Coordination of Multirobot Systems Under Temporal Constraints

by

Yunus Emre Şahin

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Electrical Engineering: Systems)
in the University of Michigan
2020

Yunus Emre Şahin
ysahin@umich.edu
ORCID iD: 0000-0003-3585-608X

To my family, and
to my beloved Salla

Acknowledgments

First of all, I would like to thank my advisor Professor Necmiye Ozay from the bottom of my heart. It would not be possible to finish this dissertation without her guidance. She was always available, extremely resourceful and helpful. I would also like to thank Professor Stavros Tripakis for hosting me in Finland at Aalto University. I really enjoyed working together with him and playing basketball with the Hellas Helsinki team. And last but not least, I would like to thank Professor Stéphane Lafortune and Professor Dimitra Panagou for accepting to be on my dissertation committee and providing invaluable feedback.

I would like to acknowledge the financial support that funded my research over the years. This dissertation is supported in part by the NASA Early Career Grant # NNX16AR65G from NASA's Space Technology Research Grants Program. Additional support came from NSF grants CNS-1446298 and ECCS-1553873, and DARPA grant N66001-14-1-4045.

I met so many amazing people in Ann Arbor. I would like to detail how amazing each of them are, but this dissertation would need to be twice as long. I would like to start by thanking the Turkish friends first: Yeliz, Ali, Alparslan, Kaan, Kıvanç, Berk, İbrahim, İsmet, Merve, Doğan, Duygu, Efe, Anıl, Ahmet, Doruk, and Anıl. I had so much fun in Ann Arbor, thanks to you all. I would like to thank my friends from the department: Greg, Mike, Julian, Kwesi, Glen, Romulo, Zexiang, Michael, Petter, Yun Jae, Brian, Kaushik, Arvind, Stephanie, Florian, and Aizhan. I have so many great memories, thanks to you. I would also like to thank my current and previous lab mates Liren, Zhe, Andrew, Craig, Sunho, Daphna, Xiangru Xu, and Farshad. Thanks for all the great discussions. I am sure I forgot to mention some people here, I am asking for their forgiveness.

I will remember Ann Arbor for many reasons, but meeting Salla was a life-changing moment. She brought so much fun and joy to my life. I learned so much from her about life, about myself, and she was my biggest support. After living in different continents for so many years, I look forward to our new chapter. I love you.

At this point, I would like to switch to my native language to thank my family. Her zaman bana inanan, benden desteğini hiçbir zaman esirgemeyen sevgili annem Serpil, babam Murtaza, ve kardeşim Cem Şahin'e ne kadar teşekkür etsem az. Size sahip olduğum için kendimi dünyanın en şanslı insanı olarak görüyorum. Sizi çok seviyorum. Sevgili anneannem ve babaannem, ikinizin de çok emeği var üzerimde. Keşke Mahmut dedem ve Hasan dedem de hayatta olsalardı da, beraber kutlayabilseydik bu güzel günü. Beni bu süreçte yalnız bırakmayan diğer tüm akrabalarıma ve arkadaşlarıma da canı gönülden teşekkür ediyorum. Hepinizi çok seviyorum.

Table of Contents

List of Figures

List of Tables

List of Appendices

Abstract

Multirobot systems have great potential to change our lives by increasing efficiency or decreasing costs in many applications, ranging from warehouse logistics to construction. They can also replace humans in dangerous scenarios, for example in a nuclear disaster cleanup mission. However, teleoperating robots in these scenarios would severely limit their capabilities due to communication and reaction delays. Furthermore, ensuring that the overall behavior of the system is safe and correct for a large number of robots is challenging without a principled solution approach. Ideally, multirobot systems should be able to plan and execute autonomously. Moreover, these systems should be robust to certain external factors, such as failing robots and synchronization errors and be able to scale to large numbers, as the effectiveness of particular tasks might depend directly on these criteria. This thesis introduces methods to achieve safe and correct autonomous behavior for multirobot systems.

Firstly, we introduce a novel logic family, called counting logics, to describe the high-level behavior of multirobot systems. Counting logics capture constraints that arise naturally in many applications where the identity of the robot is not important for the task to be completed. We further introduce a notion of robust satisfaction to analyze the effects of synchronization errors on the overall behavior and provide complexity analysis for a fragment of this logic.

Secondly, we propose an optimization-based algorithm to generate a collection of robot paths to satisfy the specifications given in counting logics. We assume that the robots are perfectly synchronized and use a mixed-integer linear programming formulation to take advantage of the recent advances in this field. We show that this approach is complete under the perfect synchronization assumption. Furthermore, we propose alternative encodings that render more efficient solutions under certain conditions. We also provide numerical results that showcase the scalability of our approach, showing that it scales to hundreds of robots.

Thirdly, we relax the perfect synchronization assumption and show how to generate paths that are robust to bounded synchronization errors, without requiring run-time communication. However, the complexity of such an approach is shown to depend on the error bound, which might be limiting. To overcome this issue, we propose a hierarchical method whose complexity does not depend on this bound. We show that, under mild conditions, solutions generated by the hierarchical method can be executed safely, even if such a bound is not known.

Finally, we propose a distributed algorithm to execute multirobot paths while avoiding collisions and deadlocks that might occur due to synchronization errors. We recast this problem as a conflict resolution problem and characterize conditions under which existing solutions to the well-known drinking philosophers problem can be used to design control policies that prevents collisions and deadlocks. We further provide improvements to this naive approach to increase the amount of concurrency in the system. We demonstrate the effectiveness of our approach by comparing it to the naive approach and to the state-of-the-art.

Chapter 1.

Introduction

There are several advantages of using multirobot systems over a single-robot system. The same task could be achieved using multiple simpler robots, instead of a single highly-capable robot, making the overall system easier and cheaper to build and maintain. Moreover, the overall system would be more reliable as simpler designs would be less prone to failures, and the loss of a single robot might be tolerated. Distributing capabilities among robots enables more modular designs and thus, making multirobot systems more flexible. Furthermore, certain tasks can only be achieved using a team of robots due to spatial constraints, such as surveilling a large area. To exemplify a few applications, multirobot systems could be used in critical search and rescue missions [8, 66, 69], surveillance [2, 47], construction automation [63, 76] and warehouse logistics [57, 115].

One school of thought focuses on the teleoperation of multirobot teams to achieve the aforementioned tasks [29, 43, 83, 92]. However, there are several issues with this approach, such as degraded performance due to latency, decreased situational awareness [16]. Furthermore, as the mission specifications get more involved, they require coordination of a large number of robots, making the problem even more challenging. Ideally, multirobot systems should be able to plan and execute autonomously to overcome these shortcomings.

Multirobot systems are studied in different communities from different viewpoints. For example, multirobot path planning (MRPP), also called multi-agent path planning (MAPP), on discrete graphs has been a popular subject of interest in artificial intelligence and robotics communities. This problem is concerned with finding collision-free paths that take each robot from their initial locations to target locations, while minimizing a cost function such as time or distance. This problem can be solved optimally using reductions to other known problems such as satisfiability [102, 103], answer set programming [27], integer-linear programming [118, 119], incremental sequential convex programming [18], or using search-based methods [12, 93, 94, 110]. However, finding optimal solution is shown to be NP-hard [101, 120]. Therefore, suboptimal solutions are used when the number of robots is large [49, 56, 58, 96, 100, 113].

MRPP in the continuous domain is also studied in the literature. Some examples include sampling-based methods [13, 50], velocity-obstacle approach [28, 108], vector fields [34, 44]. Algorithms for discrete path planning are also used in continuous domain after obtaining a discrete abstraction of the workspace [48, 98, 109]. However, a complex task, such as a search and rescue mission, cannot be formulated as a MRPP problem as it also requires high-level decisions. One needs to choose multiple waypoints and solve multiple MRPP instances.

The first step in achieving autonomy is to be able to express multirobot tasks in a high-level language. Recently, the use of temporal logics for this purpose has attracted considerable attraction. In this framework, specifications are expressed in a certain formalism, such as linear temporal logic (LTL) [38, 39, 40, 45, 46, 55, 105, 106], metric temporal logic (MTL) [41], signal temporal logic (STL) [52], spatial-temporal logic (SpaTeL) [53]. Then, controllers are algorithmically generated such that the satisfaction of the specifications is guaranteed. However, several factors prevent these techniques to be applied more frequently in multirobot setting. Firstly, commonly used temporal logics, such as LTL and STL, are not originally designed to express multirobot tasks. As a result, capturing certain specifications might require lengthy formulas. This is not desired as the complexity grows exponentially with the length of the formula. Secondly, the curse of dimensionality limits the number of robots that can be handled within this framework.

Another important aspect of multirobot coordination problems is the robustness of the solutions. Unlike single robot systems, multirobot systems might tolerate the failure of individual robots without sacrificing task fulfillment. Such a notion of robustness is examined in [23, 74, 75, 80]. On the other hand, multirobot systems might suffer from synchronization errors. If robots are not perfectly synchronized, collisions might occur, or the specifications might not be satisfied. Different strategies are proposed to avoid collision avoidance in this case, such as using buffered Voronoi cells [91, 121], reciprocal velocity-obstacles [4, 7], potential functions [36], model predictive control [30], temporal specifications [46]. However, these techniques require deviation from the nominal paths or even replanning, which might lead to deadlocks or violation of other temporal specifications. If an upper bound on the synchronization error is known, paths could be generated so that collisions and deadlocks are avoided [22]. This approach does not require replanning, but, as a result, conservative. Alternatively, robots might be forced to stop and resume [58, 97, 99, 122]. These methods can guarantee both collision and deadlock avoidance without deviating from the nominal paths. The challenge here is to increase concurrency and reduce waiting times.

1.1. Contributions and Outline

In this thesis, we study the coordination of multirobot systems under temporal constraints to address the aforementioned issues. We start by providing preliminary information that is used throughout the thesis in Section 1.2. In Chapter 2, we introduce counting logics, which enable concise expression of multirobot specifications and provide complexity analysis. Furthermore, we offer a formal definition of robust satisfaction of counting logic formulas. In Chapter 3, we assume that robots move synchronously and propose optimization-based algorithms to generate multirobot paths that ensure the satisfaction of specifications given in counting logics. In Chapter 4, we relax the synchronization assumption and show how to generate plans that are robust to synchronization errors. In Chapter 5, we propose a distributed approach to execute multirobot plans in a way that guarantees to avoid collisions and deadlocks even when the upper bound on synchronization error is not known. Finally, Chapter 6 concludes the thesis and provides some future research directions.

Chapter 2: Counting Logics

Existing methods that use temporal logic, such as LTL, to define multirobot specifications require that each robot be assigned an independent task, a tedious and error-prone process when the number of robots is large. In many applications, completion of a task depends not on identities of robots, but on the number of robots satisfying a property. Take, for example, an emergency response scenario where hundreds of autonomous vehicles are deployed to locate and help the victims. In such a scenario, it is reasonable to assume that most of the vehicles would have identical capabilities and that the identity of the vehicle is not important to the rescuers, as long as the given tasks are accomplished. On the other hand, tasks might depend on the number of robots satisfying a property. For instance, one might require sufficiently many robots to surveil a particular area to look for victims. Or, one might need to limit the number of rescuers in certain regions to avoid unsafe areas or congestion. We call this type of specification *temporal counting constraints* and propose a novel logic called *counting linear temporal logic plus (cLTL+)* to specify them. This logic consists of two layers. The inner logic defines tasks that can be satisfied by a single robot, for instance, *surveiling an area* in the previous emergency response scenario. The outer logic requires *sufficiently many* (or *not too many*) robots to satisfy tasks given as inner logic formulas. For example, one might express a task that "*at least* 2 *and not more than* 5 *robots* to surveil an area" using cLTL+.

We then formally define a notion of robust satisfaction of cLTL+ formulas, similar in spirit to [25]. Since perfect synchronization of robots might not be possible in real-life applications, we need tools to analyze the effects of synchronization errors on the satisfaction of a cLTL+ formula. The notion of robust satisfaction allows rigorous analysis of this phenomenon.

Around the same time that cLTL+ was first introduced, similar two-layered temporal logics were proposed in [65] and [116]. The logic proposed in [116], called CensusSTL, uses Signal Temporal Logic (STL) semantics instead of LTL but is otherwise similar to cLTL+. However, the focus in that paper is to infer multirobot behavior from data, and neither controller synthesis nor asynchrony are discussed. On the other hand, the authors of [65] present a decentralized controller synthesis method, but cLTL+ is strictly more expressive compared to the logic of choice. Moreover, [65] ignores collision avoidance at the synthesis-level and shifts the burden to an online lower-level controller. This approach might result in deadlocks during execution. In this thesis, we provide optimization-based solution methods for controller synthesis in Chapter 3 and in Chapter 4 such that cLTL+ specifications are satisfied, and collisions and deadlocks are avoided even when the robots move asynchronously.

We also provide an interesting fragment of cLTL+, namely *counting linear temporal logic (cLTL)*. The logic cLTL can be seen as an extension of a particular class of counting problems that deal with invariant specifications, first proposed in [67, 68]. We show that the cLTL satisfiability problem is PSPACE-complete.

Chapter 3: Path Planning Subject to Counting Constraints

After defining cLTL+, we propose optimization-based algorithms to generate individual paths that collectively satisfy specifications given in this formalism. In this chapter, we assume that robots move synchronously and show how to synthesize paths by using a mixed-integer linear programming (MILP) formulation. This approach, recasting the synthesis problem as an optimization problem, is inspired by the bounded model-checking literature [5, 42]. We demonstrate the efficacy of the MILP-based approach via numerical and experimental results.

Subsequently, we propose an alternative formulation for the particular case where the specifications are given in cLTL, and the robots have identical dynamics. The alternative solution is shown to scale much better with the number of robots. We provide numerical results showing that the number of robots has almost no effect on the solution time, and problems with hundreds of robots can be solved.

Chapter 4: Path Planning Robust to Synchronization Errors

An important consideration in multirobot coordination problems is the robustness against synchronization errors. In practical implementations, robots cannot execute their paths perfectly and might move slower or faster than intended due to various factors such as low battery levels, calibration errors and other failures. These synchronization errors might lead to collisions or deadlocks if not appropriately handled. In this chapter, we discuss how to generate trajectories that can be asynchronously executed. We first show, when the synchronization error is bounded, how to generate paths that collectively avoid collisions and deadlocks while preserving the satisfaction of the desired cLTL+ specification. Furthermore, we show that our formulation is partially complete. However, this formulation requires a priori knowledge of the bound on the synchronization error, and its complexity depends on the upper bound on the synchronization error.

To overcome the shortcomings mentioned above, we propose a hierarchical method where we limit the properties to counting temporal logic plus without 'next' operator (cLTL$+_{\setminus\bigcirc}$). In this hierarchy, a coarse plan that satisfies the logic constraints is computed first at the upper-level, followed by a lower-level task of solving a sequence of generalized multirobot path planning problems. Collision avoidance and potential asynchronous executions are also dealt with at the lower-level. When lower-level planning problems are found to be infeasible, these infeasibility certificates are incorporated into the upper-level problem to re-generate plans. With this hierarchy, we shift the computational burden of avoiding collisions (due to synchronization errors) to the lower-level, where it can be handled much more efficiently. We show that the hierarchical method ensures the satisfaction of the specifications, and its complexity does not depend on the synchronization error bound.

Chapter 5: Multirobot Plan Execution

When robots are allowed to move asynchronously, control strategies must be devised to avoid inter-robot collisions and deadlocks. In Chapter 4, we handled these problems by either assuming a bound on the synchronization errors or setting a fixed priority order between robots. Both of these ideas shift the burden of collision and deadlock avoidance to the offline planning part, and thus, are conservative. In this part of the thesis, we aim to design a distributed online protocol for collision and deadlock avoidance for multirobot systems. Given a collection of paths, we focus on devising a distributed protocol so that the robots are guaranteed to reach their targets and avoid all collisions along the way. We call this the *multirobot plan execution (MRPE)* problem.

Our key contribution is to recast a MRPE problem as an instance of the well-known drinking philosophers problem (DrPP) [15], an extension of the well-known dining philosophers problem [24]. By partitioning the workspace into a set of discrete states and treating each state as a shared resource, we derive conditions on the collection of paths such that the MRPE problem can be solved using any existing DrPP solution, such as [15, 31]. This algorithm enjoys nice properties such as fairness (starvation freeness) and deadlock freeness while also guaranteeing collision avoidance when applied to multirobot setting. However, we show that such a naive approach is conservative. It requires strong conditions on the collection of robot paths to hold and unnecessarily limits the amount of concurrent behavior. To improve the system performance and allow more concurrent behavior, we propose a new algorithm. We show that, when fed by the same paths, our algorithm achieves competitive results with the state-of-the-art [58].

Acknowledgement of Previous Publications

Chapters 2 and 3 are based on [87, 88] and [89]. Relevant publications for Chapter 4 are [86] and [87]. The results in Chapter 5 are currently under review as a journal submission and can be found in [90].

1.2. Preliminaries

Traditional algorithms for multirobot coordination tend to focus on relatively simple tasks such as reaching a goal state while avoiding collisions and unsafe regions [85, 111, 118], formation control [70], reaching a consensus [37, 64]. However, to be able to complete complex tasks such as warehouse logistics or construction, these useful building blocks are not enough. Moreover, such applications require multirobot systems to work in the vicinity of humans, and thus, ensuring correct and safe behavior is of utmost importance.

The field referred to as *formal methods* is a promising candidate to address the shortcomings mentioned earlier. Initially developed by computer scientists to ensure correct behavior of software systems, the use of formal methods to solve control problems attracted attention from the academic community in recent years [9, 71]. Briefly, a system model and a formal specification are given such that one can verify that the system either satisfies or violates the specifications. Alternatively, one can synthesize *correct-by-construction* controllers such that the overall system achieves the specifications, or provide a proof that it is not possible to do so. In this chapter, we first provide a summary of Linear Temporal Logic (LTL), a commonly used specification formalism, in Section 1.2.1. We then provide the syntax and semantics of Constraint LTL (CLTL) in Section 1.2.2, which is used in the complexity analysis of cLTL in Section 2.4. Finally, we provide a summary of transition systems that are used to model the robot dynamics in Section 1.2.3.

1.2.1. Linear Temporal Logic

An LTL formula over a set AP of atomic propositions is defined recursively as follows:

$$\phi ::= True \mid ap \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \bigcirc\phi \mid \phi_1 \,\mathcal{U}\, \phi_2, \tag{1.1}$$

where $ap \in AP$ is an atomic proposition and ϕ, ϕ_1 and ϕ_2 are LTL formulas defined according to (1.1). The symbols $\neg, \wedge, \bigcirc$ and $\mathcal{U}$ correspond to the logical operators *negation* and *conjunction*, and the temporal operators *next* and *until*, respectively. Other commonly used operators can be derived from these operators, such as *disjunction* ($\phi_1 \vee \phi_2 \doteq \neg(\neg\phi_1 \wedge \neg\phi_2)$), *release* ($\phi_1 \,\mathcal{R}\, \phi_2 \doteq \neg\left(\neg\phi_1 \,\mathcal{U}\, \neg\phi_2\right)$), *eventually* ($\Diamond\phi \doteq True \,\mathcal{U}\, \phi$), *always* ($\Box\phi \doteq \neg(\Diamond\neg\phi)$), etc.

The satisfaction of an LTL formula is evaluated over infinite traces. Given a set AP of atomic propositions, a *trace* is an infinite sequence $\sigma = \sigma(0), \sigma(1), \dots \in (2^{AP})^{\omega}$. Given a trace σ and an LTL formula ϕ, satisfaction of ϕ by σ at step t is denoted by $\sigma, t \models \phi$, and is defined as follows:

- $\sigma, t \models True$,
- for any atomic proposition $a \in AP$, $\sigma, t \models a$ if and only if $a \in \sigma(t)$,

- $\sigma, t \models \phi_1 \wedge \phi_2$ if and only if $\sigma, t \models \phi_1$ and $\sigma, t \models \phi_2$,
- $\sigma, t \models \neg\phi$ if and only if $\sigma, t \not\models \phi$,
- $\sigma, t \models \bigcirc\phi$ if and only if $\sigma, t+1 \models \phi$, and
- $\sigma, t \models \phi_1 \,\mathcal{U}\, \phi_2$ if and only if there exists $l \geq 0$ such that $\sigma, t+l \models \phi_2$ and $\sigma, t+l' \models \phi_1$ for all $0 \leq l' < l$.

If $\sigma, 0 \models \phi$, then we say that σ *satisfies* ϕ and write $\sigma \models \phi$ for short. For more information on LTL, we refer the reader to [5].

We now provide definitions for stutter equivalence and stutter invariance that are used in Chapter 4.

Definition 1.1. *A pair σ_1 and σ_2 of traces is said to be* ***stutter equivalent****, if removing consecutive repetition of identical steps makes them identical. For example, $\sigma_1 = (\{a\}, \{a\}, \{a\}, \{b\}, \{a\}, \{c\}, \{c\})^\omega$ and $\sigma_2 = (\{a\}, \{b\}, \{b\}, \{b\}, \{a\}, \{a\}, \{c\})^\omega$ are stutter equivalent, whereas σ_1 and $\sigma_3 = (\{a\}, \{a\}, \{a\}, \{b\}, \{b\}, \{c\}, \{c\})^\omega$ are not.*

Definition 1.2. *An LTL formula is called* ***stutter invariant*** *if its satisfaction does not depend on stuttering.*

The fragment of LTL without the next operator (denoted $\mathrm{LTL}_{\setminus\bigcirc}$) is stutter invariant [5]. That is, given any LTL formula $\phi \in \mathrm{LTL}_{\setminus\bigcirc}$, and stutter equivalent traces σ_1 and σ_2, σ_1 satisfies ϕ if and only if σ_2 satisfies ϕ.

