ and LTL$^{disc}[\mathcal{D}]$, where satisfaction values in $[0, 1]$ quantify quality. Although related to our use of many-valued semantics, their approach depends on user-defined functions $\mathcal{F}$ or $\mathcal{D}$, and the choice of connectives (e.g., Gödel conjunction) is not justified. In contrast, our semantics intrinsically defines robustness and explicitly motivates all logical design choices.

Finally, our work builds on rLTL [213], which has inspired subsequent research on model checking [18, 19, 20], runtime monitoring [153, 153], and synthesis [165], as well as robust variants of Prompt-LTL, LDL, probabilistic, and alternating-time logics [168, 169, 226, 157].

Back Matter

Chapter 10

Conclusion and Future Outlook

In this thesis, we developed a collection of frameworks and algorithms to compute or utilize *permissive assumptions*—on the environment, on other system components, or on the plant model—in order to make logical control synthesis more flexible, scalable, and robust. This chapter summarizes the main contributions of each chapter and outlines possible directions for future research.

In Chapter 3, we introduced *adequately permissive assumptions* (APAs) on the environment for two-player graph games. While existing approaches compute assumptions that are sufficient (i.e., under which the controller can ensure the specification) and implementable (i.e., realizable by the environment), they are often overly restrictive. In contrast, APAs additionally ensure *permissiveness*, meaning they preserve all cooperatively winning behaviors of the environment. We showed that APAs can be represented using simple local edge constraints and provided a polynomial-time algorithm to compute them for parity games. Our evaluation on standard benchmarks demonstrated the effectiveness of APAs compared to existing methods for computing environment assumptions.

In Chapters 4 and 5, we extended the concept of permissive assumptions to multi-player graph games where each player has its own objective. For such games, we formalized a permissive assume-guarantee contract for each player encoded as a *contracted specification* that allows each player to satisfy their specification locally while ensuring the global satisfaction of all players' objectives. In Chapter 4, we considered the setting with rational players that prioritize satisfying their own objective while attempting to falsify the objectives of others when possible. Using APAs to over-approximate accidental cooperation among players, we developed a negotiation-based semi-algorithm that computes *rationally contracted specifications*. Additionally, we proposed an efficient but incomplete variant of this semi-algorithm that relies on an over-approximation of APAs.

In Chapter 5, we considered the setting with cooperative players that aim to jointly satisfy their objectives. We extended the concept of APAs to *contracted strategy masks* (CSMs) that represent permissive cooperative behaviors of all players. Utilizing CSMs, we developed a polynomial-time negotiation-based algorithm that computes *cooperatively contracted specifications*. Unlike the rational setting, the cooperative negotiation algorithm is both sound and complete. We demonstrated the effectiveness of our approach

on several standard benchmarks for reactive synthesis.

In Chapter 6, we applied the concept of permissive contract represented by CSMs to the domain of human-robot interaction (HRI). We proposed a novel framework in which a robot pursuing an LTL task dynamically adapts its behavior to the human's action within the limits of the permissive contract. The robot provides feedback to the human only when necessary to ensure the satisfaction of the assumptions encoded in the contract. We validated our framework both in simulation on Overcooked-AI benchmarks and on a Franka robotic arm, demonstrating the effectiveness of our approach in generating flexible and adaptive robot behaviors.

In Chapter 7, marking the beginning of Part B, we shifted focus from assumptions on other logical components to assumptions on the physical plant model. We introduced *persistent live groups*, a new class of assumptions that capture rich liveness properties of low-level continuous controllers without requiring explicit state-space discretization. By integrating these assumptions into the plant model to be utilized in the high-level synthesis games, we developed a new two-layered framework to synthesize feedback controllers for continuous system that satisfy LTL specifications. This framework allows the hybrid controller to seamlessly switch between low-level continuous controllers based on the current context, ensuring correctness and immediate reaction to environmental changes. Our algorithm rewrites the general problem into a parity game augmented with persistent live groups, which we solve using a new quasi-polynomial time algorithm.

In Chapter 8, we addressed the challenge of scalability to large plant models in logical controller synthesis. We introduced a learning-based framework for *universal controller synthesis*. Such universal controllers are synthesized independently of specific plant models, where decisions are conditioned on *prophecies*—assumptions on the behaviors of plant models—that enable correct controller behavior across a range of plants. While previous work on universal controllers relied on complex automata-based prophecies, our approach learns concise and human-readable prophecies expressed in CTL from a set of representative nominal plants. Our experiments showed that this learning-based approach outperforms existing universal synthesis methods whenever the learned prophecies capture the essential structure of the plant class.

Finally, in Chapter 9, we introduced robust CTL (rCTL) and robust CTL* (rCTL*) to reason about specifications under partial violations of branching-time assumptions. Inspired by robust LTL, our semantics use multi-valued semantics to capture different degree of violation. We showed that rCTL is strictly more expressive than CTL, while rCTL* is as expressive as CTL*. We proved that robustness comes at no additional complexity cost: model checking, satisfiability, and synthesis remain as hard as for classical CTL and CTL*.

Future Work

Negotiation framework for partial-observation settings. Our negotiation-based approaches in Chapters 4 and 5 assume that all players have perfect information about the game state. An interesting direction for future research is to extend our negotiation

framework to partial observation settings, where each player has only limited knowledge about the current state of the game. While Section 5.4 provides initial ideas on how to utilize computed contract in partial observation settings, a comprehensive framework that integrates partial observation into the negotiation process itself remains an open challenge. This extension would significantly enhance the applicability of our negotiation-based synthesis methods to real-world multi-agent systems, including human-robot interaction, where agents often operate under uncertainty and incomplete information.

Seamless reactivity of logical control for probabilistic systems. Chapter 7 introduced a two-layered framework for synthesizing feedback controllers for continuous systems that enable seamless switching between low-level continuous controllers based on the current context. A promising direction for future research is to extend this framework to probabilistic systems, where the plant dynamics are subject to stochastic disturbances or uncertainties. Incorporating probabilistic models into the synthesis process would enable the design of controllers that can provide probabilistic guarantees on the satisfaction of LTL specifications.

Universal live controller with learned prophecies. While Chapter 8 focused on universal controllers for safety specifications, extending this framework to handle liveness specifications presents an exciting avenue for future research. Capturing liveness with prophecies is challenging, as liveness requires ensuring that certain events eventually occur across all admissible plant behaviors. Successfully addressing this challenge would broaden the applicability of universal controllers to general LTL synthesis problems.

Quality as dual of robustness in branching-time logics. Tabuada and Neider [213] described *quality* as the dual of robustness. To illustrate this point, consider the CTL formula $\Diamond\Phi \Rightarrow \Diamond\Psi$. According to the motto "more is better" we would prefer the controller to guarantee the stronger property $\Box\Diamond\Psi$ whenever the environment satisfies the stronger property $\Box\Diamond\Psi$. And similarly, $\Diamond\Box\Phi$ should lead to $\Diamond\Box\Psi$ and $\Box\Phi$ should lead to $\Box\Psi$. This raises the natural question of whether an extension of CTL or CTL* can simultaneously capture both robustness and quality.

Bibliography

[1] 2015 Seville Airbus A400M crash - Wikipedia — en.wikipedia.org. https://en.wikipedia.org/wiki/2015_Seville_Airbus_A400M_crash? [Accessed 08-08-2025].

[2] Cosmo-contracted-strategy-mask-negotiation. https://github.com/satya2009rta/cosmo.

