Assumptions in Synthesis: An Approach to Multi-Agent Planning from Spatio-Temporal Specifications

GEORG FRIEDRICH SCHUPPE

Doctoral Thesis
Stockholm, Sweden 2023

Division of Robotics, Perception and Learning
School of Electrical Engineering and Computer Science
KTH Royal Institute of Technology
SE-100 44 Stockholm, Sweden

TRITA-EECS-AVL-2023:73
978-91-8040-731-1

Public defense:
20/11/2023, 14:00 GMT+2
F3 (Flodis), Lindstedtsvägen 26 & 28

© Georg Friedrich Schuppe, November 2023, except where otherwise stated.

Tryck: Universitetsservice US AB

"You couldn't have it if you did want it," the queen said. *"The rule is, jam tomorrow and jam yesterday – but never jam today."*
"It must come sometimes to 'jam today'", Alice objected.
"No, it can't", said the Queen. *"It's jam every other day: today isn't any other day, you know."*
"I don't understand you", said Alice. *"It's dreadfully confusing!"*

Lewis Carroll, *Through the Looking-Glass*

Abstract

As the deployment of autonomous robots in real-world environments becomes increasingly prevalent, the need for these robots to operate safely and efficiently alongside human actors and other robots grows. Ensuring that robots adhere to their intended tasks and safety constraints as specified is of utmost importance, particularly when considering interactions between different robots and humans. To address this challenge, formal methods-based strategy synthesis techniques have emerged as a promising approach, offering rigorous specification languages and algorithms that generate correct-by-design strategies.

This doctoral thesis focuses on the design and development of decentralized strategy synthesis algorithms to handle multi-agent systems consisting of multiple robots, or robots and humans. By considering task planning in a shared environment, the research seeks to achieve guarantees regarding specification satisfaction while considering interactions between computational agents and human actors.

Several key considerations are addressed in this work. Scalability is a major concern in strategy synthesis methods, given their high computational costs. We avoid constructing a global, centralized model of the system to mitigate the state explosion problem. Instead, we focus on decentralized methods that ensure global integrity through communication of necessary, but minimal assumptions as least-limiting advisers.

Another challenge lies in effectively modeling human agents in the system. Human behavior is influenced by context and preferences, making it difficult to provide explicit models. We avoid explicitly modelling the human and instead propose an assume-guarantee approach, where the robot conveys only necessary advice to the human. This advice, if followed by the human, enables us to provide conditional guarantees on the satisfaction of the task specification.

Lastly, while temporal logics are a rich and expressive specification language, it can be difficult to correctly capture desired tasks and constraints. We present a specification language for object-oriented tasks that leverages spatial relations to intuitively specify desired spatial and temporal relations between objects. We show that this language can be used for monitoring and planning in various scenarios. We also demonstrate how it can be used to enable a robot to request help from a human assistant through structured language.

Overall, this doctoral thesis aims to bring the benefits of formal methods-based strategy synthesis to multi-agent systems with human actors, providing a unified methodology to compute and convey least-limiting advice in interactions between robots and between robots and humans.

Sammanfattning

I takt med att autonoma robotar blir allt vanligare i verkliga miljöer ökar behovet av att dessa robotar kan arbeta säkert och effektivt tillsammans med människor och andra robotar. Det är av yttersta vikt att se till att robotarna följer sina avsedda uppgifter och säkerhetsbegränsningar, särskilt när det gäller interaktioner mellan olika robotar och människor. För att hantera denna utmaning har formella metoder baserade på strategisyntes framstått som ett lovande tillvägagångssätt, som erbjuder rigorösa specifikationsspråk och algoritmer som genererar korrekta strategier efter design.

Denna doktorsavhandling fokuserar på design och utveckling av decentraliserade algoritmer för strategisyntes för att hantera multiagentsystem som består av flera robotar, eller robotar och människor. Genom att överväga uppgiftsplanering i en delad miljö försöker forskningen uppnå garantier för specifikationstillfredsställelse samtidigt som man överväger interaktioner mellan beräkningsagenter och mänskliga aktörer.

Flera viktiga överväganden tas upp i detta arbete. Skalbarhet är ett stort problem i strategisyntesmetoder, med tanke på deras höga beräkningskostnader. Vi undviker att konstruera en global, centraliserad modell av systemet för att mildra problemet med tillståndsexplosion. Istället fokuserar vi på decentraliserade metoder som säkerställer global integritet genom kommunikation av nödvändiga, men minimala antaganden som minst begränsande rådgivare.

En annan utmaning ligger i att effektivt modellera mänskliga agenter i systemet. Mänskligt beteende påverkas av sammanhang och preferenser, vilket gör det svårt att tillhandahålla explicita modeller. Vi undviker att explicit modellera människan och föreslår istället en anta-garantera-metod, där roboten endast förmedlar nödvändiga råd till människan. Dessa råd, om de följs av människan, gör det möjligt för oss att ge villkorliga garantier för att uppgiftsspecifikationen uppfylls.

Även om temporallogik är ett rikt och uttrycksfullt specifikationsspråk, kan det vara svårt att korrekt fånga önskade uppgifter och begränsningar. Vi presenterar ett specifikationsspråk för objektorienterade uppgifter som utnyttjar spatiala relationer för att intuitivt specificera önskade spatiala och temporala relationer mellan objekt. Vi visar att detta språk kan användas för övervakning och planering i olika scenarier. Vi visar också hur det kan användas för att göra det möjligt för en robot att begära hjälp från en mänsklig assistent genom strukturerat språk.

Sammantaget syftar denna doktorsavhandling till att föra fördelarna med formell metodbaserad strategisyntes till multi-agentsystem med mänskliga aktörer, genom att tillhandahålla en enhetlig metod för att beräkna och förmedla minst begränsande råd i interaktioner mellan robotar och mellan robotar och människor.

Acknowledgements

Although only a single author appears on the front, a doctoral thesis is hardly a solo project. It is hard to name and thank the many people that contributed to the work that culminated in this thesis, but I will try nonetheless.

I extend my deepest gratitude to my supervisor, Jana, for her invaluable guidance and unwavering support throughout this journey. Without her guidance, none of this research would have been conducted. I thank you for supporting me on all levels of academic life, be it through showing me how to self-organize, discussing weird automata constructs on the whiteboard, or sitting in an 6-hour zoom call to polish a paper until the very last paragraph before a submission deadline.

I also want to thank the collaborators of the publications appearing in this thesis. Thank you for working with me on what feels like years of journal revisions or for guiding me through design, execution and evaluation of a proper user study.

I would like to thank the members of our amazing research group, both former and existing. I truly believe that our discussions in the kitchen, office or any other room or facility helped me become a more knowledgeable and open-minded person. Thanks for engaging me in difficult whiteboard discussions, giving feedback to my paper drafts, having a beer with me or taking me to a restaurant featuring your local food. This is also the right place to mention the great colleagues in our division. Thank you for all the great events, retreats and summer schools. Thanks to the many attendants of the board game nights I tried to organize on a regular basis. I did not always succeed, but I am grateful for everyone that came out to play with me after a long, grueling day of research. A special mention goes out to the members of the original RPL board game group. Thank you for playing with me!

Outside of work, I have to express my thankfulness to my friends, who provided ample distractions from the daily slog of work. My thanks go out to the members of the Stockholm fighting game community, with whom I had many fun games, dinners and drinks over the years. I also have to thank the Stockholm MtG Legacy community for providing fun distractions every Thursday evening. Last, but not least, I want to thank my friends back in Germany, who always were happy to see me, even after I didn't show up for several months in a row.

My deepest thanks and gratitude go out to my fiancée and soon-to-be wife Lisa. You left your familiar life behind to stay with me in a foreign country. Without

your emotional support, I doubt I would have been able to surpass the difficult times full of doubt and despair. I love you more than words can express and am forever grateful that you are here with me.

Lastly, I want to express my heartfelt gratitude to my family. To my parents, Kerstin and Andree, in particular, thank you for nurturing and guiding me onto this path. Knowing that I have a loving home to return to, no matter the circumstances, fills my heart with warmth and appreciation.

To all those who have been a part of this journey, in large or small ways, thank you from the depths of my heart. Your influence appears somewhere in this thesis, even if invisible to the naked eye.

List of Papers

This thesis is based on the following papers:

A: **Georg Friedrich Schuppe** and Jana Tumova. *Multi-Agent Strategy Synthesis for LTL Specifications through Assumption Composition*. 2020 IEEE 16th International Conference on Automation Science and Engineering (CASE). IEEE, 2020.

B: **Georg Friedrich Schuppe** and Jana Tumova. *Decentralized Multi-Agent Strategy Synthesis under LTLf Specifications via Exchange of Least-Limiting Advisers*. 2021 International Symposium on Multi-Robot and Multi-Agent Systems (MRS). IEEE, 2021.

C: **Georg Friedrich Schuppe**, Ilaria Torre, Iolanda Leite and Jana Tumova. *Follow my Advice: Assume-Guarantee Approach to Task Planning with Human in the Loop*. 2023 Robotics: Science and Systems (RSS).

D: Christian Pek*, **Georg Friedrich Schuppe***, Francesco Esposito, Jana Tumova and Danica Kragic. *SpaTiaL: Monitoring and Planning of Robotic Tasks Using Spatio-Temporal Logic Specifications*. To be published at Springer: Autonomous Robots (AuRo).

The author has contributed to the following articles as well, although they are not included in this thesis:

X-1: Wei Wang, **Georg Friedrich Schuppe**, and Jana Tumova. *Decentralized Multi-agent Coordination under MITL Tasks and Communication Constraints*. Extended Abstract, 2022 ACM/IEEE 13th International Conference on Cyber-Physical Systems (ICCPS). IEEE, 2022.

X-2: Wei Wang, **Georg Friedrich Schuppe**, and Jana Tumova. *Decentralized Multi-agent Coordination under MITL Tasks and Communication Constraints*. 2023 31st Mediterranean Conference on Control and Automation (MED), IEEE, 2023.

*The authors contributed equally to this work

Contents

Acknowledgements **iii**

List of Papers **v**

Contents **vii**

1 Introduction **1**
1.1 Formal Methods and Multi-Agent Systems: Design Considerations . 2
1.2 State of the Art 5
1.3 Thesis Contributions and Structure 9

2 Preliminaries **13**
2.1 Models 14
2.2 Temporal Logic 19
2.3 Automata 24
2.4 Strategy Synthesis via Product Automata 27
2.5 Centralized Strategy Synthesis for Multi-Agent Systems 30

3 Compositional Strategy Synthesis **31**
3.1 Problem Formulation 31
3.2 Method 33
3.3 Experimental Results 40

4 Decentralized Assumption-Based Synthesis **45**
4.1 Minimal Assumptions 45
4.2 Least-Limiting Advisers for Agent-Agent Interaction 48
4.3 Least-Limiting Advisers for Human-Agent Interaction 60
4.4 Conclusion 68

5 Object-Centric Tasks **69**
5.1 SpaTiaL: An Object-Centric Specification Framework for Robotics . 70
5.2 Spatial Requests in Object-Centric Human-Robot Collaboration . . 96
5.3 Discussion and Limitations 101

6 Conclusions and Future Work **103**
6.1 Future Directions 104

Bibliography **107**

Chapter 1

Introduction

When deploying autonomous robots into the real world, we desire for them to work as intended so that they fulfill their tasks and obey safety constraints. With advancing capabilities of hardware and software for robotics, the tasks that robots are required to accomplish and constraints they have to adhere to are becoming increasingly sophisticated. In addition, with rising deployment of robots, it will become more common that multiple robots as well as human actors will share an environment. Thus, they might need to occasionally communicate or collaborate even though they were originally not necessarily meant to operate as a team. Guaranteeing that the behaviour of robots is both safe for all actors and lead to task satisfaction is important, especially when considering interactions between robots, human actors or uncontrollable elements in the environment. One increasingly popular way to approach this challenging problem is by taking strategy synthesis techniques from the field of formal methods, which combine two advantages:

(i) various complex tasks and constraints on the robot's behavior can be specified rigorously in temporal logics, such as *Linear Temporal Logic* (LTL); and

(ii) formal methods-based synthesis allows to algorithmically compute strategies that are correct by design, i.e. provably satisfy the specification.

In formal synthesis, the user describes desired behaviour of a system typically through a logical specification. In automaton-based methods, a given temporal logic formula is translated into an automaton, a discrete graph-like object. Model-based strategy synthesis methods reason on a model of the system under observation. Much like any other mathematical formalism, this model is an abstract description of the behaviour of our system. In this work, we consider *model-based, automaton-based strategy synthesis*, but other methods, e.g., optimization-based synthesis methods exist.

Formal synthesis can provide strong results, i.e. strategies that provably guarantee satisfaction of complex temporal constraints and that react to the actions and events of the environment. Furthermore, partial results or generated counterexam-

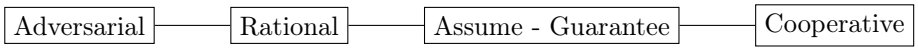


Figure 1.1: Spectrum of agent interaction.

ples can provide insight into why satisfying a specification is not possible or which events in the environment disallow satisfaction of the specification. Often, these strong results come at a cost of making strong assumptions, such as the existence of a model or little uncertainty or full observability of the environment. Also, most strategy synthesis methods are computationally very expensive and scalability of these methods is a major challenge.

The goal of this thesis is to extend the benefits of formal synthesis approaches to the task planning problem of a *multi-agent system* (MAS) consisting of multiple actors with different capabilities, constraints, goals or tasks in a shared environment.

1.1 Formal Methods and Multi-Agent Systems: Design Considerations

Applying formal synthesis techniques to multi-agent task planning problems confronts us with several key design decisions. In this section, we will highlight key considerations that this thesis focuses on and explain why they are of importance.

Modelling Approach

The first concern is that of scalability. Strategy synthesis methods are generally expensive and feature runtime complexities that are impossible to overcome by adding computational power [1]. This is especially true when reasoning about heterogeneous agents. A *centralized* approach that tries to represent the whole system in a single model is bound to run into scalability issues, especially since the runtime of most temporal logic strategy synthesis approaches scales at least exponentially with respect to the input size. To avoid the so-called *state space explosion problem*, we should aim to avoid constructing a global, centralized model of the system at hand as much as possible. *Decentralized* methods generally avoid constructing a global model, but they come with their own challenges, as they usually involve separating tasks into independent components, communication between agents or other methods that ensure global integrity of locally computed strategies.

How the interaction between agents is modelled plays a big role in how solutions can be found and what assumptions about the agents have to be made. As depicted in Figure 1.1, cooperative and adversarial approaches are the two extreme ends of the agent interaction spectrum, with so-called rational or assume-guarantee approaches being somewhere between the two.

Cooperative approaches assume full control over all agents of the system, orchestrating them jointly as a team. This approach can be very powerful, but in many

scenarios, this assumption simply does not hold, e.g. when a human actor with their own goals is a part of the system. If one of the agents is changed or deviates from the behaviour assumed by the strategy, a cooperative approach would usually need to replan.

In scenarios where it is important to react to actions of uncontrollable agents, we sometimes assume those agents to act as *adversaries*, trying to hinder the agent in question from reaching their objective. Adversarial approaches make a worst-case assumption to provide guarantees under all circumstances, even if in actuality agents are not acting in adversarial fashion. Strategies using this approach are able to handle every circumstance if they exist. In many realistic scenarios, such strategies simply do not exist [2]. For example, an autonomous car could not model other cars on the road as adversaries, as there usually is no way to avoid a crash if all other cars are aiming for that crash to happen. Similar to economic game theory, rational synthesis [3] assumes agents to act rationally, i.e. maximizing their utility first and only then acting adversarial towards other agents. Slightly closer to the cooperative approach, assume-guarantee synthesis approaches [4] have agents provide guarantees to other agents, given certain assumptions are met.

In this thesis, we research the question:

How do we model the interaction of multiple agents in such a way that we can both obtain global guarantees on specification satisfaction and retain scalability of the resulting method?

Human Agents

Another challenge is representing and modelling a human agent in the system. Since their actions and preferences are heavily influenced by the context of the situation, it is hard to provide an explicit model of human behaviour, even in simple situations. Other common assumptions, such as rationality of actors, have proven to not hold true; not in economic situations [5], but especially not in situations in the real world that might be perceived as potentially dangerous. For example, humans might take the risk of driving over the speed limit because they have an unmodeled strict deadline that they have to meet.

When multiple robots compute strategies, they might be able to exchange information in the form of numbers and logical formulas with each other. With a human actor, these mathematical objects have to be translated into another human-understandable form of communication. Whether such information is best conveyed via non-verbal means, such as light and sound, or verbal communication is another factor that has to be considered.

With these considerations in mind:

How do we represent human actors in our methods without making wrong or unnecessary assumptions about their preferences or capabilities?

Task Specification

A key design decision is the choice of specification language. This impacts both what properties can be expressed and how a solution can be obtained. These factors are usually connected, meaning that the more expressive a specification language is, the more computationally complex algorithms are needed to solve the resulting task-planning problem.

Many robotic tasks involve a complex interplay of spatial and temporal patterns [6]. Already allegedly simple tasks require robots to consider both spatial as well as temporal constraints. Menghi et. al. [7] summarize common temporal patterns of robotic tasks. These patterns include surveillance, avoidance or reacting to certain situations among others. Specifically, manipulation tasks include temporal sequences to assemble parts, recurring triggers (such as continuously moving certain parts), and time-bounded actions [8].

With these factors in mind, temporal logics seem like a suitable choice to specify high-level robotic tasks that go beyond waypoint navigation. Which temporal logic language to choose as a specification language depends on the task at hand.

Linear Temporal Logic is defined over boolean propositions. With these propositions only being able to express *true* or *false*, LTL is the simplest temporal logic language considered in this thesis. However, it can be difficult to synthesize reactive strategies for temporal logic languages richer than LTL without limitation to a fragment of the language. LTL offers reactive strategy synthesis algorithms for the full specification language.

Signal Temporal Logic (STL) is defined over continuous signals as basic building blocks and thus allows to better represent continuous constraints in the real world. In addition, their quantitative evaluation allows to reason over how close a formula is to being satisfied or violated. However, synthesizing strategies from STL specifications usually involves restriction to a weaker fragment of the language as well as a translation into an optimization problem, which can be hard to explicitly modify. In addition, STL formulae can be hard to specify, as defining constraints and goals as functions is tedious and error-prone.

Another facet of specification languages to consider is how a system can give feedback to the user. In model-based formal methods it is quite common to produce counterexamples in the case of infeasability of a task. These counterexamples can be used to refine the specification or change the underlying system. In a similar vein, the methods in this thesis make assumptions over other actors to prevent infeasibility of a given task. These assumptions have to be communicated between actors. If one of those actors is a human, they have to be able to understand the artifact that the method produces.

For the specific case of object-oriented tasks, we ask:

> How do we adapt temporal specification languages to enable users to intuitively specify object-oriented tasks and algorithms to effectively communicate requests or constraints to human operators?

1.2 State of the Art

Formal Methods in Robotics Formal methods-based synthesis from temporal logic specification has been successfully deployed in a number of robot task-planning scenarios [9] ranging from manipulation that is robust against interventions of a human operator [10] to multi-robot task planning with pre-failure warnings that give users insight as to why a specification may be violated in the future [11]. Temporal Logic-based methods applied to ensure safety and correctness of a range of control systems [12]; LTL has been used for path planning in hybrid systems [13], linear systems [14, 15], as well as sampling-based methods [16].

Multi-Agent Task Planning under LTL Specifications Task assignment, task planning, and coordination of multiple autonomous, possibly heterogeneous agents is an active research field [17] with applications in multi-robot exploration [18] or automated warehouse solutions [19]. Tasks for a multi-agent team may vary from simple reachability to more sophisticated ones described in, for instance, PDDL [20] or STRIPS [21]. However, these methods are not well-suited to specify temporal sequences of events or temporal constraints. Since we care about guaranteeing specified temporal sequences in the vicinity of other robots or humans, we aim to leverage temporal logic-based formal synthesis methods instead.

Different approaches have been used to apply temporal logic-based task planning for multi-agent systems in a scalable manner. Chen et al. [22] distribute a single LTL task between agents so they can be executed independently. This method is only complete for a a particular fragment of LTL. Ding et al. compare the advantages of a top-down and a bottom-up approach of decomposing and distributing LTL tasks in multi-robot system [23]. Guo et al. propose a leader-follower-scheme and switch leaders during execution to enforce satisfaction of all desired formulas [24, 25]. Lin et al. synthesize reactive strategies with integrated motion planning for a fragment of LTL [26]. Lacerda et al. compose Petri nets representing multi-robot team restrictions [27]. Schillinger et al. work with probabilistic models of agents capturing uncertainties, and use multiple different methods in their works [28, 29], including usage of efficient fragments of LTL, reinforcement learning to optimize the agents' behaviour, auctioning of tasks and keeping specification automaton and agent model separated. Vasile et al. schedule fleets of robots using Time Window Temporal Logic [30]. Kantaros et al. propose a sampling-based motion planning method for large-scale multi-robot systems under temporal specifications [31]. DeCastro et al. encode high level motion tasks in LTL in order to synthesize reactive strategies for a team of robots [32]. Tumova et. al. propose a receding horizon approach with a centralized product to limit the scope of planning in [33, 34].

All of these methods usually choose either the cooperative or the adversarial approach to modelling agent interactions. In this thesis, we aim to explore if other approaches to agent interaction lead to algorithms that promote scalability.

Reactive Synthesis Reactive synthesis is a field within formal synthesis where the goal is to synthesize strategies that can satisfy a specification while reacting to an uncontrollable environment. To provide guarantees on desired outcomes, the environment is assumed to be adversarial. With multi-agent systems in mind, reactive synthesis techniques are a promising way to obtain strategies for agents reacting to other, adversarial agents. The methods in this thesis are often based on key ideas in reactive synthesis. In this paragraph, we briefly summarize important developments in the field that acts as a basis for our techniques.

The first solution to the reactive synthesis problem was described in [35] based on Safra's determinization procedures [36]. While theoretically complete and correct, Safra's procedures are hard to represent efficiently in an algorithmic fashion, leading to the design of multiple procedures avoiding this kind of determinization. Those *safraless* methods provided the same theoretical worst-case-runtime, but showed improvements in implementations. The first *safraless* procedure [37] has been implemented in [38]. Other examples of safraless procedures make use of different techniques, such as antichains [39] implemented in [40] or Binary Decision Diagrams [41]. One of the main challenges of strategy synthesis remains scalability, as the problem is 2EXP-complete [35]. This doubly exponential growth does not only occur in the theoretical worst case, but also in application [1]. This issue has been tackled by limiting the expressiveness of the logical language to a more efficient fragment called GR(1) [42] or trying to subdivide the problem into smaller subproblems and composing the solutions to those. Kupferman describes a compositional method of checking for emptiness of tree automata [43]. In [41], a parity game is compositionally constructed and solved with BDD. Other works in this field include a compositional method with partial observability of the components [44].

Assumptions in Strategy Synthesis With reactive synthesis as a basis for the techniques of this thesis, we model other agents as adversaries to obtain worst-case guarantees on task satisfcation. However, adversarial behaviour can be too strict in many cases, especially with multiple, sufficiently powerful adversaries. Strictness of adversarial behaviour can be alleviated by providing guarantees only when certain assumptions on the environment are met. Refining assumptions over the environment has been used in research related to GR(1) [45]. Alur et. al. [46] refine GR(1) specifications in an iterative fashion through counterexamples. *Assume-Guarantee* synthesis generates strategies in two-player games under the assumption that the environment follows a certain specification [4]. Raman et al. provide minimal feedback for unachievable high-level robot behaviours. Here, a human designer adds assumptions by refining the specification [47, 48]. In assume-guarantee compositional design, assumptions and guarantees are used as an interface for system components [49, 50] and are usually designed by hand, but can also be obtained through automata learning [51] or abstraction refinement [52].

In order for assumptions on other parts of the system to be meaningful, they have to be as *least-limiting* or *maximally-permissive* as possible. Maximally per-

missive supervisors are used to synthesize flexible strategies for manufacturing systems [53]. The notion of least-limiting supervisors or maximally-permissive strategies has been a topic of research in other communities, as well. Tumova et. al. [54] formalized least-limiting strategy advisers for safety objectives. However, it is hard to capture the notion of least-limiting or maxmimally permissive for ω-regular objectives, as in general there exists no unique solution [55, 56]. Bernet et. al. [55] defined the notion of *permissive strategies* for parity games as strategies that subsume the behaviour of all memoryless strategies. Chatterjee et. al. [56] generate weakest sufficient assumptions on the environment. The presented works either use assumptions as an interface for communication or they generate minimal assumptions, but don't communicate them the environment. In this thesis, we generate minimal assumptions and we iteratively communicate them to other actors. To the best of our knowledge, no other works in the field have done this.

Specification Refinement In many situations, formal synthesis fails to find a correct-by-design strategy e.g., due to high uncertainty in the environment model, or infeasibility of the task. Solutions to this challenge include refining the model, i.e. adding assumptions on the robot or its environment, repairing the specification itself, or finding stategies that violate the specification as little as possible. For instance, Raman et. al. [57] automatically provide feedback through structured English expressions and allows a human user to add a refined description of the environment increasing the chances to find a correct-by-design strategy. Sharma et. al. [58] map natural language sentences to transformations of cost functions that enable users to correct goals and update motions to incorporate user preferences and recover from planning errors. Some other works propose quantitative evaluation of LTL specification satisfaction, leading to finding the least-violating plans in the given model [59, 60, 61]. To our best knowledge, none of the works on formal methods-based robot task planning with a human in the loop focuses on explicitly communicating advice, that – if followed – enables guarantees.

Task Planning with Human in the Loop Task planning for autonomous robots with a human in the loop has been approached from various angles. Many works develop a model of the human, motivated either through data or psychological findings. For instance, Buisan et. al. [62] leverage hierarchical task networks to plan for a task while also considering and emulating the human decision, action, and reaction processes. In formal methods-based task planning, many approaches rely on models of the systems they analyze, including the humans. For example, Feng et. al. [63] model a human operator in a UAV control scenario via a Markov Decision Process based on key characteristics such as the operator's workload, proficiency, and fatigue. Junges et. al. [64] use probabilistic human models obtained through reinforcement learning on previously recorded observations of human trajectories. However, explicitly modelling humans is usually not very robust to small changes

in the context. Many assumptions, such as that the humans behave rationally also do not hold in real-world experiments [5].

In contrast, some other works that focus on formal methods-based task planning with a human in the loop do not model the human explicitly, but introduce reactive re-planning, such as in [10]. Bohg et. al. [65] propose an object-centric reactive task-planning method that produces sequential manipulation plans relative to object positions that remain valid even if the objects are moved, for example by a human. He et. al. [66] synthesize strategies reacting to the human behaviour. They propose an efficient, compositional approach to perform synthesis for finite-horizon tasks.

In this thesis, we take the latter approach. We avoid to explicitly model human preferences or capabilities and instead compute only minimal assumptions and restrictions that are absolutely necessary for specification satisfaction of the robot.

Advice-Giving in Human-Robot Interaction After computing minimal assumptions on human behaviour, we have to be able to effectively communicate these assumptions to the human actor. How and when this is done represents an interesting field of research on its own. Several works study the effect of robots giving advice to human in a 1-on-1 human-robot interaction. Barlas et. al. [67] studied how robot-instructed actions alter the humans sense of agency. Strait et. al. [68] investigate the effects of robot communication strategies in advice-giving situations based on robot appearance, interaction modality and distance. Torrey et. al. [69] experimentally compared help-giving strategies. Tucker et. al. [70] optimize agents to communicate according to utility, informativeness, and complexity. These works generally study how advice should be conveyed and what factors influence the acceptance of such advice, but unlike us, they do not focus on what advice should be given in the first place.

Object-Oriented Task Planning While rich and expressive, temporal logic specifications can be tedious and hard to parse. While the complexity of temporal operators is hard to mask without losing technical correctness, we argue that handling of boolean propositions or signal functions could be improved through domain-specific primitives that are closer to natural language. In this thesis, we focus these efforts for the specific use-case of object-centric manipulation tasks, such as setting a table. In object-centric tasks, positions of objects can be described through their relation to other objects. These spatial relations can roughly be translated into natural language, e.g. *left of, close to* or *touching*. Various approaches incorporate spatial relations between objects into robotic frameworks [71, 72, 73, 74, 75, 76], e.g., what it means when two boxes are touching. These spatial relations help robots to understand desired goal configurations of objects [77] and to predict an action's outcome [78]. To this end, many approaches perform object-centric planning [79, 80, 81], where planning is performed in the task space of the object itself [82, 83, 84].

Spatial relations between objects range from high-level geometric relations, such as an object being left of another [85, 86], to relations that describe which objects support another object within a 3D structure [87, 88, 89]. Learning spatial relations from data and demonstrations is difficult [90, 91, 92] due to the variety of relations and the complexity of separating similar relations.

1.3 Thesis Contributions and Structure

In this thesis, we approach the problem of deploying autonomous systems that share the environment with other actors as a multi-agent task planning problem. Through the desire to have these systems act safely correctly satisfy their tasks, we motivate the use of formal methods. We explained key design decisions and challenges when applying formal methods to MAS task planning in Section 1.1. We develop decentralized strategy synthesis algorithms that are able to handle multiple robots as well as humans. With reactive synthesis as a starting point, we compute strategies for each agent that are able to react to unknown behaviour. We alleviate strictness of the assumed adversarial behaviour by making assumptions only where necessary. Through communication of logical assumptions, we provide conditional guarantees on the overall system given that each participant follows the assumptions made. We define minimal assumptions and communicate them as least-limiting advisers as a unified method of computing logical assumptions for both robot-robot and robot-human interaction. Lastly, we define a spefication framework for object-centric tasks that simplifies specification and communication of logical constraints between a robot and a human operator.

In Chapter 2, we define foundational elements and algorithms of the techniques presented in this thesis. We present different models and how they can be used and describe basic algorithms that we build on in the following chapters.

In Chapter 3, we propose a compositional solution to the strategy synthesis problem for LTL specifications in the cooperative (heterogeneous) multi-agent scenario. A main challenge of the general strategy synthesis approach is the state-space explosion occurring during construction of a global model for agents with different, mutually dependent goals. Given a set of agents and their individual goal specifications represented through a local model and an LTL formula, we compute a compliant set of strategies that fulfill each agents' goal specification. We exploit dependencies and independencies between agents in a multi-agent scenario to tackle scalability issues. We avoid the state-space explosion by computing individual solutions for each agent separately and then composing these solutions. During the initial strategy computation, assumptions over the states of other agents not represented in the local model are generated only where needed. These assumptions are resolved during composition of the individual solutions to assure compliance of the computed strategies. To ensure completeness of our approach, we maintain the set of all candidate strategies in a suitable representation. During composition, agents' state spaces and candidate strategies are iteratively pruned, hindering the

state-space explosion. The effectiveness of this approach is demonstrated in several simulation case studies and compared to the classical, monolithic approach. We show that this approach is effective when dependencies between agents are high.

In Chapter 4, we approach a similar problem from the opposite side, aiming to provide scalability when interactions between agents are sparse. We define the notion of minimal assumptions and use them both for interactions between computational agents and human actors. A key step is treating safety and liveness separately, by synthesizing necessary safety and fairness assumptions and iteratively exchanging them in the form of advisers between the agents and communicating them to humans.

In Section 4.3, we focus on correct-by-design robot task planning from finite Linear Temporal Logic (LTL_f) specifications with a human in the loop. Since provable guarantees are difficult to obtain unconditionally, we take an assume-guarantee perspective. Along with guarantees on the robot's task satisfaction, we compute the weakest sufficient assumptions on the human's behavior. We approach the problem via a stochastic game and leverage algorithmic synthesis of the weakest sufficient assumptions. We turn the assumptions into runtime advice to be communicated to the human. We conducted an online user study and showed that the robot is perceived as safer, more intelligent and more compliant with our approach than a robot giving more frequent advice corresponding to stronger assumptions. In addition, we show that our approach leads to less violations of the specification than not communicating with the participant at all.

In Section 4.2, we propose a decentralized solution to a high-level task-planning problem for a multi-agent system under a set of possibly dependent LTL_f specifications. We propose an approach where the problem is turned into a number of individual two and a half player stochastic games with reachability objectives. If almost-surely winning strategies cannot be found for them, we deploy so-called least-limiting advisers to restrict agents' behaviours. We avoid the state-space explosion problem by computing advisers locally in each game, independently of the model and specification of other agents. The solution is sound, but conservative. We demonstrate its scalability in a series of simulated scenarios involving cleaning of an office-like environment.

In Chapter 5, we focus on intuitive task specifications for object-oriented tasks. Various real-world robotic tasks are difficult to monitor and plan for due to the complex interplay of spatial and temporal requirements. For instance, arranging a dining table requires manipulating objects of different shapes in a predefined temporal order, e.g., first placing the mug and then filling it with coffee, while satisfying spatial relations such as forks need to placed left of plates. Existing solutions either do not consider tasks in an object-centric fashion, depend on manual definition of object relations, or are not expressive enough to capture a rich variety of tasks. To address these shortcomings, we propose the spatio-temporal framework SpaTiaL that unifies the specification, monitoring, and planning of various real-world object-oriented robotic tasks. SpaTiaL is expressive enough to capture the diverse spatial relations between objects and temporal patterns of tasks while providing

real-time monitoring and online planning. Through spatial requests, we leverage SpaTiaL's capabilites to request understandable, minimal help from human actors in an otherwise autonomous planning loop. We demonstrate the benefits of SpaTiaL in multiple real scenarios as well as in simulated object pushing and pick-and-place tasks.

Source code for all presented methods is available open-source [1]. We wrap up the thesis and conclude with directions for future research in Chapter 6.

[1] See links in corresponding chapters, all repositories available at `https://github.com/KTH-RPL-Planiacs`

Chapter 2

Preliminaries

In model-based, automata-theoretic strategy synthesis, we start with a logical specification of the desired behaviour of our system. Together with a model of the system at hand, formal strategy synthesis reasons on a *product* of the model and a representation of the specification. This workflow is depicted in Figure 2.1.

The methods in this thesis focus on discrete, finite-state models. Such models can be obtained by segmenting the environment into regions of interest or translating the capabilities of the robot into states, e.g. *gripper aligned* or *object in gripper*. The specification formula is translated into an *automaton*. Together, the model and the automaton are unified into a *product*. In the product, a strategy is searched. This strategy could be a sequence of actions or a mapping from the current state and a history of past events to an action. By executing the actions given by the strategy, the executing system can guarantee satisfying the specification.

In this chapter, we first introduce basic definitions for discrete models with finite states. We introduce different mechanisms to model uncertain outcomes,

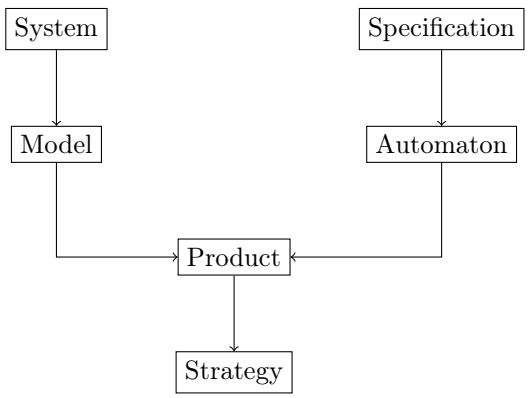


Figure 2.1: Standard workflow of strategy synthesis.

such as non-deterministic or probabilistic outcomes or games between the system and an imaginary, adversarial opponent. Secondly, we define syntax and semantics of Linear Temporal Logic and introduce automata as a discrete structure to reason about LTL formulae. Lastly, we demonstrate execution of the workflow presented in Figure 2.1 in its simplest form.

2.1 Models

Model-based strategy synthesis methods reason on a model of the system under consideration. Much like any other mathematical formalism, this model is an abstract description of the behaviour of our system. The methods presented in this thesis utilize models with discrete and finite states. Such models can be obtained through abstraction and segmentation of the environment and through manually modelling the capabilities of the system, such as a robot. For example, a common approach is to generate a triangulation of the environment for a given dynamical system such that low-level controllers are always able to navigate between triangular regions [93, 94]. But not every model has to be constructed in such manner. For example, with a holonomic robot in a simple office environment, a graph representation of rooms and corridors with a node for each room and an edge between rooms that are connected can suffice for high-level planning. For example, Hawes et. al. generate a topological map on top of a occupancy grid map [95] for the purposes of long-term autonomy.

In this section, we first describe *transition systems* as the simplest model we consider. Transitions systems utilize nondeterminism to model unknown outcomes of actions. We then continue to introduce probabilities on unknown outcomes of actions and define *Markov decision processes*. Finally, we introduce the notion of an adversarial opponent, frequently called player 2 in *stochastic games*.

Transition Systems

In computer science, transition systems are a commonly used to model the potential behavior of discrete systems. They are basically directed graphs where nodes represent states, and edges model possible transition between these states. Many different types of transition systems or *Kripke structures* are proposed in literature [96]. In this work, we consider transition systems where transitions are labelled with actions and states are labelled with atomic proposition. Without loss of generality, we also assume a single initial state. A state describes some information about a system at a certain moment of its behavior. For instance, a discrete state of a robot in an office environment could describe the current room the robot is in. The transitions might model which rooms are connected such that the robot can directly navigate from one room to another. Each transition is labelled with an action. Intuitively, we say that choosing a certain action in a certain state will lead to a state change defined by the transitions. If in some state an action has

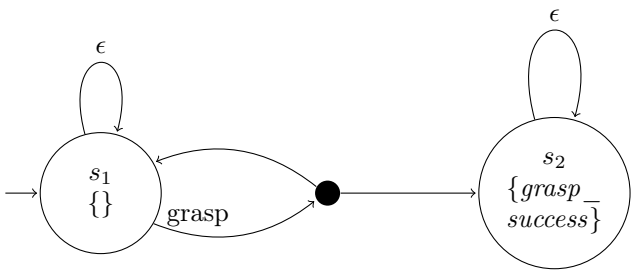


Figure 2.2: A non-deterministic transition system modelling a grasp attempt.

multiple outgoing transitions associated with it, the outcome of that action is partially unknown. For example, a robot choosing to navigate into the room to the left might end up in that room or get stuck on the way and end up in a failure state. The outcome is chosen *non-deterministically*, which means that the outcome of this selection process is not known, and no statement can be made about the likelihood with which a certain transition is selected.