1.2.2. Constraint Linear-Time Temporal Logic

Regular LTL can capture tasks such as *"current value of x is non-negative"*, however, LTL lacks in its expressiveness when it comes to tasks such as *"current value of x is eventually greater than some future value of y"*. Several temporal logics extending the expressiveness of LTL are proposed to overcome this shortcoming. This section introduces one of those temporal logics, namely *Constraint Linear-Time Temporal Logic*, which can be found in [20].

Let Var be a set of variables where each $v \in Var$ takes values from the domain D such that $v(t) \in D$ for each $t \in \mathbb{N}$. A tuple $\mathcal{D} = (D, R_1, \dots, R_m, \mathcal{I})$ is called a *constraint system* where each *$\mathcal{D}$-term constraint* $c_i = R_i(v_1, \dots, v_n)$ is a relation over the elements of D and is associated with a set $\mathcal{I}(R_i) \subseteq D^n$. To exemplify, let $Var = \{x, y\}$ take values from the domain $D = \mathbb{N}$. The symbol $<(x, y)$ is a $\mathcal{D}$-term constraint, and it defines an ordering relation between pairs of values from $\mathbb{N}^2$. Associated $\mathcal{I}(R_i) = \{(0,1), (0,2), \dots\} \subseteq \mathbb{N}^2$ is a countably infinite set, which is a set of all pairs from $\mathbb{N}^2$ where the first element is less than the second element. The syntax of the *constraint linear temporal logic parameterised by the constraint system* $\mathcal{D}$, denoted CLTL($\mathcal{D}$), is defined as follows:

$$\phi ::= c \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \bigcirc\phi \mid \phi_1 \,\mathcal{U}\, \phi_2, \tag{1.2}$$

where c is a *$\mathcal{D}$-term constraint* over the variables Var and the symbols $\neg, \wedge, \bigcirc$ and $\mathcal{U}$ are defined in the same way as regular LTL. Other commonly used operators, such as $\vee$, $\Diamond$ and $\Box$, can be derived from these operators in the usual way.

Let σ denote a particular realization of the values assigned to each variable. We say that σ satisfies c_i at time t, denoted $\sigma, t \models_{\mathcal{D}} c_i$ if $(v_1(t), \ldots, v_n(t)) \in \mathcal{I}(R_i)$. Satisfaction of a CLTL($\mathcal{D}$) formula is defined inductively as follows:

- $\sigma, t \models c$, if $\sigma, t \models_{\mathcal{D}} c$,
- $\sigma, t \models \phi_1 \wedge \phi_2$ if and only if $\sigma, t \models \phi_1$ and $\sigma, t \models \phi_2$,
- $\sigma, t \models \neg\phi$ if and only if $\sigma, t \not\models \phi$,
- $\sigma, t \models \bigcirc\phi$ if and only if $\sigma, t+1 \models \phi$, and
- $\sigma, t \models \phi_1 \,\mathcal{U}\, \phi_2$ if and only if there exists $l \geq 0$ such that $\sigma, t+l \models \phi_2$ and $\sigma, t+l' \models \phi_1$ for all $0 \leq l' < l$.

To exemplify, let $Var = \{x, y\}$ take values from the domain $D = \mathbb{N}$ and let $\mathcal{D}$-term constraint $<$ be defined in the usual way. Let σ denote a particular realization where $x(t) = t$ and $y(t) = 2t$. Then, $\sigma, t \models (x < y)$ for all t. Therefore, $\sigma, t \models \square(x < y)$ as well. On the other hand $\sigma, t \not\models (y < x)$ for any t.

Note that the semantics of CLTL($\mathcal{D}$) are almost identical to that of LTL, with the exception of the satisfaction of $\mathcal{D}$-term constraints. In fact, CLTL($\mathcal{D}$) is a generalization of the regular LTL. That is, LTL is equivalent to CLTL($\mathcal{D}$) for the particular constraint system $\mathcal{D} = (\{0, 1\}, true, \mathcal{I})$ and $\mathcal{I}(true) = 1$. Other commonly used constraint systems are of the form $\mathcal{N} = (\mathbb{N}, <, =)$ and $\mathcal{Z} = (\mathbb{Z}, <, =)$. When the syntax of CLTL($\mathcal{D}$) is extended by allowing constants to be used, to capture tasks such as $(\Diamond x < 5)$, the resulting logic is denoted by CLTL^{con}($\mathcal{D}$). For more information on the Constraint Linear-Time Temporal Logic, we refer the reader to [20].

1.2.3. Transition Systems

We mainly use transition systems to model the dynamics of robots in this thesis.

Definition 1.3. *A **transition system** is a tuple $T = (V, \mathcal{E}, AP, L)$ where V is a finite set of states, $\mathcal{E} \subseteq V \times V$ is a transition relation, AP is a finite set of atomic propositions, and $L : V \to 2^{AP}$ is a labeling function. A transition system is called* deterministic *if all transitions are controllable, i.e., if $(v, v') \in \mathcal{E}$, then there exists a controller that can steer a robot from state $v \in V$ to state $v' \in V$. Otherwise, a transition system is called* non-deterministic.

We say that v *satisfies* a or a *holds at* v if $a \in L(v)$ for $v \in V$ and $a \in AP$.

Transition systems could be obtained using abstraction methods [79, 114] or motion primitives [32, 62, 73]. Such abstract graph-based representations are commonly used for describing the behavior of robotic teams [6, 118]. In Chapter 3, we assume that robot dynamics are modeled by action deterministic transition systems. In Chapter 4 and Chapter 5, we allow a particular type of non-determinism such that $(v, v') \in \mathcal{E}$ guarantees the existence a controller that can steer a robot from state $v \in V$ to state $v' \in V$, but the duration of the transition might take an arbitrary number of finite steps.

Definition 1.4. *Given a transition system $T = (V, \mathcal{E}, AP, L)$, an infinite sequence $\pi : \pi(0), \pi(1), \pi(2), \ldots \in V^{\omega}$ of states such that $(\pi(k), \pi(k+1)) \in \mathcal{E}$ is called a **path** (or **trajectory**, or $T-$**path**). For a given trajectory π, the corresponding **trace** is defined as $\sigma(\pi) = L(\pi(0)), L(\pi(1)), \cdots \in (2^{AP})^{\omega}$.*

We say that a path π satisfies an LTL specification ϕ if the corresponding trace $\sigma(\pi) \models \phi$, and write $\pi \models \phi$. We now define stutter bisimulation equivalence between transition system. Roughly speaking, two transition systems are called *stutter bisimilar* if they have the same branching structure and every step taking by one transition system can be matched by the other, barring repetition. Formally, stutter bisimulation equivalence can be seen as a relation that maps the states and transitions of one transition system to the other.

Definition 1.5. *Transition systems* $T_1 = (V_1, \mathcal{E}_1, AP, L_1)$ *and* $T_2 = (V_2, \mathcal{E}_2, AP, L_2)$ *are said to be* stutter bisimilar *if there exists an equivalence relation* $\sim\ \subseteq \mathcal{E}_1 \times \mathcal{E}_2$ *such that for all* $(v_1, v_2) \in \sim$*:*

- $L_1(v_1) = L_2(v_2)$*,*
- *If* $(v_1, v_1') \in \mathcal{E}_1$*, then there exists a finite* T_2*-path* $v_2, u_1, \ldots, u_n, v_2'$ *for some* $n \geq 0$ *such that* $(v_1, u_i) \in\sim$ *for all* $i \in \{1, \ldots, n\}$ *and* $(v_1', v_2') \in\sim$*,*
- *If* $(v_2, v_2') \in \mathcal{E}_2$*, then there exists a finite* T_2*-path* $v_1, u_1, \ldots, u_n, v_1'$ *for some* $n \geq 0$ *such that* $(u_i, v_2) \in\sim$ *for all* $i \in \{1, \ldots, n\}$ *and* $(v_1', v_2') \in\sim$*.*

By Definition 1.5, if T_1 and T_2 are stutter bisimilar, for every trace $\sigma_1(\pi_1)$ where π_1 is a T_1-path, there exists a T_2-path π_2 such that $\sigma_1(\pi_1)$ and $\sigma_2(\pi_2)$ are stutter equivalent.

Chapter 2.

Counting Logics

The motivation behind counting logics is multirobot planning tasks that are fairly complex, but that have a particular structure that allows scalability. Consider an emergency response scenario, for instance after an earthquake, that requires deployment of hundreds of autonomous (ground and air) vehicles to provide supplies to victims. In such a scenario, the robotic team needs to provide supplies to certain areas, surveil different areas for survivors, and avoid certain regions of danger. The tasks may require *sufficiently many* robots to be in a given region simultaneously to provide the necessary support. Similarly, narrow passageways or the potential to trigger further destruction to damaged structures may require *not too many* robots to be in certain regions at the same time. The role or identity of individual robots are not essential for the satisfaction of such constraints. For instance, as long as the supplies reach their target, it does not matter which subset of the robots provides them. On the other hand, there might be additional requirements on each or some subset of robots. To exemplify, consider the following two tasks: *each robot needs to visit a charging station every now and then* or *at least one robot needs to visit region A, pick up an object and drop it at region C.* Such constraints are called *temporal counting constraints* and can be captured using cLTL+, a novel logic introduced in this thesis.

Capturing temporal counting constraints using LTL or STL, most commonly used temporal logics for multirobot systems, requires a formula whose length grows combinatorially with the number of robots. Therefore, scaling to a large number of robots is challenging. The logic cLTL+ consists of two-layers. The *inner logic* is identical to LTL and is used to describe tasks that can be satisfied by a single robot. For example, tasks such as *"avoid collisions with obstacles at all times"* or *"eventually visit region A"* can be described by the inner logic. The outer layer then specifies the evolution of the number of robots required to satisfy an inner logic formula. For example, we can specify tasks such as *"All robots* must avoid collisions with obstacles" or *"At least five robots* should eventually visit region A" using cLTL+.

Chapter overview

We provide the syntax and semantics of cLTL+ and cLTL in Section 2.1 and Section 2.3, respectively. We also introduce a notion of robust satisfaction of cLTL+ specifications in Section 2.2, and provide complexity results for cLTL in Section 2.4. Finally, we provide conclusions in Section 2.5.

2.1. Counting Linear Temporal Logic Plus (cLTL+)

This section provides the syntax and semantics for a novel two-layered logic called *Counting Linear Temporal Logic Plus (cLTL+)*.

2.1.1. Inner Logic

An inner logic formula over a set AP of atomic propositions is defined recursively as follows:

$$\phi ::= True \mid a \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \bigcirc\phi \mid \phi_1 \,\mathcal{U}\, \phi_2, \tag{2.1}$$

where $a \in AP$ is an atomic proposition and ϕ, ϕ_1 and ϕ_2 are inner logic formulas. We use Φ to denote the set of all inner logic formulas defined according to (2.1). The semantics of inner logic is identical to LTL and can be found in Section 1.2.1.

2.1.2. Outer Logic

After defining the inner logic, we now present the syntax for cLTL+ which is based on a new proposition type: a *temporal counting proposition* (tcp) is an inner logic formula paired with a nonnegative integer, i.e., $tcp = [\phi, m] \in \Phi \times \mathbb{N}$. The inner logic formula ϕ defines a task and m specifies the number of robots needed to satisfy it. For example, $tcp = [\Diamond a, 5]$ is a temporal counting proposition that evaluates to $True$ if the task "$\Diamond a$" is satisfied by at least five robots.

The following grammar can now be used to recursively define cLTL+ formulas:

$$\mu ::= True \mid tcp \mid \neg\mu \mid \mu_1 \wedge \mu_2 \mid \bigcirc\mu \mid \mu_1 \,\mathcal{U}\, \mu_2, \tag{2.2}$$

where $tcp \in \Phi \times \mathbb{N}$ is a temporal counting proposition and μ, μ_1 and μ_2 are cLTL+ formulas. Identical to inner logic, other commonly used operators can be derived from (2.2).

We now provide the semantics with the assumption that robots move synchronously. We later relax this assumption in Section 2.2. Given a collection $\Sigma = \{\sigma_1, \ldots, \sigma_N\}$ of N infinite traces, the satisfaction of a cLTL+ formula μ at time t is denoted by $\Sigma, t \models \mu$ and inductively defined as follows:

$$\begin{aligned}
&\Sigma, t \models [\phi, m] \text{ if and only if } |\{n \mid \sigma_n, t \models_{LTL} \phi\}| \geq m,\\
&\Sigma, t \models \neg\mu \text{ if and only if } \Sigma, t \not\models \mu,\\
&\Sigma, t \models \mu_1 \wedge \mu_2 \text{ if and only if } \Sigma, t \models \mu_1 \text{ and } \Sigma, t \models \mu_2,\\
&\Sigma, t \models \bigcirc\mu \text{ if and only if } \Sigma, t+1 \models \mu\\
&\Sigma, t \models \mu_1 \,\mathcal{U}\, \mu_2 \text{ iff there exists } l \geq 0 \text{ such that } \Sigma, t+l \models \mu_2\\
&\qquad\qquad\qquad\text{and } \Sigma, t+k \models \mu_1 \text{ for all } k < l.
\end{aligned} \tag{2.3}$$

The semantics of the outer logic, with the exception of satisfaction of a *tcp*, is similar to regular LTL. The intuition is that Σ represents the behaviors of N robots, where σ_n corresponds to the behavior of robot $\mathcal{R}_n$. Robot $\mathcal{R}_n$ is said to satisfy the inner logic formula ϕ at time t if $\sigma_n, t \models \phi$. Then $[\phi, m]$ is satisfied at time t if the number of robots satisfying ϕ at time t are greater than or equal to m, i.e., $|\{n \mid \sigma_{\pi_n}, t \models \phi\}| \geq m$.

We say that the collection $\Sigma = \{\sigma_1, \ldots, \sigma_N\}$ of N infinite traces satisfies the cLTL+ μ and denote it by $\Sigma \models \mu$ when $\Sigma, 0 \models \mu$. Given a collection $\Pi = \{\pi_1, \ldots, \pi_N\}$ of N

infinite paths with corresponding traces $\Sigma = \{\sigma(\pi_1), \ldots, \sigma(\pi_N)\}$, we also write $\Pi \models \mu$ if $\Sigma \models \mu$.

Example 2.1. *Assume the following specification for N robots is given in plain English: "Every robot should regularly visit the charging station and the number of robots in region A should be less than 5 until region B is populated by at least 2 robots". Mark region A, region B and the charging station, with atomic propositions a, b and c, respectively. Then the specification is expressed in cLTL+ as $\mu = [\Box\Diamond c, N] \wedge (\neg[a, 5]\, \mathcal{U}\, [b, 2])$.*

The inner logic formula $\Box\Diamond c$ is satisfied by any robot if that robot regularly visits the charging station, marked by c. Then $[\Box\Diamond c, N]$ is satisfied if at least N robots (all robots) regularly visit the charging station. Similarly, $\neg[a, 5]$ (or $[b, 2]$) is satisfied at time t, if less than 5 (or at least 2) robots satisfy proposition a (or proposition b) at that time. Combining all, μ specifies the same task that is given in plain English.

2.1.3. Comparison with Regular LTL

Given a multirobot system with N robots and a cLTL+ formula μ, we now show how to rewrite the same specifications in LTL. Doing so highlights the advantages of using cLTL+ in scenarios where robot identity is not critical for accomplishing the collective task.

Let μ be a cLTL+ formula over the set AP of atomic propositions. We first define a new set of atomic propositions $AP' = \bigcup_{a \in AP}\{a_1, a_2, \ldots a_N\}$ such that there are N new atomic propositions for each atomic proposition $a \in AP$, one for each robot.

Then, for each temporal counting proposition $tcp = [\phi, m]$ in μ, we define a new set $\{\phi_1, \phi_2, \ldots \phi_N\}$ of LTL formulas over AP', where ϕ_n is obtained by replacing every atomic proposition $a \in AP$ with the corresponding $a_n \in AP'$. We then define $tcp' \doteq \bigvee_{i=1}^{I}(\bigwedge_{j \in J_i} \phi_j)$, where $J = \{J_1, \ldots, J_I\}$ is the set of all m-element subsets of $[N]$, hence $I = \binom{N}{m}$. Note that, tcp' is equivalent to tcp, meaning that any collective execution that satisfy one will also satisfy the other.

Note that, as a result of the conversation from cLTL+ to regular LTL, the number of atomic propositions increases linearly and the length of the formula increases combinatorially with the number of robots. Since the complexity of synthesis algorithms depend on the size of the formula [5], such an approach would not scale well with the number of robots.

2.2. Robustness Against Asynchrony

Our definitions so far assume perfect synchronization of robots. However, perfect synchronization of robots is a challenging task and synchronization errors, if not handled with care, might result in violation of the specifications. For instance, assume there are 2 robots and the specification requires a certain property p to be satisfied by at least one robot at all times, i.e., $\mu = \Box[p, 1]$. Let π_i denote a path for robot r_i and

$$\begin{array}{rcl} \sigma(\pi_1) & = & \{p\} \quad \{\neg p\} \quad \{\neg p\} \quad \{\neg p\} \quad \ldots, \\ \sigma(\pi_2) & = & \{\neg p\} \quad \{p\} \quad \{p\} \quad \{p\} \quad \ldots \end{array}$$

be corresponding traces. Property p is satisfied by robot $\mathcal{R}_1$ only at time $t = 0$ and by $\mathcal{R}_2$ at all times except $t = 0$. If robots are perfectly synchronized, the specification μ would be satisfied. However, if $\mathcal{R}_2$ moves slower than intended and causes a synchronization error, μ would be violated.

When robots are allowed to move asynchronously, there are infinitely many ways a collection of infinite paths $\{\pi_1, \ldots, \pi_N\}$ could be executed. To reason about asynchronous executions, we define the following concepts.

Definition 2.1. *A mapping $k : \mathbb{N} \to \mathbb{N}$ is called a* ***local counter*** *if it satisfies the following:*

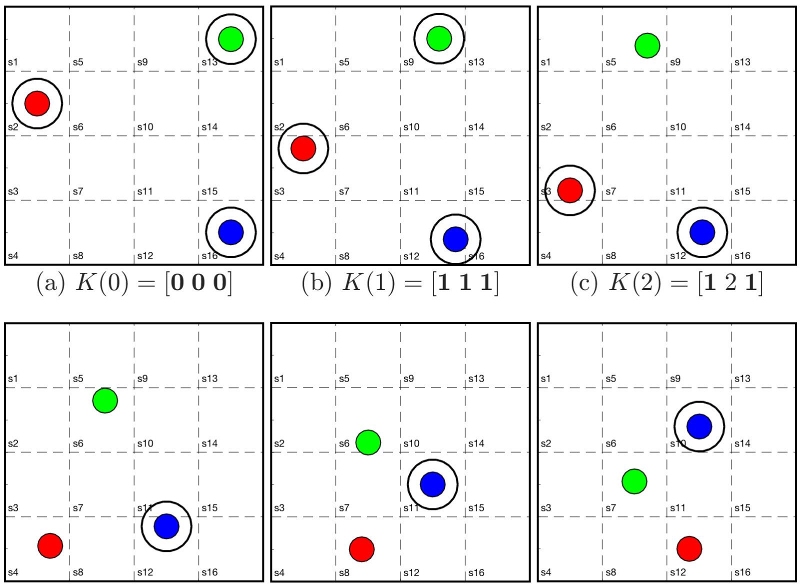
$$k(0) = 0,\ k(t) \leq k(t+1) \leq k(t) + 1,\ \lim_{t\to\infty} k(t) = \infty. \tag{2.4}$$

The set of all local counters is denoted by $\mathcal{K}$.

A local counter is used to keep track of how far single robot has moved along its trajectory. If π_n denotes the trajectory and k_n denotes the local counter of robot $\mathcal{R}_n$, the position of $\mathcal{R}_n$ at time t is given by $\pi_n(k_n(t))$. Equation (2.4) guarantees that initial conditions are respected, the order of states in a trajectory is preserved, and that robots eventually make progress.

Given a collection of trajectories, a collective execution is uniquely identified by a collection of local counters:

Definition 2.2. *An N-dimensional* ***collective execution*** *$K : \mathbb{N} \to \mathbb{N}^N$ is a mapping from global time to local counters, i.e., $K \doteq [k_1 \ldots k_N]$ where $k_n \in \mathcal{K}$ for all $n \in [N]$. The set of all N-dimensional collective executions is denoted by $\mathcal{K}_N$.*



(a) $K(0) = [\mathbf{0}\ \mathbf{0}\ \mathbf{0}]$ (b) $K(1) = [\mathbf{1}\ \mathbf{1}\ \mathbf{1}]$ (c) $K(2) = [\mathbf{1}\ 2\ \mathbf{1}]$

(d) $K(3) = [2\ 3\ \mathbf{1}]$ (e) $K(4) = [3\ 3\ \mathbf{2}]$ (f) $K(5) = [4\ 4\ \mathbf{3}]$

Figure 2.1: Illustrative example for local counters and anchor time. Frames (a) to (e) correspond to snapshots of a possible asynchronous execution taken at times $t = 0$ to $t = 5$. Robots are enumerated in the order of red, green, blue and local times of robots at each time step are shown below the corresponding frame. Anchoring robots are highlighted with a black circle and the anchor time is shown in bold.

The following example illustrates the concept of asynchronous execution and the corresponding local counters.