[3] How terrible software design decisions led to uber's deadly 2018 crash. ars technica. https://arstechnica.com/cars/2019/11/how-terrible-software-design-decisions-led-to-ubers-deadly-2018-crash/. [Accessed 08-08-2025].

[4] Simpa-sufficient-implementable-permissive-assumption. https://gitlab.mpi-sws.org/kmallik/simpa.

[5] Tesla settles lawsuit over man's death in a crash involving its semi-autonomous driving software — apnews.com. https://apnews.com/article/tesla-autopilot-lawsuit-settlement-f4c19ce05e17669de212fc262265e351. [Accessed 08-08-2025].

[6] Martín Abadi and Leslie Lamport. The existence of refinement mappings. *Theor. Comput. Sci.*, 82(2):253–284, 1991.

[7] Houssam Abbas, Yash Vardhan Pant, and Rahul Mangharam. Temporal logic robustness for general signal classes. In Necmiye Ozay and Pavithra 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 45–56. ACM, 2019.

[8] Takumi Akazaki and Ichiro Hasuo. Time robustness in MTL and expressivity in hybrid system falsification. In Daniel Kroening and Corina S. Pasareanu, editors, *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part II*, volume 9207 of *Lecture Notes in Computer Science*, pages 356–374. Springer, 2015.

[9] Shaull Almagor, Udi Boker, and Orna Kupferman. Formally reasoning about quality. *J. ACM*, 63(3), June 2016.

[10] R. Alur, S. Moarref, and U. Topcu. Compositional and symbolic synthesis of reactive controllers for multi-agent systems. *Inf. Comput.*, 261:616–633, 2018.

[11] Ashwani Anand, Kaushik Mallik, Satya Prakash Nayak, and Anne-Kathrin Schmuck. Computing adequately permissive assumptions for synthesis. In Sriram Sankaranarayanan and Natasha Sharygina, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 29th International Conference, TACAS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Paris, France, April 22-27, 2023, Proceedings, Part II*, volume 13994 of *Lecture Notes in Computer Science*, pages 211–228. Springer, 2023.

[12] Ashwani Anand, Satya Prakash Nayak, Ritam Raha, Irmak Sağlam, and Anne-Kathrin Schmuck. Quantitative strategy templates. In Meenakshi D´Souza, Raghavan Komondoor, and B. Srivathsan, editors, *Automated Technology for Verification and Analysis*, pages 65–86, Cham, 2026. Springer Nature Switzerland.

[13] Ashwani Anand, Satya Prakash Nayak, Ritam Raha, and Anne-Kathrin Schmuck. Follow the stars: Dynamic ω-regular shielding of learned policies. *CoRR*, abs/2505.14689, 2025.

[14] Ashwani Anand, Satya Prakash Nayak, and Anne-Kathrin Schmuck. Synthesizing permissive winning strategy templates for parity games. In *CAV (1)*, volume 13964 of *Lecture Notes in Computer Science*, pages 436–458. Springer, 2023.

[15] Ashwani Anand, Satya Prakash Nayak, and Anne-Kathrin Schmuck. Strategy templates - robust certified interfaces for interacting systems. In S. Akshay, Aina Niemetz, and Sriram Sankaranarayanan, editors, *Automated Technology for Verification and Analysis - 22nd International Symposium, ATVA 2024, Kyoto, Japan, October 21-25, 2024, Proceedings, Part I*, volume 15054 of *Lecture Notes in Computer Science*, pages 22–41. Springer, 2024.

[16] Ashwani Anand, Anne-Kathrin Schmuck, and Satya Prakash Nayak. Contract-based distributed logical controller synthesis. In Erika Ábrahám and Manuel Mazo Jr., editors, *Proceedings of the 27th ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2024, Hong Kong SAR, China, May 14-16, 2024*, pages 11:1–11:11. ACM, 2024.

[17] F. Ancona and A. Bressan. Patchy vector fields and asymptotic stabilization. *ESAIM: COCV*, 4:445–471, 1999.

[18] Tzanis Anevlavis, Daniel Neider, Matthew Philippe, and Paulo Tabuada. Evrostos: the rLTL verifier. In Necmiye Ozay and Pavithra 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 218–223. ACM, 2019.

[19] Tzanis Anevlavis, Matthew Philippe, Daniel Neider, and Paulo Tabuada. Verifying rLTL formulas: now faster than ever before! In Empty, editor, *57th IEEE Conference on Decision and Control, CDC 2018, Miami, FL, USA, December 17-19, 2018*, pages 1556–1561. IEEE, 2018.

[20] Tzanis Anevlavis, Matthew Philippe, Daniel Neider, and Paulo Tabuada. Being correct is not enough: Efficient verification using robust linear temporal logic. *ACM Trans. Comput. Log.*, 23(2):8:1–8:39, 2022.

[21] W. Alejandro Apaza-Perez and Antoine Girard. Compositional synthesis of symbolic controllers for attractivity specifications. In *2021 60th IEEE Conference on Decision and Control (CDC)*, pages 2008–2013, 2021.

[22] Krzysztof R. Apt and Erich Grädel, editors. *Lectures in Game Theory for Computer Scientists*. Cambridge University Press, 2011.

[23] André Arnold and Damian Niwinski. *Rudiments of μ-calculus*. Elsevier, 2001.

[24] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

[25] Mrudula Balachander, Emmanuel Filiot, and Jean-François Raskin. LTL reactive synthesis with a few hints. In *TACAS (2)*, volume 13994 of *Lecture Notes in Computer Science*, pages 309–328. Springer, 2023.

[26] Ayca Balkan, Moshe Vardi, and Paulo Tabuada. Mode-target games: Reactive synthesis for control applications. *IEEE Transactions on Automatic Control*, 63(1):196–202, 2017.

[27] Tamajit Banerjee, Rupak Majumdar, Kaushik Mallik, Anne-Kathrin Schmuck, and Sadegh Soudjani. Fast symbolic algorithms for omega-regular games under strong transition fairness. *TheoretiCS*, 2, 2023. https://www.mpi-sws.org/tr/2020-007.pdf.

[28] Howard Barringer, Dimitra Giannakopoulou, and Corina S Pasareanu. Proof rules for automated compositional verification through learning. In *SAVBS 2003*, 2003.

[29] Nicolas Basset, Jean-François Raskin, and Ocan Sankur. Admissible strategies in timed games. In Luca Aceto, Giorgio Bacci, Giovanni Bacci, Anna Ingólfsdóttir, Axel Legay, and Radu Mardare, editors, *Models, Algorithms, Logics and Tools - Essays Dedicated to Kim Guldstrand Larsen on the Occasion of His 60th Birthday*, volume 10460 of *Lecture Notes in Computer Science*, pages 403–425. Springer, 2017.

[30] Calin Belta and Sadra Sadraddini. Formal methods for control synthesis: An optimization perspective. *Annual Review of Control, Robotics, and Autonomous Systems*, 2:115–140, 2019.

[31] A. Benveniste, B. Caillaud, D. Nickovic, R. Passerone, J.-B. Raclet, P. Reinkemeier, A.L. Sangiovanni-Vincentelli, W. Damm, T.A. Henzinger, and K.G. Larsen. Contracts for system design. *Found. Trends in EDA*, 12(2-3):124–400, 2018.

[32] Julien Bernet, David Janin, and Igor Walukiewicz. Permissive strategies: from parity games to safety games. *RAIRO Theor. Informatics Appl.*, 36(3):261–275, 2002.

[33] Dietmar Berwanger. Admissibility in infinite games. In Wolfgang Thomas and Pascal Weil, editors, *STACS 2007, 24th Annual Symposium on Theoretical Aspects of Computer Science, Aachen, Germany, February 22-24, 2007, Proceedings*, volume 4393 of *Lecture Notes in Computer Science*, pages 188–199. Springer, 2007.