Definition 1 (Transition System)**.** *A (non-deterministic) transition system (TS) is a tuple* $T = (S, Act, \delta, s_0, AP, L)$*, where*

- S *is a finite set of states,*
- Act *is a finite set of actions,*
- $\delta : S \times Act \to 2^S$ *is the transition function,*
- s_0 *is the initial state,*
- AP *is a finite set of atomic propositions and*
- $L : S \to 2^{AP}$ *is a state labeling function.*

We call a transition system deterministic if $|\delta(q, a)| \leq 1, \forall q \in Q, \forall a \in Act$.

Example 1 (Non-deterministic Grasping)**.** *Depicted in Figure 2.2 is a simple model of a robot that can grasp an object. For this model, we consider a single atomic proposition grasp_success. The model starts in state* s_1*. The label* $L(s_1) = \{\}$ *indicates that the grasp has not yet succeeded. The action* ϵ *denotes waiting or not doing anything. The action* grasp *has a non-deterministic outcome. When using the action* grasp*, the state can change to* s_2*, where the grasp succeeded, but it can also change to* s_1*, meaning that it sometimes fails to grasp the object as intended.*

Markov Decision Process

With further knowledge about the system at hand, such as data from previous system executions, the transition system model can be expanded by replacing the non-deterministic resolution of action outcomes with probabilistic behaviour.

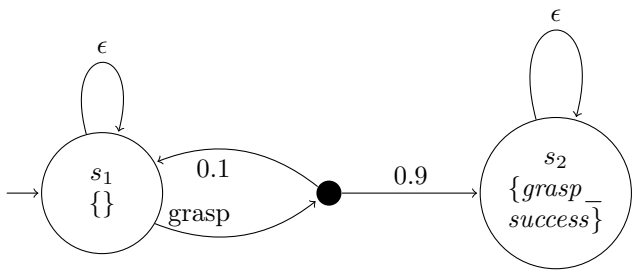


Figure 2.3: A Markov decision process modelling a grasp attempt.

Definition 2 (Markov Decision Process)**.** *For a finite set X, a probability distribution on X is a function $\delta : X \to [0,1]$ such that $\sum_{x \in X} \delta(x) = 1$. We write $Supp(\delta) = \{x \in X \mid \delta(x) > 0\}$ for the support set of δ. We denote the set of probability distributions on X by $Dist(X)$.*

A markov decision process (MDP) is a tuple $M = ((S,E), Act, \delta, s_0, AP, L)$ where

- *(S,E) is a finite, directed graph,*
- *$S = S_1 \uplus S_p$ is the states space partitioned into states controlled by the player and probabilistic states,*
- *$E = E_1 \uplus E_p$ is the set of edges, partitioned analogously,*
- *Act is a set of actions,*
- *$\delta : S_p \to Dist(S_1)$ is a probabilistic transition function,*
- *$s_0 \in S_1$ is the initial state,*
- *AP is a set of atomic propositions and*
- *$L : S_1 \to 2^{AP}$ is a* state labelling function.

We define $E_1 \subseteq S_1 \times Act \times S_2, E_2 \subseteq S_2 \times Act \times S_p, E_p \subseteq S_p \times S_1$. For all $s_p \in S_p$ and $s_1 \in S_1$, we have $(s_p, s_1) \in E$ iff $\delta(s_p)(s_1) > 0$. The state labelling function assigns to every state the atomic propositions which hold true in there. For $s \in S_1$, we define the set of enabled *actions as $Act(s) = \{a \in Act \mid (s,a,s') \in E, s' \in S\}$.*

Example 2 (Probabilistic Grasping)**.** *Depicted in Figure 2.3 is an MDP model of the same grasp attempt. Imagine that through previous observation, we concluded that the grasp attempt fails in roughly* 10% *of cases. We can represent this by making the following modifications to Figure 2.2: After choosing the action* grasp *in state s_1, the model evolves to a probabilistic state depicted in solid black. From there, two transitions are assigned with the probabilitites of either switching to state s_2 or returning back to state s_1.*

Non-Determinstic versus Probabilistic Behaviour

The key difference between TS and MDP is how unknown outcomes of actions are handled. At first glance, enhancing the model with data about probabilities seems like a strict improvement of the model. However, both non-deterministic and probabilistic outcomes can be powerful modelling tools depending on the purpose of the model.

Example 3 (Faulty Grasping)**.** *The difference is best explained through a simple example using the grasp model from Figure 2.2 and Figure 2.3: We are trying to come up with a stategy to observe the proposition grasp_success. To satisfy the formula, we have successfully grasp the object and transition to state s_2. Starting in state s_1, we can simply choose to* grasp *and hopefully transition to state s_2. In state s_2, we don't have to do anything, so we can choose the action ϵ. In the non-deterministic approach of the TS, there exists an outcome where successfully grasping and entering the state s_2 never happens, even if tried infinitely many times. In this setting, satisfying the task is impossible to guarantee, as it might occur that we never leave the state s_1. However, in the probabilistic setting, the chance to eventually leave the state s_1 is* 1*, if tried infinitely often, so the overall probability to safisfy the task is 1.*

In this example, the MDP probably models the system better assuming we can retry the grasp infinitely often. Probabilistic behaviour would however not be well-suited to model unknown outcomes related to the actions of other agents that might react to the behaviour of the system.

Games

If we want to incorporate both probabilistic outcomes of actions as well as non-deterministic behaviour of another agent into a single model, we can model the system and its interaction with the environment as a *stochastic game*.

Definition 3 (Labelled Stochastic Game)**.** *A two and a half player stochastic game (SG) is a tuple $G = ((S, E), Act, \delta, s_0, AP, L)$ where*

- *(S, E) is a finite, directed graph,*
- *$S = S_1 \uplus S_2 \uplus S_p$ is the states space partitioned into player 1, player 2, and probabilistic states,*
- *$E = E_1 \uplus E_2 \uplus E_p$ is the set of edges, partitioned analogously,*
- *Act is a set of actions,*
- *$\delta : S_p \to Dist(S_1)$ is a probabilistic transition function,*
- *$s_0 \in S_1$ is the initial state,*
- *AP is a set of atomic propositions and*
- *$L : S_1 \to 2^{AP}$ is a* state labelling function.

We define $E_1 \subseteq S_1 \times Act \times S_2, E_2 \subseteq S_2 \times Act \times S_p, E_p \subseteq S_p \times S_1$. *For all* $s_p \in S_p$ *and* $s_1 \in S_1$, *we have* $(s_p, s_1) \in E$ *iff* $\delta(s_p)(s_1) > 0$. *The state labelling function assigns to every state the atomic propositions which hold true in there. For* $s \in S_1 \cup S_2$, *we define the set of* enabled *actions as* $Act(s) = \{a \in Act \mid (s, a, s') \in E, s' \in S\}$.

If each state in S_p has a single outgoing edge, G is a *deterministic* game. In these cases, we can omit S_p from the definition. If each state in S_2 has a single outgoing edge, we can omit player 2 and their states S_2 from the game. Then, the stochastic game falls back to a Markov Decision Process. In the same fashion, removing decisions for both players yields a *Markov Chain* (MC). Therefore, we can call MDPs and MCs are one- and zero-player SGs, respectively.

The decentralized methods in Chapter 4 reason explicitly over sets of edges, so we introduce the notion here. If an explicit representation of edges E is not necessary, it can be omitted for the sake of readability.

Plays, Strategies and Traces

The following definitions of *plays*, *traces*, *strategies* and *objectives* use stochastic games, but all definitions can be applied to transition systems or MDPs in the expected way.

Definition 4 (Play)**.** *A* play *of game* G *is an infinite sequence* $\pi = s_0 s_1 s_2 \dots$ *of states such that* $(s_k, a, s_{k+1}) \in E_1 \cup E_2$ *or* $(s_k, s_{k+1}) \in E_p$ *for all* $k \geq 0$. *We use* Π *for the set of all plays.*

Definition 5 (Trace)**.** *Given a play* $\pi = s_0 s_1 s_2 \dots \in S^\omega$ *on a labelled model, we define its trace as* $\tau(\pi) = L(s_0)L(s_1)L(s_2)\dots \in (2^{AP})^\omega$.

Definition 6 (Joint Trace)**.** *Given* n *plays over labelled models such that their* $AP_i \cap AP_j = \emptyset$, $\forall i \neq j$ *and* n *traces produced by plays* $\tau_i = o_{1,i} o_{2,i} \dots \in (2^{AP_i})^\omega$, *we define the joint trace as a sequence:*

$$\tau_{joint} = (o_{1,1} \cup o_{1,2} \cup \dots \cup o_{1,n})(o_{2,1} \cup o_{2,2} \cup \dots \cup o_{2,n}) \dots$$

Definition 7 (Strategy)**.** *A deterministic finite-memory strategy for player* i *is a function* $\lambda : S_i^* \to Act$, *where* $\lambda(s_1 \dots s_i) \in Act(s_i)$. *A memoryless strategy is such that* $\lambda(s_1 \dots s_i) = \lambda(s_i)$ *for all* $s_1 \dots s_i \in S_i^*$ *and we use* $\lambda : S_i \to Act$.

In this work, we only consider *deterministic* strategies $\lambda : S^* \to S$. A strategy is *memoryless* if it does not depend on the history of the play but only on the current state and can be represented as a function $\lambda : S \to Dist(S)$. We denote by $\mathcal{A}$ and $\mathcal{B}$ the sets of strategies for player 1 and 2. Given a starting state s and strategies α, β for each player, the outcome of the game is an induced Markov Chain $\pi_s^{\alpha,\beta}$. We write $Outcome(s, \alpha, \beta) \subseteq \Pi_s$ for the set of plays that start from s and where both players follow their respective strategies α and β. Strategies λ_1, λ_2 define plays that follow them $\pi_{\lambda_1,\lambda_2} = s_0 s_1 s_2 \dots$, where $(s_k, \lambda_i(s_k), s_{k+1}) \in E_i$ if $s_k \in S_i, i = 1, 2$. For

a set of plays $\mathcal{E} \subseteq \Pi$, state s, and strategies λ_1 and λ_2, we denote the probability that a play beginning in s and following λ_1, λ_2 belongs to $\mathcal{E}$ by $Pr_s^{\lambda_1,\lambda_2}(\mathcal{E})$.

Definition 8 (Objective)**.** *An* objective *for a player is a set $\psi \subseteq \Pi$ of plays that are winning for that player.*

Given a set $F \subseteq S$ of states, a *safety* objective $Safe(F) = \{s_0 s_1 s_2 \cdots \in \Pi \mid \forall k \geq 0, s_k \in F\}$ requires only states in F to be visited. The *reachability* objective $Reach(F) = \{s_0 s_1 s_2 \cdots \in \Pi \mid \exists k \geq 0, s_k \in F\}$ requires some state in F eventually to be reached.

When using probabilistic models, we restrict the focus of this work to almost-surely winning strategies.

Definition 9 (Almost-Sure Winning)**.** *Given an objective ψ, we define a strategy λ_1 of player 1 as* almost-sure winning *from state s, if for every strategy λ_2 of the other player, $Pr_s^{\lambda_1,\lambda_2}(\psi) = 1$.*

Given an objective ψ, we denote the set of states from which player 1 has an almost-sure winning strategy as $\langle\langle 1 \rangle\rangle(\psi)$. Given an objective ψ, the *cooperative winning set* $\langle\langle 1, 2 \rangle\rangle(\psi)$ is the set of states s where there exists a strategies λ_1, λ_2 for players 1 and 2, such that $Pr_s^{\lambda_1,\lambda_2}(\psi) = 1$. Stochastic games with a reachability objective can be solved by state-of-the-art tools, such as PRISM-games [97].

2.2 Temporal Logic

Equipped with different modelling techniques, we now move on to focus on defining what a logical specification is. We specify desired properties and constraints of our system by requiring or disallowing certain sequences of events over time. This includes expressions that roughly translate to *eventually reach the goal area*, *never enter this dangerous area* or *whenever you receive a request, reply to it eventually.*

When approaching logical reasoning over events in time in a formal way, the first and foundational question is concerned with the actual formalization of time and statements governing temporal events. One fundamental distinction is the distinction between linear and branching models. In linear models, the flow of time can be depicted as a line, while branching models allow a tree-like representation, supporting the view that the past is fixed and hence linear while the future is branching into multiple possible futures. Another important decision one has to make in devising a formal system for time is that between discrete and dense or continuous models of time. In discrete models, every time instant has an immediate successor and predecessor, while in continuous models, between any two subsequent time instants, there is another instant. Discrete models of time are prevalent in computer science, while continuous models are more common in natural sciences, philosophy and control theory. A last foundational choice is that of instants or intervals as atomic units of time. Famously, Zeno's arrow paradox [98] only arises since he defined time through instants.

Tense Logic was the first formal logical language reasoning about time and introduced 1957 by Prior [99][100]. Prior's motivation for inventing tense logic was largely philosophical, and his ideas were strongly inspired by the use of tense in natural language [101]. The novel contribution of Prior's work was treating propositions as tensed, meaning they inhibit a reference to time. On a technical level, this was achieved by the introduction of temporal operators into the language. While Prior himself referred to his invention as *Tense Logic*, nowadays the more general expression *Temporal* Logic is being used [101].

In this work, we will focus on *Linear Temporal Logic*, which uses a discrete, instant-based and linear model of time, but here we will briefly elaborate on branching-time and interval-based models of time to offer a more complete overview of temporal logics in general. A more historical overview of temporal logics in philosophy can be found in Øhrstrøm and Hasle [98] or the Stanford Encyclopedia of Philosophy [101] and the references therein. The topic of branching versus linear time in the field of computer science has been discussed by Vardi [102].

Linear Temporal Logic

Originally proposed for the formal analysis of computer programs, *Linear Temporal Logic* (LTL) was introduced in 1977 by the computer scientist Amir Pnueli [103], featuring only the temporal operators X and U. While Pnueli proposed the application of temporal logics to the specification and verification of reactive and concurrent programs and systems, application areas have since then expanded to other areas, such as the task planning of autonomous systems. LTL is interpreted over linear traces of such programs or systems. This formalism is suitable for our goal of finding strategies for an autonomous system that generate traces satisfying a given specification.

Definition 10 (LTL Syntax)**.** *Formally, we define an LTL formula over a set of atomic propositions AP as:*

$$\phi \stackrel{def}{=} true \mid a \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid X\phi \mid \phi_1 U \phi_2, \quad a \in AP,$$

where *true* is a symbol for unconditional truth, $\neg$ is negation, $\wedge$ represents conjunction of two formulas, X represents a formula holding true in the *next* step and U requires a formula to hold true until another formula holds true. The full power of propositional logic is obtained through the operators $\neg$ and $\wedge$. Operators $\vee, \rightarrow, \leftrightarrow$ are defined as usual:

$$\begin{aligned}
\phi_1 \wedge \phi_2 &\stackrel{\text{def}}{=} \neg(\neg\phi_1 \vee \neg\phi_2) \\
\phi_1 \rightarrow \phi_2 &\stackrel{\text{def}}{=} \neg\phi_1 \vee \phi_2 \\
\phi_1 \leftrightarrow \phi_2 &\stackrel{\text{def}}{=} (\phi_1 \rightarrow \phi_2) \vee (\phi_2 \rightarrow \phi_1)
\end{aligned}$$

Furthermore, $F\phi$ represents ϕ holding *eventually* in the future and $G\phi$ represents ϕ *always* holding from now on.

$$F\phi \stackrel{\text{def}}{=} \mathit{true}\, U \phi$$
$$G\phi \stackrel{\text{def}}{=} \neg F \neg \phi$$

Semantics of Linear Temporal Logic

We interpret LTL formulae over discrete instants in time using the atomic propositions AP. For each discrete instant in time, we observe a subset of the atomic propositions that hold true. We call the set of all possible subsets the *alphabet* $\Sigma = 2^{AP}$. A sequence of symbols from the alphabet Σ is called a *word*.

Classical LTL is interpreted over infinite sequences words. The set of all possible infinite word words is denoted as Σ^ω [96]. The logical language LTL_f is interpreted over finite words $\sigma \in \Sigma^*$ [104].

For a word σ, $\sigma[i]$ represents the i-th symbol of that word, i.e. the subset of propositions that hold true at time step i. $\sigma[i,j]$ is the sequence of symbols between the i-th and j-th time step. We denote a word σ *satisfying* an LTL formula ϕ starting from the time instant i as $\sigma, i \models \phi$. We abbreviate $\sigma, 0 \models \phi$ as $\sigma \models \phi$, as we usually care about satisfaction of an LTL formula over the whole word from the start. The set of satisfying words is defined as:

$$\mathit{Words}(\phi) = \big\{\sigma \in \Sigma \mid \sigma \models \phi\big\}.$$

Definition 11 (Semantics for Infinite Words)**.** *Given an infinite word σ, we inductively define satisfaction as follows:*

$$\begin{aligned}
&\sigma, i \models a, a \in AP \ \textit{iff}\ a \in \sigma[i].\\
&\sigma, i \models \neg\phi \ \textit{iff}\ \sigma, i \not\models \phi.\\
&\sigma, i \models \phi_1 \wedge \phi_2 \ \textit{iff}\ \sigma, i \models \phi_1 \ \textit{and}\ \sigma, i \models \phi_2.\\
&\sigma, i \models X\phi \ \textit{iff}\ \sigma, i+1 \models \phi.\\
&\sigma, i \models \phi_1 U \phi_2 \ \textit{iff}\ \exists j, i \leq j : \sigma, j \models \phi_2\\
&\qquad \textit{and}\ \forall k, i \leq k < j : \sigma, k \models \phi_1.
\end{aligned}$$

Definition 12 (Semantics for Finite Words)**.** *For finite words, we denote the length of a word σ as $\mathit{length}(\sigma)$. We denote the last index of the word with $\mathit{last} = \mathit{length}(\sigma) - 1$. Given a finite word σ, we inductively define satisfaction as follows:*

$$\begin{aligned}
&\sigma, i \models a, a \in AP \ \textit{iff}\ a \in \sigma[i].\\
&\sigma, i \models \neg\phi \ \textit{iff}\ \sigma, i \not\models \phi.\\
&\sigma, i \models \phi_1 \wedge \phi_2 \ \textit{iff}\ \sigma, i \models \phi_1 \ \textit{and}\ \sigma, i \models \phi_2.\\
&\sigma, i \models X\phi \ \textit{iff}\ i < \mathit{last} \ \textit{and}\ \sigma, i+1 \models \phi.\\
&\sigma, i \models \phi_1 U \phi_2 \ \textit{iff}\ \exists j, i \leq j \leq \mathit{last} : \sigma, j \models \phi_2\\
&\qquad \textit{and}\ \forall k, i \leq k < j : \sigma, k \models \phi_1.
\end{aligned}$$

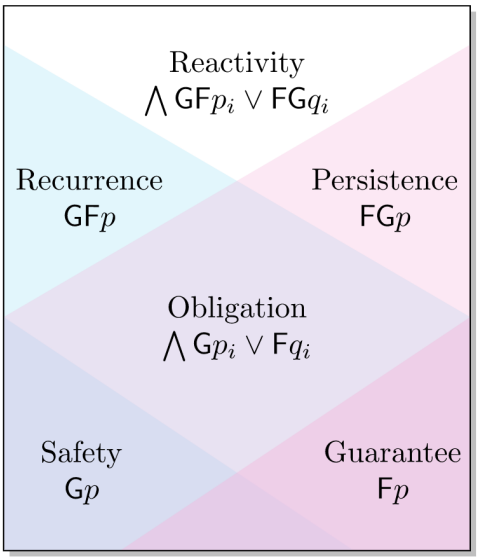


Figure 2.4: Hierarchy of Temporal Properties. Adapted from: `https://spot.lrde.epita.fr/hierarchy.html`

A Hierarchy of Temporal Properties

When developing a logical language for a specific purpose, it is important keep in mind what types of statements can be expressed through it. After proposing LTL for formal analysis of *reactive systems*, the language and its usage was thoroughly studied. In 1990, Manna proposed a *hierarchy of temporal properties* [105], where key properties expressible in LTL are put in relation to each other. The hierarchy is built from the classes pictured in Figure 2.4, where each class includes everything below it. For instance, the recurrence class includes the obligation class which also includes the safety and guarantee classes. The *Reactivity* class contains all properties expressable in LTL. With LTL_f, all expressable properties lie within the *Obligation* class.

Safety

Safety involves all temporal patterns which reason about a certain property always or never holding true in a given system. A practical example would be: "On an intersection, it should never be the case that both traffic lights show green." Safety properties can be disproven on a finite prefix of a trace, through a single counterexample of the property, e.g. a single instance where both traffic lights show green, but for proof of safety the whole trace needs to be examined.

Guarantee Guarantee or *Liveness* properties reason about a certain property happening eventually, usually involving a desired state being reached. An example: *If passenger presses the traffic light button at an intersection, the traffic lights will eventually switch to let them safely pass.* Liveness Properties can be proven on a finite prefix of a trace, through a single witness of the property, but can only be disproven by examining the whole trace. Safety and Liveness are dual properties.

Obligation Obligation is the union of Safety and Liveness. In this class, both goals (*eventually reach the goal area*), as well as constraints (*never visit this dangerous area*) can be expressed. All properties expressable in LTL_f lie within this class.

Recurrence Recurrence or *Fairness* properties are obtained by chaining the temporal operators G and F. These properties reason about certain events happening indefinitely often. For example, fairness requires that whenever several processes that share resources are run concurrently, they must be treated 'fairly' by the operating system, scheduler. In our traffic light example: *Both traffic lights should infinitely often repeat to show green.*

$$\phi = GF\,check_pressure$$

is a classic temporal pattern to model monitoring or patrolling of autonomous systems.

Persistence Persistence is the dual to recurrence. It requires a certain property to always hold from some point in the future. These properties can be used to model stabilization of a system or convergence to a set of states.

Reactivity Reactivity represents all properties that are expressable in LTL. It captures reaching goals, following constraints, as well as infinite recurrence and convergence.

Other Temporal Logics

Depending on the needs of the user, LTL can be extended in various ways.

Timed Temporal Logics *Metric Temporal Logic*[106] adds explicit real-time bounds to temporal operators. This enables users to specify exactly how long a certain property should hold or until when a task should be completed. Unfortunately, the model checking and thus strategy synthesis over interval-based semantics is undecidable [107]. This effectively makes classic MTL unsuitable to synthesize strategies for autonomous systems. However, through restriction to fragments or using different semantics, decidability can be achieved in various ways [108]. A

decidable and popular fragment, called *Metric Interval Temporal Logic* (MITL), restricts the temporal bounds to intervals of non-zero length and that are either natural numbesr or infinite [109, 110].

Signal Temporal Logic (STL) [111] further replaces boolean propositions with real-valued signals and is defined over continuous time, where all temporal operators are bounded by an interval. STL is well suited to analyze real-valued signals and has been applied in control [112] and motion planning [113]. STL specifications can often directly be used within optimization problems by exploiting its quantitative semantics [114] or to define satisfying regions in the state space [115].

Probabilistic extensions of STL can be used to directly reason about probabilitites of certain events. PrSTL [116] incorporates random variables in predicates, allowing to specify e.g. the probability of a certain event to always be below a threshold.

Branching Time Temporal Logics Branching time logics have been used in computer science, e.g. for efficient formal verification of the possible behaviour of computer systems. The most popular ones are the computation tree logics CTL and CTL* [96], which are comparable to Prior's two branching time temporal logics [101]. They are interpreted on so-called computation trees. These trees are naturally obtained as tree unfoldings of discrete transition systems. While CTL historically preceded CTL*, nowadays CTL is often viewed as a fragment of CTL*, with CTL* being more expressive as both CTL and the later introduced linear temporal logic LTL[96]. Probabilistic extensions of CTL have been used to facilitate verified motion planning for a robot in a segmented environment. Here, the branching nature of CTL is used to represent differing outcomes of high-level actions [117].

Choice of Specification Formalism

For high-level task planning and negotiation between agents, we prefer using LTL over more expressive alternatives such as MITL or STL. In this thesis, we are making heavy use of the fact that every LTL formula can be translated into a discrete, finite automaton, a mathematical structure we can manipulate and use for the goals of this work. Through restriction to a certain fragment of expressable properties in LTL, it is possible to alleviate some of the complexity of synthesis algorithms. In this thesis, we either use the full expressive power of LTL over infinite words (Chapter 3) or limit ourselves to LTL_f over finite words, falling back to the obligation class (Chapter 4 and Chapter 5).

2.3 Automata

Every LTL formula can be represented by a graph-like structure called an automaton. An automaton consists of states and transitions between the states. The

transitions are labelled with symbols of an alphabet. Given a word, we can generate a run on the automaton by sequentially looking at the next symbol of the word and following the transitions with matching symbol. Through so-called *acceptance conditions*, the automaton either accepts or rejects a word given to it. This usually involves visiting or avoiding a certain set of states. For LTL over infinite words, the acceptance conditions consider visiting such states infinitely often. For LTL_f over finite words, ending in an accepting state is enough.

Definition 13 (Automaton)**.** *An automaton over the alphabet Σ is a tuple $A = (Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, Σ is a finite set of symbols, $\delta : Q \times \Sigma \to 2^Q$ is a transition function, q_0 is the initial state, and $F \subseteq Q$ is a finite set of accepting states.*

We call an automaton *deterministic* if $|\delta(q, \sigma)| \leq 1, \forall q \in Q, \forall \sigma \in \Sigma$. We call an automaton *complete* if $|\delta(q, \sigma)| \geq 1, \forall q \in Q, \forall \sigma \in \Sigma$.

Definition 14 (Infinite Run)**.** *An infinite run of the automaton A on a word $\sigma = \sigma_0\sigma_1\sigma_2 \cdots \in \Sigma^\omega$ is an infinite sequence of states $r = q_0q_1q_2 \cdots \in Q^\omega$ with q_0 being the initial state and $\forall i \geq 0 : q_{i+1} \in \delta(q_i, \sigma_i)$.*

Definition 15 (Finite Run)**.** *A finite run of the automaton A on a word $\sigma = \sigma_0\sigma_1\sigma_2 \cdots \in \Sigma^*$ is an infinite sequence of states $r = q_0q_1q_2 \cdots \in Q^*$ with q_0 being the initial state and $\forall i \geq 0 : q_{i+1} \in \delta(q_i, \sigma_i)$.*

Given a word σ, we define the set of all possible runs of A on σ as $\text{Runs}_A(\sigma)$. If A is deterministic, $|\text{Runs}_A(\sigma)| = 1$. We denote by $|r|_q$ the number of times the state q appears in r.

Definition 16 (Acceptance Conditions)**.** *For infinite words, we consider three different acceptance conditions and name automata with such acceptance conditions accordingly:*

- *Büchi Automaton (NBA):* $\exists r \in Runs_A(\sigma) : \exists q \in F : |r|_q = \infty$
- *universal co-Büchi Automaton (UCA):* $\forall r \in Runs_A(\sigma) : \forall q \in F : |r|_q < \infty$
- *universal k-co-Büchi Automaton (UKCA):* $\forall r \in Runs_A(\sigma) : \sum_{q \in F} |r|_q \leq K$

For finite words, we accept a word if the last state in a run is an accepting state.

- *Finite Automaton:* $\exists r \in Runs_A(\sigma) : r_{last} \in F$

We denote the sets of words accepted by those acceptance conditions by $L_b(A), L_{uc}(A)$ and $L_{uc,K}(A)$. For DFA, we simply write $L_(A)$.

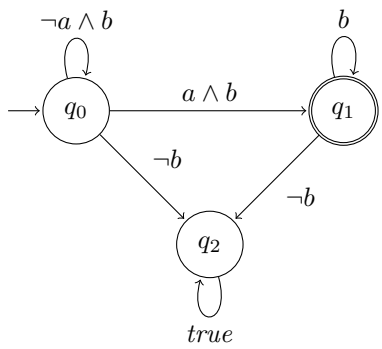


Figure 2.5: DFA for the LTL$_f$ formula $\phi_1 = Fa \wedge Gb$

Automaton Construction from LTL Formulae

For every LTL$_f$ formula ϕ, we can construct a DFA A_ϕ such that $L_b(A_\phi) = \text{Words}(\phi)$ [104]. This can be done with off-the-shelf tools such as MONA [118]. For every LTL formula ϕ, there exists an Büchi automaton A_ϕ such that $L_b(A_\phi) = \text{Words}(\phi)$ [96]. Tools like ltl2ba exist for this purpose [119]. By duality of accepting conditions, clearly $L_{uc}(A) = \overline{L_b(A)}$ holds [39]. From this, we can construct a UCA A such that $L_{uc}(A) = \text{Words}(\phi)$ by constructing a (non-)deterministic Büchi automaton for $\neg\phi$ and changing the acceptance condition, leaving the automaton as is.

$$\text{Words}(\phi) = \{\sigma \mid \sigma \not\models \neg\phi\} = \overline{L_b(A_{\neg\phi})} = L_{uc}(A_{\neg\phi})$$

Example 4 (Automaton over finite words)**.** *Depicted in Figure 2.5 is the DFA for the LTL$_f$ formula $\phi_1 = Fa \wedge Gb$. The initial state q_0 is indicated by an incoming arrow. The accepting state q_1 is indicated through a double lined circle. For a finite word σ, this DFA decides if a word belongs to the language $L(\phi_1)$ by processing the word symbol by symbol, triggering the appropriate transitions and tracking the current state. If the state after the last symbol of the word is an accepting state, the word is accepted as part of $L(\phi_1)$. For example, the word*

$$\sigma_1 = \{b\}\{b\}\{b\}\{b\}\{b\}$$

is not accepted, as the DFA remains in state q_0. Thus $\sigma_1 \notin L(\phi_1)$. This is expected, as the word does not contain an a anywhere, which we specified with Fa. While the word

$$\sigma_2 = \{b\}\{a,b\}\{b\}\{\}\{b\}$$

has an a at the second position, which triggers the transition to q_1, the symbol $\sigma[3]$ violates the constraint Gb of the formula and triggers the transition to q_2, a state which cannot be left after entering. Thus $\sigma_2 \notin L(\phi_1)$. If we take the word

$$\sigma_3 = \{b\}\{b\}\{a,b\}\{b\}\{b\},$$

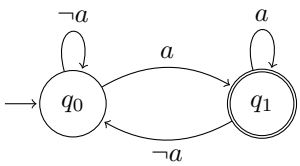


Figure 2.6: NBA for the LTL formula $\phi_2 = GFa$

instead, the symbol $\sigma[2] = \{a, b\}$ triggers the transition to state q_1 and the following symbols keep it there. Thus $\sigma_3 \in L(\phi_1)$.

Example 5 (Automaton over infinite words)**.** *As an example for an LTL formula over infinite words, Figure 2.6 shows an NBA for the LTL formula $\phi_2 = GFa$. Initial and accepting states are identical to the DFA example, but this automaton has the büchi acceptance condition, meaning that words are accepted if an accepting state is visited infinitely often. For example, take the word where a occurs every second step:*

$$\sigma_4 = \{a\}\{\}\{a\}\{\}\{a\}\{\}\dots$$

Here, the automaton switches between states q_0 and q_1 infinitely often, meaning that q_1 is visited infinitely often. Hence, $\sigma_4 \in L_b(\phi_2)$. If we examine the word

$$\sigma_5 = \{a\}\{a\}\{a\}\{\}\{\}\{\}\dots,$$

we can see that a occurs in the first three symbols, but not afterwards (here, the dots indicate infinite continuation of $\{\}$). The accepting state q_1 is visited only a finite amount of times and therefore $\sigma_5 \notin L_b(\phi_2)$.

2.4 Strategy Synthesis via Product Automata

In this section, we briefly introduce the strategy synthesis problem for LTL specifications and the basic concept of the *product automaton*. We assume to be given a labelled model such as a transition system or an MDP and a temporal logic specification such as an LTL or LTL$_f$ formula. The task is to find a strategy such that all traces that can be generated by following the strategy satisfy the specification. In order to find such a strategy, the model and the specification automaton will be unified into a *product automaton*, capturing both the evolution of the model and the specification automaton in a single structure. A simplified, schematic overview is shown in Figure 2.1.

Depending on the model and the specification automaton, finding a strategy on the product present problems of varying difficulty. Table 2.1 lists combinations of models and automata and their resulting problems. In this section, we will demonstrate the product between an NBA and a deterministic TS to illustrate the general process. The methods presented in this thesis build on this standard

	LTL_f	LTL	
	DFA	det. BA	NBA
det. TS	path-finding	lasso path	lasso path
non-det. TS	2P-game with reachability goal	2P-game with Büchi goal	Rabin game
MDP	2.5P-game with reachability goal	2.5P-game with Büchi goal	2.5P-Rabin game

Table 2.1: Different combinations of model and specification automaton lead to problems of varying difficulty.

technique. In Chapter 3, we handle non-deterministic TS and NBA for multiple agents. The methods presented in Chapter 4 and Chapter 5 are based on products between DFAs and MDPs.

While the problem to solve differs depending on the combination of model and specification language, they all share an underlying technique, constructing a *product automaton*. To illustrate this, we demonstrate how to create a product from a deterministic transition system and a BA, but the idea extends to the other presented models, as well. All planning techniques in this thesis are based on similar constructions.

Definition 17 (Product Automaton)**.** *Given a transition system*

$$T = (S, Act, \delta_{ts}, s_0, AP, L)$$

and a Büchi automaton

$$A = (Q, \Sigma, \delta_a, q_0, F)$$

over the alphabet Σ, we define the labelled product automaton

$$P = (S \times Q, Act, \delta, p_0, AP, F_p, L_p)$$

where

- *$S \times Q$ is the new state space, tracking the state of both T and A,*
- *$\delta : S \times Q \times Act \to S \times Q$ is the transition function with*

$$\delta((s, q), a) = \{(s', q') \mid s' = \delta_{ts}(s, a), q' = \delta_a(q, L(s'))\},$$

- *$p_0 = (s_0, q_0)$ is the initial state,*
- *$F_p = \{(s, q) \mid q \in F\}$ is the set of accepting states and*
- *$L_p : S \times Q \to \Sigma$ is a labelling function with*

$$L_p((s, q)) = L(s).$$

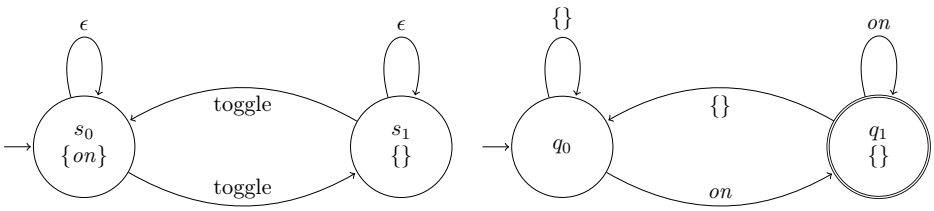


Figure 2.7: Left: A deterministic TS of a light switch. Right: A deterministic Büchi automaton for the formula $\phi = GFon$

.

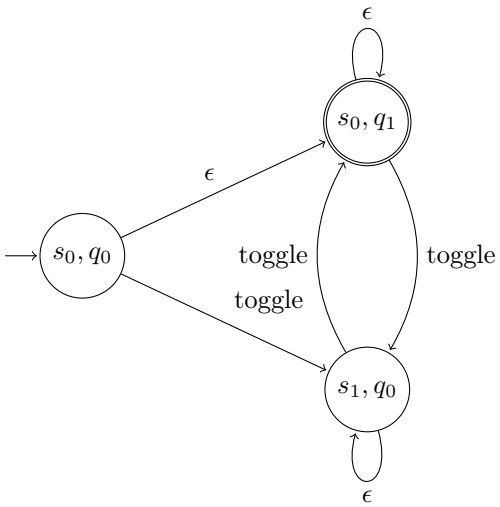


Figure 2.8: A product automaton from model and specification depicted in Figure 2.7.

Example 6 (Product Automaton)**.** *For demonstration purposes, we will use a simple, deterministic model that can toggle freely between two states. A deterministic Büchi automaton is obtained from the formula $\phi = GFon$. Both models are depicted in Figure 2.7.*

The resulting product automaton is depicted in Figure 2.8. A path on a product automaton represents both a valid evolution of T and of A. In each state, we change the state of T by choosing an action and then update the state of A through label of the new state in the transition system. Through the acceptance condition of the Büchi automaton and the corresponding set of accepting states, we can find an infinite path on the model that satisfies the specification formula. These so-called *lasso paths* [96] consist of a circular suffix that visits an accepting state and finite prefix starting from an initial state and leading to that accepting state.

Example 7 (Lasso Path)**.** *In Figure 2.8, repeatedly choosing the action ϵ leads to an accepting run, since it never leaves the accepting state after reaching there. Repeatedly choosing the action* toggle *also leads to an accepting run, since we infinitely often reach the accepting state every two steps.*

Through finding an accepting lasso path on the product automaton P, we found a series of actions that create a trace on T that is accepted by the Büchi automaton and in turn accepts the specification we were given.

After finding a valid solution in the product, it has to be projected back onto the model to obtain a strategy as defined in Definition 7. In the case of our lasso path example, the strategy is simply a sequence of actions. In more complex cases, the states of the product automaton can be used as memory for the strategy. Loosely speaking, we can keep track of the evolution of the product automaton during execution and then use the current state of the product automaton and the solution found within the product to produce the next action that should be taken.

2.5 Centralized Strategy Synthesis for Multi-Agent Systems

When multiple agents share an environment, their executions might influence each other, so the interaction between them has to be accounted for. If we assume that each agent has their own model and their own specification, we need to consider the traces of agents jointly. Assuming we can control each agent and have all models and specifications, a straightforward approach is to compose multiple transition systems $T_i, i \in [1, n]$ into a single, global transition system:

$$T = \bigotimes_{i \in [1,n]} T_i$$

with composition defined by the synchronous product operator [96]. Fulfillment of all specification formulae can be assured by conjunction of all partial specifications:

$$\phi = \bigwedge_{i \in [1,n]} \phi_i$$

From here, a strategy for all agents can be found by constructing $T \otimes A_\phi$ and applying methods from [12]. With this, we have effectively reduced the multi-agent case to a single agent case. While this approach is correct and complete in theory, scalability is an issue that cannot be overcome by adding computational power [1].