Example 2.2. *Let the following three trajectories*

$$\begin{aligned}
\pi_1 &= s_2 \quad s_3 \quad s_4 \quad s_8 \quad s_{12} \quad \ldots \\
\pi_2 &= s_{13} \quad s_9 \quad s_5 \quad s_6 \quad s_7 \quad \ldots \\
\pi_3 &= s_{16} \quad s_{12} \quad s_{11} \quad s_{10} \quad s_9 \quad \ldots .
\end{aligned}$$

denote the trajectories of a red, green, and a blue robots, respectively. An arbitrary collective execution is illustrated in Figure 2.1. Local counters are initially set as $K(0) = [0\ 0\ 0]$ *at time* $t = 0$*; that is, each robot* $\mathcal{R}_n$ *is initially positioned at* $\pi_n(0)$.

Every robot completes a transition by time $t = 1$, so local counters are updated as $K(1) = [1\ 1\ 1]$. The red and the blue robots move slower than expected and fail to complete two transitions by time $t = 2$. The green robot, on the other hand, successfully completes two transitions by time $t = 2$. Thus, local counters are updated as $K(2) = [1\ 2\ 1]$. Similarly, the values of the local counters up to $t = 5$ can be seen from Figure 2.1.

Given a collection of N infinite traces $\Sigma = \{\sigma_1, \dots, \sigma_N\}$ and a collective execution $K = [k_1, ..., k_N]$, the pair (Σ, K) is called a *collective trace.* Semantics in (2.3) are stated with the assumption that robots move synchronously. To generalize to asynchronous executions, we replace every Σ of (2.3) with (Σ, K) and modify the first line as follows:

$$(\Sigma, K), t \models [\phi, m] \text{ if and only if } |\{n \mid \sigma_n, k_n(t) \models_{LTL} \phi\}| \geq m. \tag{2.5}$$

As stated before, when robots are allowed to move asynchronously, there are infinitely many collective executions associate with a collection of trajectories. Without a bound on asynchrony, it might be impossible to achieve meaningful tasks. For this reason, we introduce the following definition.

Definition 2.3. *A collective execution $K = [k_1 \dots k_N]$ is called* ***τ-boundedly asynchronous (or, τ-bounded, in short)*** *if*

$$\max_{t \in \mathbb{N}, n, m \in [N]} (|k_n(t) - k_m(t)|) \leq \tau.$$

The set of all τ-bounded N-dimensional collective executions is denoted by $\mathcal{K}_N(\tau)$.

A collective execution $K \in \mathcal{K}_N(0)$ is called a *synchronous* execution. In a synchronous execution, all robots start and complete their transitions simultaneously. The synchronous execution $K^* = [k_1^* \dots k_N^*]$ where $k_n^*(t) = t$ for all n and t is called *globally synchronous*.

To reason about robust satisfaction of a formula, we further need to define the concept of *anchor time* for collective executions.

Definition 2.4. *For a given collective execution $K = [k_1 \dots k_N]$, the* ***anchor time mapping*** *b_K maps the time index t to the smallest local counter value $k_n(t)$, i.e., $b_K(t) = \min_n k_n(t)$.*

For a τ-bounded collective execution $K \in \mathcal{K}_N(\tau)$ and a given time step t, at least one local counter has the value $b_K(t)$ and all other local counters are limited to an interval: $k_n(t) \in [b_K(t), b_K(t) + \tau]$ for all n. For the globally synchronous collective execution K^*, the anchor time mapping is the identity mapping on $\mathbb{N}$. In Figure 2.1, "anchoring robots" at each time step are highlighted with a black circle and anchor times are written in bold.

Having defined the "anchor time", we now formally define the concept of robust satisfaction for a collection of trajectories.

Definition 2.5. *A collection of trajectories $\Sigma = \{\sigma_1, \dots, \sigma_N\}$* ***$\tau$-robustly satisfies*** *$\mu$ at time t, denoted*

$$\Sigma, t \models_\tau \mu, \tag{2.6}$$

if and only if for all $K \in \mathcal{K}_N(\tau)$ and for all $T \in b_K^{-1}(t)$,

$$(\Sigma, K), T \models \mu. \tag{2.7}$$

In other words, a specification μ is τ-robustly satisfied at time t by Σ if every τ-bounded collective execution K of Σ satisfies μ at all time instances T for which the anchor time is t. Consider the set of trajectories $\Pi = \{\pi_1, \pi_2, \pi_3\}$ and an asynchronous collective execution K given in Example 2.2. Let $\Sigma = \{\sigma_1, \sigma_2, \sigma_3\}$ be the corresponding traces for Π. For $\Sigma, 1 \models_\tau \mu$ to hold; we must have $(\Sigma, K), T \models \mu$, for all $T \in \{1, 2, 3\}$ since $b_K^{-1}(1) = \{1, 2, 3\}$. Additionally, the same argument must hold for every possible $K' \in \mathcal{K}_N(\tau)$. If $\Sigma, 0 \models_\tau \mu$, we say that the collection Σ satisfies cLTL+ formula μ and write $\Sigma \models_\tau \mu$ for short. With a slight abuse of notation we also write $\Pi \models_\tau \mu$ when a collection Π of paths is given and corresponding traces Σ are clear from the context.

2.3. Counting Linear Temporal Logic (cLTL)

In this section, we introduce a fragment of cLTL+, namely *counting linear temporal logic* (cLTL). The syntax of cLTL over a set AP of propositions is shown below:

$$\mu ::= True \mid tcp_{cLTL} \mid \neg\mu \mid \mu_1 \wedge \mu_2 \mid \bigcirc\mu \mid \mu_1 \, \mathcal{U} \, \mu_2. \tag{2.8}$$

where temporal counting proposition tcp_{cLTL} are restricted such that the inner logic formulas are atomic propositions, instead of general LTL formulas. Temporal counting propositions of this special form $tcp_{cLTL} = [a, m] \in AP \times \mathbb{N}$ are called *counting constraints.*

As a result of the restriction on the inner logic formulas, cLTL enforces robots to "synchronize". Furthermore, counting constraints render the overall control problem *permutation invariant.* This structural property was first exploited in [67] for coordination of large collections of systems in the context of scheduling thermostatically controlled loads with time-invariant counting constraints on system modes.

The following example depicts the differences between cLTL and cLTL+ formulas:

Example 2.3. *Consider the following cLTL+ formulas:* $\mu_1 \doteq \Box\Diamond[a, m]$, $\mu_2 \doteq [\Box\Diamond a, m]$, *and* $\mu_3 \doteq \Box[\Diamond a, m]$ *for* $a \in AP$.

Here temporal counting constraint of μ_1 *is* $[a, m]$ *where* a *is an atomic proposition. Hence,* μ_1 *classifies also as a cLTL formula. Task "a" can be satisfied by any robot, simply by visiting a state where* a *holds. The temporal counting proposition "*$[a, m]$*" is satisfied at time* t *if at least* m *robots to satisfy* a *at time* t*. Moreover, the temporal operators "*$\Box\Diamond$*" in the outer layer necessitate that the temporal counting proposition is satisfied infinitely many times. Thus, there should be an infinite number of instances where* a *is* simultaneously *satisfied by more than* m *robots in order for* μ_1 *to be satisfied.*

On the other hand, neither μ_2 *nor* μ_3 *can be specified in cLTL. In both formulas, the inner formula contains temporal operators which are not allowed in the cLTL syntax. The difference between* μ_1 *and* μ_2 *is that the latter relaxes the simultaneity requirement. The inner formula* $\Box\Diamond a$ *can be satisfied by any robot if the robot sat-*

isfies a infinitely many times. The integer m is the smallest number of robots that needs to satisfy the inner formula. Hence, the cLTL+ formula μ_2 requires at least m robots to satisfy a infinitely many times, but as opposed to μ_1 they need not do so simultaneously. For any given time the number of robots that satisfy a might never exceed m, or even 1. Note that any collective trajectory that satisfies μ_1 also satisfies μ_2, but the converse is not true.

The difference between μ_2 and μ_3 is more subtle. Any collective trajectory that satisfies μ_2 would also satisfy μ_3. The converse is also true if the number of robots is finite. However, in the hypothetical scenario where there are infinitely many robots, μ_3 can be satisfied even if no robot satisfies a more than once. ■

2.4. Complexity Analysis for cLTL

In this section, we examine the complexity of the cLTL satisfiability problem. Complexity results are important as they give an estimate on how hard the problem we are interested in is. We now formally define the cLTL satisfiability problem:

Problem 2.1. *Given a cLTL specification μ over atomic proposition set $AP = \{a_1, a_2, \ldots, a_k\}$, does there exist a collection $\Sigma = \{\sigma_1, \ldots, \sigma_N\}$ of traces for some $N > 0$ such that $\Sigma \models \mu$?*

The complexity of Problem 2.1 gives us an estimate of how hard the problem we are interested in is. The following Theorem shows that cLTL and LTL are in the same complexity class. This is an encouraging result since it shows that cLTL provides benefits over LTL without paying penalties.

Theorem 2.1. *The cLTL satisfiability problem is PSPACE-complete.*

Proof. Firstly, we provide an upper bound by showing that satisfiability of cLTL specifications can be reduced to satisfiability of $\text{CLTL}^{con}(\mathbb{N}, <, =)$, which is shown to be solvable in PSPACE in [21]. Let μ be a cLTL formula over atomic proposition set AP. Without loss of generality, we can express μ in PNF. To transform μ into PNF, treat each counting proposition as an atomic proposition and use standard techniques used to transform LTL formulas as in [5]. Then replace each counting proposition $tcp_{cLTL} = [a_i, m]$ with negated atomic constraint $\neg(x_i < m)$. Then, take the conjuction of the resulting formula with $\bigwedge_{x_i}(x_i < N+1)$ and denote it by μ'. Solve the resulting $\text{CLTL}^{con}(\mathbb{N}, <, =)$ satisfiability problem for μ'. This conservation can be done by introducing $2|AP|$ variables and same number of constraints as counting propositions.

Once a solution is found for the x_i values, one can extract a collection $\Sigma = \{\sigma_1, \ldots, \sigma_N\}$ as follows. For each t and for each x_i, choose the first $x_i(t)$ traces. Let σ_n be one of those traces. Adjust σ_n such that $a_i \in \sigma_n(t)$ and $a_i \notin \sigma_m(t)$ for all $m \neq n$. Consequently $x_i(t) \leq m$ implies $|\{n \mid \sigma_n, t \models_{LTL} \mu\}| \leq m$, and $x_i(t) > m$ implies $|\{n \mid \sigma_n, t \models_{LTL} \mu\}| > m$. The rest of the semantics in Equation (1.2) and Equation (2.3) are identical. Therefore, if a solution exists for μ', collection Σ obtained by this approach satisfies the cLTL formula μ, i.e., $\Sigma \models \mu$.

To provide a lower bound, we can reduce satisfiability of LTL into satisfiability of cLTL by replacing every atomic proposition a_i of a given LTL formula with $[a_i > 1]$. Since satisfiability of LTL is PSPACE-complete. □

2.5. Summary

This section introduced temporal counting constraints for multirobot systems. Such constraints are encountered when the number of robots satisfying a particular property needs to be bounded either from above or below. Existing temporal logics cannot capture counting constraints efficiently, requiring formulas whose length grows combinatorially with the number of robots as shown in Section 2.1. To overcome this problem, we introduced a new formalism, namely counting linear temporal logic plus (cLTL+).

We then introduced a notion of robust satisfaction of cLTL+ formulas. Since it might not always be possible to synchronize robots perfectly, solutions should be robust to synchronization errors. Robust satisfaction definition is useful in analyzing the effects of synchronization errors on the satisfaction of the specifications.

We also introduced a fragment of cLTL+, called cLTL. The logic cLTL can be seen as an extension of a particular class of counting problems that deal with invariant specifications. Solutions to cLTL specifications are permutation invariant. That is, the identity of robots is not important, and trajectories of any two robots can be swapped without affecting the satisfaction of the specifications. This property is later exploited in Chapter 3, to provide an optimization-based solution, which scales to systems with hundreds of robots when robots have identical dynamics. We also showed that cLTL belongs to the same complexity class as LTL and is PSPACE-complete.

Chapter 3.

Path Planning with Counting Constraints

This chapter provides the formal definition of the synchronous multirobot coordination problem with cLTL+ constraints and provides an optimization-based solution. Subsequently, an alternative solution is proposed for the special case where the specifications are given in cLTL and the robots have identical dynamics. The alternative solution is shown to scale much better with the number of robots. In fact, the number of robots has almost no effect on the solution time and problems with hundreds of robots can be solved with the alternative method as demonstrated in Section 3.7.

3.1. Synchronous coordination problem

In this short section, we formally state the problem we are interested in solving.

Problem 3.1. *Given N robots with dynamics $\{T_n = (V_n, \mathcal{E}_n, AP, L_n)\}$, initial conditions $\{\pi_n(0)\}$, and a cLTL+ formula μ over AP, synthesize a collection $\Pi = \{\pi_n\}$ such that the globally synchronous execution of Π satisfies μ, i.e., $(\Pi, K^*) \models \mu$.*

In order to solve Problem 3.1, we generate individual trajectories in a centralized fashion. Robots then follow these trajectories in a distributed fashion, using local controllers without runtime communication. As the wording of the Problem 3.1 suggests, we assume that robots will execute their paths synchronously.

3.2. Path Planning with cLTL+ Specifications

To generate trajectories we encode the robot dynamics and the cLTL+ constraints using integer linear constraints and pose the synthesis problem as an integer linear program (ILP). This approach is inspired by the bounded model-checking literature [11]. In particular, we focus the search on individual trajectories in prefix-suffix form. That is, for a given integer h, we aim to construct individual trajectories of the form $\pi_n = \pi_n(0)\pi_n(1)\ldots\pi_n(h)\ldots$ and find an integer $l \in \{0, \ldots, h-1\}$ such that for all $k \geq h$, $\pi_n(k) = \pi_n(k+l-h)$. We later show in Theorem 3.2 that this assumption that the trajectories are in prefix-suffix form is without loss of generality. In the following, we present ILP encodings of dynamic and temporal constraints.

3.2.1. Globally synchronous robot dynamics

Given the transition system $T_n = (V_n, \mathcal{E}_n, AP, L_n)$ that represents the dynamics of robot $\mathcal{R}_n$, consider the matrix A_n defined as $A_n^{j,i} = 1$ if and only if $(v^i, v^j) \in \mathcal{E}_n$

where $A_n^{j,i}$ is the term in the j^{th} row and i^{th} column of A_n. We use a Boolean vector $w_n(t) \in \{0,1\}^{|V_n|}$ with a single nonzero component to denote the state of robot $\mathcal{R}_n$ at time t. For example, if $V_n = \{v^1, v^2, v^3\}$ and that robot $\mathcal{R}_n$ is at v^2 at time t, then $w_n(t) = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}^T$. With a slight abuse of notation, we equivalently write $w_n(t) = v^2$.

Given matrices $\{A_n\}$ corresponding to $\{T_n\}$, and a set of inital conditions $\{\pi_n(0)\}$, the dynamics of robot $\mathcal{R}_n$ are captured as follows:

$$w_n(t+1) \leq A_n w_n(t), \quad \mathbf{1}^T w_n(t) = 1, \quad w_n(0) = \pi_n(0), \tag{3.1}$$

for all $n \in [N]$ and for all $t \in \{0, \dots, h-1\}$. The first inequality in (3.1) ensures that robots respect the transitions $\mathcal{E}_n$, the second term guarantees conservation of the number of robots, and the last term results from the initial condition. The trajectory π_n corresponding to the sequence $\mathbf{w}_n = w_n(0)w_n(1)\dots$ can then be extracted by locating the nonzero component in each $w_n(t)$.

3.2.2. Loop constraints

To ensure that the generated trajectories are in prefix-suffix form, we introduce h binary variables $\mathbf{z_{loop}} = \{z_{loop}(0), \dots z_{loop}(h-1)\}$ and the following constraints:

$$\begin{aligned} w_n(h) &\leq w_n(t) + \mathbf{1}(1 - z_{loop}(t)), \\ w_n(h) &\geq w_n(t) - \mathbf{1}(1 - z_{loop}(t)), \\ \sum_{t=0}^{h-1} z_{loop}(t) &= 1, \end{aligned} \tag{3.2}$$

for all $n \in [N]$ and for all $t \in \{0, \dots, h-1\}$. When these constraints are satisfied, there exists a unique $l < h$ such that $z_{loop}(l) = 1$ and $w_n(h) = w_n(l)$. For all other time instances $t \neq l$, the first two inequalities are trivially satisfied.

3.2.3. Inner logic constraints

We next recursively describe how temporal counting logic constraints can be translated into integer constraints. Let $\phi \in \Phi$ be an inner logic formula given according to (2.1) and let h be the horizon length. For each robot $\mathcal{R}_n$, we introduce h binary decision variables $z_n^{\phi}(t) \in \{0, 1\}$ for $t \in \{0, 1, \dots, h-1\}$, and ILP constraints such that $z_n^{\phi}(t) = 1$ if and only if $\pi_n, t \models \phi$. Hence, satisfaction of an inner formula ϕ by $\mathcal{R}_n$ is equivalent to $z_n^{\phi}(0) = 1$. We use the following encodings to recursively create the corresponding ILP constraints:

a (atomic proposition): Let $\phi \in AP$ be an atomic proposition and let the states of T_n be given by the set $V_n = \{v_n^1, v_n^2, \dots, v_n^{|V_n|}\}$. We define the vector $\mathbf{v}_n^{\phi} \in \{0, 1\}^{|V_n|}$

such that the i^{th} entry of $\mathbf{v}_n^\phi$ is 1 if and only if $\phi \in L(v_n^i)$. That is, $\mathbf{v}_n^\phi$ encodes the labeling function L_n. Then we introduce the following constraints for all $n \in [N]$:

$$\begin{aligned} &(\mathbf{v}_n^\phi)^T w_n(t) \geq z_n^\phi(t), \\ &(\mathbf{v}_n^\phi)^T w_n(t) < z_n^\phi(t) + 1. \end{aligned} \tag{3.3}$$

When $\phi \in L(w_n(t))$, we have $(\mathbf{v}_n^\phi)^T w_n(t) = 1$ and $z_n^\phi(t) = 1$ must hold due to the second inequality in (3.3). Conversely, if $\phi \notin L(v)$, the first inequality requires that $z_n^\phi(t) = 0$. The following encodings of Boolean and temporal operators are consistent with those in [11]:

$\neg$ *(negation):* Let $\varphi = \neg\phi$. Then for all $n \in [N]$,

$$z_n^\varphi(t) = 1 - z_n^\phi(t), \qquad t = 0, \ldots, h-1. \tag{3.4}$$

$\wedge$ *(conjunction):* Let $\phi = \bigwedge_{i=1}^{I} \phi_i$. Then for all $t = 0, \ldots, h-1$ and for all $n \in [N]$,

$$\begin{aligned} &z_n^\phi(t) \leq z_n^{\phi_i}(t), \qquad \text{for } i = 1, \ldots, I \quad \text{and,} \\ &z_n^\phi(t) \geq 1 - I + \sum_{i=1}^{I} z_n^{\phi_i}(t). \end{aligned} \tag{3.5}$$

$\vee$ *(disjunction):* Let $\phi = \bigvee_{i=1}^{I} \phi_i$. Then for all $t = 0, \ldots, h-1$ and for all $n \in [N]$,

$$\begin{aligned} &z_n^\phi(t) \geq z_n^{\phi_i}(t), \qquad \text{for } i = 1, \ldots, I \quad \text{and,} \\ &z_n^\phi(t) \leq \sum_{i=1}^{I} z_n^{\phi_i}(t). \end{aligned} \tag{3.6}$$

With a slight abuse of notation, when $\phi = \bigvee_{i=1}^{I} \phi_i$, we write $z_n^{\phi}(t) = \bigvee_{i=1}^{I} z_n^{\phi_i}(t)$ instead of stating the inequalities in (3.6). Encoding of the temporal operators is then as follows:

$\bigcirc$ *(next):* Let $\varphi = \bigcirc\phi$, then for all $n \in [N]$

$$\begin{aligned} z_n^{\varphi}(t) &= z_n^{\phi}(t+1), \qquad t = 0, \dots, h-2 \text{ and}, \\ z_n^{\varphi}(h-1) &= \text{ for } \bigvee_{t=0}^{h-1} (z_n^{\phi}(t) \wedge z_{loop}(t)). \end{aligned} \tag{3.7}$$

$\mathcal{U}$ *(until):* if $\phi = \phi_1 \,\mathcal{U}\, \phi_2$, then for all $n \in [N]$

$$\begin{aligned} z_n^{\phi}(t) &= z_n^{\phi_2}(t) \vee \left(z_n^{\phi_1}(t) \wedge z_n^{\phi}(t+1)\right), \quad \text{for all } t \le h-2, \\ z_n^{\phi}(h-1) &= z_n^{\phi_2}(h-1) \vee \left(z_n^{\phi_1}(h-1) \wedge \left(\bigvee_{t=0}^{h-1} \left(z_{loop}(t) \wedge \tilde{z}_n^{\phi}(t)\right) \right) \right), \\ \tilde{z}_n^{\phi}(t) &= z_n^{\phi_2}(t) \vee \left(z_t^{\phi_1,n} \wedge \tilde{z}_n^{\phi}(t+1) \right), \quad \text{for all } t \le h-2, \\ \tilde{z}_n^{\phi}(h-1) &= z_n^{\phi_2}(h-1), \end{aligned} \tag{3.8}$$

where $\tilde{z}_n^{\phi}(t)$ are auxiliary binary variables. As shown in [11], not introducing auxiliary variables results in trivial satisfaction of the *until* operator.