[34] Raven Beutner and Bernd Finkbeiner. Prophecy variables for hyperproperty verification. In *35th IEEE Computer Security Foundations Symposium, CSF 2022, Haifa, Israel, August 7-10, 2022*, pages 471–485. IEEE, 2022.

[35] R. Bloem, K. Chatterjee, S. Jacobs, and R. Könighofer. Assume-guarantee synthesis for concurrent reactive programs with partial information. In *TACAS*, pages 517–532. Springer, 2015.

[36] Roderick Bloem, Krishnendu Chatterjee, Karin Greimel, Thomas A. Henzinger, Georg Hofferek, Barbara Jobstmann, Bettina Könighofer, and Robert Könighofer. Synthesizing robust systems. *Acta Informatica*, 51(3):193–220, 2014.

[37] Roderick Bloem, Krishnendu Chatterjee, and Barbara Jobstmann. Graph games and reactive synthesis. In *Handbook of Model Checking*. Springer, 2018.

[38] Roderick Bloem, Hana Chockler, Masoud Ebrahimi, and Ofer Strichman. Synthesizing reactive systems using robustness and recovery specifications. In Clark W. Barrett and Jin Yang, editors, *2019 Formal Methods in Computer Aided Design, FMCAD 2019, San Jose, CA, USA, October 22-25, 2019*, pages 147–151. IEEE, 2019.

[39] V. D. Blondel and J. N. Tsitsiklis. A survey of computational complexity results in systems and control. *Automatica*, 36(9):1249–1274, 2000.

[40] Benjamin Bordais, Daniel Neider, and Rajarshi Roy. Learning branching-time properties in CTL and ATL via constraint solving. In *FM (1)*, volume 14933 of *Lecture Notes in Computer Science*, pages 304–323. Springer, 2024.

[41] Patricia Bouyer, Nicolas Markey, Jörg Olschewski, and Michael Ummels. Measuring permissiveness in parity games: Mean-payoff parity games revisited. In Tevfik Bultan and Pao-Ann Hsiung, editors, *Automated Technology for Verification and Analysis, 9th International Symposium, ATVA 2011, Taipei, Taiwan, October 11-14, 2011. Proceedings*, volume 6996 of *Lecture Notes in Computer Science*, pages 135–149. Springer, 2011.

[42] Stephen Boyd, Laurent El Ghaoui, Eric Feron, and Venkataramanan Balakrishnan. *Linear matrix inequalities in system and control theory*. SIAM, 1994.

[43] Julian Bradfield and Igor Walukiewicz. *The mu-calculus and Model Checking*, pages 871–919. Springer International Publishing, Cham, 2018.

[44] R. Brenguier, J.-F. Raskin, and O. Sankur. Assume-admissible synthesis. *Acta Informatica*, 54(1):41–83, Feb 2017.

[45] Léonard Brice, Jean-François Raskin, and Marie van den Bogaard. Subgame-perfect equilibria in mean-payoff games. In Serge Haddad and Daniele Varacca, editors, *32nd International Conference on Concurrency Theory, CONCUR 2021, August 24-27, 2021, Virtual Conference*, volume 203 of *LIPIcs*, pages 8:1–8:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[46] Léonard Brice, Jean-François Raskin, and Marie van den Bogaard. Rational verification for nash and subgame-perfect equilibria in graph games. In Jérôme Leroux, Sylvain Lombardy, and David Peleg, editors, *48th International Symposium on Mathematical Foundations of Computer Science, MFCS 2023, August 28 to September 1, 2023, Bordeaux, France*, volume 272 of *LIPIcs*, pages 26:1–26:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[47] Thomas Brihaye, Véronique Bruyère, and Julie De Pril. On equilibria in quantitative games with reachability/safety objectives. *Theory Comput. Syst.*, 54(2):150–189, 2014.

[48] Véronique Bruyère, Noémie Meunier, and Jean-François Raskin. Secure equilibria in weighted games. In Thomas A. Henzinger and Dale Miller, editors, *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, pages 26:1–26:26. ACM, 2014.

[49] Véronique Bruyère, Guillermo A. Pérez, Jean-François Raskin, and Clément Tamines. Partial solvers for generalized parity games. In Emmanuel Filiot, Raphaël M. Jungers, and Igor Potapov, editors, *Reachability Problems - 13th International Conference, RP 2019, Brussels, Belgium, September 11-13, 2019, Proceedings*, volume 11674 of *Lecture Notes in Computer Science*, pages 63–78. Springer, 2019.

[50] Véronique Bruyère, Stéphane Le Roux, Arno Pauly, and Jean-François Raskin. On the existence of weak subgame perfect equilibria. *Inf. Comput.*, 276:104553, 2021.

[51] Oscar Lindvall Bulancea, Petter Nilsson, and Necmiye Ozay. Nonuniform abstractions, refinement and controller synthesis with novel BDD encodings. *IFAC-PapersOnLine*, 51(16):19–24, 2018.

[52] Cristian S. Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasipolynomial time. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 252–263. ACM, 2017.

[53] Micah Carroll, Rohin Shah, Mark K Ho, Tom Griffiths, Sanjit Seshia, Pieter Abbeel, and Anca Dragan. On the utility of learning about humans for human-ai coordination. *Advances in neural information processing systems*, 32, 2019.

[54] Christos G Cassandras and Stéphane Lafortune. *Introduction to Discrete Event Systems*. Springer Nature, 2021.

[55] Davide G Cavezza, Dalal Alrajeh, and András György. Minimal assumptions refinement for realizable specifications. In *Formal Methods in Software Engineering*, 2020.

[56] C.Belta, B. Yordanov, and E.A. Gol. *Formal Methods for Discrete-Time Dynamical Systems*, volume 15 of *Studies in Systems, Decision and Control*. Springer International Publishing, 2017.

[57] K. Chatterjee, T.A. Henzinger, and B. Jobstmann. Environment assumptions for synthesis. In *CONCUR*, pages 147–161. Springer, 2008.

[58] Krishnendu Chatterjee, Laurent Doyen, Emmanuel Filiot, and Jean-François Raskin. Doomsday equilibria for omega-regular games. *Inf. Comput.*, 254:296–315, 2017.

[59] Krishnendu Chatterjee, Laurent Doyen, Thomas A Henzinger, and Jean-François Raskin. Algorithms for omega-regular games with imperfect information. *Logical Methods in Computer Science*, 3, 2007.

[60] Krishnendu Chatterjee and Thomas A. Henzinger. Assume-guarantee synthesis. In *TACAS*, pages 261–275. Springer, 2007.

[61] Krishnendu Chatterjee, Thomas A Henzinger, Barbara Jobstmann, and Arjun Radhakrishna. Gist: A solver for probabilistic games. In *CAV*, 2010.

[62] Krishnendu Chatterjee, Thomas A Henzinger, and Marcin Jurdziński. Games with secure equilibria. *Theoretical Computer Science*, 365(1-2):67–82, 2006.

[63] Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. Generalized parity games. In Helmut Seidl, editor, *Foundations of Software Science and Computational Structures, 10th International Conference, FOSSACS 2007, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2007, Braga, Portugal, March 24-April 1, 2007, Proceedings*, volume 4423 of *Lecture Notes in Computer Science*, pages 153–167. Springer, 2007.

[64] Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. Algorithms for Büchi games. *arXiv*, abs/0805.2620, 2008.