Chapter 3

Compositional Strategy Synthesis

In this chapter, we tackle the challenge of scalability by exploiting dependencies and independencies between agents. We propose a solution where every agent first computes possible strategies in isolation, making different assumptions over the other agents only where it is necessary. Afterwards, the agents compose their plans and check if others can follow the made assumptions. We start with the assumption that all agents are cooperative (see Figure 1.1) such that any made assumptions is suitable, as long as the agents can follow them. To ensure completeness of our approach, we maintain the set of all candidate strategies in a suitable representation. During composition, agents' state spaces and candidate strategies are iteratively pruned, hindering the state-space explosion. We evaluate the method and compare it to the monolithic approach on several examples and case studies.

3.1 Problem Formulation

We propose a solution to the multi-agent task planning problem for a collaborative team of heterogeneous agents. The capabilities and possible behaviours of each agent will be modeled as a *non-deterministic transition system*, where the non-determinism models unknown outcomes of different actions. Each agent will need to satisfy an LTL formula representing task specification and constraints. The specification of each agent might be dependent on other agents.

Problem 1. *Given n transition systems*

$$T_i = (S_i, Act_i, \delta_i, s_{ini,i}, AP_i, L_i),\ i \in [1, n],$$

such that $AP_i \cap AP_j = \emptyset,\ \forall i \neq j$ and LTL formulas $\phi_1, \ldots, \phi_n$ defined over $AP = \bigcup_{i=1}^{n} AP_i$ each, find strategies $\lambda_i : (S_1 \times \cdots \times S_n)^* \subseteq Act_i$ such that resulting joint traces complying with these strategies satisfy all ϕ_i:

$$\tau_{joint} \models \phi_1 \wedge \cdots \wedge \phi_n$$

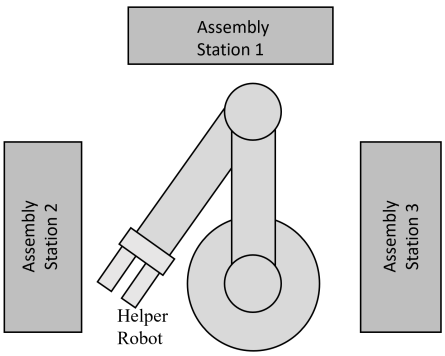


Figure 3.1: Example 8. Multiple assembly stations share a helper robot. ©2020 IEEE [120]

Example 8 (Assembly Line)**.** *Consider an industrial assembly line with assembly robots situated at different stations. Each robot performs a sequence of preparation and assembly steps to fabricate a workpiece. Some of these steps may be difficult and require assistance. For this reason, additional robot arms are installed in reach of the assembly stations. Since the assembly stations share a resource, coordination of assembly strategies is needed to avoid conflicts and minimize waiting times. An example of a setup with three assembly stations and one helper robot is illustrated in Figure 3.1. An example of a transition system modeling an assembly robot is given in Figure 3.2, where a, b, c stand for three different types of pieces that need to be prepared for assembly. This could represent polishing or applying adhesives. ab and bc stand for intermediate workpieces and abc is the finished workpiece. The transitions correspond to the robot's capabilities to either prepare a part or do a specific assembly. The self-loops allow the robot to wait for help, which the assembly of a and b to ab requires. This constraint is stated in a constraint formula, which can be added to the goal specification through a conjunction. The helper can be modeled as the transition system in Figure 3.3. It can provide assistance only to a specific assembly station at a time. The individual tasks are given in* (3.1) *and* (3.2)*. The formula for assembly station i consists of the goal specification and model constraints: keep assembling the final workpiece* abc_i*, and assemble the connection between piece a and b if assistance from the helper robot is provided. The helper task is to repeatedly return to the neutral position, to allow the robot arm to provide assistance where needed.*

$$\phi_{assembly,i} = GF\, abc_i \wedge G(a_i \wedge X(ab_i \vee abc_i) \rightarrow help_i) \tag{3.1}$$

$$\phi_{helper} = GF\, neutral \tag{3.2}$$

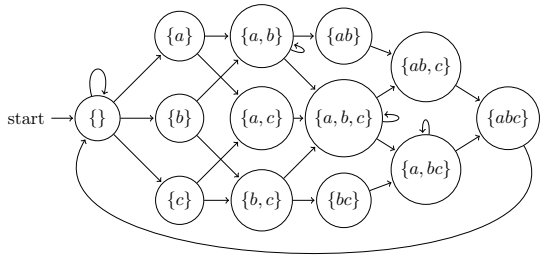


Figure 3.2: Single assembly station.

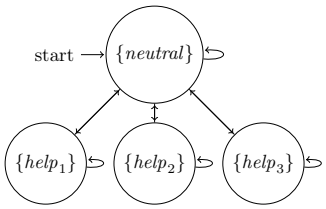


Figure 3.3: Helper robot.

Figure 3.4: Transition Systems of the assembly example. Action labels and state names are omitted due to readability. ©2020 IEEE [120]

3.2 Method

A flowchart illustrating the proposed compositional process addressing Problem 1 is shown in Figure 3.5. As a starting point, we translate each specification formula ϕ_i to an UCA A_{ϕ_i} as shown in Section 2.3. For each pair T_i and A_{ϕ_i}, we define a product *turn-based UCA (tbUCA)* P_i that captures the possible trajectories of T_i accepted by the specification automaton A_{ϕ_i}, as well as the dependencies on other agents via additional labels on the transitions of the tbUCA, which we call *assumptions*. We apply methods from [39] to effectively reduce the tbUCA with assumptions to a *safety game*, for which efficient solution algorithms exist [121]. We obtain a *masterplan* as an efficient representation of all candidate strategies. To find strategies that are mutually compatible with the assumptions of the tbUCAs $P_1, \dots P_n$, we iteratively compose the corresponding masterplans pairwise. This way, the candidate strategies that are not mutually compliant between agents are gradually pruned away. After all products have been composed, the remaining masterplan is a representation of all possible strategies satisfying Problem 1.

Candidate Strategies for a Single Agent

Product Automaton with Assumptions We extend the usual definition of a synchronous product between a transition system and an infinite word automaton (see Definition 17) by introducing additional labels to the transition function δ, called assumptions. These labels are subsets of atomic propositions of the other agents that need to hold true for the transition to happen. Thus, we represent dependencies between agents without explicitly modelling possible states of every other agent, which would lead to a state space explosion.

Definition 18 (Product tbUCA with Assumptions)**.** *Given a transition system*

$$T = (S, Act, \delta_{ts}, s_{ini}, AP_i, L)$$

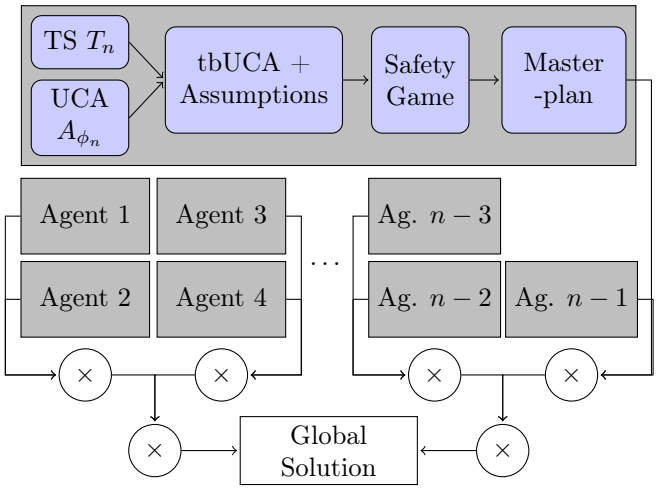


Figure 3.5: Construction of the individual solutions and their binary composition for multiple agents. ©2020 IEEE [120]

and a UCA

$$A_\phi = (U, 2^{AP_\phi}, \delta_{uc}, u_{ini}, F),$$

we define the assumption alphabet $\widehat{\Sigma} = 2^{\widehat{AP}}$*, where*

$$\widehat{AP} = AP_\phi \setminus AP_i.$$

$\sigma \in \widehat{\Sigma}$ *is called an assumption. The* product turn-based UCA (tbUCA) with assumptions *is a tuple*

$$P = (Q_1, Q_2, \Sigma_1, \Sigma_2, \delta_1, \delta_2, q_{ini}, Q_F),$$

where

- $Q_1 \subseteq S \times U$; $Q_2 \subseteq S \times U \times Act \times \widehat{\Sigma}$,
- $\Sigma_1 = Act \times \widehat{\Sigma}$; $\Sigma_2 = S$,
- $\delta_1 : Q_1 \times \Sigma_1 \to Q_2$. $\delta_1((s,u),(a,\sigma)) = \{(s,u,a,\sigma) \in Q_2 \mid \delta_{ts}(s,a) \neq \varnothing\}$,
- $\delta_2 : Q_2 \times \Sigma_2 \to Q_1$. $\delta_2((s,u,a,\sigma),s_i) = \{(s_i,u_i) \in Q_1 \mid u_i \in \delta_{uc}(u,\widehat{\sigma})\}$, *if* $s_i \in \delta_{ts}(s,a)$ *and* $\widehat{\sigma}$ *is the largest element of* 2^{AP_ϕ} *that is also a subset of* $L(s_i) \cup \sigma$,
- $q_{ini} = (s_{ini}, u_{ini}) \in Q_1$ *is the initial state and*
- $Q_F = \{(s,u) \mid (s,u) \in Q_1$ *and* $u \in F\}$.

States in Q_1 represent the current state of the transition system s and the UCA u. The deterministic transitions from Q_1 to Q_2 represent the agent choosing an action $a \in Act$ available in the current state of the transition system and making an assumption $\sigma \in \widehat{\Sigma}$ about the other agents' atomic propositions. This assumption may be seen as a requirement for the agents' cooperation. States in Q_2 are intermediate states including the chosen action and assumption on top of s and u. The transitions from Q_2 to Q_1 respect the transitions in A_ϕ and are labeled with the non-deterministic outcomes of corresponding transitions in T.

Definition 19 (Run on P)**.** *A run of P on a word $w = w_1 w_2 \cdots \in (\Sigma_1 \Sigma_2)^\omega$ is a alternating sequence of states $q_1 q_2 \cdots \in (Q_1 Q_2)^\omega$ such that q_1 is the initial state and $\forall i \geq 1 : q_{i+1} \in \delta_j(q_i, w_i)$ with $j = 2 - (i \mod 2)$).*

Definition 20 (Traces of P)**.** *The trace $\tau \in (2^{AP})^\omega$ of a run $r = q_1 q_2 \cdots \in (Q_1 Q_2)^\omega$ of the tbUCA P over the word $w \in (\Sigma_1 \Sigma_2)^\omega$ is defined as:*

$$\tau_i(r) = L(q_{2i}(1)) \cup q_{2i}(4), \forall i$$

The trace of a product tbUCA with assumptions is a sequence of the propositions holding in the current state of T and the propositions of other agents which are assumed to hold. We denote by $Traces(P)$ all traces of infinite runs on P and by $Traces_{acc}(P)$ all possible traces of words accepted by P. An important property of the traces of P is their correspondence to traces of the agents and words accepted by A_ϕ :

Lemma 1 (Accepted traces of P)**.** *Let P be a product tbUCA with assumptions of T and A_ϕ.*

$$\begin{aligned} Traces_{acc}(P) = &\{w \in (2^{AP})^\omega \mid w \in L_{uc}(A_\phi) \text{ and} \\ &\exists \tau \in Traces(T), \text{ such that } \forall j. \tau_j \subseteq w_j\}. \end{aligned}$$

Proof 1. *"$\subseteq$": For each run*

$$r = q_1 q_2 \cdots \in (Q_1 Q_2)^\omega$$

on P, it holds that

$$q_{2i-1}(1) = q_{2i}(1), q_{2i-1}(1) \in S, q_{2i+1}(1) \in \delta_{ts}(q_{2i}(1), q_{2i}(3)), \forall i \geq 1$$

and $q_1(1)$ is the initial state of T. So the sequence

$$q_1(1) q_3(1) q_5(1), \cdots \in S^\omega$$

is a path on T. Following from this and Definition 20,

$$\begin{aligned} Traces(P(T, \phi)) \subseteq &\{w \in (2^{AP})^\omega \mid \exists \tau \in Traces(T), \\ &\text{such that } \forall j. \tau_j \subseteq w_j\} \end{aligned}$$

Also,

$$q_{2i-1}(2) = q_{2i}(2), q_{2i-1}(2) \in Q, q_{2i+1}(2) \in \delta_{uc}(q_{2i}(2), L(q_{2i}(1)) \cup q_{2i}(4)), \forall i \geq 1$$

and $q_1(2)$ *is the initial state of* A*. So the sequence*

$$q_1(2)q_3(2)q_5(2), \cdots \in Q^\omega$$

is a run on A*. A run on* P *is accepting, iff the corresponding run on* A *is accepting. Every run on* P *that visits a final state* $q_f \in Q_F$ *infinitely often is rejected by* P*. This run* r *of* P *has has a corresponding run on* A*, which visits a final state in* F *infinitely often, since* $q_F(2) \in F$*. So the run is not accepting on* A *as well. From this follows*

$$Traces_{acc}(P(T, \phi)) \subseteq L_{uc}(A).$$

Together with the above result, it holds that

$$\begin{aligned} Traces_{acc}(P) \subseteq &\{w \in (2^{AP})^\omega \mid w \in L_{uc}(A_\phi) \text{ and} \\ &\exists \tau \in Traces(T), \text{ such that } \forall j.\tau_j \subseteq w_j\}. \end{aligned}$$

"$\supseteq$*": We show that for every infinite path* $t = s_1 s_2 \cdots \in S^\omega$ *on* T *and every word* $w \in \Sigma_\phi = 2^{AP_\phi}$ *with* $w \models \phi$ *and* $w_i \subseteq L(t_i), \forall i$*, there exists a run*

$$r = q_1 q_2 \cdots \in (Q_1 Q_2)^\omega$$

on P*. By Definition 16, all runs* $u = u_1 u_2 \cdots \in Q^\omega$ *on* A *over* w *are accepting. We will now construct this accepting run* $r = q_1 q_2 \cdots \in (Q_1 Q_2)^\omega$ *on* P*.* $q_1 = (s_1, u_1)$ *is the initial state. The states of the runs are defined as* $q_2 i - 1 = (s_i, u_i) \in Q_1, \forall i \geq 1$ *and* $q_2 i = (s_i, u_i, a, \sigma) \in Q_2$ *with* $a \in Act, t_{i+1} \in \delta_{ts}(s_i, a)$ *and* $\sigma \in \widehat{AP}$ *with* $u_{i+1} \in \delta_{uc}(u_i, \sigma \cup L(s_{i+1})), \forall i \geq 1$*. The run* r *is accepting since* $w \models phi$ *and so every run* u *is accepting. Since for every path on* T *and a word* w *we constructed an accepting run on* P*, it holds that*

$$\begin{aligned} Traces_{acc}(P) \supseteq &\{w \in 2^{AP} \mid w \in L_{uc}(A_\phi) \text{ and} \\ &\exists \tau \in Traces(T), \text{ such that } \forall j.\tau_j \subseteq w_j\}. \end{aligned}$$

□

A product tbUCA with assumptions can be interpreted as a 2-player turn-based game as in Definition 3. Q_1 are states of player 1 and Q_2 the states of player 2. Each run on P is a play. A tbUCA is *realizable* if there exists a winning policy for player 1 such that for all policies of player 2, the resulting play is an accepting run [35].

With respect to Problem 1, realizability of all $P_i, i \in [1, n]$ is necessary for the existence of a set of strategies λ_i fulfilling all ϕ_i, but not sufficient, as made assumptions in each P_i need to be checked for compatibility with each $P_j, j \neq i$.

From tbUCA to tbUKCA to Safety Games Filiot et al. [39] showed that it is possible to reduce the realizability problem of a tbUCA automaton to the realizability of a tbUKCA automaton, a *turn-based universal K-co-Büchi Automaton*[1], by finding a finite boundary for K.

Furthermore, they determinize the automaton to $det(P, K)$. Usually, determinisation of automata is done by using subsets of possible states as new states for the then deterministic automaton. This construction is extended with counters that count up the maximal number of accepting states which have been visited b runs ending up in a certain state. Each state $F \in \mathcal{F}_1$ ($\mathcal{F}_2$) in $det(P, K)$ is a mapping from all $q \in Q_1$ ($q \in Q_2$) to $[0, K+1]$ and called a *counting function*. In each state of $det(P, K)$, the counter of a state $q \in Q_1 \cup Q_2$ indicates the maximum number of times an accepting state has been visited over all possible runs on the prefix read that ends up in q. If no run on the prefix read ends in q, then the counter is set to -1. This way, the deterministic tbUKCA becomes a safety game $G(P, K) = (\mathcal{F}_1, \mathcal{F}_2, \Sigma_1, \Sigma_2, \Gamma_1, \Gamma_2, \Delta_1, \Delta_2, F_{ini})$ with appropriately defined Δ_1 and Δ_2, and Γ_1, Γ_2 allowing all possible $\sigma \in \Sigma_1, \Sigma_2$ in all states. We aim to avoid a visit to *Unsafe* $= \mathcal{F}_f = \{F \in \mathcal{F}_1 \cup \mathcal{F}_2 \mid \exists q, F(q) > K\}$. A more complete explanation and proofs are given in [39].

Definition 21 (Determinisation of tbUKCA (from [39]))**.** *Let P be a tbUKCA*

$$A = (\Sigma_1, \Sigma_2, Q_1, Q_2, \Delta_1, \Delta_2, q_0, Q_f)$$

with $K \in \mathbb{N}$. Each state in the resulting automaton is a mapping from all $q \in Q_1 \cup Q_2$ to $[K+1]$, called counting function. The counter of a state q is set to -1 when no run on the prefix read ends up in q. Formally,

$$det(A, K) = (\Sigma_1, \Sigma_2, \mathcal{F}_1, \mathcal{F}_2, \delta_1, \delta_2, \mathcal{F}_0, \mathcal{F}_f),$$

where:

- $\mathcal{F}_1 = \{F \mid F \text{ is a mapping from } Q_1 \text{ to } [K+1]\}$
- $\mathcal{F}_2 = \{F \mid F \text{ is a mapping from } Q_2 \text{ to } [K+1]\}$
- $\delta_i(F, \sigma) = q \mapsto \begin{cases} \max\{\min(K+1,\ F(p)+1; \mid q \in \Delta_i(p, \sigma), & F(p) \neq -1\}, \\ & \text{if } q \notin \mathcal{F}_f \\ \max\{\min(K+1,\ F(p) \mid q \in \Delta_i(p, \sigma), & F(p) \neq -1\}, \\ & \text{if } q \in \mathcal{F}_f \end{cases}$
- $\mathcal{F}_0 = q \in Q_1 \mapsto \begin{cases} -1, & \text{if } q \neq q_0 \\ 0, & \text{if } q = q_0 \wedge q \notin \mathcal{F}_f \\ 1, & \text{if } q = q_0 \wedge q \in \mathcal{F}_f \end{cases}$

[1] A tbUKCA is a turn-based universal K-co-Büchi Automaton with the accepting condition threshold defined by K as in Definition 16.

- $\mathcal{F}_f = \{F \in \mathcal{F}_1 \cup \mathcal{F}_2 \mid \exists q, F(q) > K\}$.

Lemma 2 ([39]). *Let P be a tbUCA over inputs Σ_2 and outputs Σ_1 with n states and $K = 2n(n^{2n+2} + 1)$. Then P is realizable iff player 1 has a winning policy in the safety game $G(P, K)$.*

Guided by Lemma 2, we reduce the realizability problem for P to finding a masterplan $MP(G(P, K))$ in a safety game with assumptions.

Composition of Candidate Strategies

For every agent T_i and the corresponding specification ϕ_i, we created a tbUCA with assumptions P_i and transformed into a safety game $G(P_i, K)$. We also have a masterplan MP for each $G(P_i, K)$ representing all candidate strategies. The remaining step is the composition of the masterplans, which is inspired by [122], but requires additional reasoning over the formerly made assumptions. We define composition through the product of two safety games with assumptions that "resolves" the mutual assumptions the two games impose on each other, while keeping the remaining ones.

First, let us introduce an observation and necessary definitions allowing us to map the composition of the masterplans of the safety game onto the composition of the underlying transition systems. Let P be a tbUCA with assumptions as defined in Definition 18, $G(P, K)$ a safety game constructed from P as outlined above. For every state $F \in \mathcal{F}_1$ it holds that $s_i = s_j$, for every $q_i, q_j \in Q_1$ where $F(q) \neq -1$. For every state $F \in \mathcal{F}_2$ it holds that $s_i = s_j$, $a_i = a_j$ and $\sigma_i = \sigma_j$ for every $q_i, q_j \in Q_2$ where $F(q) \neq -1$.

We denote the projection of states in $\mathcal{F}_i$ onto the states of T and A_ϕ as $States_{TS} : \mathcal{F}_i \to S$ and $States_{UCA} : \mathcal{F}_i \to 2^U$, where $i \in \{1, 2\}$. Note that thanks to the observation above, $States_{TS}(\mathcal{F})$ is a singleton for all $\mathcal{F}$. The labeling function L can be naturally extended to $G(P, K)$ by inheriting the labels from the respective states of T. We also define $Assum : \mathcal{F}_2 \to 2^{\widehat{AP}}$ as the assumed proposition.

Definition 22 (Product of Safety Games with Assumptions). *Let*

$$G^i = (\mathcal{F}_1^i, \mathcal{F}_2^i, \Sigma_1^i, \Sigma_2^i, \Gamma_1^i, \Gamma_2^i, \Delta_1^i, \Delta_2^i)$$

with propositions AP^i, assumption symbols $\widehat{AP}^i$, $i \in \{1, 2\}$, be two safety games with assumptions. The product is defined as a safety game with assumptions $G^1 \otimes_a G^2$:

- $\mathcal{F}_i^p \subseteq \mathcal{F}_i^1 \times \mathcal{F}_i^2$, $i \in \{1, 2\}$,
- $\Sigma_i^p = \Sigma_i^1 \times \Sigma_i^2; i \in \{1, 2\}$,
- $\Delta_1^p : \mathcal{F}_1^p \times \Sigma_1^p \to \mathcal{F}_2^p$,

- $\Delta_2^p : \mathcal{F}_2^p \times \Sigma_2^p \to \mathcal{F}_1^p$,
- $\Gamma_i^p : \mathcal{F}_1^p \to 2^{\Sigma_i^p}$ *with* $\Gamma_1^p((F_i^1, F_i^2)) = \{(\gamma^1, \gamma^2) \mid \gamma^1 \in \Gamma_i^1(F_i^1), \gamma^2 \in \Gamma_i^2(F_i^2)\}$, $i \in \{1, 2\}$,
- $AP^p = (AP^1 \cup AP^2)$ *is the new set of propositions,*
- $AP_r^p = (\widehat{AP}^1 \cup \widehat{AP}^2) \setminus AP^p$ *are the propositions that will be "resolved",*
- $\widehat{AP}^p = (\widehat{AP}^1 \cup \widehat{AP}^2) \setminus AP_r^p$ *is the new set of remaining assumption symbols,*
- $States_{TS}^p(F) = States_{TS}^1(F) \cup States_{TS}^2(F)$,
- $States_{UCA}^p(F) = States_{UCA}^1(F) \cup States_{UCA}^2(F)$,
- $Assum^p(F) = (Assum^1(F) \cup Assum^2(F)) \setminus AP_r$,

with $\Delta_1^p((F_1^1, F_1^2), (\gamma^1, \gamma^2)) =$

$$\{(F_2^1, F_2^2) \mid F_2^1 = \Delta_1^1((F_1^1, \gamma^1), F_1^2 = \Delta_1^2(F_1^2, \gamma^2) \\ \textit{and } Assum^1(F_2^1) \cup Assum^2(F_2^2) \textit{ is consistent.}\}$$

and $\Delta_2^p((F_2^1, F_2^2), (\gamma^1, \gamma^2)) =$

$$\{(F_2^1, F_2^2) \mid F_2^1 = \Delta_1^1((F_1^1, \gamma^1), F_1^2 = \Delta_1^2(F_1^2, \gamma^2) \textit{ and} \\ Assum^1(F_2^1) \cup Assum^2(F_2^2) \cup L^p((F_2^1, F_2^2)) \textit{ is consistent.}\}$$

Any play on the game $G^p = G^1 \otimes_a G^2$ is a play both on G^1 and G^2. Any play complying with $MP(G^p)$ is winning for player 1 and the corresponding plays on G^1 and G^2 are also winning for player 1. Furthermore, for each pair of plays on G^1 and G^2, there exists a corresponding play on G^p. If both plays on G^1 and G^2 are complying with their respective masterplans, they are both winning for player 1 in G^1 and G^2. If at every state, the assumptions of the plays are consistent (see Definition 16) with each other and with both labelling functions of their respective encoded transition systems, the constructed play on G^p is also winning for player 1 and thus a part of the masterplan of G^p. The masterplan of the product can be computed in the same way as those of the subgames. Just constructing the product without solving is not enough, as inconsistencies in made assumptions could lead to losing states in the composed masterplan. We can extend the result from [122] to our product definition.

Product pruning For the sake of computational efficiency, it is important to note that the search of a masterplan can be restricted to the reachable part of the safety game; thus, pruning away potentially large unreachable parts of the product throughout the compositions. Formally, we denote by $Reach(G, F_{ini})$ the set of states reachable from F_{ini} in G.

Definition 23 (Masterplan of Reachable Positions [122])**.** *The* masterplan of reachable positions *is defined as*

$$MP_R(G, F_{ini})(F) = \begin{cases} MP(G)(F), \textit{if } F \in \textit{Reach}(G, F_{ini}) \\ \varnothing, \ \textit{else} \end{cases}$$

Definition 24 (Restricted Games)**.** *Given a function* $\Lambda : S_1 \to 2^{\Sigma_1}$*, in the restricted game* $G[\Lambda]$*, the available moves* Γ_1 *are further restricted by* Λ*, i.e.* $G[\Lambda] = (\mathcal{F}_1, \mathcal{F}_2, \Sigma_1, \Sigma_2, {\Gamma_1}', \Gamma_2, \Delta_1, \Delta_2)$ *with* ${\Gamma_1}'(f) = \Gamma_1(f) \cap \Lambda(f)$*.*

The following lemma guides us in efficiently composing safety games with assumptions and is a slightly modified version of a lemma found in [122].

Lemma 3. *Let* $\Lambda_1 = MP(G^1)$ *and* $\Lambda_2 = MP(G^2)$*.*

$$MP(G^1 \otimes_a G^2) = MP(G^1[\Lambda_1] \otimes_a G^2[\Lambda_2]).$$

Incremental Composition In practice, to efficiently find strategies to Problem 1, we use the following property of tbUKCA [39]:

$$\forall k_1, k_2, 0 \leq k_1 \leq k_2 : L_{uc,k_1}(P) \subseteq L_{uc,k_2}(P) \subseteq L_{uc}(P).$$

Intuitively, this property allows us to find the masterplans in a incremental fashion, starting with $k = 0$ and iteratively raising k until compatible masterplans in $G(P, k)$ are calculated through composition or the theoretical boundary of $2n(n^{2n+2} + 1)$ is reached. Methods to construct strategies from the composed masterplans are known [39]. This finally leads to the complete procedure depicted as pseudocode in Algorithm 1.

3.3 Experimental Results

We implemented the algorithm in Python3. Computation has been performed on a laptop with Intel i5-7300 processor and 8GB RAM.

Comparison to the Monolithic Approach First, we revisit the running example Example 8 and compare our method to the monolithic approach described in Section 2.5. The results for varying numbers of assembly stations are given in Table 3.1. The 'run' shows that the compositional method is able to handle larger instances and is able to calculate solutions faster than the monolithic approach. The main factor in scalability is the amount of pruning due to the use of restricted games in the composition. As all assembly stations are identical, there exist multiple equivalent strategies that are all compatible during each composition. This leads to a large growth of possible strategies (and therefore considered states) during each composition step. With three assembly stations, the helper arm is used continuously (see Table 3.2). The introduction of a fourth assembly leads to a large number of waiting cycles for each assembly station which leads to a sharp increase in considered states.

Algorithm 1: Compositional Synthesis

```
Data: T_1, ..., T_n and φ_1, ..., φ_n
1  P = {P_i(T_i, A_{φ_i}), | i ∈ [1, n]} ;                    // tbUCA with asm.
2  s = max{ |P| | P ∈ P};
3  thresh = 2s(s^{2s+2} + 1) ;                                // upper bound for k
4  for k = 0 to thresh do
5      P_det = {det(P, k) | P ∈ P} ;                          // determinize P_i
       /* calculate masterplan of reachable positions for all ind.  games      */
6      G = {G^i[Λ_i] | Λ_i = MP_R(G(P_i, k), F_ini)} /* compose games until global
          game is reached                                                      */
7      while |G| > 1 do
8          if ∃i : G^i ∈ G = ∅ then
9              break ;    // one of the games has no solution, retry with higher k
10         G_new = ∅;
11         find pairing of games in G;
12         for each pair found above (G^i, G^j) do
13             G_prod = G^i ⊗_a G^j;
14             Λ_prod = MP_R(G_prod, F_ini);
15             G_new = G_new ∪ {G_prod[Λ_prod]};
16         G = G_new ; // during each iteration, the total number of games halves
17     return MP_R(G[0]) ;        // all games are composed into one global game
```

Heterogenous Capabilities We further explore a scenario where assembly stations are not identical, as the homogeneity of models is not necessary in our approach. One assembly station is now disallowed to have more than two individual pieces prepared at the same time, while the other two can only assemble *ab* before *bc* and vice versa. Heterogeneity of agents does not impose additional complexity to the computation; in contrast, synthesizing a strategy is slightly more efficient in comparison the homogeneous scenario, as the added restrictions further limit the number of possible strategies during composition (see case 'het' in Table 3.1).

Reactivity Finally, we consider a scenario where the assembly stations get a simpler assembly task of a two-component workpiece *ab*, but are unable to choose to pick up a specific component and need to react according to the component they get. This non-deterministic scenario was also handled by our approach successfully and is reported on in in 'react' in Table 3.1).

Scalability While our method could outperform the monolithic approach in Section 3.3, it also suffered from the state explosion problem with more than three assembly stations. This is due to a high number of candidate strategies in each

Monolithic

case	assembly stations	tbUCA size	states considered	time
run	1	413 (806)	2036	2.1sec
	2	20998 (50700)	-	-
	3	872983 (2353750)	-	-
	4	-	-	-
het	3	447776 (1191218)	-	-
react	3	19440 (63903)	-	-

Compositional

case	assembly stations	max tbUCA size	max states cons.	time
run	1	131 (225)	449	1.0 sec
	2	131 (225)	6008	3.1 sec
	3	131 (225)	9589	3.2 sec
	4	131 (225)	-	-
het	3	112 (196)	2449	0.8 sec
react	3	40 (68)	9923	2.6 sec

Table 3.1: Size of solutions and considered states in the monolithical and compositional method. tbUCA size is given as vertices (edges). ©2020 IEEE [120]

assembly1	assembly2	assembly3	helper
$_$	$_$	...	
$_$	a	a, c	*neutral*
$_$	a, b	a, b, c	$help_3$
c	a, b, c	ab, c	*neutral*
b, c	a, b, c	abc	$help_2$
bc	ab, c	$_$	*neutral*
a, bc	abc	c	$help_1$
abc	$_$	b, c	*neutral*
$_$	a	ab, c	$help_3$
b	a, b	abc	*neutral*
a, b	a, b, c	abc	$help_2$
a, b, c	ab, c	$_$	*neutral*

Table 3.2: Execution of a synthesized strategy with different starting points. Coloured cells indicate use of the helper. ©2020 IEEE [120]

stations	max tbUCA size	max states cons.	time
2	42 (80)	79	0.1 sec
4	42 (80)	88	0.25 sec
10	42 (80)	88	0.69 sec
20	42 (80)	88	1.43 sec
40	42 (80)	88	2.84 sec
100	42 (80)	88	8.66 sec
200	42 (80)	88	16.28 sec
500	42 (80)	88	42.57 sec

Table 3.3: "Dining Philosophers". Pruning invalid solutions early leads to huge scalability capabilities. ©2020 IEEE [120]

step of the composition process. We consider yet another scenario with a set of assembly robots organized in a ring. Each robot can either prepare their own assembly or help a neighbouring robot with theirs. A robot that has started an assembly cannot finish without assistance from both neighbouring robots, so a team strategy is needed to prevent deadlocks. The task is formalized through the following LTL formula:

$$\phi_i = \Box\Diamond n_i \wedge \Box\Diamond a_i \wedge \Box(a_i \wedge \bigcirc\neg a_i \rightarrow h_{(i-1)\%n} \wedge h_{(i+1)\%n}).$$

Our approach can handle over 500 robots, while the monolithic approach already struggles with 4 robots (see Table 3.3). Here, a high degree of dependency between the robots led to restrictive assumptions, low number of candidate strategies, and extensive pruning in intermediate products.

Analysis and Discussion

Theorem 1. *Algorithm 1 is a correct and complete solution to Problem 1.*

Proof 2. *Let T_i and $\phi_i, i \in [1, n]$ be n agent models and specifications. Let Λ_{res} be a non-empty masterplan of the final composed Game G in Algorithm 1 for some $K \in \mathbb{N}$. From Lemma 1 follows that realizability of all $P_i(T_i, A_{\phi_i})$ is a necessary condition for strategies λ_i as defined in Problem 1 to exist. Furthermore, if λ_i exist, they have to be a subset of the strategies realizing $P_i(T_i, A_{\phi_i})$. Lemma 2 shows that individual masterplans of $G_i, i \in [1, n]$ are representing all policies realizing $P_i(T_i, A_{\phi_i})$. Through repeated application of Lemma 3, the allowed moves in Λ_{res} encode allowed moves in each individual subgame G^i and their respective masterplans Λ_i. If all games G^i are composed, $\widehat{AP} = \varnothing$. If Λ_{res} is non-empty, all outcomes of a play with Λ_{res} are both winning and complying with all T_i by Definition 22. So Λ_{res} is indeed encoding all possible policies λ_i as defined in Problem 1. To conclude that there exist no strategies λ_i, the upper boundary thresh in Algorithm 1 needs to be reached.* □

Composition Order Pruning of the intermediate products is a critical factor in scalability of our approach. We thus propose to compose those intermediate products in Algorithm 1 that present the largest number of "resolved" assumptions for each pair and thus reducing the assumption alphabet as quickly as possible.

Optimality Our approach does not guarantee optimality of the resulting strategies. However, since we are iteratively raising K in our incremental composition (see Section 3.2) , we will first find candidate strategies with the lowest number of steps between of consecutive visits to states that mark a progress towards fulfillment of liveness specifications. For example, the formula $\phi = \Box\Diamond x$ holds if $\neg x$ does not hold in every step at some point in the trajectory. It would hold if x holds every third state, but it also would hold if x holds every 100th state.

Long-Term Autonomy and Replanning Our approach can be used to efficiently replan in case of a change in the environment, as only one half of the composition tree needs to be redone. One could define a heuristic which defines the composition order, to compose the robots with the lowest probability of changing during long-term execution first.

Conclusion

We presented a method that improves scalability for cooperative multi-agent task planning in cases where dependency between agents is high. Invalid solutions can be pruned early, leading to great improvements in scalability with the most extreme case shown in Table 3.3. However, it does not perform well in scenarios where agents are mostly independent.

Chapter 4

Decentralized Assumption-Based Synthesis

The method presented in Chapter 3 works best when agents have a lots of dependencies and are tightly coupled together, as it is possible to prune invalid solutions early on. If we imagine scenarios where autonomous systems incidentally share environments with other actors, interactions are going to be sparse, as each actor has their own task that they generally should be suited to fulfill. Clearly, the presented method would not perform well in such a scenario. Hence, another approach is needed that is tailored for sparse interactions. In this chapter we present a decentralized approach, where agents communicate assumptions only when necessary and only to relevant actors. The aim is to otherwise have them compute strategies mostly locally.

We model the interaction of a single agent with the environment and other agents as a stochastic game G, meaning that other actors are seen as adversaries. Modelling the behaviour of other agents as adversarial is conservative, but a reactive strategy winning this game can handle all possible behaviours of agents. In sufficiently complex scenarios, it is not uncommon for such a strategy to not exist. If there is no winning strategy for player 1, we aim to limit player 2 by imposing assumptions on their behaviour such that the existence of a strategy can be guaranteed. In order for those assumptions to be meaningful, we want them to be as unintrusive as possible. Thus, a notion of minimiality has to be defined. Chatterje et. al. explored the notion of minimal assumptions for deterministic 2-player-games [56]. We expand their definitions for stochastic games and use them as a key component in our methods.

4.1 Minimal Assumptions

Any linear-time property can be decomposed into a safety and a liveness component [96]. From this insight, Chatterje et. al. show that with only two types of

assumptions, namely safety and fairness assumptions, satisfaction of any ω-regular objective can be guaranteed [56]. The safety component of an objective ψ can be expressed as

$$\psi_s = \mathit{Safe}(\langle\langle 1,2\rangle\rangle(\psi)).$$

From all states in $\langle\langle 1,2\rangle\rangle(\psi_s)$, called live states, ψ can be satisfied almost-surely if the players cooperate. Safety assumptions restrict player 2 to ensure that for every step of a play on $\widetilde{G}$, there exists a continuation that satisfies the objective ψ almost-surely. What is remaining is the liveness component that ensures player 1 can satisfy ψ almost-surely regardless of the choices of player 2. For this, strong fairness assumptions are imposed on player 2. In our methods, these assumptions are then communicated to the other agents in the form of least-liming advisers.