3.2.4. Outer logic constraints

Similar to the inner logic, we proceed by transforming a cLTL+ formula into ILP constraints. Given a cLTL+ formula μ and a time horizon h, we create h binary decision variables $\mathbf{y}^{\mathbf{cLTL+}} = \{y^{\mu}(t)\}$, where $t \in \{0, 1, \dots, h-1\}$ and ILP constraints $ILP(\mu)$. While doing so, we ensure that $y^{\mu}(t) = 1$ if and only if $(\Pi, K^*), t \models \mu$ where

K^* is the globally synchronous collective execution. We remind the reader that since ILP constraints are created recursively, creating the constraints for formula μ requires first creating constraints for all inner logic formulas that appear in μ. We denote by $ILP(\mu)$ the set of all resulting constraints that encode the satisfaction of μ, and by $(\mathbf{z},\mathbf{y})^{\mathbf{cLTL+}}$ the set of all variables created in this process.

We provide encodings only for counting propositions since the rest of the semantics are identical. Let $\mu = [\phi, m] \in AP \times \mathbb{N}$ be a temporal counting proposition. Then

$$m > \sum_{n=1}^{N} z_n^{\phi}(t) - My^{\mu}(t) \geq m - M, \tag{3.9}$$

where M is a sufficiently large positive number, in particular, $M \geq N + 1$. Note that when $y^{\mu}(t) = 1$, the inequality on the right reduces to $\sum_{n=1}^{N} z_n^{\phi}(t) \geq m$. Moreover, the inequality on the left is trivially satisfied since $M \geq N + 1$. Conversely, when $y^{\mu}(t) = 0$, the inequality on the right is trivially satisfied and the inequality on the left reduces to $\sum_{n=1}^{N} z_n^{\phi}(t) < m$. Therefore, $y^{\mu}(t) = 1$ if and only if the number of robots that satisfy ϕ at time t is greater than or equal to m. Conversely, $(y^{\mu}(t) = 0)$ if and only if the number of robots that satisfy ϕ at time t is less than m. Therefore, the ILP constraints in (3.9) are correct and consistent with the semantics of cLTL+.

3.2.5. Overall optimization problem and its analysis

The following optimization problem is formed to generate a solution to an instance of Problem 3.1 given a horizon length h:

$$\begin{aligned} \text{Find} \quad & \{\mathbf{w}_n\}, \mathbf{z}^{\mathbf{loop}}, (\mathbf{z}, \mathbf{y})^{\mathbf{cLTL+}} \\ \text{s.t.} \quad & (3.1), (3.2), ILP(\mu) \text{ and } y^{\mu}(0) = 1. \end{aligned} \tag{3.10}$$

Next we analyze this solution approach. The following theorem shows that the solutions generated by (3.10) are sound.

Theorem 3.1. *If the optimization problem in* (3.10) *is feasible for a cLTL+ formula* μ*, then a collection* $\Pi = \{\pi_n\}_{n\in[N]}$ *of trajectories can be extracted from* $\{\mathbf{w}_n\}$ *such that* $(\Pi, K^*) \models \mu$*.*

Proof. Constraint (3.1) guarantees that the collection Π of trajectories generated from $\{\mathbf{w}_n\}$ are feasible, consistent with the initial conditions and with the system dynamics. Furthermore, (3.2) ensures that these solutions can be extended to infinite trajectories of the form $\pi_n = \pi_n(0)\ldots\pi_n(l-1)\,(\pi_n(l)\ldots\pi_n(h-1))^{\omega}$. The ILP encodings (3.3)-(3.8) of LTL formulas are sound [11], and the same encodings are also used for cLTL+ formulas by replacing $z_n^{\phi}(t)$ with $y^{\mu}(t)$, where μ is any cLTL+ formula. The only exception is that (3.3) is replaced with (3.9), which we showed to be correct. Therefore, the constraint $y^{\mu}(0) = 1$ together with $ILP(\mu)$ guarantees that $(\Pi, K^*) \models \mu$. Thus, if (3.10) is feasible, then the globally synchronous execution of Π solves Problem 3.1. □

As a corollary, it is easy to show that stutter invariance of formulas (see Definition 1.2) allows the generalization of the soundness result from globally synchronous executions to all synchronous executions.

Corollary 3.1. *If μ does not contain any next operator $\bigcirc$, neither in the inner nor in the outer logic, then $(\Pi, K) \models \mu$ for all synchronous executions $K \in \mathcal{K}_N(0)$.*

The following theorem shows that encodings presented in (3.1)-(3.10) are complete:

Theorem 3.2. *If there is a solution to Problem 3.1, then there exists a finite h such that* (3.10) *is feasible.*

Proof. In order to show that prefix-suffix form solutions are complete, we reduce Problem 3.1 to a regular LTL control synthesis problem, for which prefix-suffix solutions have been shown to be complete [5].

Section 2.1 shows that any cLTL+ formula μ over the set AP of atomic propositions can be rewritten as an equivalent LTL formula μ' over a new set AP'. Next, we create a product transition system $T' \doteq \Pi_n T_n$ with the set AP' as its atomic propositions. Now Problem 3.1 is reduced to a standard LTL synthesis problem and it can be solved using a model-checker to generate a prefix-suffix solution or to declare the non-existence of solutions (see e.g., [10]). □

The proof of Theorem 3.2 highlights the advantages of using cLTL+ in scenarios where robot identity is not critical for accomplishing the collective task. Although the problem can be reduced to a standard LTL synthesis problem as the proof suggests, the reduction results in a synthesis problem on a product transition system with size exponential in the number of robots, and with an LTL formula that is combinatorially longer than the cLTL+ formula. Indeed, without a convenient logic, just writing down that LTL formula would be a tedious and error-prone task.

Remark 3.1. *The complexity of solving ILPs is known to be NP-complete, yet there are efficient heuristics and corresponding software packages (see, e.g., [33]) that reliably solve relatively large instances with ease. An instance of* (3.10) *has $\mathcal{O}(hN(|V_n|+|\mu|))$ decision variables and constraints, where h is the solution horizon, N is the number of robots, $|V_n|$ is the number of states of the largest transition system, and $|\mu|$ is the length of the cLTL+ formula μ. Enforcing collision avoidance, as shown in Section 3.4, introduces $\mathcal{O}(hN^2|V_n|)$ additional constraints.*

Remark 3.2. *For practical implementations, we want solutions to satisfy the specifications even if ϵ number of robots fail unexpectedly during run-time. This type of robustness can easily be incorporated into the optimization formulation by modifying* (3.9) *as follows:*

$$m + \epsilon > \sum_{n=1}^{N} z_n^{\phi}(t) - My^{\mu}(t) \geq m - \epsilon - M. \tag{3.11}$$

In order to find the "most robust" *solution possible, the feasibility problem in Equation* (3.10)*, where Equation* (3.11) *is used instead of Equation* (3.9)*, can be posed as an optimization problem with ϵ as a variable to be maximized.*

Note that, solutions generated using Equation (3.11) would be robust to unexpected failures if the failing robots do not block others from progressing. This assumption is reasonable for a team of drones, but it might be limiting for team of ground robots.

3.3. Path Planning with cLTL Specifications

Given an instance of Problem 3.1, if the specification μ can be expressed in cLTL and all robots have identical dynamics, the overall coordination problem becomes permutation invariant. Then, we can define *aggregate dynamics* and generate an aggregate solution instead of individual trajectories. This structural property was first exploited in [67] for coordination of large collections of systems in the context of scheduling thermostatically controlled loads with time-invariant counting constraints on system modes. This aggregate solution can be found using more efficient encodings and then mapped to individual trajectories.

In the following, we first define a class of cLTL problems and provide the corresponding efficient encodings:

Problem 3.2. *Given N robots with identical dynamics $T = (V, \mathcal{E}, AP, L)$, initial conditions $\{\pi_n(0)\}$, and a cLTL formula μ over AP, synthesize a collection $\Pi = \{\pi_1, \dots, \pi_N\}$ of trajectories such that the globally synchronous collective execution of Π satisfies μ, i.e., $(\Pi, K^*) \models \mu$.*

When solving Problem 3.2, instead of finding a trajectory for each robot, we compute a collective behavior by deciding how many robots that should move between each pair of states at each time step. To further clarify, let the set V of states be enumerated such that $V = \{v^1, v^2, \dots, v^{|V|}\}$. We define an *aggregate state* vector $\mathbf{w} = [w^1, w^2, \dots w^{|V|}]^T$ where the i^{th} row of $\mathbf{w}$ denotes the number of robots at state v^i. Similarly, the *aggregate input* is defined as a vector $\mathbf{u} = [u_1^1, u_1^2, \dots, u_1^{|V|}, u_2^1, \dots u_2^{|V|}, \dots u_{|V|}^{|V|}]^T$ where u_i^j denotes the number of robots that transition from state v^i to v^j. Note that the aggregate input is state-dependent since the total number of robots sent from a particular state to others cannot be greater than the number of robots in that state. Furthermore, the number of robots sent from a state can only be a non-negative integer. An input satisfying these conditions is called *admissible* and $\Upsilon(\mathbf{w})$ denotes the set of all admissible inputs for a given state $\mathbf{w}$. The set $\Upsilon(\mathbf{w})$ is captured by the following set of equalities:

$$\Upsilon(\mathbf{w}) = \left\{ \{u_i^j\} : \quad \sum_{j=1}^{|S|} u_i^j = w^i, \quad u_i^j = 0 \text{ if } (v^i, v^j) \not\in \rightarrow, \quad u_i^j \in \mathbb{N} \right\}. \tag{3.12}$$

The evolution of the aggregate state is described by the following linear constraints:

$$\mathbf{w}(t+1) = B\mathbf{u}(t), \quad \mathbf{u}(t) \in \Upsilon(\mathbf{w}(t)), \tag{3.13}$$

where B is defined as $B \doteq I_{|V|} \otimes \mathbf{1}_{|V|}^T$, $I_{|V|}$ is the identity matrix of size $|V|$, and $\otimes$ is the Kronecker product.

Compared to (3.1) where the state evolution of each robot is computed individually, in (3.12) and (3.13), the decision variables are aggregate inputs, representing how many robots that move along each edge. The identity of robots is not important since cLTL specifications are permutation invariant. As a result, the number of decision variables is independent of the number of robots, which makes cLTL encodings scale well with the number of robots.

Loop constraints for aggregate states can be written as

$$\begin{aligned} \mathbf{w}(h) \leq & \ \mathbf{w}(t) + \mathbf{1}(1 - z_t^{loop}), \\ \mathbf{w}(h) \geq & \ \mathbf{w}(t) - \mathbf{1}(1 - z_t^{loop}). \end{aligned} \tag{3.14}$$

Inner logic constraints for individual robots are no longer needed since the cLTL inner logic is constrained to the grammar $\phi \in AP$. In the outer logic, only the encoding of temporal counting propositions in (3.9) needs modification. Let $\mu = [\phi, m]$ be a tcp_{cLTL} and let $V = \{v^1, \ldots, v^{|V|}\}$ be the set of states. As in (3.3) we introduce a vector $\mathbf{v}^\phi \in \{0,1\}^{|S|}$ such that the i^{th} entry of $\mathbf{v}^\phi$ is 1 if and only if $\phi \in L(v^i)$. Then, for all $t = 0, \ldots, h$, the constraints

$$\begin{aligned} &\mathbf{v}^\phi \mathbf{w}(t) \geq m - M(1 - y_t^\mu), \\ &\mathbf{v}^\phi \mathbf{w}(t) \leq m + M y_t^\mu, \end{aligned} \tag{3.15}$$

ensure that $y^\mu(t) = 1$ if and only if the number of robots that satisfy $\phi \in AP$ is greater than or equal to m. The rest of the outer logic encodings are not modified and used as before.

Given a time horizon h, the following optimization problem is formed to generate solutions to an instance of Problem 3.2:

$$\begin{aligned} \text{Find} \quad & \mathbf{u}(0), \ldots, \mathbf{u}(h-1), \mathbf{z^{loop}}, \mathbf{y^{cLTL}}, \\ \text{s.t.} \quad & (3.12), (3.13), (3.14), ILP(\mu) \text{ and } y^\mu(0) = 1 \end{aligned} \tag{3.16}$$

where $ILP(\mu)$ is the set of all resulting constraints that encode the satisfaction of μ, and $\mathbf{y}^{\mathbf{cLTL}}$ the set of all variables created by Equation (3.15).

We now show how a solution of (3.16) can be mapped to a collection $\{\pi_n\}$ of individual trajectories. Given initial conditions $\pi_n(0)$, and $\mathbf{u}(0)$, randomly choose u_i^j robots from state v^i and assign their next state as v^j. This is always possible since $\mathbf{w}(0)$ is well defined and $\mathbf{u}(0) \in \Upsilon(\mathbf{w}(0))$. Continuing in this manner, we can generate the collection $\{\pi_n\}$ whose globally synchronous collective execution satisfies the specification μ. Details of a similar construction of individual trajectories can be found in [68].

Before proceeding to the asynchronous problem, we remind the reader of two important things: (i) the ILP constraints in (3.16) are consistent with cLTL+ semantics, therefore soundness and completeness guarantees follow from Theorems 3.1 and 3.2. (ii) An instance of (3.16) has $\mathcal{O}(h(|\mathcal{E}|+|\mu|))$ decision variables and constraints where $|\mathcal{E}|$ is the number of transitions and $|\mu|$ is the length of the formula. Crucially, the number of decision variables and constraints does not depend on the number of robots. Therefore, it easily scales to very large numbers of robots as demonstrated in Section 3.7.

3.4. Collision Avoidance

We say that two robots are in collision if they occupy the same state at the same time, or if they swap their positions between two consecutive time steps. Existing approaches for collision avoidance include forcing robots to wait and resume when necessary [58, 99, 122] or deviating minimally from the nominal trajectories [4, 7, 36, 91, 121]. Although such methods would guarantee collision avoidance and progress through the synthesized paths, they might lead to violation of other specifications expressed in cLTL+. Therefore, we propose to directly incorporate collision avoidance into the trajectory synthesis.

Expressing collision avoidance as a cLTL+ specification is also not practical, in general. To clarify, consider a cLTL+ specification that limits the number of robots in each discrete state to at most 1, which can be written by introducing atomic propositions for each discrete state. Such a specification would enforce collision avoidance at discrete time instances, however, collisions can still occur if two robots swap their position. One can also try being more conservative by defining atomic propositions for sets of states and limiting the number of robots in each set to be at most 1, but this will require many atomic propositions in addition to being conservative. To avoid such issues, we handle collisions outside the logical specification by introducing additional constraints to the optimization problem.

For synchronous executions, we introduce the following constraints:

$$\begin{gathered} w_m(t) + w_n(t) \leq \mathbf{1}, \\ (w_m(t+1) + w_n(t) \leq \mathbf{1}) \vee (w_m(t) + w_n(t+1) \leq \mathbf{1}), \end{gathered} \tag{3.17}$$

for all $0 \leq t < h$ and for all n, m pairs. The first inequality in Equation (3.17) ensures that no robots occupy the same state at the same time step, and the second line of inequalities prevents robots from “swapping" states. That is, if robots $\mathcal{R}_m$ and $\mathcal{R}_n$ are at states v_m and v_n, respectively, they cannot swap their positions in the next time step.

On the other hand, when specifications are given in cLTL and all robots have identical dynamics, collision avoidance can effectively be encoded as an additional cLTL formula. In this case, one can define a new atomic proposition for each state and limit the number of robots to at most 1. As in the cLTL+ case, this requirement prevents collisions in discrete time steps. To avoid inter-sample collisions, the individual trajectories extracted from the aggregate solution are modified as follows. Assume two robots swap their positions at some point in time, i.e., $\pi_m(t) = \pi_n(t+1)$ and $\pi_m(t+1) = \pi_n(t)$. Instead of their positions, we swap their paths. That is, both paths are cut into two pieces at time t and $\mathcal{R}_m$ is assigned the path $\tilde{\pi}_m : \pi_m(0), \pi_m(1), \ldots, \pi_m(t), \pi_n(t+1), \pi_n(t+2), \ldots$ and vice versa. This modification prevents robots from swapping their positions. Furthermore, specifications are still satisfied since cLTL is permutation invariant with respect to robot identity.

3.5. Extension to Continuous-State Dynamics

Up to now, we assumed that robot dynamics are modeled by discrete transition systems. Given continuous dynamics, discrete abstraction techniques can be leveraged to obtain transition systems. However, abstraction computations are costly and do not scale well with the number of dimensions. This section provides slight modifications to the earlier encodings that allow direct treatment of continuous-state discrete-time dynamics. Assume that the robot dynamics are given as

$$w_n(t+1) = f_n(w_n(t), u_n(t)), \tag{3.18}$$

where $w_n(t) \in \mathbb{R}^{d_w}$ and $u_n(t) \in \mathbb{R}^{d_u}$ denote the state and input of robot n at time t, respectively.

The first modification is to replace the constraints in (3.1) with (3.18) for all $n \in [N]$ and for all t. The loop constraints in (3.2) are then modified as follows:

$$\begin{aligned} w_n(h) \leq& w_n(t) + M(1 - z_{loop}(t)), \\ w_n(h) \geq& w_n(t) - M(1 - z_{loop}(t)), \end{aligned} \tag{3.19}$$

where M is a sufficiently large number. Equation (3.19) creates a loop by forcing $w_n(h)$ to be equal to $w_n(t)$ for some t.

Remark 3.3. *Achieving a perfect loop closure as in* (3.19) *is not realistic in the presence of modeling errors and disturbances. Instead, these trajectories should be seen as waypoints to be tracked by a feedback controller. Ideas similar to funnel libraries [59, 117] can be used to generate such feedback control laws that can track the path computed by our MILP solution with a pre-specified bound. Such a bound can also be incorporated by appropriately expanding and shrinking continuous propositions to guarantee overall correctness as is done in [51, 117].*

Next, we modify (3.3) to accommodate continuous states. We assume that each atomic proposition $a \in AP$ corresponds to a convex polytope $\{w \in \mathbb{R}^{d_w} \mid H^a w \leq h^a\}$, where $H^a \in \mathbb{R}^{d_a \times d_w}$ and $h^a \in \mathbb{R}^{d_a}$. Then for each atomic proposition and for all t and $n \in [N]$, we replace the inequality constraints in (3.3) with the following:

$$H^a w_n(t) \leq h^a + M(1 - e_n^a(t)), \tag{3.20a}$$

$$H^a w_n(t) \geq h^a + \epsilon - M e_n^a(t), \tag{3.20b}$$

$$z_n^a(t) = \bigwedge_i e_n^{a,(i)}(t), \tag{3.20c}$$

where ϵ is infinitesimally small, M is sufficiently large, and e_n^a is a binary vector of size d_a. The i^{th} row of e_n^a is denoted by $e_n^{a,(i)}(t)$ and represents the satisfaction of the i^{th} linear constraint of the convex polytope. That is, (3.20a) and (3.20b) ensure that the i^{th} linear constraint is satisfied if and only if $e_n^{a,(i)}(t) = 1$. Then, (3.20c) guarantees that $w_n(t) \in \{w \in \mathbb{R}^{d_w} \mid H^a w \leq h^a\}$ if and only if $z_n^a(t) = 1$, i.e., all linear constraints are satisfied. No other modifications are needed to use $z_n^a(t)$ in (3.4)-(3.9).

Finally, we modify the optimization problem to account for auxiliary variables. Let $\mathbf{e}^{cLTL+}$ denote the set of all auxiliary variables created by (3.20). We form the following optimization problem to find solutions:

$$\begin{aligned} \text{Find} \quad & \{u_n(0) \dots u_n(h-1)\}, \mathbf{z}^{\mathbf{loop}}, (\mathbf{e}, \mathbf{z}, \mathbf{y})^{\mathbf{cLTL+}} \\ \text{s.t.} \quad & (3.18), (3.19), ILP(\mu) \text{ and } y^{\mu}(0) = 1. \end{aligned} \tag{3.21}$$

Remark 3.4. *Given initial condition* w_0^n *and inputs* $\{u_n(0) \dots u_n(h-1)\}$*, state* $w_n(t)$ *can be found by* (3.18)*. Hence, no decision variables are needed for the states.*

Remark 3.5. *The resulting feasibility problem is a* mixed integer linear program

(MILP) *if dynamics in* (3.18) *are linear.*

As stated above, obtaining discrete abstractions from continuous dynamics is computationally expensive: the size of the transition system typically grows exponentially with the dimensionality of robot states. Since each discrete state in the transition system introduces a binary decision variable in the discrete-space formulation, the size of the optimization problem in (3.10) can grow quickly. On the other hand, in (3.21), each continuous state is represented with a single continuous decision variable. Therefore, the number of decision variables is independent from the size of the environment. While the number of auxiliary binary decision variables introduced by (3.20) depends on the specific problem instance, the continuous approach might be favorable when compared to an abstraction approach.

As for collisions, they can be avoided by introducing additional constraints, similar to 3.4, potentially by defining a safe distance and using approaches as in [82].

3.6. Extension of cLTL+ Syntax

This section provides a straightforward extension of the cLTL+ syntax inspired by censusSTL [116]. Up to now, the logic is oblivious as to which robot satisfies what atomic proposition, or task. In practice, robots might have heterogeneous capabilities and certain tasks might only be performed by a specific subset of robots. For example, imagine a collection of drones and a reconnaissance mission that includes, among other things, taking aerial photos of a region. If not all of the drones have cameras, one might want to identify those that can take photos and require subtasks that involve photography to be completed by this subset. Similarly, in a collective of robots where one robot is designated to be the leader it may be desirable to specify that the other robots periodically have to report to the leader.