[65] Krishnendu Chatterjee, Florian Horn, and Christof Löding. Obliging games. In *Int. Conference on Concurrency Theory*, pages 284–296. Springer, 2010.

[66] Krishnendu Chatterjee and Vishwanath Raman. Assume-guarantee synthesis for digital contract signing. *Formal Aspects Comput.*, 26(4):825–859, 2014.

[67] Swarat Chaudhuri, Sumit Gulwani, and Roberto Lublinerman. Continuity analysis of programs. In Manuel V. Hermenegildo and Jens Palsberg, editors, *Proceedings of the 37th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2010, Madrid, Spain, January 17-23, 2010*, pages 57–70. ACM, 2010.

[68] Andrew Clark. Verification and synthesis of control barrier functions. In *2021 60th IEEE Conference on Decision and Control (CDC)*, pages 6105–6112, 2021.

[69] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In Dexter Kozen, editor, *Logics of Programs, Workshop, Yorktown Heights, New York, USA, May 1981*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 1981.

[70] Edmund M. Clarke, E. Allen Emerson, and A. Prasad Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263, 1986.

[71] Edmund M. Clarke, Orna Grumberg, Daniel Kroening, Doron A. Peled, and Helmut Veith. *Model checking, 2nd Edition*. MIT Press, 2018.

[72] F. H. Clarke. Lyapunov functions and discontinuous stabilizing feedback. *Annual Reviews in Control*, 35(1):13–33, 2011.

[73] F. H. Clarke, Y.S. Ledyaev, L. Rifford, and R.J. Stern. Feedback stabilization and Lyapunov functions. *SIAM Journal on Control and Optimization*, 39(1):25–48, 2000.

[74] J. Cortes. Discontinuous dynamical systems. *IEEE Control Systems Magazine*, 28(3):36–73, 2008.

[75] Patrick Cousot and Radhia Cousot. Constructive versions of Tarski's fixed point theorems. *Pacific Journal of Mathematics*, 82(1):43 – 57, 1979.

[76] Eric Dallal and Paulo Tabuada. On compositional symbolic controller synthesis inspired by small-gain theorems. In *2015 54th IEEE Conference on Decision and Control (CDC)*, pages 6133–6138, 2015.

[77] Werner Damm and Bernd Finkbeiner. Automatic compositional synthesis of distributed systems. In Cliff Jones, Pekka Pihlajasaari, and Jun Sun, editors, *FM 2014: Formal Methods*, pages 179–193, Cham, 2014. Springer International Publishing.

[78] Julie De Pril, János Flesch, Jeroen Kuipers, Gijs Schoenmakers, and Koos Vrieze. Existence of secure equilibrium in multi-player games with perfect information. In Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik, editors, *Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part II*, volume 8635 of *Lecture Notes in Computer Science*, pages 213–225. Springer, 2014.

[79] Stphane Demri, Valentin Goranko, and Martin Lange. *Temporal Logics in Computer Science: Finite-State Systems*. Cambridge University Press, USA, 1st edition, 2016.

[80] Rayna Dimitrova and Rupak Majumdar. Deductive control synthesis for alternating-time logics. In *2014 International Conference on Embedded Software (EMSOFT)*, pages 1–10, 2014.

[81] Alexandre Donzé and Oded Maler. Robust satisfaction of temporal logic over real-valued signals. In Krishnendu Chatterjee and Thomas A. Henzinger, editors, *Formal Modeling and Analysis of Timed Systems*, pages 92–106, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[82] Laurent Doyen, Thomas A. Henzinger, Axel Legay, and Dejan Nickovic. Robustness of sequential circuits. In Luís Gomes, Victor Khomenko, and João M. Fernandes, editors, *10th International Conference on Application of Concurrency to System Design, ACSD 2010, Braga, Portugal, 21-25 June 2010*, pages 77–84. IEEE Computer Society, 2010.

[83] Raimund Edlinger, Christoph Föls, and Andreas Nüchter. An innovative pick-up and transport robot system for casualty evacuation. In *IEEE International Symposium on Safety, Security, and Rescue Robotics, SSRR 2022, Sevilla, Spain, November 8-10, 2022*, pages 67–73. IEEE, 2022.

[84] Rüdiger Ehlers, Stéphane Lafortune, Stavros Tripakis, and Moshe Y. Vardi. Supervisory control and reactive synthesis: a comparative introduction. *Discret. Event Dyn. Syst.*, 27(2):209–260, 2017.

[85] Rüdiger Ehlers and Ufuk Topcu. Resilience to intermittent assumption violations in reactive synthesis. In Martin Fränzle and John Lygeros, editors, *17th International Conference on Hybrid Systems: Computation and Control (part of CPS Week), HSCC'14, Berlin, Germany, April 15-17, 2014*, pages 203–212. ACM, 2014.

[86] E. Allen Emerson and Joseph Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. *J. Comput. Syst. Sci.*, 30(1):1–24, 1985.

[87] E. Allen Emerson and Charanjit S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*, pages 368–377. IEEE Computer Society, 1991.

[88] E. Allen Emerson and Charanjit S. Jutla. The complexity of tree automata and logics of programs. *SIAM J. Comput.*, 29(1):132–158, 1999.

[89] E. Allen Emerson and Chin-Laung Lei. Modalities for model checking: Branching time logic strikes back. *Sci. Comput. Program.*, 8(3):275–306, 1987.

[90] Georgios E. Fainekos and George J. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science*, 410(42):4262–4291, September 2009.

[91] Peter Faymonville, Bernd Finkbeiner, and Leander Tentrup. Bosy: An experimentation framework for bounded synthesis. In *Proceedings of CAV*, volume 10427 of *LNCS*, pages 325–332. Springer, Springer, 2017.

[92] Lu Feng, Clemens Wiltsche, Laura Humphrey, and Ufuk Topcu. Synthesis of human-in-the-loop control protocols for autonomous systems. *IEEE Transactions on Automation Science and Engineering*, 13(2):450–462, 2016.

[93] Nathanaël Fijalkow, Nathalie Bertrand, Patricia Bouyer-Decitre, Romain Brenguier, Arnaud Carayol, John Fearnley, Hugo Gimbert, Florian Horn, Rasmus Ibsen-Jensen, Nicolas Markey, et al. Games on graphs. *arXiv preprint arXiv:2305.10546*, 2023.

[94] Emmanuel Filiot, Raffaella Gentilini, and Jean-François Raskin. Rational synthesis under imperfect information. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 422–431. ACM, 2018.

[95] Bernd Finkbeiner, Niklas Metzger, and Yoram Moses. Information flow guided synthesis. In *Proceedings of 34th International Conference on Computer Aided Verification (CAV 22)*, 2022.

[96] Bernd Finkbeiner, Niklas Metzger, Satya Prakash Nayak, and Anne-Kathrin Schmuck. Synthesis of universal safety controllers. In Arie Gurfinkel and Marijn Heule, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 31st International Conference, TACAS 2025, Held as Part of the International Joint Conferences on Theory and Practice of Software, ETAPS 2025, Hamilton, ON, Canada, May 3-8, 2025, Proceedings, Part II*, volume 15697 of *Lecture Notes in Computer Science*, pages 177–197. Springer, 2025.

[97] Bernd Finkbeiner, Niklas Metzger, Satya Prakash Nayak, and Anne-Kathrin Schmuck. Universal safety controllers with learned prophecies. In Sven Koenig,

Chad Jenkins, and Matthew E. Taylor, editors, *Fortieth AAAI Conference on Artificial Intelligence, Thirty-Eighth Conference on Innovative Applications of Artificial Intelligence, Sixteenth Symposium on Educational Advances in Artificial Intelligence, AAAI 2026, Singapore, January 20-27, 2026*, pages 36217–36226. AAAI Press, 2026.