Safety Assumptions

A safety assumption is a set $E_s \subseteq E_2$ of edges we remove from the stochastic game G in order to find an almost-sure winning strategy for objective ψ_s. Such safety assumptions should be sufficient, realizable and minimally restrictive for player 2. We transfer the results from [56] for deterministic games to stochastic games and almost-sure winning strategies:

Definition 25 (Minimality)**.** *An assumption E_s is minimal if $|E'_s| \leq |E_s|$ for all sufficient assumptions $E'_s \in E_2$.*

Definition 26 (Safe-Sufficiency)**.** *Given an objective ψ, a safety assumption $E_s \subseteq E_2$ is safe-sufficient if player 1 has an almost-sure winning strategy for the objective*

$$\begin{aligned}&\mathit{AssumeSafe}(\psi, E_s) = \psi_s \cup \\ &\{\pi = s_0 s_1 s_2 \ldots \mid \exists k \geq 0, (s_k, s_{k+1}) \in E_s\}\end{aligned}$$

The unique, minimal safety assumption can be computed through

$$E_s = \{(s, s') \in E_2 \mid s \in \langle\langle 1,2\rangle\rangle\psi \text{ and } s' \notin \langle\langle 1,2\rangle\rangle\psi\}, \tag{4.1}$$

E_s can be computed in a straightforward way by a modified reachability algorithm on graph G in quadratic time [56].

Given a stochastic game and the objective ψ for player 1, we say that a state s is *live* for player 1, if they have an almost-sure winning strategy from s for the objective $\mathit{Safe}(\langle\langle 1,2\rangle\rangle_{as}(\psi))$.

Definition 27 (Live State)**.** *A state s is live for the objective $\mathit{AssumeSafe}(\psi, E_s)$ if and only if $s \in \langle\langle 1,2\rangle\rangle(\psi)$.*

We can incorporate safety assumptions into a stochastic game by removing edges in E_s, resulting in a new stochastic game $G = ((S, E \setminus E_s), \mathit{Act}, \delta, s_0, L)$.

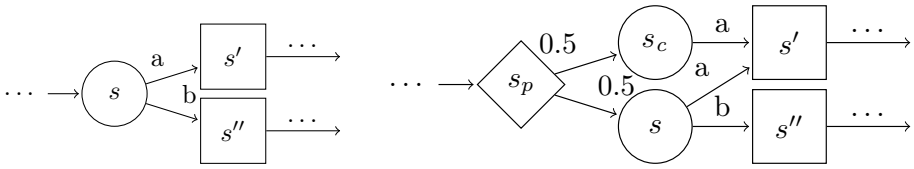


Figure 4.1: Enforcing fairness on (s, a, s') by preprending a probabilistic state. ©2021 IEEE [123]

Fairness Assumptions

A fairness assumption is a set $E_l \subseteq E_2$ of edges that need to be chosen fairly (i.e. infinitely many times upon infinitely many visits to their source). Similarly as for the safety assumptions, we desire the fairness assumptions to be sufficient and minimal. Minimality criterion is defined analogously to Definition 25.

Definition 28 (Live-Sufficiency)**.** *Given an objective ψ, a fairness assumption E_l is sufficient if player 1 has an almost-sure winning strategy for the objective*

$$\begin{aligned}&\mathit{AssumeFair}(\psi, E_l) = \psi \cup \{\pi = s_0 s_1 s_2 \dots \mid \\ &\exists (s, s') \in E_l \text{ s.t. } s_k = s \text{ for infinitely many } k \text{ but} \\ &s_{k+1} = s' \text{ only finitely often.}\}\end{aligned}$$

Informally, if a state with an outgoing edge $e \in E_l$ is visited infinitely often, we assume that e will be chosen infinitely often. A fairness assumption E_l is live-sufficient if player 1 has an almost-sure winning strategy for the objective $\mathit{AssumeFair}(\psi, E_l)$.

Lemma 4. *Let $G = ((S, E), s_i, \delta)$ be a stochastic game and ψ be an objective. If s_i is live, then there exists a minimal fairness assumption $E_l \subseteq E_2$ that is sufficient and realizable by the environment.*

In stochastic games, we can enforce fairness by the following construction: We prepend an edge $(s, a, s') \in E$ with a probabilistic state s_p. From s_p, we choose with a non-zero probability to enter a copy of state s from which only action a leading to s' can be taken; and with a non-zero probability to enter the original state s with potentially other actions enabled. An example is depicted in Figure 4.1.

Locally Minimal Fairness Assumptions Since computing a *minimal* set of fairness assumptions is NP-hard [56], we instead compute *locally minimal* fairness assumptions E_l^*. The procedure is outlined in Algorithm 2, resulting in a runtime depending linearly on $|E_2|$. We start by constructing a new stochastic game by expanding each edge $e \in \widetilde{E}_2$ to be fair, i.e. $E_{l,0}^* = E_2$, with the construction described above. If in $G'(E_{l,i}^*)$, player 1 has a winning strategy, we remove edges

from $E^*_{l,i}$ until it is not possible to do so without player 1 not having a winning strategy anymore. The resulting set $E^*_{l,i}$ is locally minimal, as no set $E' \subseteq E^*_{l,i}$ is sufficient. The order in which we remove edges from $E^*_{l,i}$ is chosen randomly.

Algorithm 2: Locally Minimal Fairness Assumptions

Data: $G = ((S, E), Act, \delta, s_0, AP_i, L,)$,
$\psi = Reach(\{(s, q) \in S' \mid q \in F\}$
1 $E^*_{l,i} = E^*_{l,0} = E_2$;
2 Construct $G'(E^*_{l,i})$ based on construction in 4.1;
3 **if** $s'_i \notin \langle\langle 1 \rangle\rangle(\psi)$ *for* $G'(E^*_{l,i})$ **then**
4 **return** False;
5 **for** $i \leftarrow 1$ **to** $|E_2|$ **do**
6 **if** $\exists e \in E^*_{l,i}$ *s.t.* $s'_i \in \langle\langle 1 \rangle\rangle(\psi)$ *for* $G'(E^*_{l,i} \setminus \{e\})$ **then**
7 $E^*_{l,i+1} = E^*_{l,i} \setminus \{e\}$;
8 **else**
9 **return** $E^*_{l,i}$;

Lemma 5. *Given minimal E_s and E_l, a winning strategy for*

$$AssumeFair(AssumeSafe(\psi, E_s), E_l)$$

is a winning strategy for the objective ψ under safety and fairness assumptions E_s and E_l.

Safety and fairness assumptions represent a key component in the following methods. They are used to communicate between computational agents, but also from computational agent to human agent.

4.2 Least-Limiting Advisers for Agent-Agent Interaction

We make use of the above notion of safety and fairness assumptions to synthesize correct-by-design strategies for multiple robots sharing an environment. In this environment, we assume that each robot has their own goals. Like in an office environment or any other workplace, we assume each actor to be able to fulfil their own goals autonomously, but interactions can occur, e.g. avoiding each other in a narrow hallway or deciding who gets to use the sink first. These assume these types of interactions to be sparse. Each robot is being tasked to fulfill an LTL specification that might be dependent on the other robots, for example avoiding collision in a critical intersection. With the behaviour of the other robots unknown, it is often infeasible to provide satisfaction guarantees in any situation, so we generate least-limiting assumptions on the behaviour of other robots and provide satisfaction guarantees

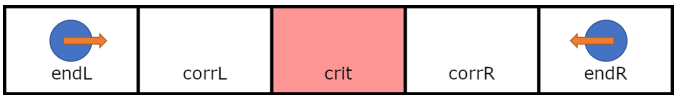


Figure 4.2: Simple MAS example. Two robots (the circles) facing each other are tasked to reach the opposite ends of a corridor, but are disallowed to enter the middle cell (crit) simultaneously. ©2021 IEEE [123]

based on these assumptions. We propose a framework where representations of the assumptions are exchanged between robots in an iterative fashion. We analyze the properties of our method and demonstrate scalability of our implementation in different use-case scenarios.

Multi-Agent System Model

Consider a group N of n (heterogeneous) agents individually modelled as labelled MDPs $\mathcal{M}_i = ((S_i = S_{i,1} \uplus S_{i,p}, E_i), Act_i, \delta_i, s_{i,0}, AP_i, L_i)$. Without loss of generality, we assume that the sets of atomic propositions $AP_1, \ldots, AP_n$ are pairwise disjunct and $AP = \bigcup_{i=1}^{n} AP_i$.

Example 9 (Running Example)**.** *Figure 4.2 illustrates two robots placed in the opposite ends of a narrow corridor. The corridor is partitioned into cells, each of which is labeled with a set of atomic propositions that hold there. A state of a robot is determined by its orientation (left, right) and the cell it occupies (*endL, corrL, crit, corrR, endR*). In each state, a robot can choose to stay, move forward, or turn* $180°$*. As a result, they change their state according to a probabilistic distribution reflecting the uncertain outcome of the chosen action, e.g., due to imprecision in actuation. We thus model each robot as an MDP, similarly as in [124][117]. Each robot has its own set of atomic propositions that can hold in its state* $AP_i = \{endL_i, corrL_i, crit_i, corrR_i, endR_i, facingR_i\}$*,* $i \in \{1, 2\}$*. An MDP for the robot in the leftmost cell is illustrated in Figure 4.3. Here, we assumed perfect execution of the actions, i.e.* $\delta(s_p, s_1) \in 0, 1$ *for each* $s_p \in S_p$ *and* $s_1 \in S_1$*.*

In our setup, each agent chooses and executes a move synchronously at each discrete time step, producing a *joint trace*:

Definition 29 (Joint Trace)**.** *Assuming multiple traces* $\tau_i = o_{i,1} o_{i,2} \cdots \in (2^{AP_i})^\omega$ *produced by plays of* $\mathcal{M}_i$*, we define the joint trace as a sequence:*

$$\tau_{joint} = (\bigcup_{i \in [1,n]} o_{i,1})(\bigcup_{i \in [1,n]} o_{i,2}) \cdots \in (2^{AP})^\omega$$

Task Specification

Each agent is given its own task to satisfy; however, the task may require occasional collaboration with others. For instance, in our running example, the two robots

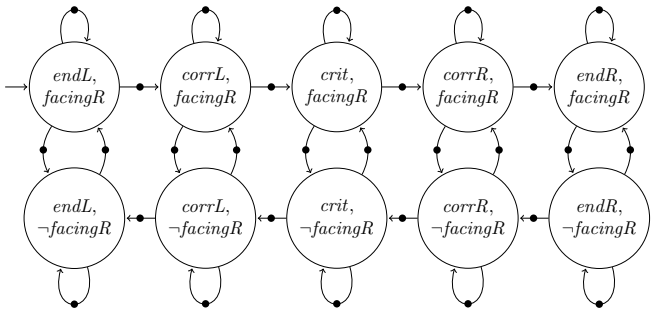


Figure 4.3: MPD of the left robot in the running example. States in S_p are depicted as small dots and the outgoing edges from them are associated with probability 1. Propositions that hold in a state are denoted inside of the state. ©2021 IEEE [123]

are given a task to reach the opposite end of the corridor. While most of the time they can move independently, they have to collaborate to avoid being in the critical section in the middle at the same time.

Formally, each agent is assigned their own LTL$_f$ task specification ϕ_i over the alphabet AP_ϕ where $AP_i \subseteq AP_\phi \subseteq AP$. We assume that each agent has full knowledge over all atomic propositions AP holding in each time step for each agent.

Example 10 (Running Example cont.)**.** *Formulas expressing the task of reaching the opposite side of the corridor and not entering the critical section at the same time are:*

$$\phi_1 = F(endR_1) \wedge G!(crit_1 \wedge crit_2)$$

$$\phi_2 = F(endL_2) \wedge G!(crit_1 \wedge crit_2)$$

Problem Definition

Problem 2. *Given a set N of n labelled MDPs $\mathcal{M}_i$ and n LTL$_f$ formulae $\phi_i, i \in N$ over AP, compute a strategy $\lambda_{i,1}$ for player 1 for each $i \in N$ such that there exists a finite prefix on the produced joint trace that satisfies all $\phi_i, i \in N$ with probability 1.*

Remark 1 (Centralized Cooperative Approach)**.** *A straightforward approach to solving problems with multiple models and multiple specifications is to compose the models $\mathcal{M}_i, i \in [1, n]$ into a single, global model $\mathcal{M} = \bigotimes_{i\in[1,n]} \mathcal{M}_i$ with composition defined by the synchronous product operator as defined in [96]. Fulfillment of all specification formulae can be assured by conjunction of partial specifications $\phi = \bigwedge_{i\in[1,n]} \phi_i$. Problem 2 can be solved by constructing $\mathcal{M} \otimes A_\phi$ and solving with standard methods. While those methods are correct and complete in theory, their exponential complexity is an issue that makes the approach practically infeasible.*

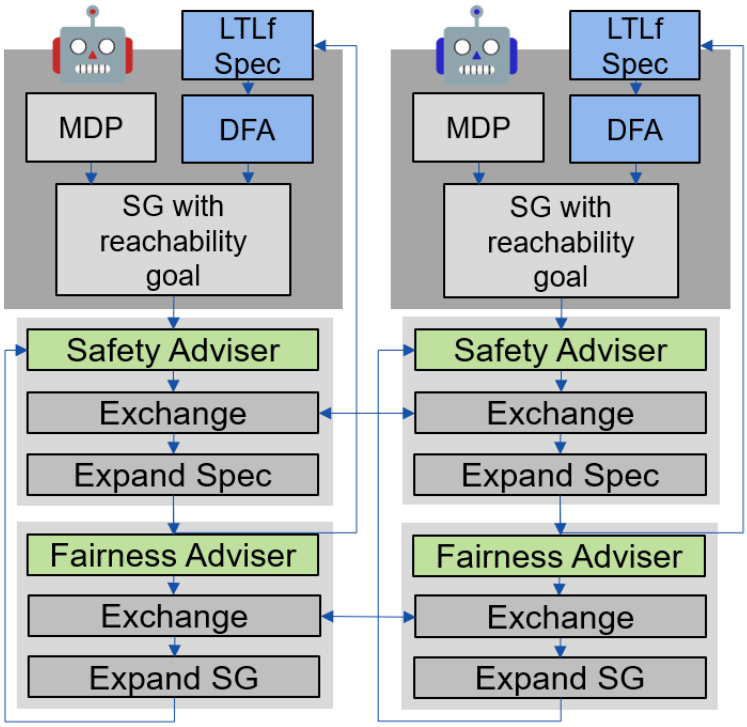


Figure 4.4: Schematic overview of the approach with two agents. ©2021 IEEE [123]

Method

A schematic overview of our approach for two agents is given in Figure 4.4 and can be scaled up to an arbitrary number of agents. Each agent i starts with the assumption that other agents are adversarial by augmenting the MDP with states $S_{i,2}$ and creating a product stochastic game with a DFA representing ϕ_i. An attempt to find an almost-sure winning strategy to this game with the objective represented by ϕ_i is made; the strategy would constitute a solution to Problem 1 for agent $i \in N$. Suppose that this attempt is not successful for one of the agents. Then, minimal, necessary assumptions on the behaviour of some of the agents are imposed and exchanged in an iterative fashion in the form of least-limiting advisers. These advisers prune edges in $E_{i,2}$ or enforce fair usage of them for some of the agents, leading to a higher chance for i to find a winning strategy. The algorithm terminates when a winning strategy for all $i \in N$ is found, meaning that we have a set of strategies that provably solve Problem 2, or if agents are unable to follow exchanged advisers, leading to concluded unsatisfiability.

Example 11 (Running Example cont.)**.** *Initially, an almost-winning strategy cannot be found for neither of the robots as the requirement to avoid entering the critical section cannot be enforced without some degree of collaboration. A safety adviser is thus synthesized by our approach that, informally, forces the robots to avoid entering the critical section when being in the adjacent state. Obeying this adviser guarantees avoiding violating* $G!(crit_1 \wedge crit_2)$*. However, a deadlock may occur this way when both robots occur in the state adjacent to the critical section. Our approach synthesises a fairness adviser that prescribes to leave the state ("back off") with a certain probability. This way, one of the robot randomly gives way to the other by turning around, signalling safe traversal of the critical section.*

Individual Strategy Synthesis

In the first step, each agent augments its MDP model $\mathcal{M}_i$ and constructs a turn-based synthesis game, modelling their interaction with other agents through a completely unrestricted player 2. At the same time, the agent's specification ϕ_i is included via DFA A_{ϕ_i} and thus translated to a reachability objective. In the following, we omit index i where clear from the context to simplify the notation.

Definition 30 (Agent-Agent Synthesis Game)**.** *Given a labelled agent MDP*

$$\mathcal{M} = ((S = S_1 \uplus S_p, E), Act, \delta, s_0, AP_i, L)$$

and a DFA

$$A_\phi = (Q, 2^{AP_\phi}, q_0, \Delta, F),$$

where $AP_i \subseteq AP_\phi \subseteq AP$*, we define the environment alphabet* $\widehat{\Sigma} = 2^{\widehat{AP}}$*, where* $\widehat{AP} = AP_\phi \setminus AP_i$*. We create a new, labelled stochastic game*

$$\widetilde{G} = ((\widetilde{S}, \widetilde{E}), Act \cup \{\epsilon\}, \tilde{\delta}, \tilde{s}_0, AP_i, \widetilde{L},),$$

as follows:

1. State space. $\widetilde{S} = \widetilde{S}_1 \uplus \widetilde{S}_2 \uplus \widetilde{S}_p$.

   a) $\widetilde{S}_1 = S_1 \times Q$ *are the states of player 1, tracking both the current state of* $\mathcal{M}$ *and* A_ϕ*;* $\tilde{s}_1 = (s_1, q)$ *for* $\tilde{s}_1 \in \widetilde{S}_1$*.*

   b) $\widetilde{S}_2 = S_p \times Q$ *are the states of player 2, choosing propositions from* $\widehat{\Sigma}$ *to represent the actions of other agents;* $\tilde{s}_2 = (s_p, q)$ *for* $\tilde{s}_2 \in \widetilde{S}_2$*.*

   c) $\widetilde{S}_p = S_p \times Q \times \widehat{\Sigma}$*;* $\tilde{s}_p = (s_p, q, \sigma)$ *for* $\tilde{s}_p \in \widetilde{S}_p$*.*

   Without loss of generality, in practice $\widetilde{S}$ *will include only the states reachable from* $\tilde{s}_0$*.*

2. Edges. $\widetilde{E} = \widetilde{E}_1 \uplus \widetilde{E}_2 \uplus \widetilde{E}_p$.

a) $\widetilde{E}_1 = \{(\tilde{s}_1, a, \tilde{s}_2) \mid \tilde{s}_1 = (s_1, q) \in \widetilde{S}_1, a \in Act, \tilde{s}_2 = (s_p, q) \in \widetilde{S}_2,$ *and* $(s_1, a, s_p) \in E\}$

b) $\widetilde{E}_2 = \{(\tilde{s}_2, \epsilon, \tilde{s}_p) \mid \tilde{s}_2 \in \widetilde{S}_2, \tilde{s}_p = (\tilde{s}_2, \sigma) \in \widetilde{S}_p,\ \sigma \in \widehat{\Sigma}\ \}$

c) $\widetilde{E}_p = \{(\tilde{s}_p, \tilde{s}_1) \in \widetilde{E} \mid \tilde{s}_p = (s_p, q, \sigma) \in \widetilde{S}_p, \tilde{s}_1 = (s_1, q') \in \widetilde{S}_1, \Delta(q, \sigma \cup L(s_1)) = q')$ *and* $s_1 \in Supp(\delta(s_p))\}$.

3. Probabilistic Transition Function. $\tilde{\delta} : \widetilde{S}_p \to Dist(\widetilde{S}_1)$. *Given* $\tilde{s}_p = (s_p, q, \sigma) \in \widetilde{S}_p$ *and* $\tilde{s}_1 = (s_1, q') \in \widetilde{S}_1$, *we define:*

$$\tilde{\delta}(\tilde{s}_p)(\tilde{s}_1) = \begin{cases} \delta(s_p)(s_1), & \text{if } \Delta(q, \sigma \cup L_s(s_1)) = q' \\ 0, & \text{otherwise.} \end{cases}$$

4. Initial State. $\tilde{s}_0 = (s_0, q_0) \in \widetilde{S}_1$.
5. State Labelling Function. $\widetilde{L} : \widetilde{S}_1 \to AP_i$. *Given* $\tilde{s}_1 = (s_1, q) \in \widetilde{S}_1$, $\widetilde{L}(\tilde{s}_1) = L(s_1)$.

Every play of $\widetilde{G}$ corresponds to a run on A_ϕ and a play on $\mathcal{M}$. Given an infinite sequence $\tilde{\pi} = \tilde{s}_0\tilde{s}_1\tilde{s}_2 \cdots \in \Pi$, the corresponding play $\pi_{\mathcal{M}} \in (S_1 \cup S_p)^\omega$ of $\mathcal{M}$ is defined through the first component $s_k \in S$ from the states $\tilde{s}_k = (s_k, q_k) \in \widetilde{S}_1 \cup \widetilde{S}_p$ of $\tilde{\pi}$. In a similar fashion, we define the corresponding run $\pi_A \in Q^\omega$ on A_ϕ through the second component $q_k \in Q$ from the states $\tilde{s}_k = (s_k, q_k) \in \widetilde{S}_1$ of the play $\tilde{\pi}$. We call a play of $\widetilde{G}$ *accepting*, if there exists an accepting prefix of the corresponding run on A_ϕ. Therefore, to satisfy specification ϕ, a run has to reach a state $(s, q) \in S$ such that $q \in F$. Once such a state is reached, the task is fulfilled. The corresponding game objective is

$$\psi = Reach(\{(s, q) \in S \mid q \in F\}) \tag{4.2}$$

and we need to decide if $s_0 \in \langle\langle 1 \rangle\rangle(\psi)$. If there exists such a strategy, it is memoryless and deterministic [125].

Lemma 6. *A memoryless deterministic almost-sure winning strategy for player 1 for the objective* $Reach(\{(s, q) \in S \mid q \in F\})$ *directly maps to a finite-memory strategy for player 1 on* $\mathcal{M}$ *that is almost-surely satisfying the specification* ϕ.

Proof 3. *(Sketch) Since every play of* $\widetilde{G}$ *corresponds to a run on* A_ϕ, *we can map the memoryless strategy on* $\widetilde{G}$ *to a finite-memory strategy on* $\mathcal{M}$, *treating the current state of* A_ϕ *as the finite memory.* □

If an almost-sure winning strategy exist for the game $\widetilde{G}$, we call the game *realizable*. Instances of such games can be solved by state-of-the-art tools, such as PRISM-games [97].

If $\widetilde{G}$ is realizable for all agents, Problem 1 is solved. If not, we limit the agents by imposing assumptions on their behaviour such that the existence of a strategy

can be guaranteed. These are communicated in the form of least-liming advisers to the other agents. We compute minimal assumptions E_s and E_l as described in Section 4.1.

From Safety Assumptions to Safety Advisers

Safety assumptions E_s are specific to the stochastic game $\widetilde{G}$ and hence cannot be directly communicated as an adviser to the other agents. Instead, we communicate the advice in the form of atomic propositions, which all agents have access to.

Definition 31 (Simplest Precondition)**.** *The simplest precondition pre for any edge $\tilde{e}_2 = (\tilde{s}_2, \tilde{s}_p) \in \widetilde{E}_2$ is defined as $pre(\tilde{e}_2) = \widetilde{L}(\tilde{s}_1)$ where $\tilde{s}_1 \in \widetilde{S}_1$ s.t. $(\tilde{s}_1, \tilde{s}_2) \in \widetilde{E}$.*

The simplest precondition is a set of atomic propositions that have to hold before an edge in the safety assumptions is taken.

Definition 32 (Safety Adviser)**.** *A* safety adviser *is a set of tuples:*

$$SafeAdv = \{(pre, \sigma) \mid pre \in AP_i, \sigma \in \widehat{\Sigma}_i\}$$

We call pre the precondition and σ the advice*; given that agent i satisfies pre, other agents should not satisfy σ in their next state.*

Incorporating Safety Advisers

After agent j broadcasts its safety adviser $SafeAdv_j$, it can assume the advice to be followed by other agents. Here, we propose a way of incorporating adviser from j into the strategy planning of an agent i. Given a safety adviser $SafeAdv_j$, agent i creates an LTL$_f$ formula for each tuple $(pre, \sigma) \in SafeAdv_j$, as follows

$$\phi_{(pre,\sigma),i} = G(pre \rightarrow \neg X proj_i(\sigma)), \tag{4.3}$$

where $proj_i(\sigma) = \{a \mid a \in \sigma \text{ and } a \in AP_i\}$. If $proj_i = \varnothing$, agent i is not affected by the advice-tuple.

We can incorporate all advisers from all agents into the specification of the agent i through conjunction:

$$\phi_{s,i} = \bigwedge_{\forall (pre,\sigma) \in SafeAdv_j, j \in N} \phi_{(pre,\sigma),i} \tag{4.4}$$

A specific case occurs for $j = i$, when instead of imposing the safety adviser through an aditional LTL$_f$ formula, we directly remove edges from $\widetilde{E}_2$.

Lemma 7. *Let $SafeAdv_j$ be a safety adviser tuple from agent j. Assuming all other agents $i \neq j$ can individually satisfy $\phi_{s,i}$, then the safety assumptions E_s of agent j are necessarily yielded.*

Proof 4. *For a given $(pre, \sigma) \in Adv_s$, assume that all agents individually satisfy $\phi_{(pre,\sigma),i}$. In any resulting joint trace $\tau = \tau_{joint}$, if $pre \subseteq \tau_k$ for some k, $proj_i(\sigma) \nsubseteq \tau_{k+1}$. Since all propositions in AP belong to some agent, $\sigma \nsubseteq \tau_{k+1}$.* □

Iterative Exchange of Safety Advisers

The iterative exchange of safety advisers is summarized in Algorithm 3. After expanding their specification with formulae (4.4), each agent constructs a new synthesis game with updated specification or prunes the game and checks for a winning strategy (lines 2-6). If there is none, the agent continues to compute additional least limiting safety advisers and broadcasts them. This is repeated until no further safety advisers are necessary or unrealizability is concluded (line 7-12).

Algorithm 3: Iterative Exchange of Safety Advisers

```
   Data: M_1, ..., M_n and φ_1, ..., φ_n
1  while any φ_i changed do
2      for i ← 1 to n do
3          compute G̃_i from M_i and φ_i;
4          prune G̃_i through all previously computed own safety advisers;
5          if s_0 ∉ ⟨⟨1,2⟩⟩(ψ) then
6              return False;
7      for i ← 1 to n do
8          compute E_{s,i} as in (4.1);
9          compute SafAdv_i as in Definition 32;
10         for j ← 1 to n do
11             if i ≠ j then
12                 φ_j = φ_j ∧ φ_{s,j} as in (4.4);
```

Lemma 8. *Algorithm 3 is guaranteed to terminate, either by not requiring additional safety advisers or by concluding unsatisfiability.*

Proof 5. *Advisers constructed as in Definition 32 only use symbols from AP, so the total amount of possible advisers is finite. If an iteration in Algorithm 3 did not lead to termination, it must have added safety advisers. Eventually, the algorithm terminates either through concluding realizability, unrealizability or by constructing all possible advisers.*

From Fairness Assumptions to Fairness Advisers

From locally minimal fairness assumptions E_l^*, we construct a fairness adviser $FairAdv$.

Definition 33 (Fairness Adviser)**.** *A* fairness adviser *is a set of tuples:*

$$FairAdv = \{(pre, \sigma) \mid pre \in AP_i, \sigma \in \widehat{\Sigma}_i\}$$

We call pre the precondition and σ the advice*; given that agent i satisfies pre, other agents should satisfy σ with non-zero probability in their next state.*

Incorporating Fairness Advisers

Given a fairness adviser $FairAdv_j$, agent i expands their synthesis game $\widetilde{G}$ to a new game $\widetilde{G}'$ by prepending each player-1 state with a probabilistic state, as in Figure 4.1, enforcing fairness on the subset of actions that lead to states $\tilde{s}$ satisfying $proj_i(\sigma) \subseteq \widetilde{L}(\tilde{s})$. If no such choice exists, then player 1 cannot fulfill the adviser in that state. In the proposed framework of this paper, we conclude that this state must be avoided and prune all player-1 choices leading to this state. Own fairness advisers $FairAdv_i$ are incorporated by applying the fairness construction in Figure 4.1 to corresponding player-2 states.

This construction guarantees that all fairness advisers are followed, and hence if a winning strategy exists for each agent's $\widetilde{G}'$ with objective ψ, then the collection of such winning strategies is a solution to Problem 1. On the other hand, the construction possibly enforces stronger constraints than needed, making the overall approach conservative.

Lemma 9. *Let (pre, σ) be a fairness adviser tuple from agent i. Assuming all agents $j \in N$ individually have an almost-sure winning strategy in $\widetilde{G}'$ with objective ψ, the fairness assumption of agent i is necessarily yielded.*

Proof 6. *If all agents $j \in N$ individually have an almost-sure winning strategy in the game $\widetilde{G}'$, then each agent chooses actions that lead to $proj_i(\sigma)$ with non-zero probability every time the preconditions pre are met, meaning that σ is fulfilled with non-zero probability in the step after pre occurs. If pre occurs infinitely often, σ will occur infinitely often with probability 1.* □

Iterative Exchange of Safety and Fairness Advisers

Incorporating fairness advisers can lead to unrealizability of the newly created game $\widetilde{G}'$, leading to the need of additional safety and/or fairness assumptions. We iteratively exchange advisers until no further advisers are necessary or unrealizability is concluded. The overall procedure is summarized in Algorithm 4.

Lemma 10. *Algorithm 4 is guaranteed to terminate, either by converging on safety and fairness advisers or by concluding unrealizability.*

Proof of Lemma 10 is analogous to the proof of Lemma 8.

Analysis

Theorem 2 (Correctness)**.** *Either Algorithm 4 returns unrealizability or the joint trace of strategies computed by satisfy all specifications almost-surely.*

Algorithm 4: Complete Adviser-Based Framework

Data: $\mathcal{M}_1, \ldots, \mathcal{M}_n$ and $\phi_1, \ldots, \phi_n$

1 **while** *any* $SafAdv_i$ *or* $FairAdv_i$ *changed* **do**
2 converge safety advisers as in Algorithm 3;
3 **for** $i \leftarrow 1$ **to** n **do**
4 compute $E^*_{l,i}$ as in Algorithm 2;
5 Construct $\widetilde{G}'(E^*_{l,i})$ based on construction in 4.1;
6 **if** $\tilde{s}'_i \notin \langle\langle 1 \rangle\rangle(\phi))$ *for* $\widetilde{G}'(E^*_{l,i})$ **then**
7 **return** False;
8 compute $FairAdv_i$ as in Definition 32;
9 **for** $i \leftarrow 1$ **to** n **do**
10 **for** $j \leftarrow 1$ **to** n **do**
11 incorporate $FairAdv_j$ as in 12;

Proof 7. *(Sketch) From Lemma 8 and Lemma 10 it follows that the algorithm terminates. Consider the case where it terminated by concluding realizability. In that case, every agent i has a winning strategy for their individual synthesis game (line 6). They must follow all broadcasted safety and fairness advisers, as they are incorporated into the game either through the specification or through expanding the game. Conversely, all of their broadcasted advisers can be assumed to be followed (Lemma 7, Proof 6). Hence, each agent's play corresponds to an accepting run on its respective DFA* $A_{\phi.i}$ *and thus the joint trace has a finite prefix that accepts all specification formulae, by which it solves Problem 1.* □

Remark 2 (Complexity)**.** *Solving a stochastic game with safety or reachability objective can be done in quadratic time [56]. Safety advisers require solving a single stochastic game with a safety objective and can thus also be computed in quadratic time. Locally minimal fairness advisers require solving multiple stochastic games, limited by the edges of current game. While fairness advisers grow the games only by a linear factor, safety advisers can grow the DFA by a doubly exponential factor. Since it is hard to put a limit on the amount of generated advisers, we do not provide a tight upper bound but in the worst case, our approach meets the complexity of reactive synthesis, which is proven to be 2EXP-complete. In all our experiments presented in Section 11, both safety and fairness adviser computation converged after one or two rounds.*

Simulation Results

We implemented our approach in Python3 using MONA [118] and *ltlf2dfa* to obtain DFA for LTL_f specifications and PRISM-games [97] to solve our stochastic game

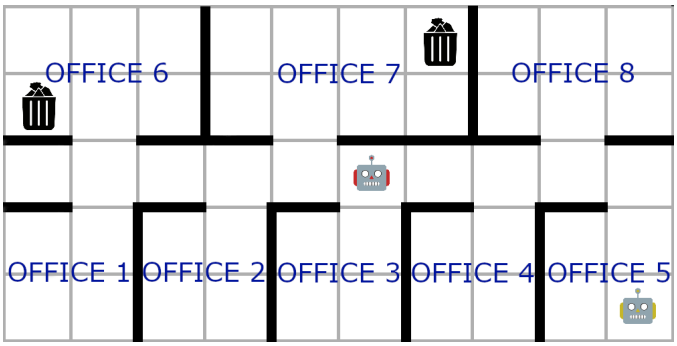


Figure 4.5: Instance of the office scenario with two robots and two bins. ©2021 IEEE [123]

		bin robots				
		1	2	3	4	5
bins	1	0.538s	0.945s	1.256s	1.699s	2.102s
	2	1.561s	2.929s	4.580s	5.784s	7.086s
	3	7.148s	12.848s	17.479s	24.686s	30.549s

Table 4.1: Average runtimes for scenario 1. ©2021 IEEE [123]

instances. Computation has been performed on an Intel i5-7600K processor and 16GB RAM. The implementation alongside installation instructions is available on Github [1].

To illustrate the scalability of the approach, we simulated an office-like environment, partitioned into 10x5 labelled cells. A schematic view of the office is shown in Figure 4.5. A state of a robot is determined by its orientation (left, right, up, down) and the cell it occupies. In each state, a robot can choose to stay, move forward, or turn 90°.

Independent Tasks

In the first scenario, each of n robots is assigned to empty all ℓ bins in the environment. The task formula for robot i is:

$$\phi_i = \bigwedge_{\forall k \in \{1,\ldots,\ell\}} F bin_{i,k},$$

where $bin_{i,k}$ is an atomic propositions encoding that the position of robot i coincides with the position of bin k.

[1] `https://github.com/gschup/ltlf-assum`

		bin robots				
		1	2	3	4	5
cleaning robots	1	7.487s	10.704s	13.929s	16.896s	21.681s
	2	12.766s	15.694s	19.724s	23.369s	29.032s
	3	20.399s	23.524s	26.839s	30.770s	37.978s
	4	29.046s	32.074s	36.786s	40.195s	47.520s
	5	43.427s	45.509s	49.680s	52.495s	66.732s

Table 4.2: Average runtimes for scenario 2. ©2021 IEEE [123]

Average runtimes of 10 runs for $n = 1, \ldots, 5$ and $\ell = 1, \ldots, 3$ with random starting positions of robots and bins are listed in Table 4.1. Since the robots are completely independent, a rising number of agents leads to a linear increase in runtime, as games are solved separately and no advisers need to be exchanged. Adding more bins to each robot's task increases the DFA size and therefore the game size and runtimes, as expected.

Spillage Cleaning tasks

In addition to n robots emptying bins, we add m cleaning robots to the scenario. Each cleaning robot $j \in \{1, \ldots, m\}$ needs to clean detected spillage in office no. o_j. It is also required to guarantee that none of the bin-emptying robots enter the affected office in order to prevent further damage. The task formula for the bin emptying robots is defined as above, the cleaning robots have the following task:

$$\phi_j = F\mathit{off}_{j,o_j} \wedge G(\bigwedge_{\forall i \in \{1,\ldots,n\}} \neg \mathit{off}_{i,o_j})$$

Average runtimes of 10 runs for $n = 1, \ldots, 5$, $m = 1, \ldots, 5$ and $\ell = 1$ with random starting positions of robots and bins are listed in Table 4.2. To guarantee task satisfaction, each cleaning robot has to broadcast safety advisers to the bin-emptying robots, disallowing them to enter the office they were assigned to clean. Adding bin-emptying robots to the scenario leads to small increases in computation time, as the computed safety advisers are efficient to implement. Adding additional cleaning robots results in a larger increase in runtime, as each needs to compute safety advisers.

Collision-free Cleaning

Finally, we consider the scenario where the bin-emptying robots are disallowed to enter the dirty office only when the cleaning robots are working there to avoid collisions, replacing ϕ_j for each of the cleaning robot with

$$\phi_j = F(\mathit{off}_{j,o_j} \wedge \bigwedge_{\forall i \in \{1,\ldots,n\}} \neg \mathit{off}_{i,o_j}).$$

		bin robots				
		1	2	3	4	5
cleaning robots	1	8.884s	11.112s	13.486s	15.869s	18.887s
	2	15.348s	18.269s	19.565s	22.699s	26.838s
	3	23.529s	25.288s	27.326s	30.395s	33.639s
	4	28.826s	32.210s	33.479s	37.456s	41.082s
	5	34.259s	39.343s	41.691s	45.421s	48.764s

Table 4.3: Average runtimes for scenario 3. ©2021 IEEE [123]

Average runtimes of 10 runs for $n = 1, \ldots, 5$, $m = 1, \ldots, 5$ and $\ell = 1$ with random starting positions of robots and bins are listed in Table 4.3. This time, fairness advisers have to be exchanged. For small games, solving the task is more efficient than in scenario 2 that required exchange of safety advisers, as safety advisers grow the size of the individual games by growing the DFA. Since implementing fairness advisers is efficient, adding additional bin emptying robots does not lead to a big increase in runtime.

4.3 Least-Limiting Advisers for Human-Agent Interaction

Since there are only two types of assumptions, we hypothesize that human actors in the environment would also be able to understand and follow advice given by a robot. We make use of the above notion of safety and fairness assumptions to synthesize correct-by-design strategies for a robot sharing the environment with one or multiple human actors. Similar to the agent-agent scenario, we assume that the human actor has their own goals. The robot is being tasked to fulfill an LTL specification that incorporates statements about the human actor in the environment. With the behaviour of the human actor unknown, it is often infeasible to provide satisfaction guarantees in any situation, so we generate least-limiting assumptions on the human behaviour and provide satisfaction guarantees based on these assumptions.

Modelling Approach

We model the robot's capabilities through a finite labelled MDP $\mathcal{M} = (S = S_1 \uplus S_p, Act, \delta, s_0, AP_r, L)$, where AP_r is a set of atomic propositions that describe the status of the robot with respect to its tasks, and $L : S \to AP_r$ is a function that labels state with atomic propositions, which hold there. For simplicity, we assume that each transition (action execution) takes one time unit and the robot's environment is observable at all times. Probabilistic transitions are used to model probabilistic outcomes of chosen actions. These probabilities could originate from modelling success rates of the actions or unmodelled disturbances of the environment.