To be able to specify such tasks, the temporal counting propositions (*tcp*) can be modified to contain the subset of robots that are designated with satisfaction of the inner logic formula. Redefine *tcp* as a tuple consisting of an atomic proposition, a non-empty set of robots and a non-negative integer, i.e., $\mu = [\phi, \mathcal{S}, m] \in \Phi \times 2^{[N]} \times \mathbb{N}$. Here satisfaction of μ at time t requires at least m robots from the subset $\mathcal{S} \in 2^{[N]}$ to satisfy ϕ at time t. By modifying *tcp*'s in this manner we can assign individual tasks to a specific subset of robots. To exemplify, given a collective $\mathcal{S}$ of drones, let $\mathcal{S}_c \in \mathcal{S}$ denote those with camera. Then the temporal counting proposition $tcp = [a, \mathcal{S}_c, m]$ would be satisfied if at least m drones from $\mathcal{S}_c$ visit regions marked by $a \in AP$ to take aerial photos.

Let $\mu = [\phi, \mathcal{S}, m]$. We modify (3.9) as follows to account for the change in *tcp* definition:

$$m > \sum_{n \in \mathcal{S}} z_n^{\phi}(t) - M y^{\mu}(t) \geq m - M. \tag{3.22}$$

Similarly, for the robustness case, we modify (4.3) as follows:

$$m > \sum_{n \in \mathcal{S}} r_n^{\phi}(t) - M y^{\mu}(t) \geq m - M. \tag{3.23}$$

It is straightforward to see that (3.22) and (3.23) preserve all of the soundness and completeness guarantees for this extension.

3.7. Examples

All experiments are run on a computer with 3.6 GHz Intel Core i7 and 128 GB RAM and YALMIP [54] is used to setup the optimization problems with Gurobi [33] as the underlying ILP solver. Our implementation can be accessed from `https://github.com/sahiny/cLTL-synth`.

3.7.1. Emergency response example

Let $N = 10$ robots be deployed in a the workspace depicted in Figure 3.1. We assume that only half of these robots—the even-numbered ones—are equipped with cameras. The workspace is discretized into 10×10 cells and each robot is modeled with a transition system with 100 states, each corresponding to a single cell. At each step, robots can either choose to stay put or to travel to any of the four neighboring cells without leaving the workspace. We remark that a monolithic LTL solution for this problem would have required constructing a transition system with 100^{10} states.

Let $\mathcal{S} = \{\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_N\}$ be the set of all robots and $\tilde{\mathcal{S}} = \{\mathcal{R}_2, \mathcal{R}_4, \ldots\}$ be the set of robots equipped with cameras. The specification is given by

$$\mu = \bigwedge_{i=1}^{8} \mu_i, \tag{3.24}$$

where each μ_i and the reasoning behind them is as follows:

- $\mu_1 = \Box\neg[D, \mathcal{S}, 1]$: collision with obstacles, which are marked with D, should be avoided,

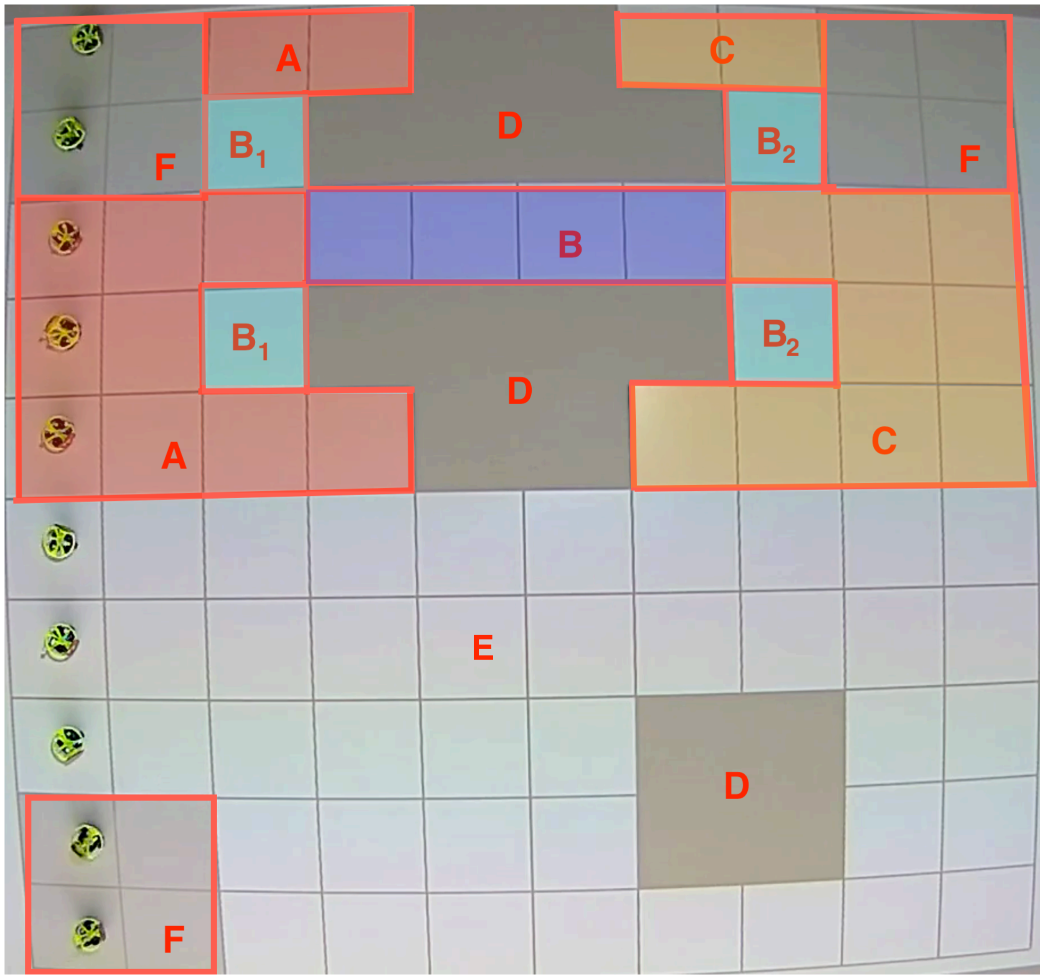


Figure 3.1: Emergency response example. Regions A, C, and E represent different neighborhoods, B represents a fragile bridge, F represents charging stations and D represents inaccessible zones.

- $\mu_2 = \Box\neg[B, \mathcal{S}, 3]$: the bridge, marked by B, must not be occupied by more than 2 robots,
- $\mu_3 = [\Box\Diamond F, \mathcal{S}, N]$: each robot should visit charging stations, marked by F, infinitely many times,
- region A and C must be populated with at least half of the robots and should

be left empty, infinitely many times:

$$\mu_4 = \Box\Diamond[A, \mathcal{S}, N/2], \qquad \mu_5 = \Box\Diamond[C, \mathcal{S}, N/2],$$

$$\mu_6 = \Box\Diamond(\neg[A, \mathcal{S}, 1]), \qquad \mu_7 = \Box\Diamond(\neg[C, \mathcal{S}, 1],$$

- $\mu_8 = (\neg[B, \mathcal{S}, 1])\, \mathcal{U} \left([B_1, \tilde{\mathcal{S}}, 1] \wedge [B_2, \tilde{\mathcal{S}}, 1]\right)$: bridge should be empty until it is inspected from both sides by robots equipped with cameras.

In addition to these specifications, we require that robots avoid collisions with each other. We assume robots move synchronously, posit a time horizon $h = 35$, and solve the optimization problem (3.1). The resulting optimization problem, which is encoded in YALMIP, has 36042 optimization variables and 13502 constraints, and is solved in 1038 seconds. Important frames obtained from the obtained solution are shown in Figure 3.2.

Even though all specifications are met by this solution for a synchronous execution, it could easily break with the introduction of asynchrony. For instance, note that region A is emptied (resp. region C is populated with more than 5 robots) only for a single time step at $t = 16$ (resp. $t = 18$). Hence, a single-step delay of a single robot could result in violation of μ_6 (resp. μ_5). Similarly, a robot enters the bridge for the first time at $t = 11$, which is the exact same time step when the bridge is inspected from both sides. If one of the robots inspecting the bridge moves slower than intended, μ_8 would be violated. These concerns motivated the study in Chapter 4 where we show how to overcome these shortcomings.

Note that all subspecifications of μ except μ_3 can be expressed in cLTL. If this

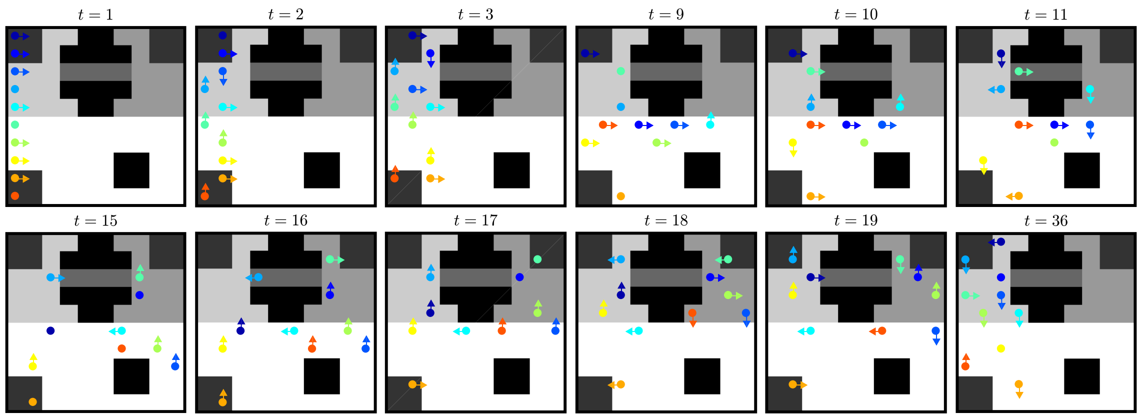


Figure 3.2: Important frames from the non-robust solution of the emergency response example. Arrows indicate the direction of movement. The loop starts at frame $t = 4$, thus the state at $t = 4$ is identical to the state at $t = 36$. Time $t = 16$ and $t = 18$ are the only time steps where region A and C are emptied and populated with more than 5 robots, respectively. The bridge is empty until two robots inspect it from different sides at $t = 11$. Every robot visits the charging station and avoids collisions.

requirement is removed, cLTL encodings can also be used to solve the same problem. In fact, a video simulating the synthesized plans, when μ_3 is removed cLTL encodings of Section 3.3 are used, can be seen at `https://youtu.be/EJ-v2yD-6_I`.

3.7.2. Numerical examples

To examine the scalability of the proposed approach with respect to different factors, we use the emergency response example explained in Section 3.7.1 and specifications in (3.24). The base example uses the following parameters: the number of robots $N = 10$ and solution horizon $h = 35$. We then vary one of these parameters at a time, and report the average solution times with 95% confidence interval values and the maximum solution times (in parentheses) in Table 3.1. These results are obtained over 20 runs with random initial conditions.

We report results for three different implementations in Table 3.1. The first implementation encodes dynamics as in Section 3.2.1. The second implementation, which is explained shortly, encodes dynamics slightly differently by taking advantage of the 4-connected grid environments. Finally, we implement the continuous-state extension proposed in Section 3.5. There are several important results that can be seen from Table 3.1. Firstly, we show that the proposed framework can handle complex temporal specifications and large number of robots. Secondly, we see that solution times scale reasonably well with N and h. Interestingly, we see that solution times change significantly with the encodings. Although, the encodings proposed in this thesis presented for arbitrary transitions system, solution times can be significantly reduced by exploiting the structure of the problem (such as using grid encodings for 4-connected workspaces).

We also note that one needs to do a binary search on h to find a solution horizon that leads to a feasible problem. For this example, we observed that the infeasibility is certified in less than a second for $h \leq 10$, and $h \geq 25$ leads to a feasible solution for all initial conditions. The minimum feasible h is dependent on the initial condition. There is usually an increase in solution times at the boundary of feasibility and infeasibility, yet this may also depend on the specific ILP solver and heuristics it implements.

In 4-connected grid environments, robots move in a two-dimensional gridded environment only horizontally or vertically. This structure can be exploited to decrease the number of decision variables as follows. Let x and y denote the length and width of this environment. For each robot and for each time step, encodings in (3.1) define a

Boolean decision variable vector of size $x \times y$ where the only non-zero entry represents the state of the robot at that time. In grid-encodings, we define two Boolean vectors with dimensions x and y for each robot and for each time step. Both vectors have only one non-zero entry, denoting the coordinates of the robot at that time. As a result, the number of decision variables in the optimization problem is significantly reduced. For example, given the parameters $N = 10$ and $h = 35$, the grid encoding has 18057 (as opposed to 36042) decision variables and 26928 (as opposed to 13502) constraints. From Table 3.1, we can see that this encoding almost always leads to faster solution times compared to the regular encoding.

For the continuous-state implementation, we use the following dynamics for each robot:

$$\begin{bmatrix} x_n(t+1) \\ y_n(t+1) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_n(t) \\ y_n(t) \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_n^1(t) \\ u_n^2(t) \end{bmatrix} \tag{3.25}$$

where $(x_n(t), y_n(t))$ represent the coordinates of $\mathcal{R}_n$ at time t. We also bound the control input as $|u_n^i(t)| \leq 1$ for $i = 1, 2$. For parameters $N = 10$ and $h = 35$, the resulting optimization problem has 22767 decision variables and 49271 constraints. Results in Table 3.1 show that continuous-state implementation scales reasonably well with the number of robots N and the solution horizon h.

The specifications of the emergency example cannot be expressed in cLTL. Therefore, we use a different example to illustrate the efficiency of cLTL encodings. Assume that robots have identical dynamics and the transition system $T = (V, \mathcal{E}, AP, L)$, where $\mathcal{E}$ is generated from an Erdös-Rényi graph with edge probability 0.25, represents the dynamics of N robots. The set V of states is partitioned into two sets of same size and labeled with $a_1 \in AP$ and $a_2 \in AP$. Each robot is assigned an initial state that is randomly selected from those labeled with a_1. Three goal regions are created such that each has $\frac{|V|}{10}$ randomly selected states and are labeled with $g_i \in AP$ for $i = 1, 2, 3$. The specification is given by the cLTL formula μ:

$$\mu = (\Diamond\Box[a_2, N/2]) \wedge \left(\bigwedge_{i=1}^{3} \Box\Diamond[g_i, N/3]\right). \tag{3.26}$$

The specification μ requires at least half of the robots to reach states marked by a_2 and stay there indefinitely. Also, each goal region must be populated by at least $N/3$ robots, infinitely often over time. The results in Table 3.2 are obtained by varying either the number of robots $N = 10$ or the time horizon $h = 20$ while keeping all the other parameters intact. Solution times in the first and second column are obtained by alternative cLTL encodings proposed in Section 3.3 and regular cLTL+ encodings, respectively. Regular cLTL+ encodings could not find solutions for $N = 500$ within the timeout threshold of 60 minutes. On the other hand, cLTL encodings scale much better with the number of robots and easily handle hundreds of robots in a matter of seconds. In fact, solution times are almost unaffected by the number of robots.

3.8. Summary

In this chapter, we presented an optimization-based method to provide multirobot paths to satisfy the specifications given in cLTL+. We assumed that robot dynamics are captured with deterministic transition systems and that robots move syn-

Table 3.1: Average solution times for the emergency response example with varying parameters. For each setting, 95% confidence intervals are obtained over 20 trials. Times are given in seconds and maximum solution times are written in parantheses.

		cLTL+ (regular)	**cLTL+ (grid)**	**cLTL+ (continuous)**
N	4	6.6 ± 3.0 (15.3)	7.9 ± 5.7 (18.3)	46.1 ± 13.2 (71.9)
	6	15.5 ± 12.3 (68.0)	18.9 ± 11.3 (37.1)	86.2 ± 64.0 (313.9)
	8	27.8 ± 8.9 (48.5)	27.3 ± 10.2 (39.4)	99.7 ± 48.8 (225.1)
	10	273.6 ± 229.2 (1056.8)	75.8 ± 25.9 (121.1)	128.4 ± 88.6 (420.8)
h	35	273.6 ± 229.2 (1056.8)	75.78 ± 25.9 (121.1)	128.4 ± 88.6 (420.8)
	40	371.9 ± 277.5 (1513.3)	82.91 ± 25.7 (118.1)	205.2 ± 112.6 (545.8)
	45	781.5 ± 885.3 (2855.9)	128.62 ± 45.8 (216.8)	189.1 ± 68.5 (318.6)
	50	593.4 ± 829.9 (3713.2)	145.57 ± 41.1 (236.5)	271.8 ± 134.1 (743.6)
	55	836.3 ± 1018.8 (3721.4)	163.38 ± 55.3 (272.8)	275.2 ± 133.8 (735.1)
	60	1188.2 ± 1758.1 (5917.7)	202.42 ± 92.5 (410.5)	468.4 ± 358.0 (2037.6)

Table 3.2: Comparison of average solutions times (in seconds) of cLTL and cLTL+ encodings.

		cLTL	**cLTL+**
N	10	10.86	2.64
	20	10.12	5.87
	50	8.99	56.13
	500	12.72	*TO*
h	20	10.86	2.64
	40	26.84	5.32
	60	60.93	7.87

chronously. We showed how to encode dynamic constraints, temporal counting constraints and collision constraints as mixed-integer linear constraints. This encoding allows us to generate a collection of trajectories by solving a feasibility problem such that the specifications are satisfied. We proved that such an approach is sound and complete.

We also proposed an alternative method for the particular case when the specifications are given in the cLTL fragment, and all robots have identical dynamics. Due to the structure of the problem —permutation invariance property of cLTL constraints— the alternative method's solution times do not depend on the number of robots. We further discussed how to generate solutions that are robust to failing robots and presented an extension to the cLTL+ syntax to allow more expressiveness.

Chapter 4.

Path Planning Robust to Synchronization Errors

In Chapter 3, we solved Problem 3.1 with the assumption that robots move synchronously. However, it is difficult to perfectly synchronize the motion of robots in real-life applications. Ideally, generated solutions should continue satisfying the specifications in the presence of "small disturbances", such as bounded synchronization errors. To reason about such time-robustness, robust satisfaction of cLTL+ formulas are introduced in Section 2.2.

In this section, we present two different methods to generate solutions that satisfy cLTL+ specifications robustly. Firstly, we show how to generate solutions that are robust to bounded synchronization errors, where the upper bound can be arbitrary but assumed to be known. We then use encodings similar to those of Chapter 3, but ensure that the solutions are robust via slight modifications. This solution method is shown to be partially complete but its computational complexity depends on the synchronization error bound. In the second part of this chapter, we propose a hierarchical method whose computational complexity do not depend on the synchronization error bound. However, as a trade-off, this approach is shown to be partially complete.

Synchronous execution assumes that multiple robots can transition from one discrete state to another at the same time. However, this is not always possible in reality where robots may move slower or faster than intended, leading to asynchronous switching times as illustrated in Figure 2.1. To exemplify, consider a task $\mu = \Diamond[\phi, m]$ that requires multiple robots to simultaneously satisfy a certain proposition ϕ at some time in the future. Let Π be a collection of trajectories and K be a synchronous collective execution. Assume that tcp $[\phi, m]$ holds for a single time step t and fails to hold for any other time instance, i.e., $(\Pi, K), t \models [\phi, m]$ for some t and $(\Pi, K), t' \not\models [\phi, m]$ for all $t' \neq t$. While such a Π satisfies μ for the synchronous execution it is not always a desirable collection. If K becomes asynchronous due to one of the robots moving at a different speed than intended, correctness guarantees would no longer be valid and μ would not be satisfied. This fact motivates searching for solutions that are robust to such asynchrony.

For most non-trivial specifications, finding a collection of trajectories that is robust to unbounded asynchrony is challenging if not impossible. However, if an upper bound on the asynchrony is assumed, one can generate robust solutions such that satisfaction of the task is guaranteed even under the worst-case scenario.