[98] Bernd Finkbeiner and Noemi Passing. Compositional synthesis of modular systems. *Innov. Syst. Softw. Eng.*, 18(3):455–469, 2022.

[99] Bernd Finkbeiner and Noemi Passing. Synthesizing dominant strategies for liveness. In Anuj Dawar and Venkatesan Guruswami, editors, *42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2022, December 18-20, 2022, IIT Madras, Chennai, India*, volume 250 of *LIPIcs*, pages 37:1–37:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[100] Bernd Finkbeiner and Sven Schewe. Bounded synthesis. *Int. J. Softw. Tools Technol. Transf.*, 15(5-6):519–539, 2013.

[101] Dana Fisman, Orna Kupferman, and Yoad Lustig. Rational synthesis. In *TACAS*, 2010.

[102] Daniel J. Fremont and Sanjit A. Seshia. Reactive control improvisation. In Hana Chockler and Georg Weissenbacher, editors, *Computer Aided Verification*, pages 307–326, Cham, 2018. Springer International Publishing.

[103] Khouloud Gaaloul, Claudio Menghi, Shiva Nejati, Lionel Briand, and Yago Isasi Parache. Combining genetic programming and model checking to generate environment assumptions. *TSE*, 2021.

[104] Khouloud Gaaloul, Claudio Menghi, Shiva Nejati, Lionel C Briand, and David Wolfe. Mining assumptions for software components using machine learning. In *ESEC/FSE*, 2020.

[105] Mihaela Gheorghiu Bobaru, Corina S Păsăreanu, and Dimitra Giannakopoulou. Automated assume-guarantee reasoning by abstraction refinement. In *International Conference on Computer Aided Verification*, pages 135–148. Springer, 2008.

[106] Dimitra Giannakopoulou, Corina S Pasareanu, and Howard Barringer. Assumption generation for software component verification. In *Proceedings 17th IEEE International Conference on Automated Software Engineering,*, pages 3–12. IEEE, 2002.

[107] Antoine Girard, Alessio Iovine, and Sofiane Benberkane. Invariant sets for assume-guarantee contracts. In *2022 IEEE 61st Conference on Decision and Control (CDC)*, pages 2190–2195, 2022.

[108] Patil Girija, John Mareena, Jin Fenny, Kandula Swapna, and Ketsaraporn Kaewkhiaolueang. Amazon robotic service (ars). 2021.

[109] Oz Gitelson, Satya Prakash Nayak, Ritam Raha, and Anne-Kathrin Schmuck. Maximal adaptation, minimal guidance: Permissive reactive robot task planning with humans in the loop. *arXiv preprint arXiv:2510.12662*, 2025.

[110] R. Goebel, R.G. Sanfelice, and A.R. Teel. *Hybrid Dynamical Systems: Modeling, Stability, and Robustness*. Princeton University Press, 2012.

[111] Rafal Goebel and Andrew R Teel. Zeno behavior in homogeneous hybrid systems. In *IEEE Conference on Decision and Control (CDC)*, pages 2758–2763, 2008.

[112] Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.

[113] David Gundana and Hadas Kress-Gazit. Event-based signal temporal logic tasks: Execution and feedback in complex environments. *IEEE Robotics and Automation Letters*, 7(4):10001–10008, 2022.

[114] Petr Hájek. *Metamathematics of Fuzzy Logic*, volume 4 of *Trends in Logic*. Kluwer, 1998.

[115] Patrik Haslum, Nir Lipovetzky, Daniele Magazzeni, Christian Muise, Ronald Brachman, Francesca Rossi, and Peter Stone. *An introduction to the planning domain definition language*, volume 13. Springer, 2019.

[116] Binghan He, Jaemin Lee, Ufuk Topcu, and Luis Sentis. BP-RRT: Barrier pair synthesis for temporal logic motion planning. In *2020 59th IEEE Conference on Decision and Control (CDC)*, pages 1404–1409, 2020.

[117] Kyle Hsu, Rupak Majumdar, Kaushik Mallik, and Anne-Kathrin Schmuck. Multi-layered abstraction-based controller synthesis for continuous-time systems. In *HSCC'18*, pages 120–129. ACM, 2018.

[118] Yuchong Huang, Ning Xu, Meng Guo, Jie Li, and Lincheng Shen. Distributed and reactive controller synthesis for multi-agent systems under finite horizon temporal logic tasks. *IEEE Transactions on Automation Science and Engineering*, 22:13485–13500, 2025.

[119] Swen Jacobs, Guillermo A. Pérez, Remco Abraham, Véronique Bruyère, Michaël Cadilhac, Maximilien Colange, Charly Delfosse, Tom van Dijk, Alexandre Duret-Lutz, Peter Faymonville, Bernd Finkbeiner, Ayrat Khalimov, Felix Klein, Michael Luttenberger, Klara J. Meyer, Thibaud Michaud, Adrien Pommellet, Florian Renkin, Philipp Schlehuber-Caissier, Mouhammad Sakr, Salomon Sickert, Gaëtan Staquet, Clément Tamines, Leander Tentrup, and Adam Walker. The reactive synthesis competition (SYNTCOMP): 2018-2021. *CoRR*, abs/2206.00251, 2022.

[120] Pushpak Jagtap, Sadegh Soudjani, and Majid Zamani. Formal synthesis of stochastic systems via control barrier certificates. *IEEE Transactions on Automatic Control*, 66(7):3097–3110, 2021.

[121] Sebastian Junges, Nils Jansen, Joost-Pieter Katoen, Ufuk Topcu, Ruohan Zhang, and Mary Hayhoe. Model checking for safe navigation among humans. In *International Conference on Quantitative Evaluation of Systems*, pages 207–222. Springer, 2018.

[122] Marcin Jurdziński and Ranko Lazić. Succinct progress measures for solving parity games. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–9. IEEE, 2017.

[123] Eleni Kelasidi, Pål Liljebäck, Kristin Ytterstad Pettersen, and Jan Tommy Gravdahl. Innovation in underwater robots: Biologically inspired swimming snake robots. *IEEE Robotics Autom. Mag.*, 23(1):44–62, 2016.

[124] Eleni Kelasidi, Signe Moe, Kristin Ytterstad Pettersen, Anna M. Kohl, Pål Liljebäck, and Jan Tommy Gravdahl. Path following, obstacle detection and obstacle avoidance for thrusted underwater snake robots. *Frontiers Robotics AI*, 6:57, 2019.

[125] Mahmoud Khaled and Majid Zamani. pFaces: an acceleration ecosystem for symbolic control. In *HSCC'19*, pages 252–257. ACM, 2019.

[126] HK Khalil. Nonlinear systems, printice-hall. *Upper Saddle River, NJ*, 3, 1996.

[127] Eric S. Kim, Sadra Sadraddini, Calin Belta, Murat Arcak, and Sanjit A. Seshia. Dynamic contracts for distributed temporal logic control of traffic networks. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 3640–3645, 2017.

[128] Joachim Klein, Christel Baier, and Sascha Klüppelholz. Compositional construction of most general controllers. *Acta Informatica*, 52(4-5):443–482, 2015.

[129] Dexter Kozen. Results on the propositional mu-calculus. *Theor. Comput. Sci.*, 27:333–354, 1983.

[130] Steve Kremer and Jean-François Raskin. A game-based verification of non-repudiation and fair exchange protocols. *J. Comput. Secur.*, 11(3):399–430, 2003.