The robot is given a high-level task formulated as an LTL_f specification ϕ over AP_ϕ. AP_ϕ is a set of atomic propositions that include AP_r as well as other propositions regarding the status of the environment, or the human in the loop. This means the robot's task may involve subtasks that depend on actions of the human. The robot's aim is to satisfy the formula almost-surely; our aim is to find a strategy under which it does so along with assumptions on the behavior of the human in the loop.

Example 12. *A simple example of a robot in an environment is given in Figure 4.6. The robot can move in the bottom six grey cells, interact with the five ingredients on the table in the center and it can also deliver at the designated delivery tray at the bottom. This is modeled as an MDP illustrated in Figure 4.7. The states S of $\mathcal{M}$ describe the position of the robot, expressed through the cell it currently occupies. In each state, the robot can move to an adjacent cell in either of the four cardinal directions or interact with the ingredient or delivery trays. The self-loops indicate not taking an action and idling for one time unit. In this particular case, the probability of each depicted transition is 1 and the action names are omitted for readability.*

The robot's task is to interact with all the ingredients of the table in the center once. The specification formula

$$\phi = \phi_g \wedge \phi_c \tag{4.5}$$

is split into goals and constraints. The goal formula ϕ_g describes that the robot should interact with all ingredient trays, regardless of their order and that it requires help from the human when interacting with the ketchup. The constraint formula ϕ_c forbids the collision when both actors reach into the same tray at the same time at all other ingredient trays. Clearly, to accomplish this task, the robot will need to assume some cooperation from the human.

$$\begin{aligned} \phi_g = {} & F\ buns_r \wedge F\ patty_r \wedge F\ lettuce_r \\ & \wedge\ F(ketchup_r \wedge ketchup_h) \wedge F\, tomato_r \end{aligned} \tag{4.6}$$

$$\begin{aligned} \phi_c = {} & G\ \neg(buns_r \wedge buns_h) \wedge G\ \neg(patty_r \wedge patty_h) \\ & \wedge\ G\ \neg(lettuce_r \wedge lettuce_h) \wedge G\ \neg(tomato_r \wedge tomato_h) \end{aligned} \tag{4.7}$$

In this work, we do not aim to explicitly represent the human's states, goals or intention through a model. For task planning purposes, we abstract the influence of the human's actions on the environment relevant to the robot's task through a set of atomic propositions and their changes. We denote these atomic propositions $AP_h = AP_\phi \setminus AP_r$. The robot considers the human to be able to freely change an arbitrary subset of atomic proposition from AP_h in every step and the assumptions that we aim to synthesize take the form of restrictions on changes to this set.

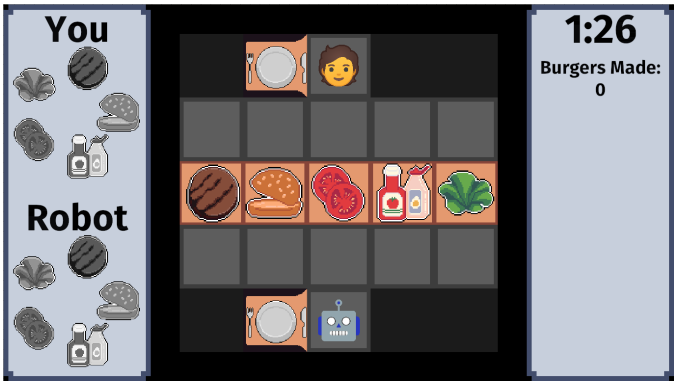


Figure 4.6: The study game screen with the playing field in the center, ingredient inventory on the left, timer and burger counter on the right.

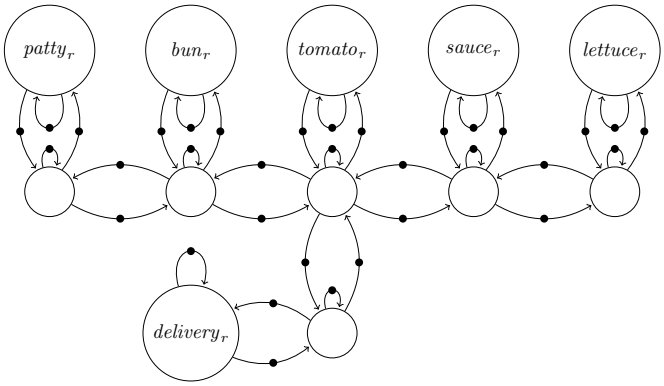


Figure 4.7: Robot MDP of the study game. States in S_p are depicted as small dots and the outgoing edges from them are associated with probability 1. Propositions that hold in a state are denoted inside of the state.

We model the interaction through a stochastic game. Starting from the initial state, the robot represented through player 1 makes a decision on its move, takes an action from Act and induces a change to propositions from AP_r. Meanwhile, the human represented through player 2 may make a change to propositions from AP_h. Note that this change can be triggered by one or more actions of the human. The remaining half player corresponds to the resolution of probabilistic outcomes of the robot MDP's actions. While a concurrent game seems to be the most suitable choice, we opt for turn-based games as they are simpler to handle algorithmically with the objectives at hand. At the same time, in a turn-based game, player 2 is a stronger adversary, so winning resulting strategies for player 1 in a turn-based game are conservative, but also winning strategies for player 1 in a concurrent game,

where player 2 has to choose without knowledge of the action of player 1.

Definition 34 (Interaction Model)**.** *Given a robot represented as a labelled MDP* $\mathcal{M} = ((S = S_1 \uplus S_p, E), Act, \delta, s_0, AP_r, L)$ *and a set of propositions controlled by the human actor* $\Sigma_h = 2^{AP_h}$*, we create a labelled stochastic game*

$$\widetilde{G} = ((\widetilde{S}, \widetilde{E}), Act \cup \{\epsilon\}, \tilde{\delta}, s_0, AP_r, L)$$

as follows:

1. State space. $\widetilde{S} = S_1 \uplus S_p \uplus \widetilde{S}_p$*, where* $\widetilde{S}_p = S_p \times \Sigma_h$ $\tilde{s}_p = (s_p, \sigma)$ *is the state of the game after the robot took an action and the human chose atomic propositions that hold true now. Without loss of generality, in practice* $\widetilde{S}$ *will include only the states reachable from* $\tilde{s}_0$*.*
2. Actions. *The actions of the game are the actions of the robot; the actions of humans as well as the half-player are not labeled.*
3. Edges. $\widetilde{E} = E \uplus E_p \uplus \widetilde{E}_p$*.*
   - $E_p = \{(s_p, \tilde{s}_p) \mid s_p \in S_p, \tilde{s}_p = (s_p, \sigma) \in \widetilde{S}_p\}$ *represent a change in the propositions triggered by the human.*
   - $\widetilde{E}_p = \{(\tilde{s}_p, s_1) \mid \tilde{s}_p \in \widetilde{S}_p, s_1 \in S_1, s_1 \in Supp(\delta(s_p))\}$ *resolve the probabilistic outcome of the actions of the robot MDP.*
4. Probabilistic Transition Function. $\tilde{\delta} : \widetilde{S}_p \to Dist(S_1)$*. Given* $\tilde{s}_p = (s_p, \sigma) \in \widetilde{S}_p$*, we define:*
$$\tilde{\delta}(\tilde{s}_p)(s_1) = \delta(s_p)(s_1).$$

Strategy Synthesis

In the next step, we introduce the specification of the robot into the game. First, we translate ϕ into a DFA A_ϕ. We then construct a stochastic game with a reachability objective that is determined from the accepting states of the DFA. In this game, finding an almost-surely winning strategy translates to satisfying the specification ϕ.

Definition 35 (Agent-Human Synthesis Game)**.** *Given the interaction model* $\widetilde{G} = ((\widetilde{S}, \widetilde{E}), Act \cup \{\epsilon\}, \tilde{\delta}, s_0, AP_r, L)$ *and a DFA* $A_\phi = (Q, 2^{AP_\phi}, q_0, \Delta, F)$*, where* $AP_r \subset AP_\phi, AP_h = AP_\phi \setminus AP_r$ *and* $\Sigma_h = 2^{AP_h}$*, we create a labelled stochastic game*

$$\widehat{G} = ((\widehat{S}, \widehat{E}), Act \cup \{\epsilon\}, \hat{\delta}, \hat{s}_0, AP_r, \widehat{L}),$$

as follows:

1. State space. $\widehat{S} = \widetilde{S} \times Q$*. All states carry information about the current state of the interaction model and the DFA.*

2. Edges. $\widehat{E} = \widehat{E}_1 \uplus \widehat{E}_2 \uplus \widehat{E}_p$. *The state of the DFA is updated on edges in* $\widehat{E}_p$. *All other edges keep their original mapping, carrying over the current DFA state.*

   a) $\widehat{E}_1 = \{(\hat{s}_1, a, \hat{s}_2) \mid \hat{s}_1 = (\tilde{s}_1, q) \in \widehat{S}_1, a \in Act, \hat{s}_2 = (\tilde{s}_p, q) \in \widehat{S}_2,$ *and* $(\tilde{s}_1, a, \tilde{s}_p) \in \widetilde{E}\}$

   b) $\widehat{E}_2 = \{(\hat{s}_2, \epsilon, \hat{s}_p) \mid \hat{s}_2 = (\tilde{s}_2, q) \in \widehat{S}_2, \hat{s}_p = (\tilde{s}_p, q) \in \widehat{S}_p,$ *and* $(\tilde{s}_2, \epsilon, \tilde{s}_p) \in \widetilde{E}\}$

   c) $\widehat{E}_p = \{(\hat{s}_p, \hat{s}_1) \mid \hat{s}_p = (s_p, q, \sigma) \in \widehat{S}_p, \hat{s}_1 = (\tilde{s}_1, q') \in \widehat{S}_1, \Delta(q, \sigma \cup \widetilde{L}(\tilde{s}_1)) = q')$ *and* $\tilde{s}_1 \in Supp(\tilde{\delta}(\tilde{s}_p))\}$.

3. Probabilistic Transition Function. $\hat{\delta} : \widehat{S}_p \to Dist(\widehat{S}_1)$. *Given* $\hat{s}_p = (s_p, q, \sigma) \in \widehat{S}_p$ *and* $\hat{s}_1 = (\tilde{s}_1, q') \in \widehat{S}_1$, *we define:*

$$\hat{\delta}(\hat{s}_p)(\hat{s}_1) = \begin{cases} \tilde{\delta}(\tilde{s}_p)(\tilde{s}_1), & \text{if } \Delta(q, \sigma \cup \widetilde{L}_s(\tilde{s}_1)) = q' \\ 0, & \text{otherwise.} \end{cases}$$

4. Initial State. $\hat{s}_0 = (s_0, q_0)$.
5. State Labelling Function. $\widehat{L} : \widehat{S}_1 \to AP_r$. *Given* $\hat{s}_1 = (s_1, q) \in \widehat{S}_1$, $\widehat{L}(\hat{s}_1) = L(s_1)$.

The objective of the game is to reach an accepting state of the DFA:

$$\psi = Reach(\{(s, q) \in S \mid q \in F\}), \tag{4.8}$$

If the robot can find an almost-surely winning strategy in $\widetilde{G}$ for ψ, it satisfies ϕ regardless of what the human does. If not, additional assumptions on the human's actions, or more precisely their effect on the atomic proposition, are necessary to offer guarantees.

We synthesize minimal assumptions as described above. A key insight is that given a set of the weakest sufficient assumptions E_s, E_l, there exists an almost-surely winning strategy in $\widehat{G}$ for the objective

$$AssumeFair(AssumeSafe(\psi, E_s), E_l).$$

In other words, assuming that edges from E_s are never taken, while edges form E_l are fairly taken, specification ϕ is guaranteed to be satisfied. In other words, if the human is able and willing to never change the atomic propositions that trigger edges E_s in $\widehat{G}$ and to eventually change the atomic propositions that trigger edges E_l in $\widehat{G}$, the robot will be able to provably accomplish its task.

Communicating the Assumptions via Advice

After computing the weakest sufficient assumptions E_s and E_l and the corresponding robot's strategy, the assumptions have to be communicated to the human in a way they can easily understand and follow. Hence, instead of communicating these sets directly, we propose to monitor the robot and the human (i.e. the state of the game $\widetilde{G}$) and issue advice at runtime. In particular, if one of the outgoing edges of the current state is found in the safety assumption set E_s, we identify the propositions σ from $(\tilde{s_2}, \tilde{s_p}) \in E_s$, and $\tilde{s_p} = (s_p, q, \sigma)$. The safety assumption is then communicated as forbidden changes to the atomic propositions. In Example 12, this could be for instance communicating to the human that they should not interact with the patty when the robot does so. The fairness assumptions can be communicated as a suggestion to make a particular change to atomic propositions. In Example 12, this could be for instance suggesting to the human that they should eventually interact with the sauce while the robot waits for their help. If this advice is followed at all times, the assumptions are fulfilled and hence the guarantees hold. We give a concrete example of communicating the assumptions via advice below in Section 4.3.

Using Least-Limiting Advice in an Online User Study

The full source code for game construction, solving and assumption computation is available on GitHub[2]. We used Python 3 to construct and manipulate the stochastic games and PRISM-games [97] to solve instances. Source code of the study and raw result data are in a separate repository[3]. The interactive game is written in Rust and compiled to WASM, so participants could play in their browser. The repository also includes other experimental materials such as questionnaires.

We designed an online user study where participants play an interactive game with a robot. We examine if humans are able and willing to follow computed assumptions given by the robot, how they evaluate the experience, and the effect of communicating assumptions on the specification satisfaction.

Interactive Game Design

A snapshot of the interactive game is illustrated in Figure 4.6. The robot's capabilities, MDP model, and specification are discussed in Example 12. Before playing the game, the participants are given instructions that explain the game in detail. They control the character represented by the human icon via arrow keys. Every time the participant moves, the robot moves as well.

The participants are tasked to assemble hamburgers alongside their robotic coworker and to collectively produce as many burgers as possible in a given time limit. For this, they have to collect all necessary ingredients from ingredient trays

[2] `https://github.com/KTH-RPL-Planiacs/human-advisers`
[3] `https://github.com/KTH-RPL-Planiacs/human-adviser-study`

and then deliver the burger to the plate at the top. An inventory of ingredients is displayed on the left of the screen. Collected ingredients are coloured, otherwise they are greyed out. On the right, a timer shows the remaining time until the game is over and a counter displays the total amount of delivered burgers.

The participants are aware that for safety reasons, the human and the robot should not reach into the same tray at the same time. If the participant and the robot move into the same ingredient tray at the same time, they crash and the screen fades to black, the currently collected ingredients are lost and the robot's and participant's positions are reset to the starting positions. The synthesis game $\widetilde{G}$ restarts to the initial state also every time the robot satisfies the specification. Advice is communicated via pictograms. Safety advice forbids certain atomic proposition changes (coming, e.g. from the safety assumption E_s) is communicated by a red cross as illustrated in Figure 4.8 in the middle. Fairness advice suggests that a certain atomic proposition should change eventually (coming, e.g. from the fairness assumption E_l) is communicated by a green arrow as illustrated in Figure 4.8 in the middle. Finally, advice that dictates to the human its next move is communicated via a black arrow as illustrated in Figure 4.8 in the bottom.

Study design

We compare our approach (*LeastLimiting*) to two baselines:

- *NoAdvice*: The robot does not issue any advice.
- *NextMove*: The robot communicates to the user exactly which move to take in every step. The moves are determined by computing a cooperative strategy for the synthesis game. In other words, we treat the human as fully controllable. Following the *NextMove* advice leads to provable satisfaction of the robot's specification Equation 4.5; this approach however does not focus on the advice to be the least limiting.

The main purpose of our study was to answer (1) whether the *LeastLimiting* advice lead to less specification violations than *NoAdvice* and (2) whether the *LeastLimiting* advice were perceived as less limiting than the *NextMove* advice. We recorded the following data in-game:

- total steps taken by the human
- number of burgers produced by the robot
- number of burgers produced by the human
- number of specification violations measured through violation of safety constraints in (4.7)

After playing the game for two minutes, we asked the participants to rate their impressions of the robot on semantic differential scales from 1-5 between different adjective pairs. The questionnaire aims to study three aspects: perceived intelligence, perceived safety and perceived hindrance. All attribute pairs are shown in

Figure 4.8: From left to right: Next-Move Advice, Safety Advice and Fairness Advice

	Least Limiting	Next Move	None
Perceived Intelligence			
Incompetent-Competent	4.05 (0.91)	3.42 (1.54)	3.81 (1.14)
Ignorant-Knowledgeable	4.22 (0.85)	3.56 (1.52)	3.97 (1.00)
Irresponsible-Responsible	3.89 (0.96)	3.33 (1.31)	3.78 (1.07)
Unintelligent-Intelligent	3.78 (1.16)	3.44 (1.44)	3.78 (1.22)
Perceived Safety			
Anxious-Relaxed	4.17 (1.00)	3.11 (1.47)	3.94 (1.31)
Agitated-Calm	4.06 (0.98)	3.03 (1.54)	4.00 (1.27)
Quiescent-Surprised	3.78 (1.05)	3.39 (1.22)	3.44 (1.18)
Perceived Hindrance			
Unpredictable-Predictable	3.95 (0.91)	3.53 (1.34)	3.81 (1.14)
Incompliant-Compliant	3.86 (0.95)	2.97 (1.40)	3.69 (1.21)
Unrestrictive-Restrictive	3.86 (0.95)	3.89 (1.01)	3.47 (1.23)

Table 4.4: Means and standard deviations (in parentheses) of the questionnaire ratings.

Table 4.4. For perceived intelligence and safety, we used the corresponding parts from the Godspeed questionnaire [126]. Since the "perceived hindrance" scale does not belong to a validated questionnaire, we analysed its component items individually. In addition, we also collected anonymous demographics data.

Study Results

We conducted the study on Amazon Mechanical Turk, and invited 120 participants evenly split over the three conditions. Of the 120 participants, 50 self-described as female, 57 as male, and 2 as non-binary; their age ranged from 24 to 72 years old (median = 36). The vast majority (N = 85) self-described as native English speakers, 5 as native-like, 14 as fluent, and 5 did not respond. Finally, we asked them about their experience with playing video games: 42 people reported playing every day, 46 weekly, 14 monthly, and 7 did not respond. We paid each participant $2 plus a performance bonus based on the amount of burgers they produced in the game. The study took on average 18 minutes, with the game taking a fixed two minutes. We conducted the study in accordance with the ethical guidelines of the hosting institution. After data collection, some participants had to be excluded due

to poor performance, or for signing up to the study twice. As a result, the final dataset contained data from 109 participants.

All analyses were performed in R version 4.2.1. For the questionnaire data, we conducted one-way ANOVAs to see if the adviser mode had any effect on the different scales. We found that adviser mode significantly influenced perceived safety ($F(2, 105) = 6.27$, $MSE = 0.63$, $p = .003$, $\hat{\eta}^2_G = .107$); post-hoc pairwise comparisons using the Tukey HSD test showed that the LeastLimiting condition was perceived as safer than the Next Move condition ($p = .009$). The effect of the adviser mode approached significance at the .05 significance level for perceived intelligence ($F(2, 106) = 2.88$, $MSE = 1.16$, $p = .061$, $\hat{\eta}^2_G = .052$), with the LeastLimiting condition being perceived as more intelligent than the Next Move condition (Tukey HSD $p = .05$). Finally, for perceived hindrance, we only found an effect of adviser mode in the Incompliant - Compliant adjective pair ($F(2, 106) = 5.65$, $MSE = 1.44$, $p = .005$, $\hat{\eta}^2_G = .096$), with the LeastLimiting condition being perceived as more compliant than the Next Move condition (Tukey HSD $p = .005$).

We conducted a Poisson regression on the number of safety violations, and we found that there were significantly fewer violations in the LeastLimiting condition than in the No-Advice baseline ($b = 0.62$, 95% CI $[0.08, 1.18]$, $z = 2.21$, $p = .027$). At the same time, burger productivity remained roughly the same between the two. The total amount of burgers produced were 142 in the LeastLimiting condition, 154 in the NoAdvice condition and 76 in the NextMove condition.

Conclusion - User Study

Safety and fairness assumptions can be communicated as advice to human actors. Our study showed that participants are able to follow the instructions generated from otherwise hard to understand elements of a mathematical object. An important challenge is the complexity of information that can be attached to a single assumption. Since an edge can be characterized by an arbitrary boolean logical formula, translation into pictograms might not always be feasible as it was in our scenario. Section 5.2 tries to alleviate this issue in object-centric tasks by providing advisers generated from spatial relations.

4.4 Conclusion

We utilize minimal assumptions as a unified basis for communication between multiple computational agents, but also between computational agents and human actors. In both cases, minimal assumptions need to transformed into a suitable representation that is understandable by other actors and thus does not rely on the internal stochastic game of the agent. In agent-agent communication, least-limiting advisers serve as a necessary, but overapproximative tool to tackle the scalability challenge of multi-agent task planning. In human-robot interaction, least-limiting advisers represent a way of communication, but also an approach to modelling the human without imposing an explicit model of their preferences or capabilitites.

Chapter 5

Object-Centric Tasks

Object manipulation is a big application area for autonomous robots. Ranging from food preparation as in the previous burger examples to industrial assembly, examples for object-centric robot tasks are not hard to find. Many manipulation tasks involve sequences such as multi-step assemblies. Temporal logics could be benefical to correctly specify such sequences of steps.

However, the boolean atomic propositions that LTL is based on are usually used to describe discrete states of a system. For autonomous systems in the real world, this usually means involvement of some sort of abstraction from the continuous domain. As a simple example, the boolean proposition *battery_almost_empty* could be true if the battery is below a certain threshold. When dealing with object-centric tasks, such as assembly or object manipulation, such abstractions can be unwieldy. In addition to satisfying a specification, we might also care about finding an optimal position or reasoning about how close to violation a part of a specification is. For such applications, *Signal Temporal Logic* comes equipped with spatial robustness metrics. Boolean propositions are replaced with continous signal functions that allow to evaluate the world in a more natural manner. However, it can be quite hard to read and specify constraints and goals via STL, as a temporal logic formula with multiple nested functions can be hard to parse.

In this chapter, we focus on intuitive task specifications for object-oriented tasks and present our framework SpaTiaL. We establish spatial relations (e.g. *fork leftOf plate*) that are both close to natural language and well defined through signed distances between polygons. With spatial relations as it's foundational building block, SpaTiaL can be used to specify and reason about object-centric tasks more intuitively.

We demonstrate it's usefulness in several experiments, ranging from monitoring to online planning. Finally, we define the notion of a *minimal request* and use SpaTiaL have a manipulation robot ask a human operator for help in case the planner cannot find or the executing platform cannot fulfill a necessary step for task satisfaction.

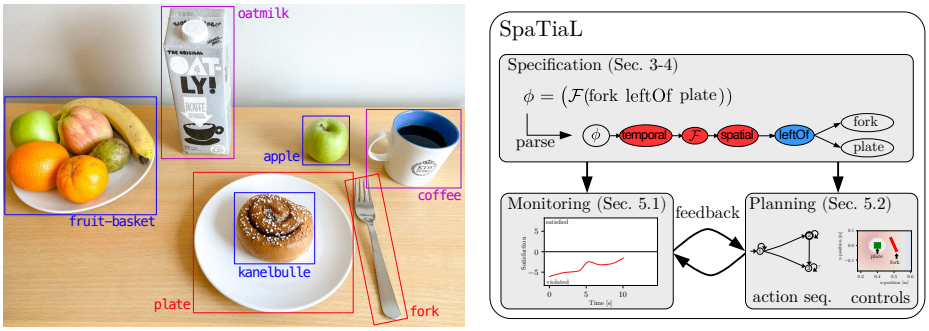


Figure 5.1: Example of a kitchen task that requires a complex interplay of spatial and temporal constraints. The robot needs to arrange the objects in a predefined temporal order, e.g., first place the fork left of the plate. Moreover, it needs to consider time bounds such as placing the filled coffee mug within 10 seconds to the right side but close to the plate after the kanelbulle has been placed within the plate. This specification requires designers to deal with various object shapes, object detection uncertainties, and temporal sequencing. SpaTiaL parses user specifications and monitors their satisfaction in real-time, providing feedback to planners.

5.1 SpaTiaL: An Object-Centric Specification Framework for Robotics

Spatial Relations between Objects

Many tasks require robots to arrange objects in desired configurations relative to each other. For instance, an object $\mathcal{O}_i$ needs to be placed close to and left of object $\mathcal{O}_j$. Our goal is to define spatial relations in continuous space to eliminate discretization effects [127, 128] and to provide quantitative semantics, i.e., a continuous measure about the extent that a scenario satisfies a spatial relation. In the following four subsections, we use signed distances, projections and cosine similarity to derive spatial relations between objects and introduce quantitative semantics to measure the satisfaction of a specification given a state s_t of the scenario (more details on monitoring can be found in Section 5.1).

Distance-Based Relations

To derive distance-based spatial relations, such as an $\mathcal{O}_i$ is close to $\mathcal{O}_j$, we rely on signed distance computations between convex objects that are used in computational geometry for robotic applications [129, 130, 131]. These approaches define constraints between geometric shapes through distances between them, e.g., an object cannot be closer to another object by a predefined margin [132]. We derive

spatial relations by exploiting the signed distance between objects. The signed distance is computed using the distance and penetration depth. The distance between two objects is defined as:

Definition 36 (Distance)**.** *The distance* $\mathrm{d}(\mathcal{O}_i, \mathcal{O}_j)$ *between two objects,* $\mathcal{O}_i$ *and* $\mathcal{O}_j$*, is the norm of the smallest translation so that both objects intersect:*

$$\mathrm{d}(\mathcal{O}_i, \mathcal{O}_j) := \inf \left\{ \|T \in \mathbb{R}^d\| \mid (T \oplus \mathcal{O}_i) \cap \mathcal{O}_j \neq \emptyset \right\}.$$

Intuitively, the distance $\mathrm{d}(\mathcal{O}_i, \mathcal{O}_j)$ is larger than zero when $\mathcal{O}_i$ and $\mathcal{O}_j$ do not intersect and zero otherwise. In the case of $\mathcal{O}_i \cap \mathcal{O}_j \neq \emptyset$, we are interested in computing the distance required to get both objects out of contact, denoted as the penetration depth:

Definition 37 (Penetration Depth)**.** *The penetration depth* $\mathrm{pd}(\mathcal{O}_i, \mathcal{O}_j)$ *between two objects,* $\mathcal{O}_i$ *and* $\mathcal{O}_j$*, is the norm of the smallest translation so that they do not intersect:*

$$\mathrm{pd}(\mathcal{O}_i, \mathcal{O}_j) := \inf \left\{ \|T \in \mathbb{R}^d\| \mid (T \oplus \mathcal{O}_i) \cap \mathcal{O}_j = \emptyset \right\}.$$

The distance and penetration depth can be efficiently computed with the Gilbert-Johnson-Keerthi algorithm (GJK) [133, Sec. 9.5] in linear or logarithmic time complexity with respect to the number of vertices. GJK is a popular choice in real-time collision checking and uses Minkowski differences between convex polygons to compute distances in the polygon's configuration space. Definition 36 and Definition 37 are complementary to each other and can never simultaneously be non-zero. Finally, we define the signed distance as [134]:

Definition 38 (Signed Distance)**.** *The signed distance* $\mathrm{sd}(\mathcal{O}_i, \mathcal{O}_j)$ *between two objects,* $\mathcal{O}_i$ *and* $\mathcal{O}_j$*, is given by:*

$$\mathrm{sd}(\mathcal{O}_i, \mathcal{O}_j) := \mathrm{d}(\mathcal{O}_i, \mathcal{O}_j) - \mathrm{pd}(\mathcal{O}_i, \mathcal{O}_j).$$

The signed distance between two objects is positive if they do not intersect, zero when they are touching, and negative when their interiors intersect. Figure 5.2 illustrates these three cases. The signed distance allow us to define spatial relations for describing if two objects are in a certain distance or overlap:

Proposition 1 (Distance and Overlap Relations)**.** *Two objects* $\mathcal{O}_i$ *and* $\mathcal{O}_j$ *are in* ϵ*-range (*$\mathrm{closeTo}_\epsilon$*) of each other if:*

$$\mathcal{O}_i \,\mathrm{closeTo}_\epsilon\, \mathcal{O}_j := \mathrm{sd}(\mathcal{O}_i, \mathcal{O}_j) \leq \epsilon,$$

and overlapping (ovlp*) if:*

$$\mathcal{O}_i \,\mathrm{ovlp}\, \mathcal{O}_j := \mathrm{sd}(\mathcal{O}_i, \mathcal{O}_j) \leq 0.$$

Proof 8. *The proof follows directly from the properties of the signed distance between two objects.*

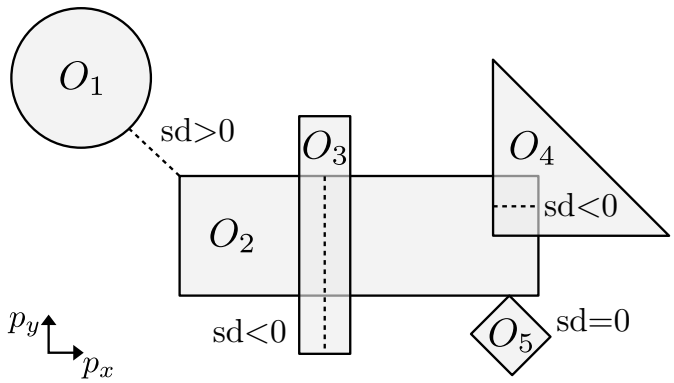


Figure 5.2: Illustration of signed distances sd (dashed lines) between various objects: $sd > 0$ when objects do not intersect (e.g., $\mathrm{sd}(\mathcal{O}_1, \mathcal{O}_2)$), $sd = 0$ when objects touch (e.g., $\mathrm{sd}(\mathcal{O}_2, \mathcal{O}_5)$), and $sd < 0$ when the interior of objects intersects (e.g., $\mathrm{sd}(\mathcal{O}_2, \mathcal{O}_3)$).

By exploiting the convexity of the object shapes, we can also determine whether an object is enclosed in another object:

Proposition 2 (Enclosed Within Relation)**.** *The shape of an object $\mathcal{O}_i$ is fully enclosed in (*enclIn*) the shape of object $\mathcal{O}_j$ if:*

$$\mathcal{O}_i \,\mathrm{enclIn}\, \mathcal{O}_j := \forall p \in \mathrm{hull}(\mathcal{O}_i) : \mathrm{sd}(p, \mathcal{O}_j) \leq 0$$

Proof 9. *When $\forall p \in \mathrm{hull}(\mathcal{O}_i) : \mathrm{sd}(p, \mathcal{O}_j) \leq 0$, then all points of the hull of $\mathcal{O}_i$ are enclosed in $\mathcal{O}_j$. Since our objects are convex, it follows that every line segment of $\mathcal{O}_i$ is also enclosed, i.e., $\forall p_1, p_2 \in \mathrm{hull}(\mathcal{O}_i), \forall : \alpha \in [0, 1] : (1 - \alpha)p_1 + \alpha p_2 \in \mathcal{O}_j$.*

Figure 5.3 illustrates the proof of Proposition 2 for two objects $\mathcal{O}_i$ and $\mathcal{O}_j$.

Definition 39 (Derived Distance-based Spatial Relations)**.** *The spatial relations* far from *(*farFrom*),* touch *(*touch *, with numeric parameter),* partial overlap *(*partOvlp *), and* closer to than *(*closerTo*) are derived from the distance-based relations:*

$$\begin{aligned}
\mathcal{O}_i \,\mathrm{farFrom}_\epsilon\, \mathcal{O}_j &:= \neg \mathcal{O}_i \,\mathrm{closeTo}_\epsilon\, \mathcal{O}_j \\
\mathcal{O}_i \,\mathrm{touch}\, \mathcal{O}_j &:= \mathcal{O}_i \,\mathrm{closeTo}_\epsilon\, \mathcal{O}_j \wedge \\
&\quad\; \mathcal{O}_i \,\mathrm{farFrom}_{-\epsilon}\, \mathcal{O}_j, \\
\mathcal{O}_i \,\mathrm{partOvlp} &:= \mathcal{O}_i \,\mathrm{ovlp}\, \mathcal{O}_j \wedge (\neg \mathcal{O}_i \,\mathrm{enclIn}\, \mathcal{O}_j) \\
\mathcal{O}_i \,\mathrm{closerTo}\, \mathcal{O}_j \text{ than } \mathcal{O}_k &:= \mathrm{sd}(\mathcal{O}_i, \mathcal{O}_j) \leq \mathrm{sd}(\mathcal{O}_j, \mathcal{O}_k)
\end{aligned}$$

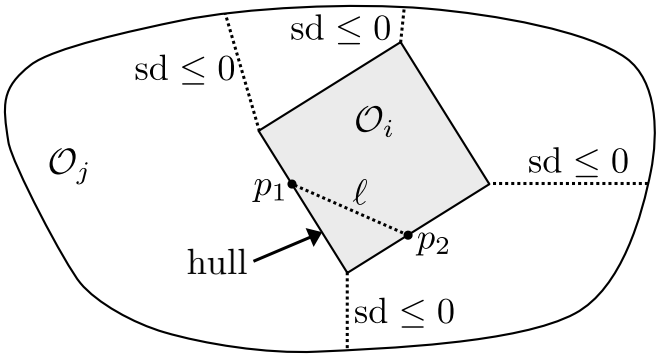


Figure 5.3: The objects $\mathcal{O}_i$ and $\mathcal{O}_j$ are convex, i.e., every line segment ℓ between any two points on the hull is also enclosed. Therefore, $\mathcal{O}_i$ is enclosed in $\mathcal{O}_j$ if the signed distance of every point on the hull is negative or zero.

Projection-Based Relations

The distance-based spatial relations already allow us to specify a variety of object configurations and to derive additional relations, such as an object being far away or in a certain range of distance values to another object. To place objects more precisely, we are further interested in specifying directions, e.g., an object is left of another one. We can derive such relations by projecting the objects onto a common coordinate system axis and determine their order.

Definition 40 (Projection)**.** *The projection operator* $\mathrm{proj}_{\mathrm{ax}}(\mathcal{O})$ *projects the shape of an object* $\mathcal{O}$ *onto the axis* ax*.*

Since our objects are compact and convex sets, the projection operator returns a closed interval. We use interval orders to determine the left-to-right precedence relation between two given intervals [135]:

Definition 41 (Interval Precedence)**.** *An interval* $\mathcal{I}_1 \subset \mathbb{R}$ *partially precedes another interval* $\mathcal{I}_2 \subset \mathbb{R}$ *if* $\min(\mathcal{I}_1) \leq \min(\mathcal{I}_2)$*, and fully precedes it if* $\max(\mathcal{I}_1) \leq \min(\mathcal{I}_2)$*.*

Thus, when we project two objects onto a given axis, we can determine whether $\mathcal{O}_i$ is partially or fully left of $\mathcal{O}_j$ with respect to axis ax:

Definition 42 (Precedence Relations)**.** *An object* $\mathcal{O}_i$ *partially precedes (*partPrec*) an object* $\mathcal{O}_j$ *with respect to axis* ax *if*

$$\begin{aligned}\mathcal{O}_i \,\mathrm{partPrec}_{\mathrm{ax}}\, \mathcal{O}_j := &\min\big(\mathrm{proj}_{\mathrm{ax}}(\mathcal{O}_i)\big)\\ &- \min\big(\mathrm{proj}_{\mathrm{ax}}(\mathcal{O}_j)\big) \leq 0,\end{aligned}$$

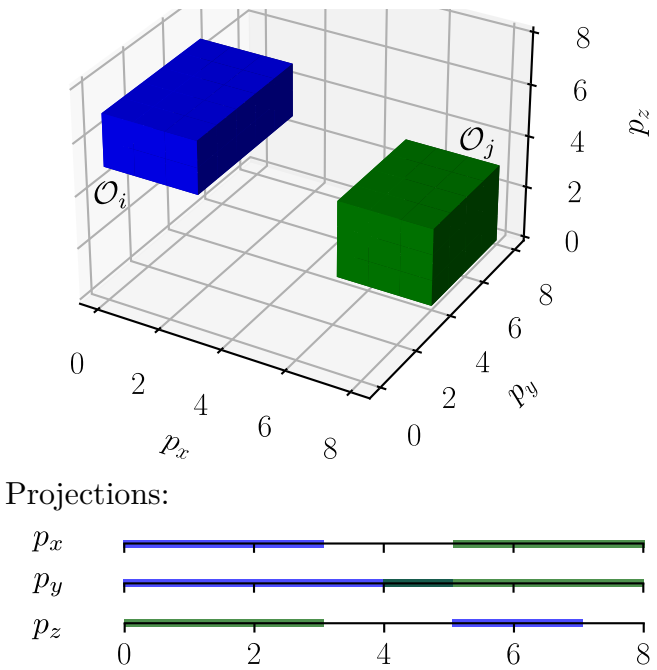


Figure 5.4: We can determine the order relation of two objects $\mathcal{O}_i$ and $\mathcal{O}_j$ by projecting them onto an axis $\text{ax} \in \{p_x, p_y, p_z\}$ and determining the order of the intervals. Here, $\mathcal{O}_i$ is left of $\mathcal{O}_j$ for the axes p_x and p_y and $\mathcal{O}_j$ left of $\mathcal{O}_i$ for axis p_z.

and fully precedes (prec) $\mathcal{O}_j$ *if*

$$\begin{aligned}\mathcal{O}_i \,\text{prec}_{\text{ax}}\, \mathcal{O}_j := &\max\left(\text{proj}_{\text{ax}}(\mathcal{O}_i)\right) \\ &- \min\left(\text{proj}_{\text{ax}}(\mathcal{O}_j)\right) \leq 0.\end{aligned}$$

With the precedence relations, users can derive spatial relations for a given application by choosing different axes, e.g., determining if an object is above another one when projecting it to the height-axis. Figure 5.4 illustrates how we can use the projection and interval order to determine whether an object is left of another object with respect to a given axis.

Finally, we derive additional projection-based spatial relations. Users can define various coordinate frames to express the projection-based spatial relations, given as a parameter. Without loss of generality, we consider a 2D environment with a single reference frame with axes p_x and p_y in the following paragraphs.