Before presenting modified encodings that incorporate robustness to asynchrony, we remind the reader that the robots are allowed to stutter as indicated by the definition of the local counter in (2.4). Therefore, any inner logic formula containing '$\bigcirc$' can easily be violated by a single robot. Hence, we restrict our attention to the case where inner logic formulas are given in LTL without the "next ($\bigcirc$)" operator ($\text{LTL}_{\setminus\bigcirc}$). We further assume that a cLTL+ formula is given in positive normal form (PNF) according to the following syntax:

$$\mu ::= True \mid tcp \mid \mu_1 \wedge \mu_2 \mid \mu_1 \vee \mu_2 \mid \bigcirc\mu \mid \mu_1 \,\mathcal{U}\, \mu_2 \mid \mu_1 \,\mathcal{R}\, \mu_2. \tag{4.1}$$

Remark 4.1. *The negation operator can be omitted without loss of generality for two reasons. First, any LTL formula can be transformed into positive normal form (PNF) [5], where the negation operator appears only before atomic propositions. Since the syntax of cLTL+ is identical to LTL, hence any cLTL+ formula can also be written in PNF where negation only appears before tcp's. Second, given an arbitrary temporal counting proposition $\mu = [\phi, m]$, the statement $\neg\mu$ can be replaced by $\mu' = [\neg\phi, N + 1 - m]$. Clearly, if there are at least $N + 1 - m$ robots satisfying $\neg\phi$, then ϕ is satisfied by less than m robots; hence, $\mu \equiv \mu'$. Thus, the omission of the negation operator is without loss of generality.*

Finally, we formally define the robust version of Problem 3.1 as follows:

Problem 4.1. *Given N robots with dynamics $\{T_n = (V_n, \mathcal{E}_n, AP, L_n)\}$, initial conditions $\{\pi_n(0)\}$, a cLTL+ formula μ given in PNF over $\text{LTL}_{\setminus\bigcirc}$, and an upper bound on the asynchrony τ, synthesize a collection $\Pi = \{\pi_1, \ldots, \pi_N\}$ of trajectories π_n that τ-robustly satisfies μ, i.e., $\Pi \models_\tau \mu$.*

4.1. Robust Encodings

We propose slight modifications to the encodings presented in Section 3.1 to generate a collection of trajectories that are τ-robust. Firstly, we define τ new Boolean vectors $w_n(h+1), w_n(h+2) \dots w_n(h+\tau)$ to represent the state of robot $\mathcal{R}_n$ "after the loop" such that $w_n(h+k) = w_n(l+k)$ for all $k = 0, 1, \dots, \tau$ where $l < h$ is the first index of the suffix loop. Secondly, given a temporal counting proposition $\mu = [\phi, m]$, we introduce a new decision variable $r_n^\phi(t)$ for each $z_n^\phi(t)$:

$$r_n^\phi(t) = \bigwedge_{k=0}^{\tau} z_n^\phi(t+k), \qquad \text{for } 0 \leq t < h. \tag{4.2}$$

These new variables $r_n^\phi(t)$ can be seen as the "robust" versions of $z_n^\phi(t)$. In order for $r_n^\phi(t) = 1$ to hold, robot $\mathcal{R}_n$ needs to satisfy the inner logic formula ϕ not only at time step t, but also for the next τ steps, i.e., $z_n^\phi(k') = 1$ for all $t \leq k' \leq t + \tau$. Consequently, $r_n^\phi(t) = 1$ implies that $\mathcal{R}_n$ satisfies ϕ at all time steps for which the anchor time is t.

To further clarify, let $K \in \mathcal{K}_N(\tau)$ be an arbitrary τ-bounded execution and T be an arbitrary time step such that $b_K(T) = t$. Firstly, the state of $\mathcal{R}_n$ at this time step is given by its local counter $k_n(T)$. Therefore, $\mathcal{R}_n$ satisfies ϕ at time T if and only if $z_n^\phi(k_n(T)) = 1$. Secondly, $t \leq k_n(T) \leq t + \tau$ must hold for all n due to τ-boundedness of K. Then, $r_n^\phi(t) = 1$ implies $z_n^\phi(k_n(T)) = 1$. That is, $\mathcal{R}_n$ satisfies ϕ at time T if $r_n^\phi(k_n(T)) = 1$. Note that, this is true for all T for which the anchor time is t, i.e., for all $T \in b_K^{-1}(t)$. Therefore, $r_n^\phi(t) = 1$ ensures that $\mathcal{R}_n$ satisfies ϕ at all time steps for which the anchor time is t.

We now define the modified outer logic constraints. As before, these constraints are constructed recursively. Let $\mu = [\phi, m]$ be a *tcp* with $m > 1$. Then (3.9) is modified as

$$m > \sum_{n=1}^{N} r_n^{\phi}(t) - My^{\mu}(t) \geq m - M. \tag{4.3}$$

In Equation (4.3), the term $y^{\mu}(t) = 1$ if and only if $\sum_{n=1}^{N} r_n^{\phi}(t) \geq m$. Thus, $y^{\mu}(t) = 1$ implies that there are at least m robots that satisfy ϕ at any time step T for which the anchor time is t. The last statement is true for every τ-bounded execution and for every time step. Thus, $y^{\mu}(t) = 1$ implies that μ is τ-robustly satisfied at anchor time t.

For the special case where $\mu = [\phi, 1]$, we use

$$1 > \sum_{n=1}^{N} r_n^{\phi}(t) - M\tilde{y}^{\mu}(t) \geq 1 - M, \tag{4.4a}$$

$$N > \sum_{n=1}^{N} z_n^{\phi}(t) - M\bar{y}^{\mu}(t) \geq N - M, \tag{4.4b}$$

$$y^{\mu}(t) = \tilde{y}^{\mu}(t) \vee \bar{y}^{\mu}(t). \tag{4.4c}$$

The inequalities in (4.4a) are identical to (4.3) for $m = 1$. For $\tilde{y}^{\mu}(t) = 1$ to hold, at least 1 robot needs to satisfy ϕ for at least $\tau + 1$ consecutive time steps. While this is sufficient for τ-robust satisfaction of μ, as shown in Theorem 4.1, it is not necessary. The collection can robustly satisfy μ for $m = 1$, even if $r_n^{\phi}(t) = 1$ fails to hold for any robot. Intuitively, $m = 1$ is a special case due to the definition of anchor time. For a τ-bounded asynchronous execution at anchor time t, all local counters are restricted to an interval but none of them are precisely known, a priori. However, at least one of the local counters must be equal to t by definition of the anchor time. Therefore, if every robot satisfies ϕ at the t^{th} step of their trajectory, there would be at least one robot satisfying ϕ at all time steps for which the anchor time is t. That is, μ is τ-robustly satisfied if the inequalities in Equation (4.4b) are satisfied for $\bar{y}^{\mu}(t) = 1$. Equation (4.4c) states that either one of these conditions is enough to robustly satisfy μ.

In the synchronous setting, satisfying a temporal counting proposition μ only for an instant would be enough. However, this is not desirable since robots might not be perfectly synchronized. Equations (4.3) and (4.4) ensure that all τ-bounded executions satisfy μ at all time instances with anchor time t, by replacing each $z_n^{\phi}(t)$ with its robust counterpart $r_n^{\phi}(t)$. As a result, even in the worst case of asynchrony, there is an instant where μ is satisfied.

Encodings of some of the outer level operators are also modified slightly. Using encodings from Section 3.1 might lead to conservatism due to robust encodings of inner logic formulas. While such conservatism is expected due to the unknown nature of asynchronous executions, we can mitigate the conservatism to some extent by modifying the outer logic encodings. For conjunction and next operators, no modification is needed: if $\mu = \mu_1 \wedge \mu_2$ and $\eta = \bigcirc\mu$ where each μ_i is a cLTL+ formula in PNF form, then $y^{\mu}(t) = y^{\mu_1}(t) \wedge y^{\mu_2}(t)$ and $y^{\eta}(t) = y^{\mu}(t+1)$.

Disjunction is encoded in two different ways: For general formulas $\mu = \bigvee_i \mu_i$, we use the standard disjunction encodings:

$$y^{\mu}(t) = \bigvee_i y^{\mu_i}(t). \tag{4.5}$$

For the special case where all arguments are temporal counting propositions, i.e, $\mu = \bigvee_i \mu_i$ such that $\mu_i = [\phi_i, m_i]$, the encoding

$$y^{\mu}(t) = \bigvee_i y^{\mu_i}(t) \vee \left(\sum_{n=1}^{N} r_n^{(\bigvee_i \phi_i)}(t) > \sum_i (m_i - 1) \right) \tag{4.6}$$

is used. The motivation behind the additional term in (4.6) is that a collection $\{\pi_n\}$ might not τ-robustly satisfy neither μ_1 nor μ_2 but can still τ-robustly satisfy $\mu_1 \vee \mu_2$ as demonstrated by the following example:

Example 4.1. *Let $\mu = \mu_1 \vee \mu_2 = [\phi_1, 2] \vee [\phi_2, 2]$ be a cLTL+ formula and let a collection $\Pi = \{\pi_1, \pi_2, \pi_3\}$ be given with the following traces:*

$$\sigma(\pi_1) = \{\phi_1\}\ \{\phi_1\}\ \{\phi_1\}\ \ldots$$

$$\sigma(\pi_2) = \{\phi_1\}\ \{\phi_2\}\ \{\phi_2\}\ \ldots$$

$$\sigma(\pi_3) = \{\phi_2\}\ \{\phi_2\}\ \{\phi_2\}\ \ldots$$

For $\tau = 1$, any arbitrary τ-bounded asynchronous execution satisfies either μ_1 or μ_2 for all time steps for which the anchor time is $t = 0$. Therefore, $\Pi \models_\tau \mu$ by Definition 2.5. On the other hand, the collection Π does not robustly satisfy neither μ_1 nor μ_2 at anchor time $t = 0$.

Equation (4.6) limits the number of robots who neither satisfy ϕ_1 nor ϕ_2 at anchor time t. By doing so, it ensures that either μ_1 or μ_2 is satisfied by the collection. Furthermore, (4.6) reduces to standard encodings for $\tau = 0$ as expected. If the disjunction μ contains both *tcp*s and other formulas, then one can re-write μ as $\mu_{tcp} \vee \mu_{or}$ where $\mu_{tcp} = \bigvee_i [\phi_i, m_i]$ to leverage the less conservative encodings in (4.6).

Due to changes in the outer disjunction encodings, the outer "until" operator is modified as well. Let $\eta = \mu_1 \,\mathcal{U}\, \mu_2$ where μ_i is a cLTL+ formula for $i = 1, 2$. Then

$$\begin{aligned}
y^{\eta}(t) &= y^{\mu_1 \vee \mu_2}(t) \wedge \left(y^{\mu_2}(t) \vee y^{\eta}(t+1)\right), \quad \text{for all } t \leq h-2, \\
y^{\eta}(h-1) &= y^{\mu_1 \vee \mu_2}(h-1) \wedge \left(y^{\mu_2}(h-1) \vee \left(\bigvee_{t=0}^{h-1} \left(z_{loop}(t) \wedge \tilde{y}^{\eta}(t) \right) \right) \right), \\
\tilde{y}^{\eta}(t) &= y^{\mu_1 \vee \mu_2}(t) \wedge \left(y^{\mu_2}(t) \vee \tilde{y}^{\eta}(t+1)\right), \quad \text{for all } t \leq h-2, \\
\tilde{y}^{\eta}(h-1) &= y^{\mu_2}(h-1).
\end{aligned} \tag{4.7}$$

The encodings in (4.7) are obtained by first using the distributive property of the Boolean disjunction operator and then rewriting the expression using the new robust disjunction encodings. As a sanity check, assume $y^{\mu_2}(t) = 1$, i.e. that μ_2 is τ-robustly satisfied at anchor time t. Then, $y^{\mu_1 \vee \mu_2}(t) = 1$ must hold due to (4.6) and (4.5), and $y^{\eta}(t) = 1$ must hold due to first line of (4.7). This is expected as when μ_2 is satisfied, η is satisfied by definition of cLTL+ semantics. If μ_2 is *not* τ-robustly satisfied at anchor time t, (4.7) enforces η and $\mu_1 \vee \mu_2$ (instead of only μ_1 as in (3.8)) to be τ-robustly satisfied at anchor times $t+1$ and t, respectively. During execution, at anchor time t, if μ_2 is satisfied, η is satisfied due to the semantics of cLTL+. Otherwise, if μ_1 is satisfied, we require η to hold at the next time step, similar to the standard encodings of “until". As with the robust disjunction encodings, (4.7) reduces to the standard encodings for $\tau = 0$.

Furthermore, we provide encodings for the “release” operator that are identical to the standard encodings used in the literature: if $\eta = \mu_1 \,\mathcal{R}\, \mu_2$, then

$$\begin{aligned}
y^{\eta}(t) &= y^{\mu_2}(t) \wedge \left(y^{\mu_1}(t) \vee y^{\eta}(t+1)\right), \qquad \text{for all } t \leq h-2, \\
y^{\eta}(h-1) &= y^{\mu_2}(h-1) \wedge \left(y^{\mu_1}(h-1) \vee \left(\bigvee_{t=0}^{h-1} \left(z_{loop}(t) \wedge \tilde{y}^{\eta}(t)\right) \right) \right), \\
\tilde{y}^{\eta}(t) &= y^{\mu_2}(t) \wedge \left(y^{\mu_1}(t) \vee \tilde{y}^{\eta}(t+1)\right), \qquad \text{for all } t \leq h-2, \\
\tilde{y}^{\eta}(h-1) &= y^{\mu_2}(h-1).
\end{aligned} \tag{4.8}$$

The release operator requires that μ_2 is satisfied up to and including the first time step where μ_1 is satisfied for the first time. The key difference from the until operator is that η may hold even if μ_1 is never satisfied, provided that μ_2 is satisfied indefinitely.

Given an instance of Problem 4.1 and a horizon length h, let $ILP_{\tau}(\mu)$ be the set of ILP constraints and $(\mathbf{z}, \mathbf{r}, \mathbf{y})^{\mathbf{cLTL+}}$ the decision variables created by using the robust encodings (4.3)-(4.8). We obtain the robust solution by solving the following optimization problem:

$$\begin{aligned}
\text{Find} \quad & \{\mathbf{w}_n\}, \mathbf{z}^{\mathbf{loop}}, (\mathbf{z}, \mathbf{r}, \mathbf{y})^{\mathbf{cLTL+}} \\
\text{s.t.} \quad & (3.1), (3.2), ILP_{\tau}(\mu) \text{ and } y^{\mu}(0) = 1.
\end{aligned} \tag{4.9}$$

The following theorems show that the solution method proposed for the asynchronous case is sound, and also complete under certain conditions. The proofs are provided in the Appendix A.

Theorem 4.1. *If the optimization problem in* (4.9) *is feasible for a cLTL+ formula* μ *given in PNF over* $LTL_{\setminus\bigcirc}$*, then a collection* $\Pi = \{\pi_1, \ldots, \pi_N\}$ *of trajectories can be extracted such that* $\Pi \models_\tau \mu$*. That is, the modified encodings in* (4.2)-(4.8) *are sound.*

As shown in Example 4.1, the disjunction operator introduces some conservatism. Furthermore, the disjunction operation is used in the encodings of "until" and "release". Therefore, completeness results from Section IV are no longer valid in the asynchronous setting. The next result clarifies the conditions when the robust encodings are complete:

Theorem 4.2. *Given a cLTL+ formula* μ *given in PNF over* $LTL_{\setminus\bigcirc}$*, if all of the following hold, then there exists a finite* h *such that* (4.9) *has a solution (i.e., the modified encodings are complete).*

- *there exists a collection* $\Pi = \{\pi_1, \ldots, \pi_N\}$ *of trajectories in prefix-suffix form that* τ*-robustly satisfies* μ*, i.e.,* $\Pi \models_\tau \mu$*,*
- AP *is a set of mutually exclusive atomic propositions, i.e., for all* $\phi_1, \phi_2 \in AP$*;* $\phi_1 \wedge \phi_2 = False$*,*
- *the specification* μ *over* AP *is on the form*

$$\mu = True \mid tcp \mid \mu_1 \wedge \mu_2 \mid tcp_1 \vee tcp_2 \mid tcp_1 \, \mathcal{U} \, tcp_2 \mid \bigcirc\mu \tag{4.10}$$

  where $tcp, tcp_1, tcp_2 \in AP \times \mathbb{N}$ *and* μ, μ_1, μ_2 *are obtained according to* (4.10)*.*

The commonly used "$\Diamond$(*eventually*)" operator can also be defined without losing completeness: $\Diamond[\phi, m] \doteq [\neg\phi, N - m + 1] \, \mathcal{U} \, [\phi, m]$. In most real world applications, several tasks are required to be completed in conjunction, which can be expressed as in (4.10). Furthermore, many interesting specifications including safety ($\square$), liveness ($\square\lozenge$), etc., can be captured in the form of (4.10) for a given time horizon h. For example, safety specifications can be encoded as $\square[\phi, m] = [\phi, m] \wedge \bigcirc[\phi, m] \wedge \cdots \wedge \bigcirc^{h-1}[\phi, m]$[1].

Remark 4.2. *The alternative solution method proposed in Section 2.3 uses more efficient encodings when the specifications are given in cLTL. However, these encodings use aggregate dynamics, therefore it is not possible to keep track of identities of the robots during synthesis. Hence, robust solutions cannot be generated with this alternative method.*

Robustifying the trajectories increases the complexity as a function of τ. In particular, an instance of (4.9) has $\mathcal{O}(\tau N(|V_n| + h|\mu|))$ additional decision variables and $\mathcal{O}(\tau N^2 h|V_n|)$ additional constraints compared to (3.10). The effect of these additional variables and constraints on solution time is investigated in Section 4.1.1.

4.1.1. Emergency Response Example Revisited

This section demonstrates that the encodings in Section 4.1 generate solutions that are robust to bounded synchronization errors. All experiments are run on a computer with 3.6 GHz Intel Core i7 and 128 GB RAM and YALMIP [**yalmip**] is used to setup the optimization problems with Gurobi [33] as the underlying ILP solver. Our implementation can be accessed from `https://github.com/sahiny/cLTL-synth`.

[1]The notation $\bigcirc^{h-1}$ corresponds to $(h-1)$ concatenated $\bigcirc$ operators

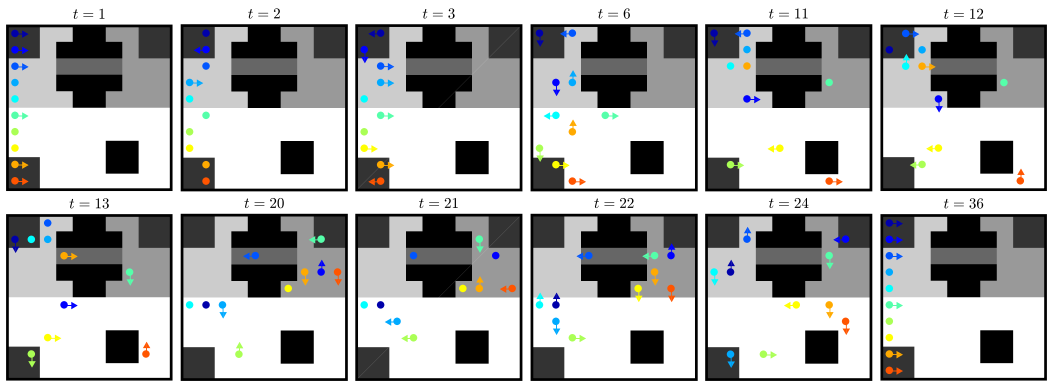


Figure 4.1: Important frames from robust solution of the emergency response example. Arrows indicate direction of movement. The loop starts at frame $t = 1$, which is identical to frame $t = 36$. The number of robots in region A (C) is 5 (0) between $t = 1$ and $t = 3$ and 0 (5) between $t = 20$ and $t = 22$, which implies that μ_4 to μ_7 are robustly satisfied at anchor time $t = 1$. No robots use the narrow passage until it has been examined by both sides between $t = 11$ and $t = 13$, and the number of robots on the bridge never exceeds 2; hence μ_2 and μ_8 are robustly satisfied. Every robot visits the charging station and avoids collisions.

We revisit the emergency response example from Section 3.7. To prevent violation of specifications due to asynchronous motion of robots, we set $\tau = 2$ and solve the resulting instance of Problem 4.1. As it is shown in Fig. 4.1, this time the number of robots in A (resp. in C) is greater than or equal to 5, starting from $t = 1$ until $t = 3$ (resp. from $t = 20$ until $t = 22$). Furthermore, when the number of robots in region A is greater than or equal to 5, there are no robots in region C, and vice versa. Therefore, even in the worst case of bounded asynchrony, there will be at least one time instance where A is populated with 5 robots and another time instance where A is empty. The same arguments hold for region C, as well. Additionally, the robots are more careful when crossing and the bridge: the bridge is first inspected at $t = 11$ and no robots enter the bridge until $t = 13$. Thus, the specification μ is satisfied even in the worst case of asynchrony.

We have implemented the trajectories extracted from the robust solution on real ground robots in the Robotarium [77]. In this experiment, robots track their respective trajectories using feedback from a top-mounted camera, and do not communicate with each other during runtime. The asynchrony bound τ is taken to be 2. To ensure this bound is satisfied, we implemented a monitor that enforces “leading robots" to wait when necessary. While we used the top-mounted camera in a centralized manner for the monitoring purpose, an asynchrony bound can be learned by analyzing the system behavior or can be enforced using only local controllers. For example, assume T_{max} is an upper-bound on the time it takes to complete one transition. Then, at time T, each robot is guaranteed to complete at least $\lfloor T/T_{max} \rfloor$ transitions, where $\lfloor T/T_{max} \rfloor$ is the greatest integer that is less than or equal to T/T_{max}. To limit the asynchrony to τ discrete steps, one can implement local controllers such that a robot is forced to wait if it were to complete $k \geq (\lfloor T/T_{max} \rfloor + \tau)$ steps up to time T. The video of the experiment can be viewed from `https://youtu.be/u8G-ewEEO6E`. As can be seen from the video, robots satisfy their tasks and avoid collisions despite the asynchrony.

We know show the effect of robustness parameter on solution times. To do so, we keep every other parameter intact and solve the emergency scenario example for $\tau = \{0, 1, 2\}$. The results from Table 4.1 show that solution times greatly increase as the parameter τ increases. In particular for the continuous-state implementation, solution times do not scale well with the robustness parameter τ. This might be explained by the high number of constraints, which is due to the additional constraints introduced by each polytopic obstacle in the continuous-state setting, compared to

Table 4.1: Average solution times for the emergency response example with varying parameters. Also shown is 95% confidence interval values obtained over 20 trials. Times are given in seconds and maximum solution times are written in parantheses.