[131] Hadas Kress-Gazit, Georgios E Fainekos, and George J Pappas. Temporal-logic-based reactive mission and motion planning. *IEEE transactions on robotics*, 25(6):1370–1381, 2009.

[132] Hadas Kress-Gazit, Morteza Lahijanian, and Vasumathi Raman. Synthesis for robots: Guarantees and feedback for robot behavior. *Annual Review of Control, Robotics, and Autonomous Systems*, 1(1):211–236, 2018.

[133] Jan Kretínský, Tobias Meggendorfer, Maximilian Prokop, and Ashkan Zarkhah. Semml: Enhancing automata-theoretic LTL synthesis with machine learning. In *TACAS (1)*, volume 15696 of *Lecture Notes in Computer Science*, pages 233–253. Springer, 2025.

[134] O. Kupferman and M.Y. Vardi. Synthesis with incomplete information. In Howard Barringer et al., editor, *2nd International Conference on Temporal Logic*, pages 91–106, Manchester, July 1997.

[135] Orna Kupferman and Noam Shenwald. The complexity of LTL rational synthesis. In Dana Fisman and Grigore Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I*, volume 13243 of *Lecture Notes in Computer Science*, pages 25–45. Springer, 2022.

[136] Orna Kupferman and Moshe Y. Vardi. μ-calculus synthesis. In Mogens Nielsen and Branislav Rovan, editors, *Mathematical Foundations of Computer Science 2000, 25th International Symposium, MFCS 2000, Bratislava, Slovakia, August 28 - September 1, 2000, Proceedings*, volume 1893 of *Lecture Notes in Computer Science*, pages 497–507. Springer, 2000.

[137] Karoliina Lehtinen, Paweł Parys, Sven Schewe, and Dominik Wojtczak. A recursive approach to solving parity games in quasipolynomial time. *Logical Methods in Computer Science*, 18, 2022.

[138] Shen Li, Daehyung Park, Yoonchang Sung, Julie A Shah, and Nicholas Roy. Reactive task and motion planning under temporal logic specifications. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 12618–12624. IEEE, 2021.

[139] Xiaoru Li, Xiaohong Li, Guangquan Xu, Jing Hu, and Zhiyong Feng. Formal analysis of fairness for optimistic multiparty contract signing protocol. *J. Appl. Math.*, 2014:983204:1–983204:10, 2014.

[140] Yinan Li and Jun Liu. ROCS: A robustly complete control synthesis tool for nonlinear dynamical systems. In *HSCC'18*, pages 130–135. ACM, 2018.

[141] D. Liberzon. *Switching in systems and control*. Birkhaüser, 2003.

[142] Martin Lück. Quirky quantifiers: Optimal models and complexity of computation tree logic. *Int. J. Found. Comput. Sci.*, 29(1):17–62, 2018.

[143] David Luenberger. An introduction to observers. *IEEE Transactions on automatic control*, 16(6):596–602, 2003.

[144] Ana Maria Mainhardt and Anne-Kathrin Schmuck. Assume-guarantee synthesis of decentralised supervisory control. *IFAC-PapersOnLine*, 55(28):165–172, 2022. 16th IFAC Workshop on Discrete Event Systems WODES 2022.

[145] Rupak Majumdar, Kaushik Mallik, Anne-Kathrin Schmuck, and Damien Zufferey. Assume–guarantee distributed synthesis. *IEEE TCAD*, 2020.

[146] Rupak Majumdar, Kaushik Mallik, Anne-Kathrin Schmuck, and Damien Zufferey. Assume–guarantee distributed synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(11):3215–3226, 2020.

[147] Rupak Majumdar, Necmiye Ozay, and Anne-Kathrin Schmuck. On abstraction-based controller design with output feedback. In *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*, pages 1–11, 2020.

[148] Rupak Majumdar, Nir Piterman, and Anne-Kathrin Schmuck. Environmentally-friendly gr (1) synthesis. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 229–246. Springer, 2019.

[149] Rupak Majumdar and Indranil Saha. Symbolic robustness analysis. In Theodore P. Baker, editor, *Proceedings of the 30th IEEE Real-Time Systems Symposium, RTSS 2009, Washington, DC, USA, 1-4 December 2009*, pages 355–363. IEEE Computer Society, 2009.

[150] Rupak Majumdar and Anne-Kathrin Schmuck. Supervisory controller synthesis for nonterminating processes is an obliging game. *IEEE Transactions on Automatic Control*, 68(1):385–392, 2023.

[151] Kaushik Mallik, Anne-Kathrin Schmuck, Sadegh Soudjani, and Rupak Majumdar. Compositional synthesis of finite-state abstractions. *IEEE Transactions on Automatic Control*, 64(6):2629–2636, 2019.

[152] Shahar Maoz, Jan Oliver Ringert, and Rafi Shalom. Symbolic repairs for GR(1) specifications. In *ICSE*, 2019.

[153] Corto Mascle, Daniel Neider, Maximilian Schwenger, Paulo Tabuada, Alexander Weinert, and Martin Zimmermann. From LTL to rLTL monitoring: improved monitorability through robust semantics. *Formal Methods in System Design*, oct 2022.

[154] Noushin Mehdipour, Matthias Althoff, Radboud Duintjer Tebbens, and Calin Belta. Formal methods to comply with rules of the road in autonomous driving: State of the art and grand challenges. *Automatica*, 152:110692, 2023.

[155] Noushin Mehdipour, Cristian Ioan Vasile, and Calin Belta. Average-based robustness for continuous-time signal temporal logic. In *58th IEEE Conference on*

Decision and Control, CDC 2019, Nice, France, December 11-13, 2019, pages 5312–5317. IEEE, 2019.

[156] Thibaud Michaud and Maximilien Colange. Reactive synthesis from LTL specification with spot. In *Proceedings Seventh Workshop on Synthesis, SYNTCAV 2018*, volume xx of *Electronic Proceedings in Theoretical Computer Science*, page xx, 2018.

[157] Aniello Murano, Daniel Neider, and Martin Zimmermann. Robust alternating-time temporal logic. In Sarah Alice Gaggl, Maria Vanina Martinez, and Magdalena Ortiz, editors, *Logics in Artificial Intelligence - 18th European Conference, JELIA 2023, Dresden, Germany, September 20-22, 2023, Proceedings*, volume 14281 of *Lecture Notes in Computer Science*, pages 796–813. Springer, 2023.

[158] Karan Muvvala, Qi Heng Ho, and Morteza Lahijanian. Beyond winning strategies: Admissible and admissible winning strategies for quantitative reachability games. *ICJAI*, 2025.

[159] Karan Muvvala and Morteza Lahijanian. Admissibility over winning: A new approach to reactive synthesis in robotics. *arXiv e-prints*, pages arXiv–2410, 2024.

[160] Karan Muvvala, Andrew M. Wells, Morteza Lahijanian, Lydia E. Kavraki, and Moshe Y. Vardi. Stochastic games for interactive manipulation domains. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2513–2519, 2024.

[161] John F. Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, 36(1):48–49, 1950.

[162] Satya Prakash Nayak, Lucas N Egidio, Matteo Della Rossa, Anne-Kathrin Schmuck, and Raphael M Jungers. Context-triggered abstraction-based control design. *IEEE Open Journal of Control Systems*, 2:277–296, 2023.

[163] Satya Prakash Nayak, Daniel Neider, Rajarshi Roy, and Martin Zimmermann. Robust computation tree logic. In Jyotirmoy V. Deshmukh, Klaus Havelund, and Ivan Perez, editors, *NASA Formal Methods - 14th International Symposium, NFM 2022, Pasadena, CA, USA, May 24-27, 2022, Proceedings*, volume 13260 of *Lecture Notes in Computer Science*, pages 538–556. Springer, 2022.