Definition 43 (Derived Projection-based Spatial Relations)**.** *The spatial relations* left of *(*leftOf*),* right of *(*rightOf*),* below of *(*below*),* above of *(*above*), and* between

*(*between*) are derived from the projection relations:*

$$\begin{aligned}
\mathcal{O}_i \,\text{leftOf}\, \mathcal{O}_j &:= \mathcal{O}_i \,\text{prec}_{p_x}\, \mathcal{O}_j \\
\mathcal{O}_i \,\text{rightOf}\, \mathcal{O}_j &:= \neg\mathcal{O}_i \,\text{prec}_{p_x}\, \mathcal{O}_j \\
\mathcal{O}_i \,\text{below}\, \mathcal{O}_j &:= \mathcal{O}_i \,\text{prec}_{p_y}\, \mathcal{O}_j \\
\mathcal{O}_i \,\text{above}\, \mathcal{O}_j &:= \neg\mathcal{O}_i \,\text{prec}_{p_y}\, \mathcal{O}_j \\
\mathcal{O}_i \,\text{between}_{\text{ax}}\, \mathcal{O}_j \text{ and } \mathcal{O}_k &:= \mathcal{O}_j \,\text{prec}_{\text{ax}}\, \mathcal{O}_i \wedge \\
&\quad\; \mathcal{O}_i \,\text{prec}_{\text{ax}}\, \mathcal{O}_k
\end{aligned}$$

Note that the partial versions of the relations are obtained by using the partial precedence relation. These versions allow one to check, e.g., whether a part of an object is left of another object.

Angle-Based Relations

In some applications, we want to orient objects in a desired way, e.g., a knife needs to point upwards. This requires us to compare the orientation vector u_i of an object $\mathcal{O}_i$ with the desired orientation vector u_d. In data analysis, the cosine similarity measures the relative angle between two vectors. For computational efficiency and to enforce a positive measure, we use the Euclidean distance approximation of the cosine distance:

Definition 44 (Euclidean Cosine Distance)**.** *The Euclidean approximation* ecd *of the cosine distance between two vectors u_1 and u_2 is given by*

$$\text{ecd}(u_1, u_2) = {}^{(||u_1 - u_2||_2^2)}/_{2},$$

where $||u_1||_2 = ||u_2||_2 = 1$.

This definition allows us to define a relation that describes whether two objects are aligned in the same orientation:

Definition 45 (Orientation Relation)**.** *An object $\mathcal{O}_i$ with angle vector u_i is oriented as object $\mathcal{O}_j$ with angle vector u_j if*

$$\mathcal{O}_i \,\text{oriented}\, \mathcal{O}_j := \text{ecd}(u_i, u_j) \leq \kappa,$$

where κ is a user-defined parameter to allow small numerical deviation from the desired orientation.

Figure 5.5 illustrates the ecd between two angle vectors u_1 and u_2 and how we use the parameter κ to account for numerical deviations.

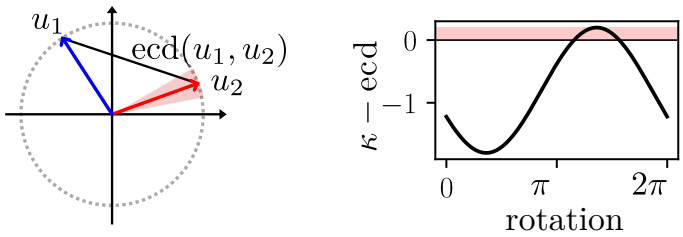


Figure 5.5: The Euclidean cosine distance ecd provides a measure for the angle between two unit vectors, u_1 and u_2. The shaded area corresponds to being $\kappa-$close to the desired angle of u_2. The right figure shows the orientation relation when rotating u_1.

Spatial Compositions and Quantitative Semantics

Users can create complex spatial specifications by composing our spatial relations with Boolean operators:

Definition 46 (Spatial Relation Grammar)**.** *Spatial relation specifications are recursively defined using the following grammar:*

$$\begin{aligned}\mathcal{R} := \top | \mathcal{O}_1 \mathrm{rel}_2 \mathcal{O}_2 | \mathcal{O}_1 \mathrm{rel}_3 \mathcal{O}_2, \mathcal{O}_3 | \neg \mathcal{R} | \\ \mathcal{R}_1 \wedge \mathcal{R}_2 | \mathcal{R}_1 \vee \mathcal{R}_2,\end{aligned}$$

where rel_2 *and* rel_3 *are spatial relations between two or three objects, respectively. The vertical bar* $|$ *represents a choice between expressions and* $\top$ *represents unconditional truth. Through the recursive definition, spatial relations can be nested, negated or combined through logical conjunction and disjunction in arbitrary fashion.*

Similar to the quantitative semantics of STL [111], we define quantitative semantics for our spatial relations by converting their constraint formulation (i.e., a form of $g \leq c$, where g is a function and c a constant) into a satisfaction formulation. We consider the state s_t of all objects at time t to compute the satisfaction.

Definition 47 (Quantitative Semantics for Spatial Relations)**.** *We define the quantitative semantics* $\rho(s_t, \mathcal{R})$ *of a spatial relation specification* $\mathcal{R}$ *as:*

$$\begin{aligned}\rho(s_t, \top) &= \infty \\ \rho(s_t, \neg\mathcal{R}) &= -\rho(s_t, \mathcal{R}) \\ \rho(s_t, \mathcal{R}_1 \wedge \mathcal{R}_2) &= \min(\rho(s_t, \mathcal{R}_1), \rho(s_t, \mathcal{R}_2)) \\ \rho(s_t, \mathcal{R}_1 \vee \mathcal{R}_2) &= \max(\rho(s_t, \mathcal{R}_1), \rho(s_t, \mathcal{R}_2))\end{aligned}$$

We define ρ for single spatial relations:

$$\begin{aligned}
\rho(s_t, \mathcal{O}_i \,\mathrm{closeTo}_\epsilon\, \mathcal{O}_j) &= \epsilon - \mathrm{sd}(\mathcal{O}_i^t, \mathcal{O}_j^t) \\
\rho(s_t, \mathcal{O}_i \,\mathrm{ovlp}\, \mathcal{O}_j) &= -\,\mathrm{sd}(\mathcal{O}_i, \mathcal{O}_j) \\
\rho(s_t, \mathcal{O}_i \,\mathrm{closerTo}\, \mathcal{O}_j \text{ than } \mathcal{O}_k) &= \\
&\quad \mathrm{sd}(\mathcal{O}_j, \mathcal{O}_k) - \mathrm{sd}(\mathcal{O}_i, \mathcal{O}_j) \\
\rho(s_t, \mathcal{O}_i \,\mathrm{enclIn}\, \mathcal{O}_j) &= -\max_{p \in \mathrm{hull}(\mathcal{O}_i)} \mathrm{sd}(p, \mathcal{O}_j) \\
\rho(s_t, \mathcal{O}_i \,\mathrm{partPrec}_{\mathrm{ax}}\, \mathcal{O}_j) &= \min\big(\mathrm{proj}_{\mathrm{ax}}(\mathcal{O}_j)\big) \\
&\quad - \min\big(\mathrm{proj}_{\mathrm{ax}}(\mathcal{O}_i)\big) \\
\rho(s_t, \mathcal{O}_i \,\mathrm{prec}_{\mathrm{ax}}\, \mathcal{O}_j) &= \min\big(\mathrm{proj}_{\mathrm{ax}}(\mathcal{O}_j)\big) \\
&\quad - \max\big(\mathrm{proj}_{\mathrm{ax}}(\mathcal{O}_i)\big)
\end{aligned}$$

The quantitative semantics for the other spatial relations are computed analogously by recursively applying our introduced semantics.

We can use our geometry-based spatial relations also between objects over time, e.g., to describe the motion of an object. As an example, we consider the composition of a new unary spatial relation that describes if an object is moving side-wards over time.

Definition 48 (Time-Relative Relations)**.** *We define an object $\mathcal{O}_i^t$ moving to the right side as*

$$\mathcal{O}_i^t \,\mathrm{movingRight} \Leftrightarrow \mathcal{O}_i^{t-1} \,\mathrm{leftOf}\, \mathcal{O}_i^t.$$

This time-relative property can be defined for all spatial relations and allows users to compose highly complex relations. Since we operate on geometric objects, we can also create phantom objects, such as goal or forbidden regions, to specify desired goal configurations or constraints.

Temporal Behaviour of Spatial Relations

With the temporal operators from logics such as LTL or MITL, we further allow users to specify the interplay of spatial relations in time.

Definition 49 (Syntax of Temporal Relations)**.** *Temporal behaviour of spatial relations is recursively defined using the following syntax:*

$$\phi := \top \,|\, \mathcal{R} \,|\, \neg\phi \,|\, \phi_1 \wedge \phi_2 \,|\, \mathcal{X}\phi \,|\, \phi_1 \mathcal{U} \phi_2 \,|\, \phi_1 \mathcal{U}_{[a,b]} \phi_2,\ 0 \leq a < b$$

Similar to Definition 46, the vertical bar | represents a choice between expressions and $\top$ represents unconditional truth. Through the recursive definition, SpaTiaL formulas can be nested, negated or combined through logical conjunction and disjunction in arbitrary fashion. The *next*-operator $\mathcal{X}\phi$ requires a spatial relation

to hold in the next time step. $\phi_1 \mathcal{U} \phi_2$ requires ϕ_1 to hold *until* ϕ_2 holds. Using the operators $\neg$ and $\wedge$, we obtain the full power of propositional logic. The operators $\vee, \rightarrow, \leftrightarrow$ are defined as usual [96]. Furthermore, $\mathcal{F}\phi = \top \mathcal{U} \phi$ represents ϕ holding true *eventually* in the future and $\mathcal{G}\phi = \neg\mathcal{F}\neg\phi$ *always* holding from now on. Bounded temporal operators additionally use a time interval $[a, b]$ to define a time window in which formulas have to hold. Similar to their unbounded variants, $\mathcal{F}_{[a,b]}\phi = \top \mathcal{U}_{[a,b]}\phi$ and $\mathcal{G}_{[a,b]}\phi = \neg\mathcal{F}_{[a,b]}\neg\phi$.

We interpret a temporal formula ϕ over finite sequences of states. We denote a sequence of states from time t to $t + k$ as $s_{t:t+k} = s_t s_{t+1} \dots s_{t+k}$.

Definition 50 (Quantitative Semantics of Temporal Relations)**.** *We define* $\rho(s_{t:t+k}, \phi)$ *as:*

$$\begin{aligned}
\rho(s_{t:t+k}, \top) &= \infty \\
\rho(s_{t:t+k}, \mathcal{R}) &= \rho(s_t, \mathcal{R}) \\
\rho(s_{t:t+k}, \neg\phi) &= -\rho(s_{t:t+k}, \phi) \\
\rho(s_{t:t+k}, \phi \wedge \psi) &= \min(\rho(s_{t:t+k}, \phi), \rho(s_{t:t+k}, \psi)) \\
\rho(s_{t:t+k}, \mathcal{X}\phi) &= \rho(s_{t+1:t+k}, \phi), k > 0 \\
\rho(s_{t:t+k}, \phi_1 U \phi_2) &= \max_{t' \in [t,t+k]} (\min(\rho(s_{t':t+k}, \phi_1), \\
&\quad \min_{t'' \in [t,t']} (\rho(s_{t'':t'}, \phi_2)))), \\
\rho(s_{t:t+k}, \phi_1 U_{[a,b]} \phi_2) &= \rho(s_{(t+a):\min(t+k,t+b)}, \phi_1 U \phi_2),
\end{aligned}$$

where $0 \leq a \leq b$ *and* $a \leq k$.

From this definition, we can derive additional semantics for the derived operators:

$$\begin{aligned}
\rho(s_{t:t+k}, \phi_1 \vee \phi_2) &= \max(\rho(s_{t:t+k}, \phi_1), \rho(s_{t:t+k}, \phi_2)), \\
\rho(s_{t:t+k}, \mathcal{F}\phi) &= \max_{t' \in [t,t+k]} (\rho(s_{t':t+k}, \phi)), \\
\rho(s_{t:t+k}, \mathcal{G}\phi) &= \min_{t' \in [t,t+k]} (\rho(s_{t':t+k}, \phi)).
\end{aligned}$$

A sequence of states $s_{t:t+k}$ satisfies a formula ϕ, denoted as $s_{t:t+k} \models \phi$, if $\rho(s_{t:t+k}, \phi) \geq 0$. We denote the set of all traces that satisfy ϕ as $L(\phi)$.

Remark 3 (Comparison to other TLs)**.** *SpaTiaL takes concepts and operators from established temporal logics, such as LTL$_f$, MITL and STL. As spatial relations are real-valued functions, our language expresses a fragment of STL with robust semantics [136] over discrete intervals of time. The syntax and semantics of unbounded temporal relations are in line with LTL$_f$ [104] and TLTL [137], which express temporal properties over finite intervals of time. Compared to MITL [106], our bounded temporal operators are defined over discrete time instead of continuous time. Applying timed automata-based planning approaches to SpaTiaL remains an interesting avenue for future research.*

Monitoring and Planning with SpaTiaL

The quantitative semantics of SpaTiaL allow users to monitor the satisfaction of a given specification on a sequence of states. This monitoring approach can be applied to video data with tracked objects, as we demonstrate in Section 15 andSection 15. In these experiments, we use SpaTiaL to monitor both the completion of an object manipulation task with complex spatial constraints, as well as real-time monitoring of social distancing behaviour, where we showcase temporal constraints. Beyond monitoring, SpaTiaL can also be used to derive a plan to satisfy a specification. In this section, we outline how to use SpaTiaL for online monitoring and present a heuristic planning method that is independent of the underlying robot dynamics.

Online Monitoring

SpaTiaL can evaluate the satisfaction of a specification online with respect to the information about objects through its quantitative semantics. These semantics measure the extent to which a scenario satisfies or violates a specification and can be used, e.g., to obtain fine-grained information on the robot's task progress during operation. Let us demonstrate the monitoring on an example:

Example 13 (Monitoring an Object's Motion)**.** *Consider a workspace with a cube $\mathcal{O}_{\text{cube}}$ that needs to be moved to the right into a pre-defined goal region $\mathcal{O}_{\text{goal}}$. The spatial relation specification for this example is $\mathcal{R}_{\text{bp}} = (\mathcal{O}_{\text{cube}}\, \text{enclIn}\, \mathcal{O}_{\text{goal}})$.*

Figure 5.6a illustrates the example and the motion of the object $\mathcal{O}^t_{\text{cube}}$ over time steps t. The computed satisfaction value for the specification ϕ_{bp} is shown in Figure 5.6b (blue graph). While moving towards the goal region, the satisfaction value increases. Values below zero indicate a violation of the specification while values greater or equal zero indicate satisfaction. As soon as $\mathcal{O}_{\text{cube}}$ is in the goal region, the satisfaction value is positive. After leaving the goal region, the specification is violated again. In Figure 5.6b, we also show the satisfaction values when replacing the enclIn relation with other spatial relations.

The computational complexity of the monitoring scales similary as the robust evaluation of STL as described in [136]. To this end, the evaluation is linearly in the number of nodes N_{tree} in the parsed syntax tree (evaluation of SpaTiaL relations) for each time step evaluation. If more objects are contained in the specification, e.g., due to the decomposition of non-convex objects, the syntax tree contains more nodes. The evaluation of a relations by using the GJK algorithm scales in the worst-case linearly in the number of vertices in the polygons.

When evaluating specifications over time, the complexity increases to $O(N_{\text{tree}} \cdot N_{\text{states}} \cdot D^h)$, where N_{states} denotes the number of states in the time history, h the maximum length of a path in the syntax tree, and D a positive constant denoting the number of samples needed to evaluate the bounded time operators. Thus, the evaluation of complex specifications may suffer from some degree of exponential complexity.

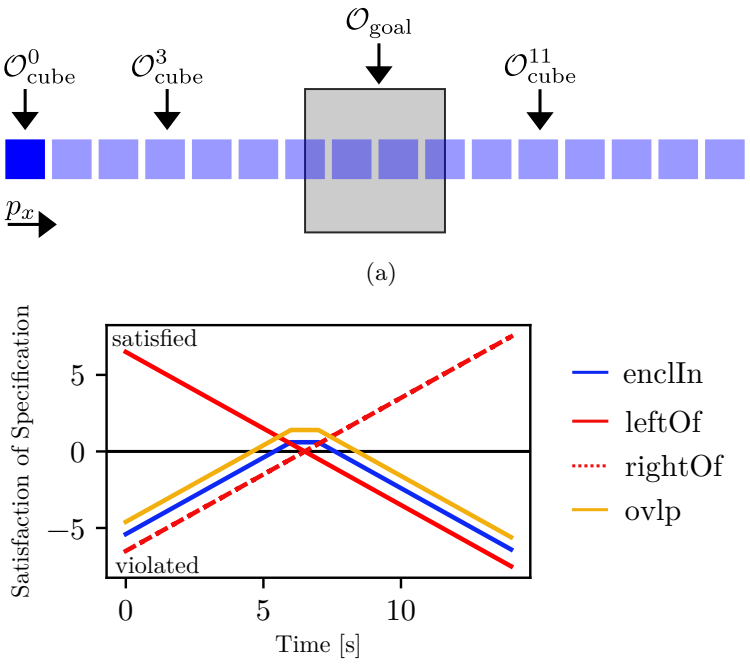


Figure 5.6: The blue block $\mathcal{O}_{\text{cube}}$ needs to be pushed into the goal region $\mathcal{O}_{\text{goal}}$, encoded as $\phi_{\text{bp}} = (\mathcal{O}_{\text{cube}}\,\text{enclIn}\,\mathcal{O}_{\text{goal}})$. (a) shows the motion of $\mathcal{O}^t_{\text{cube}}$ over time. (b) shows the computed satisfaction value for ϕ_{bp} (in blue), when using a leftOf relation, rightOf, and ovlp relation.

To address this complexity, we have implemented two strategies: 1) we use a dynamic programming approach to cache previously evaluated nodes in the syntax tree, reducing the complexity to a constant lookup when looking back in time; and 2) the evaluation of the specification over a sliding time window similar to [136]. This sliding window approach bounds the number of temporal evaluations and can be used in many robotic tasks, since one is not necessarily interested in all information since the start of the robotic system.

To monitor individual parts of complex specifications, we automatically parse every SpaTiaL specification into a tree structure, since SpaTiaL uses a Backus-Naur grammar to compose specifications. This is done through the Python libary *lark*. Figure 5.7 shows an example tree generated from the specification ϕ_{ex}. By color coding and overlaying satisfaction values, users can precisely inspect which individual parts of the specification are satisfied or violated. For instance, the negation branch of the specification tree in Figure 5.7 is violated, since the object $\mathcal{O}_A$ is still positioned above $\mathcal{O}_B$. Moreover, this tree visualization provides an easier

$$\phi_{\text{ex}} = \big(\mathcal{F}(\mathcal{O}_{\text{A}}\ \text{leftOf}\ \mathcal{O}_{\text{B}}) \wedge (\neg(\mathcal{O}_{\text{A}}\ \text{above}\ \mathcal{O}_{\text{B}}))\big)$$

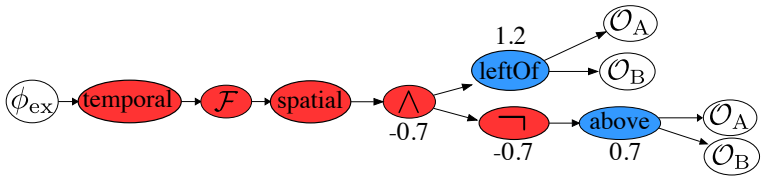


Figure 5.7: Automatically generated tree from the specification ϕ_{ex}. Through color coding, we allow users to see which part's of a specification are violated (red) and satisfied (blue).

way to interpret the specification and allows users further to compose specifications by concatenating trees.

Automaton-Based Online Planning

The object-centric nature of SpaTiaL can be used to facilitate high-level planning agnostic of the underlying robot platform. In this section, we demonstrate how SpaTiaL can be used for planning, execution and monitoring of complex tasks. For this, we present a greedy planning algorithm that monitors the execution of the plan and is able to replan if the execution of a step fails or if a step turns out to be impossible to execute. We apply automaton-based methods from temporal logic-based strategy synthesis to create a greedy planning method that aims to satisfy a syntactic subset of SpaTiaL specifications. To achieve this, we translate SpaTiaL specification into an LTL_f formula by abstracting spatial relations into atomic propositions. From there, we automatically construct a *Deterministic Finite Automaton* (DFA) to represent the temporal structure. Every LTL_f formula can be represented by a DFA [96, 104] by off-the-shelf tools such as MONA [118]. We use DFAs to repeatedly plan the next high-level action, e.g., move object $\mathcal{O}_A$ left of $\mathcal{O}_B$. Through the quantitative semantics of SpaTiaL, we translate the abstract high-level action of transitioning between states in the DFA into a formulation more suited for low-level execution. The high-level actions are planned independently of the underlying robot dynamics. This makes the plan agnostic to the exeuting robot. In our experiments in Section 15, we use gradient maps as an example on how to complete those high-level actions. We analyze the resulting method and find it to be sound, but not optimal or complete due to separation of robot dynamics from the high-level planning.

Definition 51 (DFA)**.** *A DFA is a tuple $A = (Q, \Sigma, \delta, q_0, F)$ where Q is the set of states, Σ the alphabet, $\delta : Q \times \Sigma \to Q$ the transition function, q_0 the initial state and F the set of accepting states.*

Provided a sequence of symbols from the alphabet Σ, a DFA generates a *run*.

Definition 52 (Run)**.** *Given a trace $\tau = \tau_0\tau_1 \dots \tau_n$, where $\tau_i \in \Sigma$, a finite run of a DFA is the sequence of states $q_0 q_1 \dots q_{n+1}$ such that $q_{i+1} = \delta(q_i, \rho_i)$ for all $0 \leq i \leq n$. This run is accepting if $q_{i+1} \in F$.*

When a run is accepting, the sequence of symbols provided to the DFA satisfies the specification [96]. In other words, when we reach an accepting state in the DFA, we have satisfied the specification.

In order to facilitate planning, we omit usage of bounded temporal operators and the *next*-operator $\mathcal{X}$. When omitting bounded temporal operators, every SpaTiaL formula ϕ can be translated into an LTL$_f$ formula ϕ_t by mapping every spatial relation $\mathcal{R}$ to an atomic proposition.

Example 14. *Consider the SpaTiaL formula*

$$\phi = \mathcal{F}(\mathcal{O}_0 \,\mathrm{closeTo}_\epsilon\, \mathcal{O}_1) \wedge \mathcal{G}\neg(\mathcal{O}_0 \,\mathrm{touch}\, \mathcal{O}_2) \tag{5.1}$$

with spatial relations

$$r_1 = (\mathcal{O}_0 \,\mathrm{closeTo}_\epsilon\, \mathcal{O}_1),$$
$$r_2 = (\mathcal{O}_0 \,\mathrm{touch}\, \mathcal{O}_2).$$

The temporal structure of ϕ can be expressed by

$$\phi_t = \mathcal{F}\mathit{goal} \wedge \mathcal{G}\neg\mathit{collide} \tag{5.2}$$

over atomic propositions $AP = \{goal, collide\}$ and a mapping function $m : AP \to \mathcal{R}$, $m(goal) = r_1, m(collide) = r_2$.

Additionally, through omitting the *next*-operator $\mathcal{X}$, we obtain *stutter-insensitive* temporal property.

Lemma 11 (Stutter-Insensitive Properties)**.** *Every LTL$_f$ formula without $\mathcal{X}$ can be translated into a DFA $A = (Q, \Sigma = 2^{AP}, \delta, q_i, F)$ where for every state $q \in Q \setminus q_i$, there exists a* self-loop *transition such that $q = \delta(q, \sigma)$ for some $\sigma \in \Sigma$ [96, 138].*

Remark 4 (Properties without temporal operators)**.** *Formulae where spatial relations occur without temporal operators, e.g. $\phi = \mathcal{O}_0 \,\mathrm{touch}\, \mathcal{O}_2$ correspond to A_ϕ where q_i does not have a self-loop. An initial observation provides the truth value of such properties and enables a transition into a state with a self-loop before we plan our first step.*

We translate the corresponding LTL$_f$ formula ϕ_t into a DFA A_{ϕ_t} that precisely accepts $L(\phi_t)$, e.g., using the tool MONA [118]. The resulting automaton for (5.1) is depicted in Figure 5.8.

Utilizing the structure of A_{ϕ_t} following from Lemma 11, we determine a sequence of actions aiming to satisfy ϕ. With A_{ϕ_t} handling the temporal structure of ϕ, we are able to satisfy ϕ by determining a path to an accepting state in the DFA. This

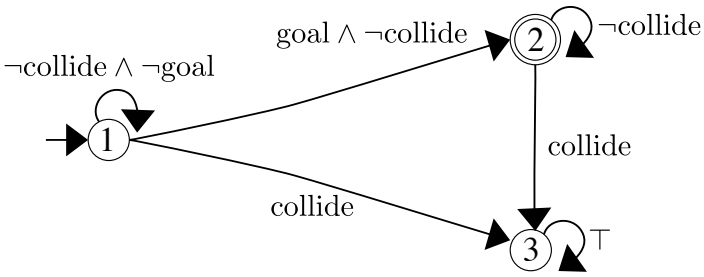


Figure 5.8: DFA for Example 14 where state 1 and state 2 are the initial and accepting states, respectively.

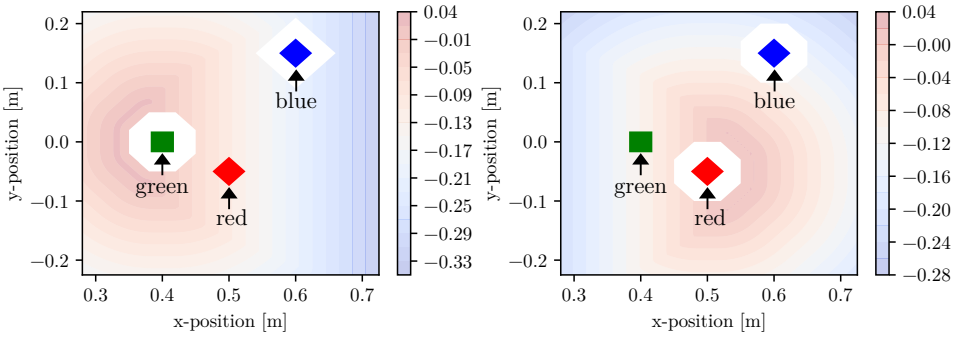


Figure 5.9: Gradient Maps for the spatial relation $\phi_p = (\mathit{green}\,\text{rightOf}\,\mathit{red}) \wedge (\text{d}(\mathit{green}, \mathit{red}) \leq 0.05)$ with spatial constraint $\phi_c = \bigwedge_{i \neq j}(\text{d}(i,j) < 0.01), i, j \in \{\mathit{green}, \mathit{red}, \mathit{blue}\}$ from the perspective of the green cube and the red cube.

path indicates which relations need to be satisfied sequentially. After an initial observation, every state has a self-loop. We use the conditions of the self-loop as constraints for the next execution step. This way, we avoid triggering unwanted transitions. The procedure is outlined in Algorithm 5. First, we construct A_{ϕ_t} and keep track of the current state (up to line 6). Using standard methods, we find a shortest path to an accepting state inside the DFA. The path length is defined through the number of required transitions. From this path, we select a transition (q_c, q_{next}) that we wish to enable in a future step (line 10). If we are in an accepting state, i.e., $q_c \in F$, then we stay in that state, i.e., $q_{next} = q_c$ (line 8). Let

$$\Sigma_t \subseteq \Sigma = \{\sigma \in \Sigma \mid \delta(q_c, \sigma) = q_{next}\} \tag{5.3}$$

be the *progress set*. By satisfying any $\sigma \in \Sigma_p$, we transition into the planned state, bringing us closer to an accepting state. A symbol σ is *satisfied*, if and only if $\rho(m(x)) > 0$ for all $x \in \sigma$. We define

$$\Sigma_c \subseteq \Sigma = \{\sigma \in \Sigma \mid \delta(q_c, \sigma) \neq q_{next} \wedge \delta(q_c, \sigma) \neq q_c\} \tag{5.4}$$

Algorithm 5: Automaton-Based Monitoring and Planning

Data: SpaTiaL formula ϕ

1 construct DFA $A_{\phi_t} = (Q, \Sigma, \delta, q_0, F)$
2 prune infeasible transitions offline (see Remark 5)
3 $q_c = q_0$; `// set current state to initial state`
4 **while** $q_c \notin F$ **do**
5 $\quad \sigma_i = \{r \in AP \mid \rho(m(r), i) \geq 0\}$
6 $\quad q_c = \delta(q_c, \sigma_i)$; `// update current state`
7 $\quad$ find shortest path
$\quad\quad q_1 q_2 \ldots q_n \in Q^*$ s.t. $q_n \in F, q_1 = q_c$ and $\forall i \exists \sigma : (q_i, \sigma, q_{i+1}) \in \delta$
8 $\quad q_{next} = q_2$
9 $\quad$ **if** *No path exists* **then**
10 $\quad\quad$ break
11 $\quad \Sigma_p = \{\sigma \in \Sigma \mid \delta(q_c, \sigma) = q_{next}\}$
12 $\quad \Sigma_c = \{\sigma \in \Sigma \mid \delta(q_c, \sigma) \neq q_{next} \wedge \delta(q_c, \sigma) \neq q_c\}$
13 $\quad$ execute (Σ_p, Σ_c) (see Lemma 12)
14 $\quad$ **if** *unable to execute* (Σ_p, Σ_c) **then**
15 $\quad\quad$ set $\delta(q_c, \sigma) = \varnothing, \forall \sigma \in \Sigma_p$; `// prune edge`

as the *constraint set*. By never satisfying any $\sigma \in \Sigma_c$, we guarantee staying in q_c until we satisfy any $\sigma \in \Sigma_p$. This is always possible since there always exist some σ such that $(q_c, \sigma, q_c) \in \delta$ due to Lemma 11.

Guided by the DFA, we have reduced the problem of satisfying ϕ to the problem of sequentially aiming to satisfy any $\sigma_p \in \Sigma_p$ while avoiding to satisfy any $\sigma_c \in \Sigma_c$. This planning approach is similar to the approach presented in [138] without the domain- and problem-specific constraints and pruning rules.

Lemma 12 (Satisfying Spatial Relations)**.** *Given a progress set Σ_p and a constraint set Σ_c, we can plan to transition to the desired next state of the DFA by satisfying*

$$\max_{\sigma_p \in \Sigma_p} (\rho(\sigma_p)) \geq 0$$

with subject to the constraint

$$\min_{\sigma_c \in \Sigma_c} (\rho(\sigma_c)) < 0$$

where $\rho(\sigma) = \min_{r \in \sigma}(\rho(m(r)))$.

Lemma 12 aims at satisfying spatial relations that trigger a planned transition while avoiding triggering any unwanted transitions. In general, finding a solution to satisfy the equations presented in Lemma 12 depends on the dynamics of the executing robot. However, by abstracting between high-level planning and low-level

control, we allow users to apply their own methods to execute the planned next action.

Search-based planning using gradient maps We demonstrate a solution that is straightforward to implement for the case where satisfying the equations in Lemma 12 is possible by moving a single object only. By choosing an object $\mathcal{O}$ and virtually moving it on a grid of the workspace, we compute a *gradient map* by evaluating spatial subformulae in Σ_p and Σ_c for all positions of $\mathcal{O}$.

Definition 53 (Gradient Maps)**.** *Given Σ_p and Σ_c, an object $\mathcal{O}$, and a point $p \in \mathbb{R}^2$, we write $\widetilde{\mathcal{O}}$ for the the object $\mathcal{O}$ with its center displaced to p. We write $\tilde{s}_t$ for a state s_t with $\mathcal{O}$ replaced by $\widetilde{\mathcal{O}}$.*

$$\mathcal{G}_{\Sigma_p,\Sigma_c,\mathcal{O}}(p) = \begin{cases} -\infty, & \textit{if } \max_{\sigma_c \in \Sigma_c}(\rho(\tilde{s}_t, \sigma_c)) \geq 0 \\ \max_{\sigma_p \in \Sigma_p}(\rho(\tilde{s}_t, \sigma_p)), & \textit{otherwise.} \end{cases}$$

Gradient maps are a uniform sampling approach in the task-space of a selected object to determine the gradient of satisfying a given spatial relation with respect to the object's position. Varying rotations can be integrated by computing gradient maps for different rotations, similar as in [139].

Example 15 (Gradient Maps)**.** *Consider a rectangular workspace with three colored blocks. Each block is described by its color, its position and its orientation. Given a progress set $\Sigma_p \subseteq \Sigma$ induced by $\phi_p = (\textit{green}\,\mathrm{rightOf}\,\textit{red}) \wedge (\mathrm{d}(\textit{green}, \textit{red}) \leq 0.05)$ and a constraint set characterized by $\phi_c = \bigwedge_{i \neq j}(\mathrm{d}(i,j) < 0.01), i, j \in \{\textit{green}, \textit{red}, \textit{blue}\}$, we depict gradient maps for the green block and for the red block in Figure 5.9. The white cut-out regions around the blocks are the result of the constraint set and represent the area where ϕ_c would be satisfied if we moved the currently considered object into that region. Since ϕ_p can be satisfied both by moving the green block right of the red block or by moving the red block left of the green block, both gradient maps have positive values. The gradient map for the blue block (not depicted) does not have any positive values.*

If for a given Σ_p, Σ_c no gradient map has any positive values for any considered object, the corresponding transition is deemed infeasible and pruned. The steps of Algorithm 5 and the construction of gradient maps can be repeated until a goal state is reached. Depending on the application, the gradient maps can be used to find either a continuous path or a new position for a single object in order to transition to the desired next state of the DFA. Two examples using this procedure are demonstrated in Section 15.

Remark 5 (Pruning the DFA)**.** *Some symbols σ and therefore transitions might be infeasible, especially when physical constraints are not represented in the specification. An example for such a transition might be requiring an object to be both close and far from another object at the same time. Additional offline pruning of*

infeasible transitions might be done through specific domain knowledge, depending on the application. In the following experiments, we employ only online pruning of edges that turn out to be infeasible through Lemma 12 (see Algorithm 5 line 15).

Analysis of the Planning Approach

In the following paragraphs, we analyze the termination, optimality, completeness, and soundness properties of our proposed planning approach.

Termination Assuming the executing robot can either complete or determine infeasibility of a planned step (Σ_p, Σ_c) in finite time, the planning algorithm Algorithm 5 is guaranteed to terminate in finite time. This follows directly from the finite amount of transitions in the DFA and the iterative pruning of infeasible transitions.

Optimality The chosen path is greedily chosen with respect to the least amount of transitions in the DFA. Without additional reasoning on the dynamics of the executing robot, it is not possible to infer further information about the quality of the solution. Within the scope of our experiments, reaching an accepting state with the least amount of transitions led to solutions where the robot performs the least amount of primitive actions possible, i.e. moving the fewest elements in the scenario.

Completeness If any solution exists to satisfy a given specification ϕ, it can be expressed as a sequence of Σ_p and Σ_c (see Algorithm 5 and Lemma 12). This follows directly from the observation that the DFA is without loss of generality complete, meaning it covers all combinatorial evaluations of spatial relations in the specification ϕ. Since the planning algorithm does not take the low-level dynamics of the executing robot into consideration, the algorithm cannot be complete. In situations with complex restrictions on the robot's dynamics, planning approaches that take these restrictions into account are preferable to the presented robot-agnostic approach.

Soundness If a sequence of executable actions that lead to an accepting state of A_{ϕ_t} can be found, the generated trace of the execution is satisfying ϕ_t and transitively satisfying the specification ϕ. This follows directly from known results on the translation of LTL_f formulae to DFA [96, 104].

Experiments

To demonstrate the capabilities of SpaTiaL, we designed three different experiments with recorded data, simulations, and a robot. Each experiment focuses on different aspects of SpaTiaL:

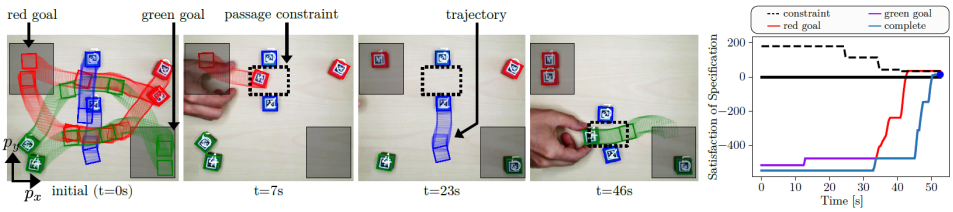


Figure 5.10: Results of our experiment in which we monitor a pushing task. (a) shows the camera frames and trajectories (full trajectory for $t = 0s$ and partial ones with length $3s$ otherwise) of the cubes for selected time steps. The trajectories show the future positions of the cubes. (b) illustrates the satisfaction value of the specification and its sub-parts over the time of the recording.

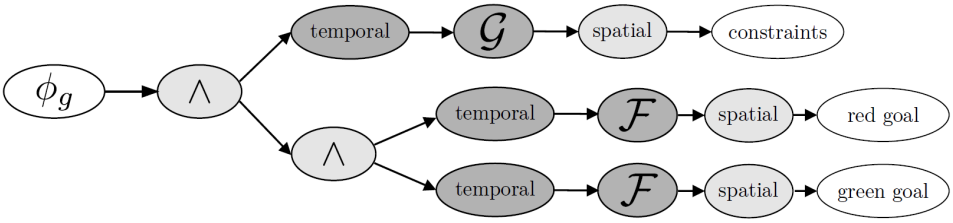


Figure 5.11: Tree representation of the parsed specification ϕ_{blocks}. The recursive computation of the quantitative semantics allows us to also monitor individual subtrees.

1. We monitor object pushing and pick-and-place tasks. This experiment highlights the creation of specifications, the use of quantitative semantics, and accounting for object detection uncertainties;
2. We monitor social distancing of agents in the Stanford drone dataset. This experiment highlights the usage of temporal properties with time bounds;
3. We use SpaTiaL for planning in robotic tasks. This experiment illustrates the high-level temporal and low-level control abstractions.

Additional experiments are in our code repository and videos of our experiments in the article's media attachment.