τ	**cLTL+ (regular)**	**cLTL+ (grid)**	**cLTL+ (continuous)**
0	273.6 ± 229.2 (1056.8)	75.8 ± 25.9 (121.1)	128.4 ± 88.6 (420.8)
1	635.3 ± 548.8 (1733.4)	120.3 ± 44.4 (195.6)	$1859.1 \pm 2319.1^{*}$ (6000)
2	4638.2 ± 874.8 (5967.7)	98.5 ± 52.5 (225.4)	$4420.5 \pm 2343.7^{**}$ (6000)

* 5 runs exceeded the time threshold of 6000 seconds.
** 14 runs exceeded the time threshold of 6000 seconds.
(Solution times for timed-out runs are taken to be 6000 seconds.)

the regular discrete-state implementation. We address this scalability problems in Section 4.2.

4.2. Hierarchical Approach

The complexity of the method proposed in Section 4.1 is shown to depend on the synchronization error bound τ. As a result, this method is computationally expensive. The problem becomes intractable if the size of a transition system T_n is large. In this section, we propose a *hierarchical* approach that scales better with the size of the transition systems. For the simplicity, assume all T_n are identical, and use T to denote this transition system. All the results in this section generalize to the case where T_n are not identical. We show that, under mild assumptions, the complexity of this hierarchical method does not depend on τ. However, as a trade-off, we need to limit the specifications to *temporal logic plus without 'next' operator (cLTL+$_{\backslash\bigcirc}$)*. Our method is illustrated in Algorithm 4.1. In the following, we give an overview of the algorithm and then explain its parts in detail.

Let $T = (V, \mathcal{E}, AP, L)$ be a transition system, $\mathcal{R} = \{\mathcal{R}_1, \ldots, \mathcal{R}_N\}$ be a set of robots, $S_0 : \mathcal{R} \to V$ be the mapping that maps robots to their initial conditions, μ be a (cLTL+$_{\backslash\bigcirc}$) formula and $\tau = 0$. From now on, we refer to an instance of Problem 4.1 by a tuple $(T, \mathcal{R}, S_0, \mu, \tau)$. Given $(T, \mathcal{R}, S_0, \mu, 0)$, we first compute a new transition system T^{abs}, called *abstraction* of T. How to compute T^{abs} is described in Section4.2.1. The motivation behind computing an abstraction is that it has a smaller size compared to the original transition system; hence, it would decrease the computational resources required to solve Problem 4.1. After adjusting the initial conditions of T^{abs} as S_0^{abs} and we solve a slightly modified version of Problem 4.1 instance $(T^{abs}, \mathcal{R}, S_0^{abs}, \mu, 1)$, relaxing the collision avoidance constraint. A solution to this instance is called an *abstract plan*, which is a collection of T^{abs}-paths. These *abstract paths* satisfy the logic constraints and can be seen as guidelines. Rather than explicitly assigning each robot a path, they indicate what propositions it needs to satisfy and in which order. We then replace each abstract path with a stutter trace equivalent T-path to generate a solution to $(T, \mathcal{R}, S_0, \mu, 0)$.

Construction of the abstraction ensures the existence of stutter trace equivalent T-paths. However, a collection of such paths is not guaranteed to be collision free. To prevent collisions, we solve a sequence of path planning problems in the lower level. If all the path planning problems are feasible, a solution to $(T, \mathcal{R}, S_0, \mu, 0)$ can be extracted. On the other hand, if the abstract plan is not feasible, we generate a counter example to be used in the higher level and obtain a different abstract plan. These steps are repeated until a solution is found or the algorithm terminates with no solution. In the following, we explain the steps of the main algorithm in greater detail.

4.2.1. Abstraction

Given a transition system $T = (V, \mathcal{E}, AP, L)$, we define an equivalence relation $\sim$ on V as follows:

$$u \sim v \text{ if and only if } u = v \text{, or } L(u) = L(v) \text{ and there exist } T\text{-paths } \pi_{uv} \text{ and } \pi_{vu} \\ \text{from } u \text{ to } v \text{ and from } v \text{ to } u \text{ such that } \sigma_{\pi_{uv}}(t) = \sigma_{\pi_{vu}}(t) = L(u) \text{ for all } t. \tag{4.11}$$

Relation $\sim$ partitions V into equivalence classes $V_1, ..., V_C$, for some $C \in \mathbb{N}$, such that all V_i are pairwise disjoint and all states in each V_i are equivalent, with $V = \cup_{i=1}^{C} V_i$. In words, nodes in a class satisfy the same property and are strongly connected. We create a state v_i^{abs} for each equivalence class V_i and denote the set of all such states by V^{abs}. To map the states of V to states of V^{abs}, we define $\mathcal{M} : V \to V^{abs}$:

$$\mathcal{M}(v) \doteq v_i^{abs} \text{ if } v \in V_i. \tag{4.12}$$

By definition, inverse of $\mathcal{M}$ maps the states of V^{abs} to equivalence classes, i.e., $\mathcal{M}^{-1}(v_i^{abs}) = V_i$. Next we define *abstraction of* T, denoted by $T_{abs} \doteq (V^{abs}, \mathcal{E}^{abs}, AP, L^{abs})$, such that:

$$\begin{aligned}\mathcal{E}^{abs} \doteq& \{(v_i^{abs}, v_j^{abs}) \mid \exists (u,v) \in \mathcal{E}, u \in \mathcal{M}^{-1}(v_i^{abs}) \text{ and } v \in \mathcal{M}^{-1}(v_j^{abs})\}, \\ L^{abs}(v_i^{abs}) \doteq& L(v) \text{ for any } v \in \mathcal{M}^{-1}(v_i^{abs}).\end{aligned} \tag{4.13}$$

The mapping L^{abs} is well-defined because $L(v)$ is guaranteed to be the same no matter which $v \in \mathcal{M}^{-1}(v_i^{abs})$ is chosen, by definition of equivalence (4.11). Furthermore, the existence of T^{abs} is guaranteed because equivalence classes form a partition of V and in the worst case, each state $v \in V$ would belong to a different equivalence class. In that case, T^{abs} would be identical to T. Computation of abstractions can be done efficiently (see Algorithm 37 in [5]). Initial conditions can be adjusted simply defining $S_0^{abs}(\mathcal{R}_n) \doteq \mathcal{M}(S_0(\mathcal{R}_n))$ for each $\mathcal{R}_n \in \mathcal{R}$.

Equivalence relation defined in (4.13) is the coarsest stutter bisimulation for T (see Lemma 7.96 in [5]) and abstraction T^{abs} is stutter bisimulation equivalent to T (see Theorem 7.102 in [5]) as stated by the following theorem:

Theorem 4.3. *Let T be a transition system and T^{abs} be its abstraction. For any T-path π, there exists a stutter trace equivalent T^{abs}-path π^{abs}. Conversely, for any T^{abs}-path π^{abs}, there exists a stutter trace equivalent T-path π.*

Proof of Theorem 4.3 can be found in Appendix A.

4.2.2. Higher Level Solution

In the higher-level, we solve a slightly different version of Problem 4.1 instance $(T^{abs}, \mathcal{R}, S_0^{abs}, \mu, 1)$ to generate a collection of paths $\{\pi_1^{abs}, \ldots, \pi_N^{abs}\}$. First, we do not enforce collision avoidance since each state in V^{abs} corresponds to a set of states in the original transition system and could hold more than a single robot. Second, we impose additional constraints coming from counter-examples. These constraints are explained in more detail in Section4.2.4. We then form an integer linear program (ILP) as it is explained in [89]. Solving the subsequent feasibility problem generates a collection $\Pi^{abs} = \{\pi_n^{abs}, \ldots, \pi_N^{abs}\}$ of T^{abs}-paths that 1-robustly satisfies μ, i.e., $\Pi^{abs} \models_1 \mu$. We call such a collection an *abstract plan*. We remind the reader that each *abstract path* π_n^{abs} has a prefix-suffix form as cLTL+$\backslash\bigcirc$formulas are interpreted over infinite horizons, i.e., there exists a non-negative integer $l^{abs} < h^{abs}$ such that for all n

$$\pi_n^{abs} = \pi_n^{abs}(0) \ldots \pi_n^{abs}(l^{abs})(\pi_n^{abs}(l^{abs}+1) \ldots \pi_n^{abs}(h^{abs}))^{\omega}. \tag{4.14}$$

In other words, each robot is assigned a lasso shaped path that can be traversed indefinitely. As shown in [5], if $(T^{abs}, \mathcal{R}, S_0^{abs}, \mu, 1)$ has a solution, then there exists a large enough h^{abs} such that there exists a solution in the form of (4.14).

4.2.3. Lower Level Solution

Theorem 4.3 guarantees that, for each T^{abs}-path π_n^{abs}, there exists a stutter trace equivalent T-path π_n. Each state of the abstraction T^{abs} corresponds to a set of states in the original transition system T. If an robot moves one from one state to another in the abstraction, it needs to move from one region to another in the original transition system. By construction of the T^{abs}, the existence of a path between such two regions is guaranteed. However, a collection of these paths might be in collision.

To generate a collection of collision-free and stutter trace equivalent T-paths, we solve a sequence of *generalized multirobot path planning (GMRPP)* problems. Following is the formal definition of GMRPP that we use:

Problem 1. *Let a transition system $T = (V, \mathcal{E}, AP, L)$, a set of robots $\mathcal{R} = \{\mathcal{R}_1, \ldots, \mathcal{R}_N\}$, a time horizon $h \in \mathbb{N}$ and injective mappings $x_I, x_G, X_I, X_G : \mathcal{R} \to 2^V$ be given. Find a collection of collision-free T-paths $\{\pi_1, \ldots \pi_N\}$ such that for all $\mathcal{R}_n \in \mathcal{R}$, $\pi_n(0) \in x_I(\mathcal{R}_n)$, $\pi_n(h) \in x_G(\mathcal{R}_n)$ and for each robot there exists a positive integer $0 < l_n < h$ where $\pi_n(t) \in X_I(\mathcal{R}_n)$ for all $0 \leq t \leq l_n$ and $\pi_n(t) \in X_G(\mathcal{R}_n)$ for all $l_n < t \leq h$.*

We characterize an instance of Problem 1 by a tuple $(T, \mathcal{R}, h, x_I, x_G, X_I, X_G)$. Each robot $\mathcal{R}_n \in \mathcal{R}$ needs to start from state within $x_I(\mathcal{R}_n) \subset X_I(\mathcal{R}_n) \subset V$ and reach a state in $x_G(\mathcal{R}_n) \subset X_G(\mathcal{R}_n) \subset V$. While doing so, robot $\mathcal{R}_n$ should stay in set of states $X_I(\mathcal{R}_n) \cup X_G(\mathcal{R}_n)$ for all times and it should not return back to $X_I(\mathcal{R}_n)$ once in $X_G(\mathcal{R}_n)$. Furthermore, collisions with other robots must be avoided. The intuition here is that set of states $X_I(\mathcal{R}_n)$ and $X_G(\mathcal{R}_n)$ correspond to two consecutive states on an abstract path. We use $x_I(\mathcal{R}_n)$ (and $x_G(\mathcal{R}_n)$) in case initial (and final) state needs to be explicitly specified. As it moves from one abstract state to the other, to prevent jittering, an robot leaving X_I should not return back.

GMRPP is a generalization of the classical multi robot path planning (MRPP) problem where one assigns a single initial state and a single goal state to each robot and assumes that the set of 'safe' states are same for all robots. Despite that, many efficient MRPP algorithms, such as [58, 100, 118], can easily be modified to accommodate for these differences. We use an ILP based method similar to [118] to solve GMRPP problems. For all $\mathcal{R}_n \in \mathcal{R}$ and for all $0 \leq t \leq h^{abs}$, we set

$$\begin{aligned} x_I^0(\mathcal{R}_n) = S_0(\mathcal{R}_n), \quad X_I^t(\mathcal{R}_n) = \mathcal{M}^{-1}(\pi_n^{abs}(t)), \\ x_G^t(\mathcal{R}_n) = X_G^t(\mathcal{R}_n) = \mathcal{M}^{-1}(\pi_n^{abs}(t+1)) \end{aligned} \tag{4.15}$$

Starting from $t = 0$, let $\{\beta_1^t, \dots, \beta_N^t\}$ be a solution to $(T, \mathcal{R}, h, x_I^t, x_G^t, X_I^t, X_G^t)$. For all $t > 0$ and $\mathcal{R}_n \in \mathcal{R}$, we set

$$x_I^t(\mathcal{R}_n) = \beta_n^{t-1}(h) \tag{4.16}$$

and solve the next GMRPP instance. For the special case $t = h^{abs}$, we set $x_G^t(\mathcal{R}_n) = \beta_n^{l^{abs}}(0)$ to 'close the loop'. If all GMRPP instances can be solved, we define for all integers $0 \leq t < h^{abs}$ and $0 \leq \alpha < h$

$$\pi_n(th + \alpha) \doteq \beta_n^t(\alpha), \quad \pi_n(h^{abs}h) \doteq \pi_n(lh). \tag{4.17}$$

Intuitively, π_n is concatenation of β_n^t. Each π_n, similar to π_n^{abs}, is in prefix-suffix form, i.e., $\pi_n = \pi_n(0), \dots, \pi_n(lh) \left(\pi_n(lh+1), \dots, \pi_n(h^{abs}h)\right)^\omega$. Note that $\pi_n(t)$ is well-defined for all t and is a valid T-path.

If $(\mathcal{R}, T, h, x_I^t, x_G^t, X_I^t, X_G^t)$ has no solutions for some t, we roll back and update

$$x_G^{t-1}(\mathcal{R}_n) \leftarrow x_G^{t-1}(\mathcal{R}_n) \setminus \{\beta_n^{t-1}(h)\}. \tag{4.18}$$

Algorithm 4.1 Hierarchical algorithm

```
1: $input$: $(T, \mathcal{R}, S_0, \mu, 0), h^{abs}, h$
2: $Counter\_Examples \leftarrow \{\}$
3: $T^{abs} \leftarrow abstract(T)$
4: $top$:
5: if $is\_high\_level\_feasible$ then
6:     $\{\pi_1^{abs}, \dots, \pi_N^{abs}\} = high\_level(T^{abs}, \mathcal{R}, S_0^{abs}, \mu, 1, Counter\_Examples)$
7: else
8:     return Infeasible
9: $t \leftarrow 0$
10: $x_I^t(\mathcal{R}_n) \leftarrow S_0(\mathcal{R}_n)$
11: while $t < h^{abs}$ do
12:     $X_I^t(\mathcal{R}_n) \leftarrow \pi_n^{abs}(t)$
13:     $x_G^t(\mathcal{R}_n) \leftarrow X_G^t(\mathcal{R}_n) \leftarrow \pi_n^{abs}(t+1)$
14: $rollback$:
15:     if $is\_GMRPP\_feasible$ then
16:         $\{\beta_n^t\} = GMRPP(\mathcal{R}, G, h, x_I^t, x_G^t, X_I^t, X_G^t)$
17:         $x_I^{t+1}(\mathcal{R}_n) \leftarrow \beta_n^t(h)$
18:         $t \leftarrow t + 1$
19:     else
20:         if $t > 0$ then
21:             $x_G^t(\mathcal{R}_n) \leftarrow x_G^{t-1}(\mathcal{R}_n) \setminus \{x_I^t(\mathcal{R}_n)\}$
22:             $t \leftarrow t - 1$
23:             goto $rollback$
24:         else
25:             $Counter\_Examples \leftarrow Counter\_Examples \cup \{\pi_1^{abs}, \dots, \pi_N^{abs}\}$
26:             goto $top$
27: $\pi_n = concatenate(\beta_n^0, \dots, \beta_n^{h^{abs}})$
28: return $\{\pi_1, \dots, \pi_N\}$
```

We call an abstract plan *infeasible* if instance $(T, \mathcal{R}, h, x_I^0, x_G^0, X_I^0, X_G^0)$ has no solutions. In that case, we generate a counter example that prevents the same abstract plan to be generated. Details of this process are explained in Section4.2.4.

4.2.4. Counter Examples

Given a collection of T^{abs}-paths $\{\tilde{\pi}_1^{abs}, \dots, \tilde{\pi}_N^{abs}\}$, assume the lower level algorithm failed. In that case, the following constraints are generated and added to the higher level:

$$\neg \left(\bigwedge_{n} \left(\bigwedge_{t} \pi_n^{abs}(t) = \tilde{\pi}_n^{abs}(t) \right) \right) \tag{4.19}$$

Constraint (4.19) imposes that the same abstract plan will not be encountered again. However, this method removes only one abstract plan at a time, which might be inefficient. Algorithm would converge faster if a number of infeasible abstract plans can be eliminated all at once, similar to Irreducibly Inconsistent Set idea in [95]. When lower level fails for an abstract plan $\{\tilde{\pi}_1^{abs}, \ldots, \tilde{\pi}_N^{abs}\}$, we generate instances $(\mathcal{R}, T, h, x_I^t, x_G^t, X_I^t, X_G^t)$ for all t such that $x_I^t(\mathcal{R}_n) = X_I^t(\mathcal{R}_n) = \tilde{\pi}_n^{abs}(t)$ and $x_G^t(\mathcal{R}_n) = X_G^t(\mathcal{R}_n) = \tilde{\pi}_n^{abs}(t)$. If $(\mathcal{R}, T, h, x_I^t, x_G^t, x_S^t)$ is infeasible, generate the following constraint to prevent such an abstract step from being generated in the future:

$$\bigwedge_{i} \left(\left(\bigwedge_{n} \pi_n^{abs}(i) = \tilde{\pi}_n^{abs}(t) \right) \implies \neg \left(\bigwedge_{n} \pi_n^{abs}(i+1) = \tilde{\pi}_n^{abs}(t+1) \right) \right). \tag{4.20}$$

Additionally, if more than $|V_i|$ robots are assigned to abstract state v_i^{abs} at any time, it is obvious that collisions cannot be avoided. We impose appropriate constraints to prevent such trivial counter examples.

4.2.5. Correctness of the Hierachical Method

Following proposition shows that the hierarchical method proposed in Section 4.2 is sound for synchronous executions.

Theorem 4.4. *Given a Problem 4.1 instance $(T, \mathcal{R}, S_0, \mu, 0)$ where μ is a cLTL$+_{\setminus\bigcirc}$ formula, assume the collection $\Pi = \{\pi_1, \ldots, \pi_N\}$ is generated by Algorithm 4.1. Then*

Π *is a solution to* $(T, \mathcal{R}, S_0, \mu, 0)$*, that is,* Π *are collision-free,* $\pi_n(0) = S_0(\mathcal{R}_n)$ *for all* $\mathcal{R}_n \in \mathcal{R}$*, and* $\Pi \models_0 \mu$*.*

Proof of Theorem 4.4 can be found in Appendix A.

4.2.6. Handling Asynchrony

This section shows how to deal with asynchrony. First, a small modification to Algorithm 4.1 necessary to solve Problem 4.1 for $\tau = 1$ is presented. Next, we show that, Problem 4.1 can be solved for arbitrary τ under mild assumptions.

Given $(T, \mathcal{R}, S_0, \mu, 0)$, assume an abstract plan is generated at the higher level. For all t and all pairs of $n, m \in [N]$, we enforce in each GMRPP

$$\beta_n(t+1) \neq \beta_m(t). \tag{4.21}$$

With this modification, when the asynchrony between robots is 1-bounded, these paths can be executed without collisions in an open-loop fashion, no communication or sensing needed at run-time. Furthermore, as shown in the following proposition, any 1-bounded asynchronous execution of collection $\{\pi_1, \ldots, \pi_N\}$, generated according to (4.17), would satisfy the specification μ.

Theorem 4.5. *Given a Problem 4.1 instance* $(T, \mathcal{R}, S_0, \mu, 1)$*, assume the collection* $\Pi = \{\pi_1, \ldots, \pi_N\}$ *is generated by Algorithm 4.1 where 4.21 is enforced in the lower level solution. Then* Π *is a solution to* $(T, \mathcal{R}, S_0, \mu, 1)$*, that is,* Π *are collision-free,* $\pi_n(0) = S_0(\mathcal{R}_n)$ *for all* $\mathcal{R}_n \in \mathcal{R}$*, and* $\Pi \models_1 \mu$*.*

Proof of Theorem 4.5 can be found in Appendix A.

Remark 4.3 (Termination). *Algorithm 4.1 is guaranteed to successfully terminate as the number of abstract plans for a given h^{abs} is finite. If a solution does not exist for a certain h^{abs}, the higher-level problem will eventually become infeasible as more counter examples are generated and the algorithm will terminate.*

Next, we show that, under mild assumptions, these trajectories can be implemented such that Problem 4.1 can be solved for arbitrary τ.

4.2.7. Generalization to Arbitrary Asynchrony

Assume all robots can communicate with each other and can indefinitely stay in any state, i.e., $(v, v) \in \mathcal{E}$ for all $v \in V$. Also assume that paths generated at the low level satisfy (4.21). Specification μ would be satisfied for all τ by $\{\pi_1, \dots, \pi_N\}$ when all robots use the following execution policy. If $\pi_n(t) = \pi_m(t')$ for some $t > t'$, robot $\mathcal{R}_n$ does not enter state $\pi_n(t)$ until robot $\mathcal{R}_m$ reaches $\pi_m(t' + 1)$. Otherwise, $\mathcal{R}_n$ moves to the subsequent state on its path. Note that, generated paths might not be collision-free under τ-bounded asynchrony. Nonetheless, the policy above would prevent collisions and would not result in deadlock as shown in [58]. We further require that robots 'synchronize at abstract steps', meaning that robot $\mathcal{R}_n$ move to $\pi_n(ht + 1)$ only after all robots $\mathcal{R}_m$ reach $\pi_m(ht)$.

Note that the complexity of this hierarchical method does not depend on τ. The higher level problem is solved for $\tau = 1$ and this is enough to satisfy the specification for any τ as long as robots avoid collisions and synchronize at abstract steps.