[164] Satya Prakash Nayak, Daniel Neider, Rajarshi Roy, and Martin Zimmermann. Robust computation tree logic. *Innov. Syst. Softw. Eng.*, 21(2):595–617, 2025.

[165] Satya Prakash Nayak, Daniel Neider, and Martin Zimmermann. Robustness-by-construction synthesis: Adapting to the environment at runtime. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation. Verification Principles*, pages 149–173, Cham, 2022. Springer International Publishing.

[166] Satya Prakash Nayak and Anne-Kathrin Schmuck. Most general winning secure equilibria synthesis in graph games. In Bernd Finkbeiner and Laura Kovács, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 30th International Conference, TACAS 2024, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2024, Luxembourg City, Luxembourg, April 6-11, 2024, Proceedings, Part III*, volume 14572 of *Lecture Notes in Computer Science*, pages 173–193. Springer, 2024.

[167] Daniel Neider and Ivan Gavran. Learning linear temporal properties. In *2018 Formal Methods in Computer Aided Design (FMCAD)*, pages 1–10. IEEE, 2018.

[168] Daniel Neider, Alexander Weinert, and Martin Zimmermann. Robust, expressive, and quantitative linear temporal logics: Pick any two for free. In Jérôme Leroux and Jean-François Raskin, editors, *Proceedings Tenth International Symposium on Games, Automata, Logics, and Formal Verification, GandALF 2019, Bordeaux, France, 2-3rd September 2019*, volume 305 of *EPTCS*, pages 1–16, 2019.

[169] Daniel Neider, Alexander Weinert, and Martin Zimmermann. Robust, expressive, and quantitative linear temporal logics: Pick any two for free. *Inf. Comput.*, 285(Part):104810, 2022.

[170] Petter Nilsson and Aaron D. Ames. Barrier functions: Bridging the gap between planning from specifications and safety-critical control. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 765–772, 2018.

[171] Pawel Parys. Parity Games: Zielonka's Algorithm in Quasi-Polynomial Time. In Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen, editors, *44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019)*, volume 138 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:13, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[172] Lasse Peters, Andrea Bajcsy, Chih-Yuan Chiu, David Fridovich-Keil, Forrest Laine, Laura Ferranti, and Javier Alonso-Mora. Contingency games for multi-agent interaction, 2023.

[173] Kittiphon Phalakarn, Sasinee Pruekprasert, and Ichiro Hasuo. Winning strategy templates for stochastic parity games towards permissive and resilient control. In *International Colloquium on Theoretical Aspects of Computing*, pages 197–214. Springer, 2024.

[174] Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of reactive(1) designs. In *Proceedings of the 7th International Conference on Verification, Model Checking, and Abstract Interpretation*, VMCAI'06, page 364–380, Berlin, Heidelberg, 2006. Springer-Verlag.

[175] André Platzer. *Logical Foundations of Cyber-Physical Systems*. Springer, 2018.

[176] Amir Pneuli and Roni Rosner. Distributed reactive systems are hard to synthesize. In *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*, pages 746–757. IEEE, 1990.

[177] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57. IEEE Computer Society, 1977.

[178] Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *Conference Record of the Sixteenth Annual ACM Symposium on Principles of Programming Languages, Austin, Texas, USA, January 11-13, 1989*, pages 179–190. ACM Press, 1989.

[179] Amir Pnueli and Roni Rosner. On the synthesis of an asynchronous reactive module. In Giorgio Ausiello, Mariangiola Dezani-Ciancaglini, and Simona Ronchi Della Rocca, editors, *Automata, Languages and Programming, 16th International Colloquium, ICALP89, Stresa, Italy, July 11-15, 1989, Proceedings*, volume 372 of *Lecture Notes in Computer Science*, pages 652–671. Springer, 1989.

[180] Adrien Pommellet, Daniel Stan, and Simon Scatton. Sat-based learning of computation tree logic. In *International Joint Conference on Automated Reasoning*, pages 366–385. Springer, 2024.

[181] Graham Priest. Dualising intuitionictic negation. *Principia: an international journal of epistemology*, 13(2):165–184, 2009.

[182] Ritam Raha, Rajarshi Roy, Nathanaël Fijalkow, Daniel Neider, and Guillermo A Pérez. Synthesizing efficiently monitorable formulas in metric temporal logic. In *International Conference on Verification, Model Checking, and Abstract Interpretation*, pages 264–288. Springer, 2023.

[183] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM J. Control Optim.*, 25(1):206–230, January 1987.

[184] G. Reissig, A. Weber, and M. Rungger. Feedback refinement relations for the synthesis of symbolic controllers. *TAC*, 62(4):1781–1796, 2017.

[185] Frederic Anthony Robinson, Mari Velonaki, and Oliver Bown. Smooth operator: Tuning robot perception through artificial movement sound. In *Proceedings of the 2021 ACM/IEEE international conference on human-robot interaction*, pages 53–62, 2021.

[186] Alëna Rodionova, Ezio Bartocci, Dejan Nickovic, and Radu Grosu. Temporal logic as filtering. In Alessandro Abate and Georgios Fainekos, editors, *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control, HSCC 2016, Vienna, Austria, April 12-14, 2016*, pages 11–20. ACM, 2016.

[187] M.Z. Romdlony and B. Jayawardhana. Stabilization with guaranteed safety using control Lyapunov–barrier function. *Automatica*, 66:39–47, 2016.

[188] M. Rungger and M. Zamani. SCOTS: A tool for the synthesis of symbolic controllers. In *HSCC*, pages 99–104. ACM, 2016.

[189] Dorsa Sadigh, Nick Landolfi, Shankar S Sastry, Sanjit A Seshia, and Anca D Dragan. Planning for cars that coordinate with people: leveraging effects on human actions for planning and active information gathering over human internal state. *Autonomous Robots*, 42(7):1405–1426, 2018.

[190] Dorsa Sadigh, Shankar Sastry, Sanjit A Seshia, and Anca D Dragan. Planning for autonomous cars that leverage effects on human actions. In *Robotics: Science and systems*, volume 2, pages 1–9. Ann Arbor, MI, USA, 2016.

[191] Sayan Saha and A. Agung Julius. An MILP approach for real-time optimal controller synthesis with metric temporal logic specifications. In *2016 American Control Conference (ACC)*, pages 1105–1110, 2016.

[192] Nathan E. Sanders, Elif Şener, and Karen B. Chen. Robot-related injuries in the workplace: An analysis of osha severe injury reports. *Applied Ergonomics*, 121:104324, 2024.

[193] Ricardo G Sanfelice. *Hybrid feedback control*. Princeton University Press, 2020.

[194] David Schmelter, Joel Greenyer, and Jörg Holtmann. Toward learning realizable scenario-based, formal requirements specifications. In *REW*, 2017.

[195] Anne-Kathrin Schmuck, Philippe Heim, Rayna Dimitrova, and Satya Prakash Nayak. Localized attractor computations for infinite-state games. In Arie Gurfinkel and Vijay Ganesh, editors, *Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part III*, volume 14683 of *Lecture Notes in Computer Science*, pages 135–158. Springer, 2024.

[196] Anne-Kathrin Schmuck, Thomas Moor, and Rupak Majumdar. On the relation between reactive synthesis and supervisory control of non-terminating processes. *Discret. Event Dyn. Syst.*, 30(1):81–124, 2020.