Implementation

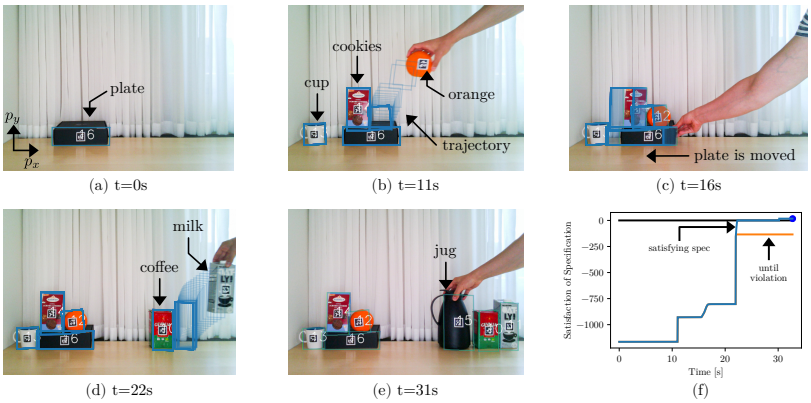
We implemented a SpaTiaL parser and interpreter in Python 3, using Lark [140] to parse the logic, Shapely [141] for geometric computations, and MONA [118] with LTLf2DFA [142] to construct DFAs for planning. Our software is published under the MIT license[1]. We provide an API to parse SpaTiaL specifications, define

[1] `https://github.com/KTH-RPL-Planiacs/SpaTiaL`

objects through sets of convex polygons, and to automatically interpret the formula to obtain satisfaction values, as well as DFA planning functionalities. We have also released the full code for all experiments, including gradient map computation and our simulation setup in PyBullet [143].

Monitoring Robotics Tasks

SpaTiaL allows user to monitor the satisfaction of specifications in real time for various applications. In our first two experiments, we monitor a block pushing and a pick-and-place task. We recorded both tasks using a camera with 30fps and localized objects in the scene using AprilTags [144]. The computation times per frame of both experiments can be found in Table 5.1 when analyzing the video with 30fps and 10fps (i.e., skipping every 2 frames). We used a machine with macOS, an Intel i5 2GHz Quadcore CPU, 32GB of DDR4 memory, and Python 3.8.



(a) t=0s (b) t=11s (c) t=16s (d) t=22s (e) t=31s (f)

Figure 5.12: Results of our experiment in which we monitor a pick-and-place task. (a) to (e) show the camera frames and trajectories (partial ones with length of $3s$) of the objects in the scene for selected time steps. The trajectories show the future positions of the objects. (f) illustrates the satisfaction value of the original specification and the modified one with an *until* violation over the time of the recording.

Block pushing task In this task, a human is pushing small blocks on an even surface towards a goal configuration while not violating constraints (see setup in Figure 5.10). We placed 2 red, 2 green, and 2 blue blocks in the scene and specify additional phantom objects that serve as goal areas. The specification requires the human to push the red and green blocks to their corresponding goal regions *redGoal* and *greenGoal*, respectively. At the same time, we impose a task constraint: when moving red/green blocks to the other side, the human needs to push the blocks

through the passage that is formed by the two blue blocks. This specification ϕ_{blocks} is encoded using the eventually and always operators as shown in Table 5.2.

Figure 5.10 illustrates selected camera frames. At $t = 7s$, the human is pushing a red block through the passage to not violate the mission constraint. However, even though the human is moving one red block towards the goal, the satisfaction value is not yet changing. This observation is due to the fact that the computed satisfaction value corresponds to the worst violation of the subparts of ϕ_{blocks}, i.e., every *and* operator in the quantitative semantics is essentially a minimum operation. In this scenario, the human did not move the red block that is further away from its goal region. Our monitoring algorithm monitors each subpart of the syntax tree parsed from the specification. Section 5.1 outlines how the evaluated syntax tree allows one to obtain this fine-grained information. Figure 5.11 shows the parsed tree which represents the three individual components that specify the red goal, green goal, and the constraints. For instance, when the human moves the green block at around $12s$, the subpart of the specification that considers the green blocks is changing (see purple line in Figure 5.10b).

All spatial relations are evaluated in continuous space while considering the currently available environment information. At $t = 23s$, the human is moving the blue blocks of the passage. SpaTiaL adapts to this change so that the remaining red and green block also need to be moved through the new passage. From $t = 33s$ to $t = 43s$, the last red block is successfully pushed into its goal region, as indicated by the satisfaction graph of the red goal in Figure 5.10b. Finally, the specification ϕ_{blocks} is satisfied at $t = 50.1s$ with a value of 15.35.

Object pick and place task In our second monitoring experiment, we have a closer look at the temporal operator *until* U and evaluations of spatial relations over time. Therefore, we consider a pick-and-place task in which kitchen items need to be placed according to a given specification (similar to the task shown in Figure 5.1). In our example, the plate needs to be moved close to the coffee mug; however, the plate is only allowed to be moved when the objects *cookies* and *orange* have been placed on the plate. This requirement is encoded in the specification ϕ_{pnp} using the until operator in Table 5.2. Moreover, ϕ_{pnp} specifies how the other objects *jug*, *coffee*, *mug*, and *milk* need to be placed. SpaTiaL allows users to easily define

Table 5.1: Mean and standard deviation of the computation time per frame in our monitoring experiments over five runs.

Experiment	Framerate	Mean	StdDev
Block pushing	30fps	32.12ms	0.2ms
	10fps	12.7ms	0.44ms
Pick-and-place	30fps	71.15ms	0.57ms
	10fps	16ms	0.86ms

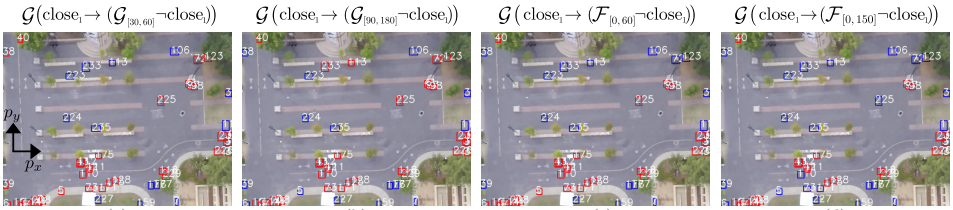


Figure 5.13: Excerpts of our experiment where we monitor which agents in the environment respect social distancing to other objects. The images show the frame 9000 (5min) of the dataset for our four specifications ϕ_1 to ϕ_4 and the too close definition $close_1$. Blue and red colored bounding boxes denote satisfying or violating the specification, respectively. The visualized IDs of the objects correspond to the IDs in the dataset.

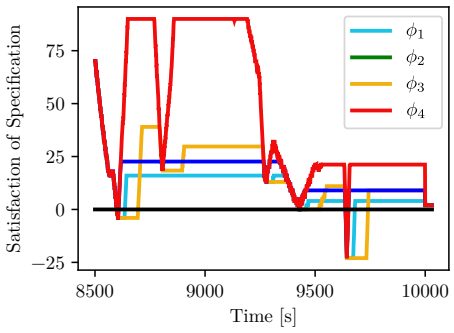


Figure 5.14: Satisfaction values of the object with ID 224 with respect to the four specifications ϕ_1 to ϕ_4 and the closeness definition close_1, where two objects are close when their bounding boxes overlap.

new relations. In this experiment, we defined a new spatial relation *moved* that indicates if an object has been moved over time using time-relative spatial relations:

$$\mathcal{O} \,\text{moved}\, \mathcal{O}[-1] := \mathcal{O} \,\text{enclIn}\, \mathcal{O}[-1] \oplus \mathcal{C}_\epsilon, \tag{5.5}$$

where $\mathcal{C}_\epsilon$ is a circle with radius $\epsilon = 25$ pixel to enlarge an object to account for object detection uncertainties of the camera detection. The relation in (5.5) checks whether the object has moved with respect to its occupancy in the previous time step.

Figure 5.12a to Figure 5.12e illustrate selected time steps of our pick-and-place experiment with partial trajectories (length: $3s$) of the objects. The computed satisfaction value is shown in Figure 5.12f over the course of the experiment with a duration of $35s$. As soon as the required objects *cookies* and *orange* have been correctly placed on the plate, the human is allowed to move the plate (see images

for $t = 11s$ and $t = 16s$). In contrast, we also tested a modified version of the specification in which we require that the human places the *orange* left of the *cookies* on the plate. This modified specification is violated since the human is moving the plate without reaching the required spatial relation (see orange line in Figure 5.12f).

Monitoring Agents in the Environment

We can also use SpaTiaL to monitor agents in an environment. Bounded temporal operators allow users to specify that certain spatial relations need to hold or not hold within a given time interval. We choose a surveillance application in which a drone is recording an environment to demonstrate the bounded time operators and the nesting of temporal operators. In our case, we focus on detecting violations of social distancing using the Stanford Drone dataset [145]. We select the bookstore data where a drone is hovering and recording the bounding boxes of pedestrians, cyclists, and other objects.

We consider four different social distancing specifications which use different temporal operators and time bounds. They are based on checking whether an object is coming too close to another object in the current scene. To identify when objects are too close, we consider two variations, close_1 and close_2:

$$\text{close}_1 := (\textit{ego}\,\text{ovlp}\,\textit{others}), \tag{5.6}$$

$$\text{close}_2 := (\textit{ego}\,\text{closeTo}_{\epsilon_\text{d}}\,\textit{others}), \tag{5.7}$$

where *ego* denotes the currently selected object for investigation, *others* corresponds to all other objects in the scene, and $\epsilon_\text{d} = 15$ is the preferred minimum distance between bounding boxes of the objects (chosen empirically using distance measurements in the pixel space of the image). Our considered social distancing specifications are:

(ϕ_1) $\big(\mathcal{G}\ \text{close} \rightarrow (\mathcal{G}_{[30,60]}\neg\text{close})\big)$,

(ϕ_2) $\big(\mathcal{G}\ \text{close} \rightarrow (\mathcal{G}_{[90,180]}\neg\text{close})\big)$,

(ϕ_3) $\big(\mathcal{G}\ \text{close} \rightarrow (\mathcal{F}_{[0,60]}\neg\text{close})\big)$,

(ϕ_4) $\big(\mathcal{G}\ \text{close} \rightarrow (\mathcal{F}_{[0,150]}\neg\text{close})\big)$,

where $\text{close} \in \{\text{close}_1, \text{close}_2\}$. Specifications ϕ_1 and ϕ_2 encode that whenever an object is too close to another, it needs to be always distant to other objects during 1 to 2 seconds or 3 to 6 seconds after contact (since the video has been recorded with 30fps). Specifications ϕ_3 and ϕ_4 specify that whenever an object is too close to another, it needs to be eventually distant to other objects within 2 seconds or 5 seconds after contact.

Figure 5.13 illustrates the evaluation of the four specifications on frame number 9000. We colored the bounding boxes to encode whether the object is satisfying

(blue color) or violating (red color) the specification. The temporal operator *always* or *eventually* allow us to encoded different levels of strictness of social distancing. This observation can be seen from the fact that the specifications with the *eventually* operator have a higher percentage of satisfying objects in the scene. Table 5.3 provides a more detailed analysis of the number of satisfying or violating objects. Finally, Figure 5.14 shows the satisfaction value for the object with ID 224 over time.

Task and Motion Planning

In the following three experiments, we demonstrate the online monitoring and planning technique outlined in Section 5.1. The first experiment features a simulated robot arm pushing blocks and illustrates the individual steps of DFA planning and the gradient maps to find continuous paths to push objects along. The second experiment features a pick-and-place specification, requiring a simulated robot to arrange a table. The third experiment demonstrates a table setting task on ABB YuMi robots.

Pushing Task and Motion Planning We consider a robot arm mounted on a planar surface as depicted in Figure 5.15. The robots is required to push three colored blocks within a rectangular workspace to fulfill the SpaTiaL specification:

$$\begin{aligned}\phi =& \mathcal{F}(g \,\text{rightOf}\, r \wedge g \,\text{rightOf}\, b)\\ &\wedge \neg(g \,\text{rightOf}\, r \wedge g \,\text{rightOf}\, b)\mathcal{U}(r \,\text{above}\, b)\\ &\wedge \mathcal{G} \bigwedge_{i \neq j} \mathrm{d}(i,j) \geq 0.03, i,j \in \{r,g,b\},\end{aligned}$$

where r, g and b denote the red, green, and blue blocks, respectively. Intuitively speaking, this specification requires the green block to eventually be right of both the red and the blue block at the same time. We disallow the green block to be right of the other blocks until the red block is above the blue block (which is not true in the initial configuration). Additionally, no block should ever come too close to another block. From this specification, we automatically generate a DFA representing the temporal structure as described in Section 5.1. The resulting automaton is depicted in Figure 5.16, with the following mapping from atomic propositions to spatial subformulae:

$$\begin{aligned}a &\coloneqq g \,\text{rightOf}\, r \wedge g \,\text{rightOf}\, b\\ b &\coloneqq r \,\text{above}\, b\\ c &\coloneqq \bigwedge_{i \neq j} \mathrm{d}(i,j) \geq 0.03, i,j \in \{r,g,b\}\end{aligned}$$

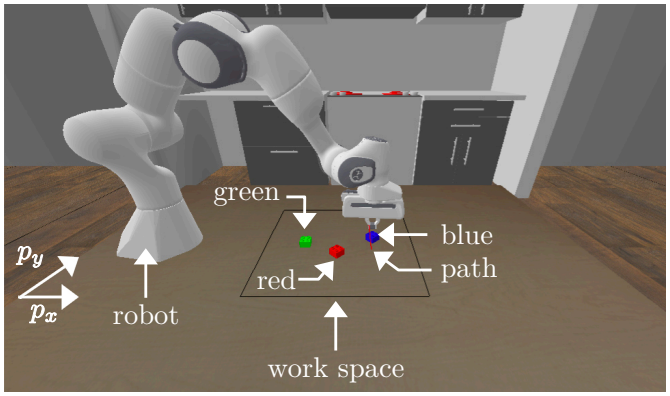


Figure 5.15: Simulator environment for the pushing TAMP experiment.

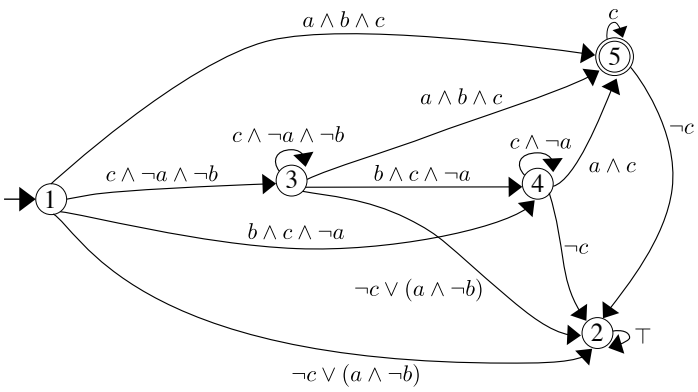


Figure 5.16: The generated DFA for the pushing task. Except for the initial state 1, each state has a self-loop. The initial state not having a self-loop is an implementation detail that is circumvented by providing an initial observation.

Given the initial configuration of objects depicted in Figure 5.15 and Figure 5.17, the planning algorithm starts in DFA state 3. The shortest path to a satisfying configuration is by taking the transition $3 \rightarrow 5$ to the accepting state, requiring all the spatial subformulae to hold true at the same time. Since it is not possible to satisfy all subformulae directly by moving only a single object, the transition is assessed as infeasible and pruned. The next (and only) path to an accepting state is $3 \rightarrow 4 \rightarrow 5$.

Figure 5.17(a) shows the gradient map for the blue block generated from the goal and constraint set for transition $3 \rightarrow 4$. The dashed line indicates the border to a satisfying value. The shortest pushing path that avoids all constrained areas and triggers the desired transition is computed through Jump-Point-Search [146].

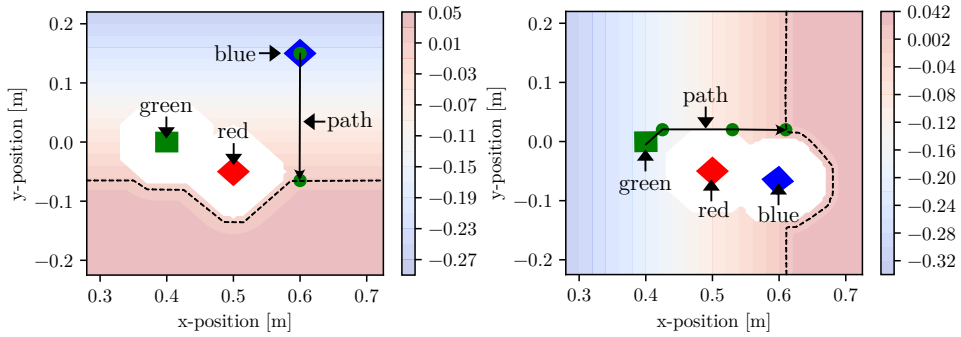


Figure 5.17: In the pushing task experiment, the robot has to satisfy a given specification by pushing the blocks around the workspace.

The path avoids all areas that violate the constraints in Σ_c, depicted in white. Moving the blue block below the red block satisfies the Until-requirement of the specification. The robot executes the plan and pushes the blue block along the computed path. Our implemented pushing controller uses the feedback of our monitoring algorithm in continuous space to adjust control inputs and account for low level control effects where blocks are not pushed properly into satisfying regions. Note that we do not need to specify the exact order of which block needs to be pushed; the planning algorithm has the freedom to decide suitable actions on the fly. For example, the planning algorithm may decide to push the red block above the blue block instead, as this action also leads to specification satisfaction. Afterwards, the DFA state is updated through observation and the current state evolves to 4. From here, the planner can reach the satisfying state 5 by pushing the green block right of both other blocks, depicted in Figure 5.17(b). After execution, the specification is satisfied and the algorithm terminates.

Pick-and-Place Task and Motion Planning The second experiment considers a similar setup, where a robot arm is mounted on a planar surface. The robot is tasked to arrange the different objects on the table. The robot can pick and place objects in a radial area around the base. The objects are identified by the labels: *bottle*, *banana*, *mug*, *gelatin*, *sugarbox*, *can*, *crackerbox*, *kanelbulle* and *plate*. The SpaTiaL specification $\phi_{\text{plan,pnp}}$ is given in Table 5.2, and requires the robot, e.g., to place the Swedish cinnaman bun *kanelbulle* within the *plate*. The resulting DFA has 33 states and 275 transitions. The shortest paths to an accepting state are unsatisfiable through manipulation of a single object and are pruned online. When the computed gradient maps contain positive values, the robot chooses to place the considered object at the position which maximises the value of the gradient map. After executing the action, the DFA is updated with the latest observations. Since we monitor the evolution of the workspace during execution, our planning method is able to detect failing grasps or misplaced objects. Execution failures in

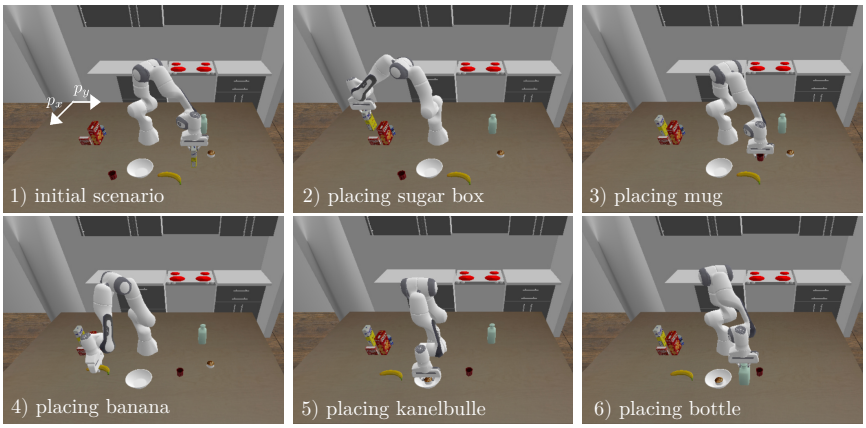


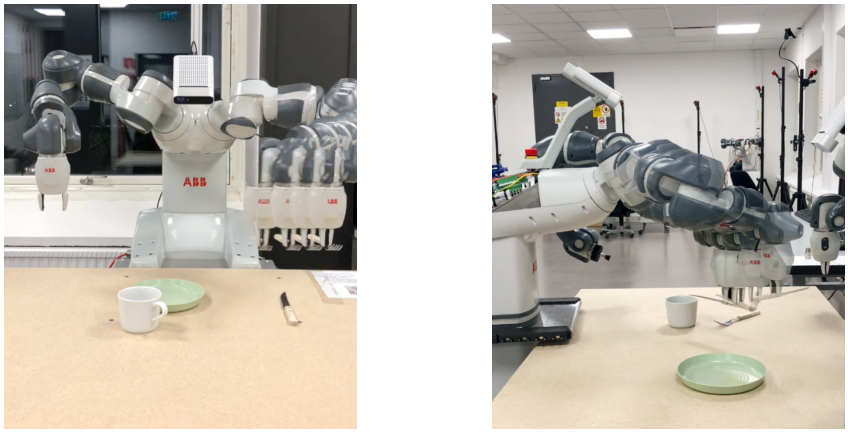
Figure 5.18: Several snapshots of the simulator environment during execution of the pick-and-place TAMP experiment. For visibility, the camera is offset by a 90 deg angle in comparison to the workspace orientation.

the simulation range from tipping over neighbouring objects or losing grasp of the current object. After failed executions, a new path to an accepting state is found and execution continues with an alternative next step or another try of the same execution step. An exemplary execution progression is depicted in Figure 5.18. Since the specification does not impose any constraint on the order in which the desired spatial relationships are fulfilled, each execution can slightly differ.

Our planning experiments illustrate how to facilitate TAMP over SpaTiaL specifications through repeated observation and planning of the next step through the DFA and gradient maps.

Experiments with an ABB YuMi Robot After our successful simulation experiments, we implemented our approach on an ABB YuMi Robot to validate planning with real sensor data and low-level controllers. The robot's table setting task specification ϕ_{ABB} can be found in Table 5.2. The objects are detected using camera images.

Grasping objects is a difficult problem and depends on the robots kinematics, end-effector dynamics and shape of the object. We deliberately designed SpaTiaL to not consider those dynamical effects and focus on the object placement itself. Yet, we can consider the grasping of objects in our ABB YuMi experiments: we use a Behavior tree approach to combine symbolic planning and geometric reasoning. As in the other experiments, our planning automaton first determines which spatial relation should be satisfied next. This triggers the behavior tree controller of the robot. The gradient maps provide the behavior tree the desired object pose. The picking pose is determined by another behavior tree controller when the tree determines the behavior to pick or release an object. In this way, we can be



(a) (b)

Figure 5.19: Experiments with an ABB YuMI robot. (a) the robot places a fork left of the plate. (b) the orientates and laces a knife.

controller-agnostic and allow users to use their designated controllers to determine the picking pose while SpaTiaL solely provides feedback in continuous space about how objects should be arranged.

Fig. 5.19 shows two snapshots from our two robot experiments. In the first experiment (see Fig. 5.19a, the robot executes the desired table setting task by rearranging the objects according to ϕ_{ABB}, e.g., placing the knife right of the plate with orientation pointing away from the robot. We use a behavior tree-based controller to execute the high-level actions that the automaton approach plans. In the second experiment (seeFigure 5.19b, we use a mobile YuMi robot for the same specification ϕ_{ABB}. However, we place the knife on another table in this scenario. Our satisfaction values guide the behavior tree-based controller of our mobile robot to first pick up the knife and then to transport it to goal table. The online monitoring in our example planning approach automatically reflects the successful placement of the knife to continue rearranging the remaining objects.

5.2 Spatial Requests in Object-Centric Human-Robot Collaboration

The planning approach presented in Section 5.1 allows us to handle the temporal structure of the specification formula through an automaton and reduces satisfying the specification into sequentially satisfying different spatial relations. However, the capabilities of the exectuing robot might be limited or satisfying a spatial relation might require more than one object to be moved. In those cases, our presented planning approach cannot find a suitable action to satify the task. Instead of ter-

Specification	Definition
ϕ_{blocks}	$(\mathcal{F}(\textit{red}\,\text{enclIn}\,\textit{redGoal})) \wedge (\mathcal{F}(\textit{green}\,\text{enclIn}\,\textit{greenGoal}))$ $\wedge(\mathcal{G}((\textit{red1}\,\text{closeTo}\,\textit{blue1}) \rightarrow (\textit{red1}\,\text{below}\,\textit{blue1})))$ $\wedge(\mathcal{G}(\textit{red2}\,\text{closeTo}\,\textit{blue2}) \rightarrow (\textit{red2}\,\text{above}\,\textit{blue2})))$ $\wedge(\mathcal{G}((\textit{green1}\,\text{closeTo}\,\textit{blue1}) \rightarrow (\textit{green1}\,\text{below}\,\textit{blue1})))$ $\wedge(\mathcal{G}((\textit{green2}\,\text{closeTo}\,\textit{blue2}) \rightarrow (\textit{green1}\,\text{above}\,\textit{blue2})))$
ϕ_{pnp}	$(\mathcal{F}((\textit{mug}\,\text{leftOf}\,\textit{plate}) \wedge (\textit{mug}\,\text{ovlp}\,\textit{plate})))$ $\wedge((\neg(\textit{plate}\,\text{moved}\,\textit{plate}[-1]))U((\textit{cookies}\,\text{above}\,\textit{plate})$ $\wedge(\textit{cookies}\,\text{ovlp}\,\textit{plate}) \wedge (\textit{orange}\,\text{above}\,\textit{plate}) \wedge (\textit{orange}\,\text{ovlp}\,\textit{plate}))$ $\wedge(\mathcal{F}(\textit{coffee}\,\text{rightOf}\,\textit{plate}) \wedge (\textit{milk}\,\text{closeTo}\,\textit{coffee}) \wedge (\textit{milk}\,\text{rightOf}\,\textit{coffee}))$ $\wedge(\mathcal{F}(\textit{jug}\,\text{leftOf}\,\textit{coffee}) \wedge (\textit{jug}\,\text{closeTo}\,\textit{coffee}))$
$\phi_{\text{plan,pnp}}$	$\mathcal{F}(\textit{kanelbulle}\,\text{enclIn}\,\textit{plate})$ $\wedge\mathcal{F}(0.1 \leq \textit{banana}\,\text{d}\,\textit{plate} \leq 0.3 \wedge \textit{banana}\,\text{leftOf}\,\textit{plate} \wedge \textit{banana}\,\text{below}\,\textit{plate})$ $\wedge\mathcal{F}(0.1 \leq \textit{mug}\,\text{d}\,\textit{plate} \leq 0.3 \wedge \textit{mug}\,\text{leftOf}\,\textit{plate} \wedge \textit{mug}\,\text{above}\,\textit{plate})$ $\wedge\mathcal{F}(0.1 \leq \textit{bottle}\,\text{d}\,\textit{plate} \leq 0.3 \wedge \textit{bottle}\,\text{leftOf}\,\textit{plate} \wedge \textit{bottle}\,\text{above}\,\textit{plate})$ $\wedge\mathcal{F}(\textit{sugarbox}\,\text{d}\,\textit{plate} \geq 0.4 \wedge \textit{sugarbox}\,\text{d}\,\textit{crackerbox} \leq 0.2)$
ϕ_{ABB}	$(\mathcal{G}((\textit{bowl}\,\text{d}\,\textit{fork} \geq 0.001) \wedge (\textit{bowl}\,\text{d}\,\textit{knife} \geq 0.001) \wedge (\textit{bowl}\,\text{d}\,\textit{mug} \geq 0.001)$ $\wedge(\textit{mug}\,\text{d}\,\textit{knife} \geq 0.001) \wedge (\textit{mug}\,\text{d}\,\textit{fork} \geq 0.001) \wedge (\textit{knife}\,\text{d}\,\textit{fork} \geq 0.001)))$ $\wedge(\mathcal{F}(((\textit{bowl}\,\text{d}\,\textit{center} \leq 0.01) \wedge (\mathcal{F}((\textit{fork}\,\text{leftOf}\,\textit{center}) \wedge (\textit{fork}\,\text{d}\,\textit{center} \leq 0.10)))$ $\wedge(\mathcal{F}((\textit{mug}\,\text{above}\,\textit{center}) \wedge (\textit{mug}\,\text{rightOf}\,\textit{center}) \wedge (\textit{mug}\,\text{d}\,\textit{center} \leq 0.20)))$ $\wedge(\mathcal{F}((\textit{knife}\,\text{rightOf}\,\textit{center}) \wedge (\textit{knife}\,\text{d}\,\textit{bowl} \leq 0.10))))))$

Table 5.2: Considered specifications in the experiments.

Predicate	Specification	Satisfying Obj.	Violating Obj.
ovlp	$\mathcal{G}\big(\text{close}_1 \rightarrow (\mathcal{G}_{[30,60]}\neg\text{close}_1)\big)$	35 (53%)	31 (47%)
	$\mathcal{G}\big(\text{close}_1 \rightarrow (\mathcal{G}_{[90,180]}\neg\text{close}_1)\big)$	41 (62%)	25 (38%)
	$\mathcal{G}\big(\text{close}_1 \rightarrow (\mathcal{F}_{[0,60]}\neg\text{close}_1)\big)$	43 (65%)	23 (35%)
	$\mathcal{G}\big(\text{close}_1 \rightarrow (\mathcal{F}_{[0,150]}\neg\text{close}_1)\big)$	49 (74%)	18 (26%)
closeTo	$\mathcal{G}\big(\text{close}_2 \rightarrow (\mathcal{G}_{[30,60]}\neg\text{close}_2)\big)$	25 (38%)	41 (62%)
	$\mathcal{G}\big(\text{close}_2 \rightarrow (\mathcal{G}_{[90,180]}\neg\text{close}_2)\big)$	30 (45%)	36 (55%)
	$\mathcal{G}\big(\text{close}_2 \rightarrow (\mathcal{F}_{[0,60]}\neg\text{close}_2)\big)$	31 (47%)	35 (53%)
	$\mathcal{G}\big(\text{close}_2 \rightarrow (\mathcal{F}_{[0,150]}\neg\text{close}_2)\big)$	40 (61%)	26 (39%)

Table 5.3: Evaluation results of the four social distancing specifications ϕ_1 to ϕ_4 for the two too close definitions.

minating and reporting a failure, the robot could ask for help, e.g. from another, more capable robot or a human operator, so they can complete an otherwise impossible step. A planning method that is able to leverage such requests should however try to be as autonomous as possible, i.e. complete as much of the task on its own. Otherwise, it is trivial to solve every given task by requesting full completion through the helping operator. Also, the human assisstant needs to be able to understand the request. In this section, we extend the planning method presented in Section 5.1 with the capability to send requests as discussed above.

Algorithm 6: Online Planning of SpaTiaL-formula

Data: SpaTiaL formula ϕ

1 construct DFA $A_{\phi_t} = (Q, \Sigma, \delta, q_0, F)$
2 prune infeasible transitions offline;
3 $E_{inf} = \varnothing$; // infeasible edges
4 $q_c = q_0$; // set current state to initial state
5 **while** $q_c \notin F$ **do**
6 $\quad \sigma_i = \{r \in AP \mid \rho(m(r), i) \geq 0\}$; // observation
7 $\quad q_c = \delta(q_c, \sigma_i)$; // update current state
8 $\quad$ find shortest path $q_1 q_2 \ldots q_n \in Q$ s.t. $q_n \in F, q_1 = q_c$ and $\forall i \exists \sigma : (q_i, \sigma, q_{i+1}) \in \delta$ and $(q_i, \sigma) \notin E_{inf}$
9 $\quad$ **if** *path exists* **then**
10 $\quad\quad q_{next} = q_2$;
11 $\quad\quad \Sigma_p = \{\sigma \in \Sigma \mid \delta(q_c, \sigma) = q_{next}\}$
12 $\quad\quad \Sigma_c = \{\sigma \in \Sigma \mid \delta(q_c, \sigma) \neq q_{next} \wedge \delta(q_c, \sigma) \neq q_c\}$
13 $\quad\quad$ execute to achieve (Σ_p, Σ_c)
14 $\quad\quad$ **if** (Σ_p, Σ_c) *not possible* **then**
15 $\quad\quad\quad E_{inf} = E_{inf} \cup \{(q_c, \sigma) \mid \sigma \in \Sigma_p\}$;
16 $\quad$ **else**
17 $\quad\quad$ find smallest request (see Definition 54);
18 $\quad\quad$ **if** *smallest request exists* **then**
19 $\quad\quad\quad$ request help;
20 $\quad\quad\quad$ wait until request is fulfilled;
21 $\quad\quad$ **else**
22 $\quad\quad\quad$ **return** ; // spec unsatisfiable

Online Planning with Requests

A full overview of the algorithm is given in Algorithm 6. First, we construct A_{ϕ_t}. The set of marked edges E_i represents unfeasible edges. In the beginning, no edges are marked. While not in an accepting state, we repeatedly update the current state (line 6-7), try to find a next action to execute (line 9-15) or make a request if the planner cannot find a feasible next step (line 17-22). First, we search for a shortest path to an accepting state that does not use any marked edges using standard methods. The path length is defined through the number of required transitions. From this path, we select the first transition (q_c, q_{next}) that we wish to enable in a future step (line 10). If we are in an accepting state, i.e., $q_c \in F$, then we stay in that state, i.e., $q_{next} = q_c$.

Planning the Next Action We define the next action as a tuple of the progress and the constraint set defined as in Equation 5.3 and Equation 5.4. By satisfying any $\sigma \in \Sigma_p$, we transition into the planned state, bringing us closer to an accepting state. By never satisfying any $\sigma \in \Sigma_c$, we guarantee staying in q_c until we satisfy any $\sigma \in \Sigma_p$. We call the tuple (Σ_p, Σ_c) a *planned action.*

Marking Unfeasible Edges If a planned action (Σ_p, Σ_c) cannot be achieved, the outgoing transition from the current state q_c corresponding to Σ_p is marked as unfeasable(line 15). This can either be the result of the low-level planner not being able to find a suitable new position for one of the objects or a failure report from the executing platform. In the next iteration of the planning loop, we search a new path to an accepting state. If no such path exists, there is no next feasible action.

Requesting Help If no next feasible action exists, we either terminate or request help. We check each outgoing, but marked edge from the current state. If the resulting next state has a path to an accepting state, that marked transition is viable for a request. If no such edge exist, the specification is not satisfiable anymore and we terminate. In every other case, we have to choose an action (Σ_p, Σ_c) to request. We wish the robot to be as autonomous as possible in satisfying its own task, so we define the notion of a *smallest* request.

Definition 54 (Smallest Request)**.** *From current state q_c, we first find the set of symbols that let us remain in that state:*

$$\Sigma_i = \{\sigma \mid \delta(q_c, \sigma) = q_c\}$$

We define the size of a request (Σ_p, Σ_c) with respect to the current state as:

$$|(\Sigma_p, \Sigma_c)| = \min_{\forall \sigma_i \in \Sigma_i, \forall \sigma_p \in \Sigma_p} (|(\sigma_i \setminus \sigma_p) \cup (\sigma_p \setminus \sigma_i)|)$$

The smallest request then is the tuple (Σ_p, Σ_c) such that:

$$\arg\min_{q_{next}}(|(\Sigma_p, \Sigma_c)|)$$

After issuing a request for action (Σ_p, Σ_c), the robot has to wait for the operator to finish the request. While waiting for the user to finalize the request, the robot monitors the scene and updates the DFA accordingly. After the request is finished, the planning loop can continue as described above.

Use-Case Demonstration

We implemented Algorithm 6 on a Baxter Robot from Rethink Robotics. Baxter is mounted in front of a table. On this table, the useable workspace is divided into equal rectangles. These areas are denoted as e.g. *top_ left_ corner.* In this

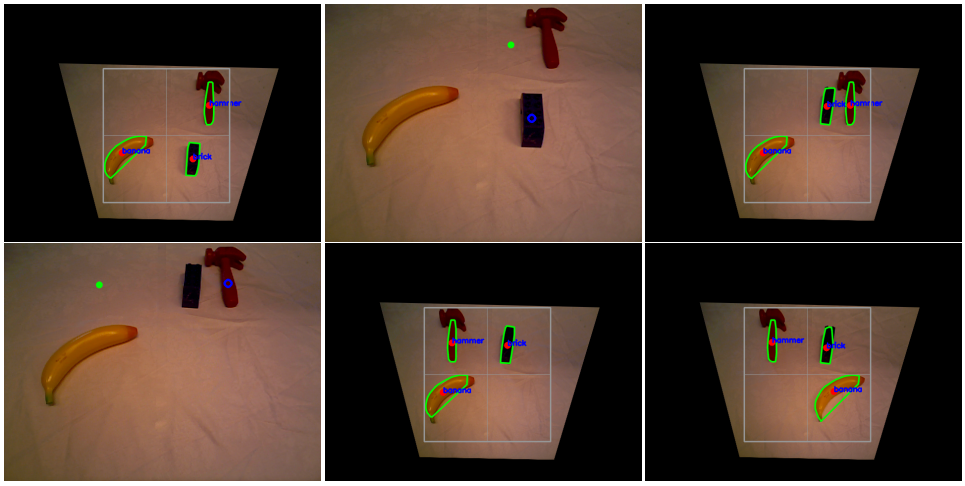


Figure 5.20: Baxter observing the workspace and planning steps to satisfy the task. Between the 5th and 6th image, a human assistant fulfilled the request shown in Figure 5.21.

implementation, baxter is able to detect and grasp three objects: a hammer, a brick and a banana. In a first, exploratory demonstration, we forbid the banana being grasped, as it might damage the fruit if it were real. Baxter is tasked with the following specification:

$$
\begin{aligned}
\phi = {} & F\,hammer \text{ enclIn } top_left_corner \\
& \wedge F\,brick \text{ enclIn } top_right_corner \\
& \wedge F\,banana \text{ enclIn } bottom_right_corner \\
& \wedge G\neg hammer \text{ ovlp } banana \\
& \wedge G\neg hammer \text{ ovlp } brick \\
& \wedge G\neg brick \text{ ovlp } banana
\end{aligned}
$$

In Figure 5.20, we depict steps of the execution from the perspective of the robot. After moving the brick and the hammer into suitable positions, the robot realizes that it cannot fulfill the specifcation without moving the banana, so it emits a request depicted in Figure 5.21. The request only contains information relevant to the next action and is constructed from the symbols of Σ_p.

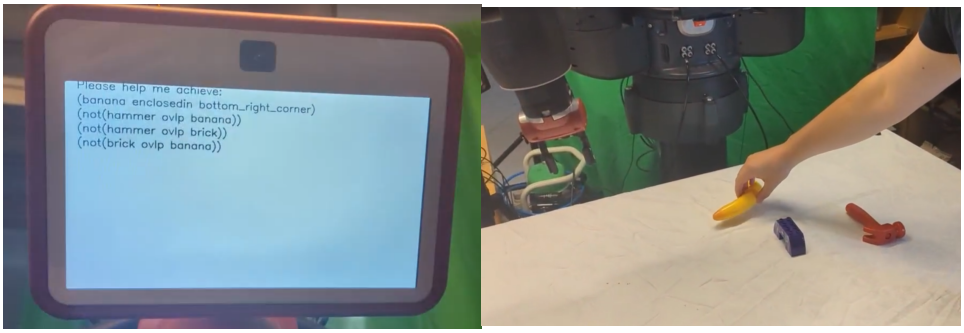


Figure 5.21: Baxter displays a request to help satisfy the specification. A human assistant places the banana in the requested spot.

5.3 Discussion and Limitations

We presented SpaTiaL, a temporal logic-based language to intuitively specify object-centric tasks and constraints. Our results showed that we can use SpaTiaL to specify, monitor and plan rich spatio-temporal robotic tasks, such as object pushing, pick and place, and agent monitoring tasks.

We propose various spatial relations, such as leftOf, rightOf and above, to specify object placements. Currently, users can only address tasks which can be described with the existing set of spatial relations. However, this set can be extended by defining new spatial relations, such as presented in our experiments. In the future, we would like to add more spatial relations and support for occlusions and partial observability.

Our quantitative semantics only capture spatial and temporal aspects. For many robotics tasks, it may be important to include additional object properties in the specification, e.g., an object's color, weight, or temperature. Since SpaTial is a subset of STL, these properties can be translated into a signal function that provides a satisfaction value for every timestep and could be incorporated.

Compared to classical TAMP approaches such as PDDLstream [147], SpaTiaL focuses on object-centric tasks, allowing users to be robot-agnostic and disregard robot dynamics for simplicity. This comes with the limitation that SpaTiaL cannot synthesize plans directly as opposed to the reasoning system in PDDL. Moreover, SpaTiaL specifications can only define pre-conditions on the robot's state that can be grounded on geometrical representations and not yet identify dynamical illogical specifications. Yet, SpaTiaL operates in continuous space and can be combined with almost any planner and controller. By determining the desired object configuration, a dedicated planner and controller can take care of the successful satisfaction of the specification.

This combination with a planner and controller, however, cannot guarantee completeness (see also Sec. 15) as opposed to integrated symbolic and control ap-

proaches such as logic geometric programming [148]. These integrated approaches can also consider object physics and plan object grasps. On the other hand, SpaTiaL provides a larger spatial and temporal expressivity. Users can specify a wide variety of tasks with the rich set of spatial relations, easily incorporate new relations, and use bounded and unbounded temporal operators. This makes it possible to use SpaTiaL for a wide variety of tasks, alone as a specification language or monitoring approach.

To improve the panning with SpaTiaL, the planning algorithm can be improved by providing general pruning rules for infeasible DFA transitions. Currently, the tool MONA can generate the DFAs from SpaTiaL specifications in milliseconds. However, finding object target positions to satisfy Lemma 12, e.g., by sampling, is more costly and can take from seconds to minutes. To address the computation times, future research includes investigating optimization-based approaches to satisfy spatial relations in our planning approach.

Chapter 6

Conclusions and Future Work

In this thesis, we extended the benefits of formal synthesis approaches to MAS task planning problems consisting of robotic and human agents with different capabilities, constraints, goals or tasks in a shared environment.

In Section 1.1, we presented major design decisions alongside three research questions:

> How do we model the interaction of multiple agents in such a way that we can both obtain global guarantees on specification satisfaction and retain scalability of the resulting method?

Approaching the issue of scalability, we presented two approaches that start from opposite ends of the spectrum presented in Figure 1.1. In our compositional method as presented in Chapter 3, agents are modelled as cooperative actors. This method can alleviate scalability issues in scenarios where agents are highly interdependent and invalid solutions can be pruned early. In scenarios where interactions are sparse, the assumption-based methods presented in Chapter 4 shine instead. Here, agents start as adversaries. Through computation of minimal assumptions and communication of least-limiting advisers, agents can keep their strategy planning as local as possible. A key step is the separation of assumptions and advisers into safety and fairness aspects.

> How do we represent human actors in our methods without making wrong or unnecessary assumptions about their preferences or capabilities?

The presented notion of least-limiting advisers also provides a theoretically sound way to convey critical information to a human actor, as presented an studied in Section 4.3. Since only two types of advisers exist, the assumptions are structurally simple enough to convey during runtime. Through a virtual pilot study, we demonstrated that agents using our method were perceived as more intelligent, safer and more compliant compared to stricter baseline methods. Our assume-guarantee approach prevents making unnecessary assumptions about human behaviour and

does not depend on an explicit human model. While our method certainly proved to be successful in our study, the communication of made assumptions could still present an issue. Computed assumptions can take the shape of arbitrary boolean formulas, which can be too complicated to present as pictograms. Searching to make usage of temporal logics more intuitive, both in specification as well as in communication of artifacts produced by our algorithms, led to the third question:

> How do we adapt temporal specification languages to enable users to intuitively specify object-oriented tasks and algorithms to effectively communicate requests or constraints to human operators?

When dealing with object-centric tasks, specification of desired properties and constraints through signal functions can be quite cumbersome. We presented SpaTiaL, a spatio-temporal framework that combines techniques from computational geometry and temporal logics to provide a unified framework to specify, monitor and plan rich spatio-temporal tasks for robotic applications. It operates in an object-centric fashion and provides various spatial relations between objects, such as an object is left of or overlaps with another object. In various experiments, we used SpaTiaL to monitor tasks in real-time and for online planning where the executing robot can request necessary help from other actors through spatial relations.

Overall, through this thesis, we provide a novel approach for autonomous systems to efficiently plan for task satisfaction and communication necessary information between computational, but also human actors. We envision these works to enable smart and transparent reactive systems that fulfill their tasks autonomously and can reason about the information they communicate.

6.1 Future Directions

Many aspects of the presented methods and ideas can be improved on, be it in terms of strengthening and generalizing the theoretical results, improving the usefulness of methods or further studying their impact through extended user studies. In this section, we briefly outline directions for future work, presented in no particular order.

Extending Least-Limiting Advisers to ω-regular Objectives

The theory presented in Chapter 4 could be extended to use LTL formulae of infinite traces, leading to stochastic games with ω-regular objectives. Such games have been studied extensively and theoretical results are established [149]. Our main reason to focus on LTL_f specifications in this line of work was the availability of mature and cabable model checkers such as PRISM-games [97]. To the best of our knowledge, no tool comparable in maturity and stability exists for stochastic games with ω-regular objectives. Through development of such a model checker,

the results of Chapter 4 can easily be generalized and strengthened, allowing users to specify even richer behaviours.

Richer User Studies

The user study conducted in Section 4.3 gives indication that least-limiting advisers might be a useful approach to provide guarantees in a human-robot interaction. However, giving the autonomous system the power to impose assumptions on the human behaviour could potentially feel restrictive or condescending to human users. The effect of the robot computing and giving advice in other domains, such as navigating corridors and making way in doorways, could be wildly different so it should be studied further. Additionally, further research effort could be spent to study and improve communication of least-limiting advisers through exploring different modes of communication, such as light, sound or generated verbal expressions. A user study could reveal if spatial requests as presented in Section 5.2 are understandable to human participants and if this type of advice could be handled by novice users.

Stronger Planning for SpaTiaL

The planning methods for SpaTiaL presented in Section 5.1 use a very simplistic uniform sampling approach to find new positions for objects. Efficiency of sampling could be improved by sampling that is biased or guided by the robustness metric. A major weakness of the current planning approach as implemented in Section 15 is that it tries to trigger a transition in the DFA by moving a single object. The current implementation cannot handle cases where progress in the DFA requires multi-step manipulation, such as clearing an area before moving an object into that area. Smarter sampling methods could alleviate this issue by efficiently searching a bigger space of sequential object changes.

Spatial Preferences

While the framework we presented in Chapter 5 comes with quantitative robustness metrics, those robustness values are not always a good indicator for the most suitable position for an object. Consider the following example: When setting a table, a human would say that the fork has to be left of the plate. Through maximizing robustness of the expression *fork leftof plate*, we obtain a point on the edge of the workspace. The further the fork is on the left side of the plate, the higher the robustness metric. In actuality, the user probably wanted the fork close to the plate, as well. In reality, there are many more constraints to the position of the fork in a classical western table setting task, such as angle of the fork or alignment with other utensils. Encoding all of this information into a specification formula can be quite cumbersome. SpaTiaL could still be used qualitatively to guarantee that the constraints are fulfilled logically, but the optimal position could be found through

any other method. For example, a modern machine learning approach could learn a suitable position for the fork from example demonstration or available images of set tables.

Improvements for Spatial Requests

Spatial requests are currently only displayed as text (see Figure 5.21). We could improve the understanding of the requested subtask by displaying supporting imagery. Another way to improve the spatial requests would be to incorporate the perspective of the assistant, as spatial relations like *leftOf* depend on the perspective of the user. While the robot waits for request completion of a request as described in Section 5.2, it could guide the human by issuing constraint violation warnings if the human is close to violating a constraint. This could improve request execution through a human actor and increase confidence in the understanding of the request.

Bibliography

[1] Christoph Schulte Althoff, Wolfgang Thomas, and Nico Wallmeier. "Observations on determinization of Büchi automata". In: *International Conference on Implementation and Application of Automata*. Springer. 2005, pp. 262–272.

[2] Marius Kloetzer and Calin Belta. "Dealing with nondeterminism in symbolic control". In: *International Workshop on Hybrid Systems: Computation and Control*. Springer. 2008, pp. 287–300.

[3] Dana Fisman, Orna Kupferman, and Yoad Lustig. "Rational synthesis". In: *Tools and Algorithms for the Construction and Analysis of Systems: 16th International Conference, TACAS 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings 16*. Springer. 2010, pp. 190–204.

[4] Krishnendu Chatterjee and Thomas A Henzinger. "Assume-guarantee synthesis". In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 2007, pp. 261–275.

[5] Colin F Camerer. *Behavioral game theory: Experiments in strategic interaction*. Princeton university press, 2011.

[6] Claudio Menghi et al. "PsALM: specification of dependable robotic missions". In: *Proc. of the IEEE/ACM Int. Conf. on Software Engineering*. 2019, pp. 99–102.

[7] Claudio Menghi et al. "Specification patterns for robotic missions". In: *IEEE Transactions on Software Engineering* 47.10 (2019), pp. 2208–2224.

[8] Nikolaus Correll et al. "Analysis and observations from the first amazon picking challenge". In: *IEEE Transactions on Automation Science and Engineering* 15.1 (2016), pp. 172–188.

[9] Hadas Kress-Gazit, Morteza Lahijanian, and Vasumathi Raman. "Synthesis for robots: Guarantees and feedback for robot behavior". In: *Annual Review of Control, Robotics, and Autonomous Systems* 1 (2018), pp. 211–236.

[10] Shen Li et al. "Reactive task and motion planning under temporal logic specifications". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 12618–12624.

[11] David Gundana and Hadas Kress-Gazit. "Event-Based Signal Temporal Logic Tasks: Execution and Feedback in Complex Environments". In: *IEEE Robotics and Automation Letters* 7.4 (2022), pp. 10001–10008.

[12] Calin Belta, Boyan Yordanov, and Ebru Aydin Gol. *Formal methods for discrete-time dynamical systems*. Vol. 89. Springer, 2017.

[13] Georgios E Fainekos, Hadas Kress-Gazit, and George J Pappas. "Hybrid controllers for path planning: A temporal logic approach". In: *Proceedings of the 44th IEEE Conference on Decision and Control*. IEEE. 2005, pp. 4885–4890.

[14] Paulo Tabuada and George J Pappas. "Model checking LTL over controllable linear systems is decidable". In: *International Workshop on Hybrid Systems: Computation and Control*. Springer. 2003, pp. 498–513.

[15] Marius Kloetzer and Calin Belta. "A fully automated framework for control of linear systems from temporal logic specifications". In: *IEEE Transactions on Automatic Control* 53.1 (2008), pp. 287–297.

[16] Y. Kantaros and M. M. Zavlanos. "Distributed Optimal Control Synthesis for Multi-Robot Systems under Global Temporal Tasks". In: *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*. 2018, pp. 162–173.

[17] Brian P Gerkey and Maja J Matarić. "A formal analysis and taxonomy of task allocation in multi-robot systems". In: *The International Journal of Robotics Research* 23.9 (2004), pp. 939–954.

[18] Laëtitia Matignon, Laurent Jeanpierre, and Abdel-Illah Mouaddib. "Coordinated multi-robot exploration under communication constraints using decentralized markov decision processes". In: *Twenty-sixth AAAI conference on artificial intelligence*. 2012.

[19] Hang Ma and Sven Koenig. "Optimal target assignment and path finding for teams of agents". In: *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems. 2016, pp. 1144–1152.

[20] Constructions Aeronautiques et al. "PDDL| the planning domain definition language". In: *Technical Report, Tech. Rep.* (1998).

[21] Richard E Fikes and Nils J Nilsson. "STRIPS: A new approach to the application of theorem proving to problem solving". In: *Artificial intelligence* 2.3-4 (1971), pp. 189–208.

[22] Yushan Chen, Xu Chu Ding, and Calin Belta. "Synthesis of distributed control and communication schemes from global LTL specifications". In: *2011 50th IEEE Conference on Decision and Control and European Control Conference*. IEEE. 2011, pp. 2718–2723.

[23] Xu Chu Ding et al. "Automatic deployment of robotic teams". In: *IEEE Robotics & Automation Magazine* 18.3 (2011), pp. 75–86.

[24] Meng Guo, Jana Tumova, and Dimos V Dimarogonas. "Cooperative decentralized multi-agent control under local LTL tasks and connectivity constraints". In: *53rd IEEE conference on decision and control*. IEEE. 2014, pp. 75–80.

[25] Meng Guo and Dimos V Dimarogonas. "Multi-agent plan reconfiguration under local LTL specifications". In: *The International Journal of Robotics Research* 34.2 (2015), pp. 218–235.

[26] Alireza Partovi, Rafael Rodrigues da Silva, and Hai Lin. "Reactive Integrated Mission and Motion Planning for Mobile Robotic Manipulators". In: *2018 Annual American Control Conference (ACC)*. IEEE. 2018, pp. 3538–3543.

[27] Bruno Lacerda and Pedro U Lima. "Petri net based multi-robot task coordination from temporal logic specifications". In: *Robotics and Autonomous Systems* 122 (2019), p. 103289.

[28] Philipp Schillinger, Mathias Bürger, and Dimos V Dimarogonas. "Decomposition of finite LTL specifications for efficient multi-agent planning". In: *Distributed Autonomous Robotic Systems*. Springer, 2018, pp. 253–267.

[29] Philipp Schillinger, Mathias Bürger, and Dimos V Dimarogonas. "Hierarchical LTL-Task MDPs for Multi-Agent Coordination through Auctioning and Learning". In: *The international journal of robotics research* (2019).

[30] Alessio Mosca et al. "Multi-robot routing and scheduling with temporal logic and synchronization constraints". In: *Proceedings of the 2019 2nd International Conference on Control and Robot Technology*. 2019, pp. 40–45.

[31] Yiannis Kantaros and Michael M Zavlanos. "Stylus*: A temporal logic optimal control synthesis algorithm for large-scale multi-robot systems". In: *The International Journal of Robotics Research* 39.7 (2020), pp. 812–836.

[32] Jonathan A DeCastro et al. "Collision-free reactive mission and motion planning for multi-robot systems". In: *Robotics research*. Springer, 2018, pp. 459–476.

[33] Jana Tumova and Dimos V Dimarogonas. "Multi-agent planning under local LTL specifications and event-based synchronization". In: *Automatica* 70 (2016), pp. 239–248.

[34] Jana Tumova and Dimos V Dimarogonas. "A receding horizon approach to multi-agent planning from local LTL specifications". In: *2014 American Control Conference*. IEEE. 2014, pp. 1775–1780.

[35] Amir Pnueli and Roni Rosner. "On the synthesis of a reactive module". In: *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. ACM. 1989, pp. 179–190.

[36] Shmuel Safra. "On the complexity of omega-automata". In: *[Proceedings 1988] 29th Annual Symposium on Foundations of Computer Science*. IEEE. 1988, pp. 319–327.

[37] Orna Kupferman and Moshe Y Vardi. "Safraless decision procedures". In: *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*. IEEE. 2005, pp. 531–540.

[38] Barbara Jobstmann and Roderick Bloem. "Optimizations for LTL synthesis". In: *2006 Formal Methods in Computer Aided Design*. IEEE. 2006, pp. 117–124.

[39] Emmanuel Filiot, Naiyong Jin, and Jean-François Raskin. "Antichains and compositional algorithms for LTL synthesis". In: *Formal Methods in System Design* 39.3 (2011), pp. 261–296.

[40] Aaron Bohy et al. "Acacia+, a tool for LTL synthesis". In: *International Conference on Computer Aided Verification*. Springer. 2012, pp. 652–657.

[41] Saqib Sohail and Fabio Somenzi. "Safety first: A two-stage algorithm for LTL games". In: *2009 Formal Methods in Computer-Aided Design*. IEEE. 2009, pp. 77–84.

[42] Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. "Synthesis of reactive (1) designs". In: *International Workshop on Verification, Model Checking, and Abstract Interpretation*. Springer. 2006, pp. 364–380.

[43] Orna Kupferman, Nir Piterman, and Moshe Y Vardi. "Safraless compositional synthesis". In: *International Conference on Computer Aided Verification*. Springer. 2006, pp. 31–44.

[44] Wouter Kuijper and Jaco Van de Pol. "Compositional control synthesis for partially observable systems". In: *International Conference on Concurrency Theory*. Springer. 2009, pp. 431–447.

[45] Roderick Bloem et al. "How to handle assumptions in synthesis". In: *arXiv preprint arXiv:1407.5395* (2014).

[46] Rajeev Alur, Salar Moarref, and Ufuk Topcu. "Counter-strategy guided refinement of GR (1) temporal logic specifications". In: *2013 Formal Methods in Computer-Aided Design*. IEEE. 2013, pp. 26–33.

[47] V. Raman and H. Kress-Gazit. "Automated feedback for unachievable high-level robot behaviors". In: *2012 IEEE International Conference on Robotics and Automation*. 2012, pp. 5156–5162. DOI: 10.1109/ICRA.2012.6224807.

[48] V. Raman and H. Kress-Gazit. "Towards minimal explanations of unsynthesizability for high-level robot behaviors". In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2013, pp. 757–762. DOI: 10.1109/IROS.2013.6696436.

[49] Kasra Ghasemi, Sadra Sadraddini, and Calin Belta. "Compositional synthesis via a convex parameterization of assume-guarantee contracts". In: *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*. 2020, pp. 1–10.

[50] Marta Kwiatkowska et al. "Assume-guarantee verification for probabilistic systems". In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 2010, pp. 23–37.

[51] Sagar Chaki et al. "Automated assume-guarantee reasoning for simulation conformance". In: *International Conference on Computer Aided Verification*. Springer. 2005, pp. 534–547.

[52] Anvesh Komuravelli, Corina S Păsăreanu, and Edmund M Clarke. "Assume-guarantee abstraction refinement for probabilistic systems". In: *International Conference on Computer Aided Verification*. Springer. 2012, pp. 310–326.

[53] YuFeng Chen et al. "Design of a maximally permissive liveness-enforcing Petri net supervisor for flexible manufacturing systems". In: *IEEE Transactions on automation science and engineering* 8.2 (2010), pp. 374–393.

[54] Jana Tumova and Dimos V Dimarogonas. "Synthesizing least-limiting guidelines for safety of semi-autonomous systems". In: *2016 IEEE 55th Conference on Decision and Control (CDC)*. IEEE. 2016, pp. 5714–5719.

[55] Julien Bernet, David Janin, and Igor Walukiewicz. "Permissive strategies: from parity games to safety games". In: *RAIRO-Theoretical Informatics and Applications* 36.3 (2002), pp. 261–275.

[56] Krishnendu Chatterjee, Thomas A Henzinger, and Barbara Jobstmann. "Environment assumptions for synthesis". In: *International Conference on Concurrency Theory*. Springer. 2008, pp. 147–161.

[57] Vasumathi Raman and Hadas Kress-Gazit. "Automated feedback for unachievable high-level robot behaviors". In: *2012 IEEE International Conference on Robotics and Automation*. IEEE. 2012, pp. 5156–5162.

[58] Pratyusha Sharma et al. "Correcting Robot Plans with Natural Language Feedback". In: *Proceedings of Robotics: Science and Systems*. New York City, NY, USA, 2022. DOI: 10.15607/RSS.2022.XVIII.065.

[59] Jana Tumova et al. "Least-violating control strategy synthesis with safety rules". In: *Proceedings of the 16th international conference on Hybrid systems: computation and control*. 2013, pp. 1–10.

[60] Jana Tumova et al. "Maximally satisfying LTL action planning". In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2014, pp. 1503–1510.

[61] Matthew R Maly et al. "Iterative temporal motion planning for hybrid systems in partially unknown environments". In: *Proceedings of the 16th international conference on Hybrid systems: computation and control*. 2013, pp. 353–362.

[62] Guilhem Buisan and Rachid Alami. "A Human-Aware Task Planner Explicitly Reasoning About Human and Robot Decision, Action and Reaction". In: *Companion of the 2021 ACM/IEEE International Conference on Human-Robot Interaction*. 2021, pp. 544–548.

[63] Lu Feng et al. "Synthesis of human-in-the-loop control protocols for autonomous systems". In: *IEEE Transactions on Automation Science and Engineering* 13.2 (2016), pp. 450–462.

[64] Sebastian Junges et al. "Model checking for safe navigation among humans". In: *International Conference on Quantitative Evaluation of Systems*. Springer. 2018, pp. 207–222.

[65] Toki Migimatsu and Jeannette Bohg. "Object-Centric Task and Motion Planning in Dynamic Environments". In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 844–851. DOI: 10.1109/LRA.2020.2965875.

[66] Keliang He et al. "Efficient Symbolic Reactive Synthesis for Finite-Horizon Tasks". In: *2019 International Conference on Robotics and Automation (ICRA)*. 2019, pp. 8993–8999. DOI: 10.1109/ICRA.2019.8794170.

[67] Zeynep Barlas. "When robots tell you what to do: Sense of agency in human- and robot-guided actions". In: *Consciousness and cognition* 75 (2019), p. 102819.

[68] Megan Strait, Cody Canning, and Matthias Scheutz. "Let Me Tell You! Investigating the Effects of Robot Communication Strategies in Advice-Giving Situations Based on Robot Appearance, Interaction Modality and Distance". In: *Proceedings of the 2014 ACM/IEEE International Conference on Human-Robot Interaction*. HRI '14. Bielefeld, Germany: Association for Computing Machinery, 2014, pp. 479–486. ISBN: 9781450326582. DOI: 10.1145/2559636.2559670. URL: https://doi.org/10.1145/2559636.2559670.

[69] Cristen Torrey, Susan R Fussell, and Sara Kiesler. "How a robot should give advice". In: *2013 8th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*. IEEE. 2013, pp. 275–282.

[70] Mycal Tucker et al. "Towards human-agent communication via the information bottleneck principle". In: *arXiv preprint arXiv:2207.00088* (2022).

[71] Karinne Ramirez-Amaro et al. "Enhancing human action recognition through spatio-temporal feature learning and semantic rules". In: *Proc. of the IEEE-RAS Int. Conf. on Humanoid Robots*. 2013, pp. 456–461.

[72] Karinne Ramirez-Amaro, Michael Beetz, and Gordon Cheng. "Transferring skills to humanoid robots by extracting semantic representations from observations of human activities". In: *Artificial Intelligence* 247 (2017), pp. 95–118.

[73] Maximilian Diehl, Chris Paxton, and Karinne Ramirez-Amaro. "Automated Generation of Robotic Planning Domains from Observations". In: *Proc. of the IEEE/RSJ Int. Conf, on Intelligent Robots and Systems.* 2021, pp. 6732–6738.

[74] Wentao Yuan et al. "SORNet: Spatial Object-Centric Representations for Sequential Manipulation". In: *Conference on Robot Learning.* 2022, pp. 148–157.

[75] Weiyu Liu et al. "StructFormer: Learning Spatial Structure for Language-Guided Semantic Rearrangement of Novel Objects". In: *Proc. of the Int. Conf. on Robotics and Automation.* 2022, pp. 6322–6329.

[76] Chris Paxton et al. "Predicting stable configurations for semantic placement of novel objects". In: *Conf. on Robot Learning.* 2022, pp. 806–815.

[77] Gregory Izatt and Russ Tedrake. "Generative modeling of environments with scene grammars and variational inference". In: *Proc. of the IEEE Int. Conf. on Robotics and Automation.* 2020, pp. 6891–6897.

[78] Fabian Paus, Teng Huang, and Tamim Asfour. "Predicting pushing action effects on spatial object relations by learning internal prediction models". In: *Proc. of the IEEE Int. Conf. on Robotics and Automation.* 2020, pp. 10584–10590.

[79] Coline Devin et al. "Deep object-centric representations for generalizable robot learning". In: *Proc. of the IEEE Int. Conf. on Robotics and Automation.* 2018, pp. 7111–7118.

[80] Mohit Sharma et al. "Learning to Compose Hierarchical Object-Centric Controllers for Robotic Manipulation". In: *Proc. of the Conference on Robot Learning.* Vol. 155. 2020, pp. 822–844.

[81] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. "CLIPort: What and where pathways for robotic manipulation". In: *Conf. on Robot Learning.* 2022, pp. 894–906.

[82] Lucas Manuelli et al. "KPAM: KeyPoint Affordances for Category-Level Robotic Manipulation". In: *Proc. of the Int. Symposium on Robotics Research.* Vol. 20. Springer, 2019, pp. 132–157.

[83] Toki Migimatsu and Jeannette Bohg. "Object-centric task and motion planning in dynamic environments". In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 844–851.

[84] Alejandro Agostini and Dongheui Lee. "Efficient state abstraction using object-centered predicates for manipulation planning". In: *arXiv preprint arXiv:2007.08251* (2020).

[85] Philipp Jund et al. "Optimization beyond the convolution: Generalizing spatial relations with end-to-end metric learning". In: *Proc. of the IEEE Int. Conf. on Robotics and Automation.* 2018, pp. 4510–4516.

[86] Sergio Guadarrama et al. "Grounding spatial relations for human-robot interaction". In: *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*. 2013, pp. 1640–1647.

[87] Rainer Kartmann et al. "Extraction of physically plausible support relations to predict and validate manipulation action effects". In: *IEEE Robotics and Automation Letters* 3.4 (2018), pp. 3991–3998.

[88] Rasoul Mojtahedzadeh et al. "Support relation analysis and decision making for safe robotic manipulation tasks". In: *Robotics and Autonomous Systems* 71 (2015), pp. 99–117.

[89] Swagatika Panda, AH Abdul Hafez, and CV Jawahar. "Single and multiple view support order prediction in clutter for manipulation". In: *Journal of Intelligent & Robotic Systems* 83.2 (2016), pp. 179–203.

[90] Benjamin Rosman and Subramanian Ramamoorthy. "Learning spatial relationships between objects". In: *The Int. Journal of Robotics Research* 30.11 (2011), pp. 1328–1342.

[91] Danny Driess et al. "Learning Geometric Reasoning and Control for Long-Horizon Tasks from Visual Input". In: *Proc. of the IEEE Int. Conf. on Robotics and Automation*. 2021, pp. 14298–14305.

[92] Xinghang Li et al. "Embodied Semantic Scene Graph Generation". In: *Conf. on Robot Learning*. 2022, pp. 1585–1594.

[93] Calin Belta, Volkan Isler, and George J Pappas. "Discrete abstractions for robot motion planning and control in polygonal environments". In: *IEEE Transactions on Robotics* 21.5 (2005), pp. 864–874.

[94] Georgios E Fainekos et al. "Temporal logic motion planning for dynamic robots". In: *Automatica* 45.2 (2009), pp. 343–352.

[95] Nick Hawes et al. "The strands project: Long-term autonomy in everyday environments". In: *IEEE Robotics & Automation Magazine* 24.3 (2017), pp. 146–156.

[96] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT press, 2008.

[97] M. Kwiatkowska et al. "PRISM-games 3.0: Stochastic Game Verification with Concurrency, Equilibria and Time". In: *Proc. 32nd International Conference on Computer Aided Verification (CAV'20)*. Vol. 12225. LNCS. Springer, 2020, pp. 475–487.

[98] Peter Øhrstrøm and Per Hasle. *Temporal logic: from ancient ideas to artificial intelligence*. Vol. 57. Springer Science & Business Media, 2007.

[99] Arthur N Prior. *Time and modality*. OUP Oxford, 1957.

[100] Arthur N Prior and Per FV Hasle. *Papers on time and tense*. Oxford University Press on Demand, 2003.

[101] Valentin Goranko and Antje Rumberg. "Temporal Logic". In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta. Summer 2020. Metaphysics Research Lab, Stanford University, 2020.

[102] Moshe Y Vardi. "Branching vs. linear time: Final showdown". In: *International conference on tools and algorithms for the construction and analysis of systems*. Springer. 2001, pp. 1–22.

[103] Amir Pnueli. "The temporal logic of programs". In: *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*. IEEE. 1977, pp. 46–57.

[104] Giuseppe De Giacomo and Moshe Y Vardi. "Linear temporal logic and linear dynamic logic on finite traces". In: *IJCAI'13 Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*. Association for Computing Machinery. 2013, pp. 854–860.

[105] Zohar Manna and Amir Pnueli. "A hierarchy of temporal properties (invited paper, 1989)". In: *Proceedings of the ninth annual ACM symposium on Principles of distributed computing*. 1990, pp. 377–410.

[106] Ron Koymans. "Specifying real-time properties with metric temporal logic". In: *Real-time systems* 2.4 (1990), pp. 255–299.

[107] Thomas A Henzinger. *The temporal specification and verification of real-time systems*. stanford university, 1991.

[108] Joël Ouaknine and James Worrell. "On the decidability of metric temporal logic". In: *20th Annual IEEE Symposium on Logic in Computer Science (LICS'05)*. IEEE. 2005, pp. 188–197.

[109] Rajeev Alur, Tomás Feder, and Thomas A Henzinger. "The benefits of relaxing punctuality". In: *Journal of the ACM* 43.1 (1996), pp. 116–146.

[110] Thomas A Henzinger, J-F Raskin, and P-Y Schobbens. "The regular real-time languages". In: *Automata, Languages and Programming: 25th International Colloquium, ICALP'98 Aalborg, Denmark, July 13–17, 1998 Proceedings 25*. Springer. 1998, pp. 580–591.

[111] Alexandre Donzé and Oded Maler. "Robust satisfaction of temporal logic over real-valued signals". In: *Proc. of the Int. Conf. on Formal Modeling and Analysis of Timed Systems*. 2010, pp. 92–106.

[112] Lars Lindemann and Dimos V Dimarogonas. "Control barrier functions for signal temporal logic tasks". In: *IEEE control systems letters* 3.1 (2018), pp. 96–101.

[113] Vasumathi Raman et al. "Model predictive control with signal temporal logic specifications". In: *Proc. of the IEEE Conf. on Decision and Control*. 2014, pp. 81–87.

[114] Fernando S Barbosa et al. "Guiding autonomous exploration with signal temporal logic". In: *IEEE Robotics and Automation Letters* 4.4 (2019), pp. 3332–3339.

[115] Lars Lindemann and Dimos V Dimarogonas. "Robust motion planning employing signal temporal logic". In: *Proc. of the American Control Conf.* 2017, pp. 2950–2955.

[116] Dorsa Sadigh and Ashish Kapoor. "Safe control under uncertainty with probabilistic signal temporal logic". In: *Proceedings of Robotics: Science and Systems XII.* 2016.

[117] Morteza Lahijanian, Sean B Andersson, and Calin Belta. "Temporal logic motion planning and control with probabilistic satisfaction guarantees". In: *IEEE Transactions on Robotics* 28.2 (2011), pp. 396–409.

[118] Jesper G Henriksen et al. "Mona: Monadic second-order logic in practice". In: *International Workshop on Tools and Algorithms for the Construction and Analysis of Systems.* Springer. 1995, pp. 89–110.

[119] Paul Gastin and Denis Oddoux. "Fast LTL to Büchi automata translation". In: *Computer Aided Verification: 13th International Conference, CAV 2001 Paris, France, July 18–22, 2001 Proceedings 13.* Springer. 2001, pp. 53–65.

[120] Georg Friedrich Schuppe and Jana Tumova. "Multi-Agent Strategy Synthesis for LTL Specifications through Assumption Composition". In: *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE).* IEEE. 2020, pp. 533–540.

[121] Franck Cassez et al. "Efficient on-the-fly algorithms for the analysis of timed games". In: *International Conference on Concurrency Theory.* Springer. 2005, pp. 66–80.

[122] Emmanuel Filiot, Nayiong Jin, and Jean-François Raskin. "Compositional algorithms for LTL synthesis". In: *International Symposium on Automated Technology for Verification and Analysis.* Springer. 2010, pp. 112–127.

[123] Georg Friedrich Schuppe and Jana Tumova. "Decentralized Multi-Agent Strategy Synthesis under LTL f Specifications via Exchange of Least-Limiting Advisers". In: *2021 International Symposium on Multi-Robot and Multi-Agent Systems (MRS).* IEEE. 2021, pp. 119–127.

[124] Bruno Lacerda, David Parker, and Nick Hawes. "Optimal Policy Generation for Partially Satisfiable Co-Safe LTL Specifications." In: *IJCAI.* 2015, pp. 1587–1593.

[125] Joost-Pieter Katoen. "The probabilistic model checking landscape". In: *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science.* 2016, pp. 31–45.

[126] Christoph Bartneck et al. "Measurement instruments for the anthropomorphism, animacy, likeability, perceived intelligence, and perceived safety of robots". In: *International journal of social robotics* 1 (2009), pp. 71–81.

[127] Andrew M Wells et al. "Learning feasibility for task and motion planning in tabletop environments". In: *IEEE Robotics and Automation Letters* 4.2 (2019), pp. 1255–1262.

[128] Neil T Dantam et al. "Incremental task and motion planning: A constraint-based approach". In: *Robotics: Science and systems*. Vol. 12. 2016, pp. 1–11.

[129] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[130] Helen Oleynikova et al. "Signed distance fields: A natural representation for both mapping and planning". In: *RSS Workshop: Geometry and Beyond-Representations, Physics, and Scene Understanding for Robotics*. 2016.

[131] Danny Driess et al. "Learning models as functionals of signed-distance fields for manipulation planning". In: *Conf. on Robot Learning*. 2022, pp. 245–255.

[132] Matt Zucker et al. "CHOMP: Covariant hamiltonian optimization for motion planning". In: *The Int. Journal of Robotics Research* 32.9-10 (2013), pp. 1164–1193.

[133] Christer Ericson. *Real-time collision detection*. CRC Press, 2004.

[134] John Schulman et al. "Motion planning with sequential convex optimization and convex collision checking". In: *The Int. Journal of Robotics Research* 33.9 (2014), pp. 1251–1270.

[135] James F Allen. "Maintaining knowledge about temporal intervals". In: *Communications of the ACM* 26.11 (1983), pp. 832–843.

[136] Alexandre Donzé, Thomas Ferrere, and Oded Maler. "Efficient robust monitoring for STL". In: *Int. Conf. on Computer Aided Verification*. Springer. 2013, pp. 264–279.

[137] Xiao Li, Cristian-Ioan Vasile, and Calin Belta. "Reinforcement learning with temporal logic rewards". In: *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*. 2017, pp. 3834–3839.

[138] Yiannis Kantaros et al. "Reactive temporal logic planning for multiple robots in unknown environments". In: *Proc. of the IEEE Int. Conf. on Robotics and Automation*. 2020, pp. 11479–11485.

[139] Tomas Lozano-Perez. "Spatial planning: A configuration space approach". In: *Autonomous robot vehicles*. Springer, 1990, pp. 259–271.

[140] Erez Shinan et al. *Lark Github Repository*. `https://github.com/lark-parser/lark`. Accessed: 2021-07-20. 2021.

[141] Sean Gillies et al. *Shapely: manipulation and analysis of geometric objects*. 2007. URL: `https://github.com/shapely/shapely`.

[142] Francesco Fuggitti. *LTLf2DFA Github Repository*. `https://github.com/whitemech/LTLf2DFA`. Accessed: 2021-07-20. 2021.

[143] Erwin Coumans and Yunfei Bai. *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. `https://pybullet.org`. 2022.

[144] John Wang and Edwin Olson. "AprilTag 2: Efficient and robust fiducial detection". In: *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*. 2016, pp. 4193–4198.

[145] Alexandre Robicquet et al. "Learning social etiquette: Human trajectory understanding in crowded scenes". In: *Proc. of the European conference on computer vision*. Springer. 2016, pp. 549–565.

[146] Daniel Harabor and Alban Grastien. "Online graph pruning for pathfinding on grid maps". In: *Proc. of the AAAI Conf. on Artificial Intelligence*. Vol. 25. 2011.

[147] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. "PDDL-Stream: Integrating Symbolic Planners and Blackbox Samplers via Optimistic Adaptive Planning". In: *Proc. of the Int. Conf. on Automated Planning and Scheduling*. 2020, pp. 440–448.

[148] Marc Toussaint. "Logic-geometric programming: An optimization-based approach to combined task and motion planning". In: *Proc. of the Int. Joint Conf. on Artificial Intelligence*. 2015.

[149] Krishnendu Chatterjee and Thomas A Henzinger. "A survey of stochastic ω-regular games". In: *Journal of Computer and System Sciences* 78.2 (2012), pp. 394–413.