[197] Anne-Kathrin Schmuck, K. S. Thejaswini, Irmak Saglam, and Satya Prakash Nayak. Solving two-player games under progress assumptions. In Rayna Dimitrova, Ori Lahav, and Sebastian Wolff, editors, *Verification, Model Checking, and Abstract Interpretation - 25th International Conference, VMCAI 2024, London, United Kingdom, January 15-16, 2024, Proceedings, Part I*, volume 14499 of *Lecture Notes in Computer Science*, pages 208–231. Springer, 2024.

[198] Philippe Schnoebelen. The complexity of temporal logic model checking. In Philippe Balbiani, Nobu-Yuki Suzuki, Frank Wolter, and Michael Zakharyaschev, editors, *Advances in Modal Logic 4, papers from the fourth conference on "Advances in Modal logic," held in Toulouse, France, 30 September - 2 October 2002*, pages 393–436. King's College Publications, 2002.

[199] Georg Friedrich Schuppe, Ilaria Torre, Iolanda Leite, and Jana Tumova. Follow my advice: Assume-guarantee approach to task planning with human in the loop. In *Robotics: Science and Systems*, 2023.

[200] Sanjit A. Seshia, Natasha Sharygina, and Stavros Tripakis. Modeling for verification. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 75–105. Springer, 2018.

[201] Jeff S Shamma and Kuang-Yang Tu. Set-valued observers and optimal disturbance rejection. *IEEE Transactions on Automatic Control*, 44(2):253–264, 2002.

[202] Pratyusha Sharma, Balakumar Sundaralingam, Valts Blukis, Chris Paxton, Tucker Hermans, Antonio Torralba, Jacob Andreas, and Dieter Fox. Correcting robot plans with natural language feedback. In *Robotics: Science and Systems*, 2022.

[203] Kaige Shi, Jindong Chang, Shuo Feng, Yali Fan, Zilai Wei, and Guoqiang Hu. Safe human dual-robot interaction based on control barrier functions and cooperation functions. *IEEE Robotics and Automation Letters*, 9(11):9581–9588, 2024.

[204] E.D. Sontag. A Lyapunov-like characterization of asymptotic controllability. *SIAM Journal on Control and Optimization*, 21(3):462–471, 1983.

[205] E.D. Sontag. A "universal" construction of Artstein's theorem on nonlinear stabilization. *Systems & Control Letters*, 13(2):117–123, 1989.

[206] Mohit Srinivasan, Samuel Coogan, and Magnus Egerstedt. Control of multi-agent systems with finite time control barrier certificates and temporal logic. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 1991–1996, 2018.

[207] Jan-Henrik Steg. On identifying subgame-perfect equilibrium outcomes for timing games. *Games Econ. Behav.*, 135:74–78, 2022.

[208] Robert S. Streett and E. Allen Emerson. An automata theoretic decision procedure for the propositional mu-calculus. *Inf. Comput.*, 81(3):249–264, 1989.

[209] Fei Sun, Necmiye Ozay, Eric M. Wolff, Jun Liu, and Richard M. Murray. Efficient control synthesis for augmented finite transition systems with an application to switching protocols. In *2014 American Control Conference*, pages 3273–3280, 2014.

[210] Paulo Tabuada. *Verification and Control of Hybrid Systems - A Symbolic Approach*. Springer, 2009.

[211] Paulo Tabuada, Ayca Balkan, Sina Y. Caliskan, Yasser Shoukry, and Rupak Majumdar. Input-output robustness for discrete systems. In Ahmed Jerraya, Luca P. Carloni, Florence Maraninchi, and John Regehr, editors, *Proceedings of the 12th International Conference on Embedded Software, EMSOFT 2012, part of the Eighth Embedded Systems Week, ESWeek 2012, Tampere, Finland, October 7-12, 2012*, pages 217–226. ACM, 2012.

[212] Paulo Tabuada, Sina Yamac Caliskan, Matthias Rungger, and Rupak Majumdar. Towards robustness for cyber-physical systems. *IEEE Trans. Autom. Control.*, 59(12):3151–3163, 2014.

[213] Paulo Tabuada and Daniel Neider. Robust linear temporal logic. In Jean-Marc Talbot and Laurent Regnier, editors, *25th EACSL Annual Conference on Computer Science Logic, CSL 2016, August 29 - September 1, 2016, Marseille, France*, volume 62 of *LIPIcs*, pages 10:1–10:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

[214] Danielle C. Tarraf, Alexandre Megretski, and Munther A. Dahleh. A framework for robust stability of systems over finite alphabets. *IEEE Trans. Autom. Control.*, 53(5):1133–1146, 2008.

[215] Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific journal of Mathematics*, 5(2):285–309, 1955.

[216] Michael Ummels. Rational behaviour and strategy construction in infinite multiplayer games. In S. Arun-Kumar and Naveen Garg, editors, *FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science, 26th International Conference, Kolkata, India, December 13-15, 2006, Proceedings*, volume 4337 of *Lecture Notes in Computer Science*, pages 212–223. Springer, 2006.

[217] Mojtaba Valizadeh, Nathanaël Fijalkow, and Martin Berger. Ltl learning on gpus. In *International Conference on Computer Aided Verification*, pages 209–231. Springer, 2024.

[218] Tom van Dijk. Oink: An implementation and evaluation of modern parity game solvers. In Dirk Beyer and Marieke Huisman, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, Part I*, volume 10805 of *Lecture Notes in Computer Science*, pages 291–308. Springer, 2018.

[219] Vasileios Vasilopoulos, Yiannis Kantaros, George J. Pappas, and Daniel E. Koditschek. Reactive planning for mobile manipulation tasks in unexplored semantic environments. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6385–6392, 2021.

[220] Alan R. Wagner. Robot-guided evacuation as a paradigm for human-robot interaction research. *Frontiers Robotics AI*, 8:701938, 2021.

[221] Wei Xiao, Calin A. Belta, and Christos G. Cassandras. High order control Lyapunov-barrier functions for temporal logic specifications. In *2021 American Control Conference (ACC)*, pages 4886–4891, 2021.

[222] Changjian Zhang, David Garlan, and Eunsuk Kang. A behavioral notion of robustness for software systems. In Prem Devanbu, Myra B. Cohen, and Thomas Zimmermann, editors, *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, pages 1–12. ACM, 2020.

[223] Yixiao Zhang, Victor Nan Fernandez-Ayala, and Dimos V. Dimarogonas. Multi-robot human-in-the-loop control under spatiotemporal specifications. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4841–4847, 2024.

[224] Zhangli Zhou, Shaochen Wang, Ziyang Chen, Mingyu Cai, Hao Wang, Zhijun Li, and Zhen Kan. Local observation based reactive temporal logic planning of human-robot systems. *IEEE Transactions on Automation Science and Engineering*, 22:643–655, 2025.

[225] Wieslaw Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theor. Comput. Sci.*, 200(1-2):135–183, 1998.

[226] Martin Zimmermann. Robust probabilistic temporal logics. *arXiv*, abs/2306.05806, 2023.

Curriculum Vitae

Research Interests

Formal verification and synthesis of cyber-physical systems, Temporal Logics, Game Theory

Education

[2021–]	Doctoral student (Computer Science), Max Planck Institute for Software Systems Kaiserslautern, Germany.	
[2019–2021]	Masters in Computer Science, Chennai Mathematical Institute Chennai, India.	CGPA: 9.94/10.
[2016–2019]	Bachelors in Mathematics and Computer Science, Chennai Mathematical Institute Chennai, India.	CGPA: 8.48/10.