4.2.8. Examples

In this section, we demonstrate the efficacy of the hierarchical method by comparing its performance to both the robust encodings in Section4.1 and to [95]. All experiments are run on a laptop with 2.5 GHz Intel Core i7 and 16 GB RAM. Gurobi [33] is used as the underlying ILP solver. Our code is accessible at `https://github.com/sahiny/cLTL-hierarchical`.

We borrow the multirobot scenario from [95] and compare the performance of SMC-based method of [95] with the methods presented in Section 4. Results show that the non-hierarchical method of Section 4.1 can only solve the problem up to $N = 2$ robots because it suffers from the size of the transition system and consequently long solution horizon. On the other hand, the hierarchical method in Section 4.2 performs better than [95].

Example 1:

Assume N robots share the same workspace that is illustrated in Fig. 4.2. For each trial, robots are randomly initialized from the region marked with x_I, and specifications are given as:

$$\mu = \Box\Diamond[r_1, N] \wedge \Box\Diamond[r_2, N/2] \wedge \Box\Diamond[r_3, N/2]. \tag{4.22}$$

In words, we require all robots to regularly (infinitely many times) meet at r_1. Similarly, at least half of the robots should regularly meet both at r_2 and r_3. In [95], robot dynamics are modeled as chains of integrators and a satisfiability modulo convex (SMC) programming based method is proposed. Non-hierarchical method in Section 4.1 and the hierarchical method Section 4.2 in do not directly handle continuous dynamics. Hence, we grid the workspace into 30 by 30 squares of same size. Robots are allowed to move horizontally or vertically to neighboring states or stay in their current position. Note that this behavior is consistent with the continuous dynamics. The abstract transition system for this example is illustrated in Figure 4.4, where transitions are shown with solid black arrows. We solve the problem for increasing N. Computation times averaged over 10 trials are shown in Table 4.2. SMC-based method in [95] can solve the problem only up to $N = 5$ robots under 30 minutes as it can be seen from the second column. Hierarchical method proposed in Section 4.2, on the other hand, can solve the same problem up to $N = 12$ robots.

Example 2:

We then modify the specifications and add the additional constraint that region marked with r_3 should be empty until both g_1 and g_2 are populated with at least one robot at the same time:

$$\mu' = \mu \wedge (\neg[r_3, 1]\, \mathcal{U}\ ([g_1, 1] \wedge [g_2, 1])) \tag{4.23}$$

Note that [95] cannot handle arbitrary cLTL+$_{\backslash\bigcirc}$formulas and expressing the same specification using regular LTL is not trivial. While any cLTL+$_{\backslash\bigcirc}$formula can be transformed into regular LTL, as shown in Section 2.1.3, the length of the LTL formula specifying the same task could be exponentially longer, significantly increasing the

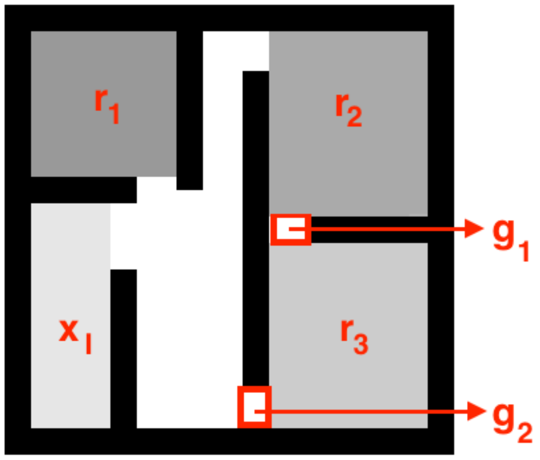


Figure 4.2: A simple workspace. Taken from [95]

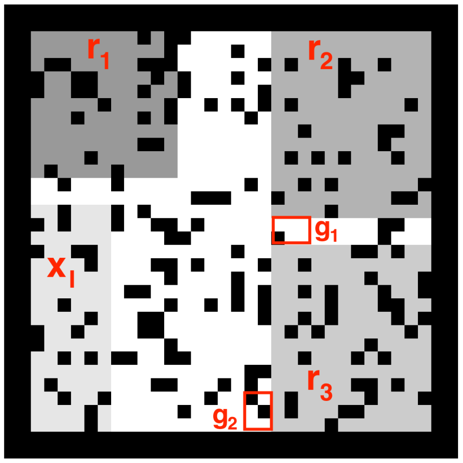


Figure 4.3: A sample workspace with randomly generated obstacles

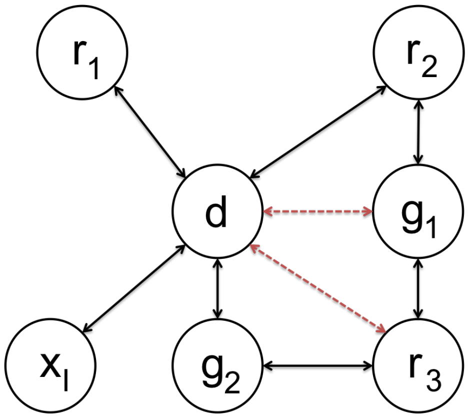


Figure 4.4: Coarsest stutter bisimulations of the workspaces. Transition system obtained for Figure 4.2 is shown with solid arrows, and additional transitions for Figure 4.3 are shown in dashed arrows

computation times. Computation times for varying N are shown in Table 4.2.

Example 3:

Next, we keep x_I, r_1, r_2, r_3, g_1 and g_2 as they are and randomly select 20% of the states as obstacles. The abstract transition system for this example is illustrated again in Figure 4.4, where the difference from Example 1 is the addition of two dashed arrows. Computation times for varying number of robots, and both specifications μ and μ' are

$$\begin{tikzcd}
 & r_1 \arrow[dr, leftrightarrow] & & r_2 \arrow[dll, leftrightarrow] \arrow[d, leftrightarrow] \\
 & & d \arrow[r, leftrightarrow, dashed] \arrow[dr, leftrightarrow, dashed] \arrow[dl, leftrightarrow] \arrow[d, leftrightarrow] & g_1 \arrow[d, leftrightarrow] \\
 & x_I & g_2 \arrow[r, leftrightarrow] & r_3
\end{tikzcd}$$

Table 4.2: Run-time comparison of different implementations (seconds)

	SMC-based [95]	**Hierarchical Section4.2**				**Non-hierarchical Section4.1**
	Fig 1	Fig 1		Fig 2		Fig 1
N	(μ)	(μ)	(μ')	(μ)	(μ')	(μ)
4	444.35	92.52	121.69	95.16	86.85	timeout
6	timeout	236.74	439.41	199.24	242.35	timeout
8	timeout	507.97	619.02	664.94	729.58	timeout
10	timeout	801.64	1665.95	1139.82	1275.62	timeout
12	timeout	1727.47	timeout	1499.17	timeout	timeout

again shown in Table 4.2. A video simulating the synthesized plans with synchronized robots can be seen at `https://www.youtube.com/watch?v=SrPDQMRmcNU`. We then assume same plans are executed asynchronously. At each step robots are delayed with $p = 0.3$ probability. Using the policy proposed in Section 4.2.7, robots are able to satisfy the specifications while avoiding collisions, as it can be seen from `https://www.youtube.com/watch?v=xO8xK9pXUKI`.

4.3. Summary

In this chapter, we proposed two different methods to solve Problem 4.1. While the non-hierarchical method is shown to be partially complete, its complexity depends on the synchronization error bound, thus, its scalability is limited for large syncronization error bounds. On the other hand, the hierarchical method's complexity does not depend on the synchronization error bound. Moreover, due to the use of abstraction, the hierarchical method can scale to very large transition systems. However, as a trade-off, the specifications are limited to cLTL$+_{\backslash\bigcirc}$ and the robots need to be able to stay indefinitely at any state for solutions to be correct. Furthermore, there is another trade-off between solving logic constraints and path planning while using the hierarchical method. Using a smaller abstraction, high-level plans can be generated faster and more complex specifications can be handled. On the other hand, a more refined abstraction can be used if lower-level path generation is the bottleneck, as it results in easier multirobot reachability problems.

Chapter 5.

Multirobot Plan Execution

We discussed the possible effects of synchronization errors in Chapter 4 and proposed methods to generate robust solutions by assuming that the synchronization errors are bounded. In real-life scenarios, the robots can move on their individual paths with different and time-varying speeds and their speed profiles are not known a priori. Therefore, it might not be possible to know or limit the synchronization error in practice.

Let us focus our attention to the following simple case. Given a collection of paths, one for each robot, devise a distributed protocol so that the robots are guaranteed to reach their targets and avoid all collisions along the way. We call this the *multirobot plan execution* problem. In fact, we encountered this problem in Section 4.2.7, and presented a potential *execution policy*, which is taken from [58], to prevent collisions and deadlocks.

Collision and deadlock prevention methods can be divided into two main groups.

In the first group, robots are allowed to replan their paths at run-time [72, 91, 107]. In this case, simpler path planning algorithms can be used, leaving the burden of collision avoidance to the run-time controllers. However, this approach might lead to deadlocks in densely crowded environments. Moreover, when the specifications are complex, changing paths might even lead to violations of the specifications. Therefore, replanning paths on run-time is not always feasible.

Alternatively, collisions and deadlocks can be avoided without needing to replan on run-time [22, 58, 81, 84, 87, 123, 124]. For instance, if the synchronization errors can be bounded, [22] and Section 4.1 show how to synthesize paths that are collision and deadlock-free. This is achieved by overestimating the positions of robots and treating them moving obstacles. However, this is a conservative approach as the burden of collision and deadlock avoidance is moved to the offline planning part.

In [58], authors provide a control policy, which is shown to be collision and deadlock-free under mild conditions on the collection of paths. This method is based on finding a fixed ordering of the robots for all possible conflicts. Such a fixed ordering prevents collisions and deadlocks, however, it is limiting as the performance of the multirobot system depends highly on the exact ordering. If one of the robots experiences a failure at run-time and starts moving slowly, it might become the bottleneck of the whole system. In fact, we demonstrate the effects of such a scenario on the system performance and provide numerical results that show the robustness of our method.

When the collection of paths are known a priori, one can also find all possible collision and deadlock configurations, and prevent the system from reaching those. For instance, distributed methods in [123] and [124] find deadlock configurations by abstracting robot paths into a edge-colored directed graph. However, this abstaction step might be conservative. Imagine a long passage which is not wide enough to fit more than one robot, and two robots crossing this passage in the same direction. The entire passage would be abstracted as a single node, and even though robots can enter the passage at the same time and follow each other safely, they would not be allowed to do so. Instead, robots have to wait for the other to clear the entire passage before entering. Moreover, [124] require that no two nodes in the graph are connected by two or more different colored edges. This strong restriction limits the method's applicability to classical multirobot path execution problems where robots move on a graph and same two nodes might be connected with multiple edges in each direction.

As connectivity and autonomous capabilities of vehicles improve, cooperative intersection management problems draw significant attention from researchers [1, 14, 26, 125]. These problems are similar to MRPE problem as both require coordinating multiple vehicles to prevent collisions and deadlocks. Compared to traditional traffic light-based methods, cooperative intersection management methods offer improved safety, increased traffic flow and lower emissions. We refer the reader to [17] for a recent survey on this topic and main solution approaches. Although they seem similar, the setting of intersection management problems are tailored specifically for the existing road networks, and thus, cannot be easily generalized to MRPE problems where robots/vehicles might be moving in non-structured environments.

The key insight of the chapter is to recast the MRPE problem as a resource allocation problem. There are similar methods such as [81], which requires a centralized controller, and [84], which needs cells to be large enough to allow collision-free travel of up to two vehicles, instead of only one. We base our method on the well-known drinking philosopher algorithm [15], an extension of the well-known dining philosophers problem [24]. We show that any existing DrPP solution can be used to solve the MRPE problem if drinking sessions are constructed carefully. However, such methods require strong conditions on a collection of paths to hold, and limit the amount of concurrency in the system. To relax the conditions and to improve the performance, we provide a novel approach by taking the special structure of MRPE problems into account. We show that our method is less conservative than the naive approach, and provide numerical results to confirm the theoretical findings. Our approach leads to control policies that can be deployed in a distributed form.

5.1. Multirobot Plan Execution Problem

We first define the multirobot plan execution problem formally: Let a set $\mathcal{R} = \{\mathcal{R}_1, \ldots, \mathcal{R}_N\}$ of robots share a workspace that is partitioned into set $\mathcal{V}$ of discrete cells. Two robots are said to be in *collision* if they occupy the same cell at the same time. We assume that a finite path is given for each robot, and π_n denotes the path associated with $\mathcal{R}_n$. We use $\pi_n end$ and $curr(r_n)$ to denote the final cell of π_n and the number of successful transitions completed by $\mathcal{R}_n$, respectively. We also define $next(\mathcal{R}_n) \doteq curr(r_n) + 1$. The motion of each robot is governed by a *control policy*, which issues one of the two commands at every time step: (1) *STOP* and (2) *GO*. The *STOP* action forces a robot to stay in its current cell. If the *GO* action is chosen, the robot starts moving. This robot might or might not reach to the next cell within one time step, however, we assume that a robot eventually progresses if *GO* action is chosen constantly. This non-determinism models the uncertainities in the environment, such as battery levels or noisy sensors/actuators, which might lead to robots moving faster or slower than intended. We now formally define the problem we are interested in solving:

Problem 5.1. *Given a collection* $\Pi = \{\pi_1, \ldots \pi_N\}$ *of paths, design a contol policy for each robot such that all robots eventually reach their final cells while avoiding collisions.*

There are many control policies that can solve Problem 5.1. For the sake of performance, policies that allow more concurrent behavior are preferred. In the literature, two metrics are commonly used to measure the performance: *makespan (latest arrival time)* and *flowtime (total arrival times)*. Given a set of robots $\mathcal{R} = \{\mathcal{R}_1, \ldots, \mathcal{R}_N\}$, if robot $\mathcal{R}_n$ takes t_n time steps to reach its final state, makespan and flowtime values are given by $\max_{1 \leq n \leq N} t_n$ and $\sum_{n=1}^{N} t_n$, respectively. These values decrease as the amount of concurrency increases. However, it might not be possible to minimize both makespan and flowtime at the same time, and choice of policy might depend on the application.

We reformulate Problem 5.1 as an instance of drinking philosophers problem. For the sake of completeness, this problem is explained briefly in Section 5.2.

5.2. Drinking Philosophers Problem

The drinking philosophers problem is a generalization of the well-known *dining philosophers problem* proposed by [24]. These problems capture the essence of conflict resolution, where multiple resources must be allocated to multiple processes. Given a set of processes and a set of resources, it is assumed that each resource can be used by at most one process at any given time. In our setting, processes and resources correspond to robots and discrete cells that partition the workspace, respectively. Similar to mutual exclusive use of the resources, any given cell can be occupied by at most one robot to avoid collisions. In the DrPP setting, processes are called *philosophers*, and shared resources are called *bottles*. A philosopher can be in one of the three *states*: (1) *tranquil*, (2) *thirsty*, or (3) *drinking*. A *tranquil* philosopher may stay in this state for an arbitrary period of time or *become thirsty* at any time it wishes. A thirsty philosopher *needs* a non-empty subset of bottles to drink from. This subset, called *drinking session*, is not necessarily fixed, and it could change over time. After acquiring all the bottles in its current drinking session, a thirsty philosopher *starts drinking*. When it no longer needs any bottles, after using them for a finite time, the philosopher goes back to tranquil state. The goal of the designer is to find a set of rules for each philosopher for acquiring and releasing bottles. A desired solution would have the following properties:

- *Liveness:* A thirsty philosopher eventually starts drinking. In our setting liveness implies that each robot is eventually allowed to move.
- *Fairness:* There is no fixed priority or partial ordering of philosophers or bottles

and the same set of rules apply to all philosophers. In multirobot setting, fairness indicate that all robots are treated equally.

- *Concurrency:* Any pair of philosophers must be allowed to drink at the same time, as long as they drink from different bottles. Analogously, no robot waits unnecessarily if it wants to move to an empty cell.

We base our method on the DrPP solution proposed in [31]. For the sake of completeness, we provide a brief summary of their solution, but refer the reader to [31] for the proof of correctness and additional details.

Each philosopher has a unique integer id and keeps track of two non-decreasing integers: session number s_num and the highest received session number max_rec. These integers are used to keep a strict priority order between the philosophers. Conflicts are resolved according to this order, in favor of the philosopher with the higher priority. To ensure liveness and fairness, this priority order changes according to the following rules.

Let p and r be two philosophers and b be a bottle shared between p and r. Define req_b as the request token associated with b. It is said that p *has higher priority than* r (denoted $p \prec r$) if and only if $s\_num_p < s\_num_r$, or $s\_num_p = s\_num_r$ and $id_p < id_r$. That is, smaller session number indicates higher priority, and in the case of identical session numbers, philosopher with the smaller id has the higher priority. Assume that p needs b (denoted $need_p(b)$) to start drinking and does not currently hold b (denoted $\neg hold(b)$). Then, p sends the message (req_b, s_num_p, id_p) to r. Upon receiving such a message, r releases b if (i) r does not need b or (ii) r is not drinking and $p \prec r$. If r does not immediately release b, then b is released once r no longer needs it. All philosophers are initialized in tranquil state with $s\_num_p = max\_rec_p = 0$ and follow the rules in Algorithm 5.1 to satisfy the aforementioned requirements.

Algorithm 5.1 Drinking Philosopher Algorithm by [31]

```
R1: $becoming\_thirsty$ with session $\mathcal{S}$
    for each bottle $b \in \mathcal{S}$ do $need_p(b) \leftarrow true$
    $s\_num_p \leftarrow max\_rec_p + 1$
R2: start $drinking$
    when holding all needed bottles do
    become $drinking$
R3: $becoming\_tranquil$, honoring deferred requests
    for each consumed bottle $b$ do
    $[need_p(b) \leftarrow false;$
    if $hold_p(req_b)$ then $[Send(b); hold_p(b) \leftarrow false]$
R4: $requesting\ a\ bottle$
    when $need_p(b); \neg hold_p(b); hold_p(req_b)$ do
    $Send(req_b, s\_num_p, id_p); hold_p(req_b) \leftarrow false$
R5: $receiving\ a\ request$ from $r$, resolving a conflict
    upon reception of $(req_b, s\_num_r, id_r)$ do
    $hold_p(req_b) \leftarrow true;$
    $max\_rec_p \leftarrow \max(max\_rec_p, s\_num_r)$
    if
    1) $\neg need_p(b)$ or,
    2) $\big(p$ is $thirsty$ and $(s\_num_r, id_r) < (s\_num_p, id_p)\big)$
        $[Send(b); hold_p(b) \leftarrow false]$
R6: $receive\ bottle$
    upon reception of $b$ do
        $hold_p(b) \leftarrow true$
```

5.3. Multirobot Plan Execution as a Drinking Philosophers Problem

In this section we recast the multirobot plan execution problem as an instance of drinking philosophers problem. We first show that naive reformulation using existing DrPP solutions leads to conservative control policies. We then provide a solution that is based on Algorithm 5.1.

Given a set $\mathcal{V} = \{v_1, \ldots, v_{|\mathcal{V}|}\}$ of cells and a collection $\Pi = \{\pi_1, \ldots \pi_N\}$ of paths, cells that appear in more than one path are called *shared.* We denote the set of shared cells by $\mathcal{V}_{shared}$, and define the set of *free cells* as $\mathcal{V}_{free} \doteq \mathcal{V} \setminus \mathcal{V}_{shared}$. To avoid collisions, a shared cell must be occupied at most by one robot at any given time. Inspired by this mutual exclusion requirement, we see the robots as philosophers and shared cells as the bottles.

Given any two arbitrary robots, we define a bottle for each cell that is visited by both. For example, if the k^{th} cell $v_k \in \mathcal{V}$ is visited both by $\mathcal{R}_m$ and $\mathcal{R}_n$, we define the bottle $b^k_{m,n}$. We denote the set of cells visited by both $\mathcal{R}_m$ and $\mathcal{R}_n$ by $\mathcal{V}_{m,n} \doteq \{v \mid \exists\ t_m, t_n : \pi_m t_m = \pi_n t_n = v \in \mathcal{V}_{shared}\}$. It must be noted that for a shared cell $v_k \in \mathcal{V}_{m,n}$, there exists a single bottle shared between $\mathcal{R}_n$ and $\mathcal{R}_m$, and both $b^k_{m,n}$ and $b^k_{n,m}$ refer to the same object. We use $\mathcal{B}_{m,n}$ and $\mathcal{B}_m$ to denote the set of all bottles $\mathcal{R}_m$ shares with $\mathcal{R}_n$ and with all other robots, respectively. With slight abuse of notation, we use $\mathcal{B}_m(V)$ to denote all the bottles associated with the cells in $V \subseteq \mathcal{V}$ that $\mathcal{R}_m$ share with others, that is, $\mathcal{B}_m(V) = \{b^k_{m,n} \in \mathcal{B}_m \mid v_k \in V\}$. We use the following example to illustrate the concepts above.

Example 5.1. *In the scenario depicted in Figure5.1, the robot $\mathcal{R}_1$ shares one bottle with $\mathcal{R}_2$, $\mathcal{B}_{1,2} = \{b^2_{1,2}\}$, three bottles with $\mathcal{R}_4$, $\mathcal{B}_{1,4} = \{b^1_{1,4}, b^2_{1,4}, b^4_{1,4}\}$, and one bottle with $\mathcal{R}_5$, $\mathcal{B}_{1,5} = \{b^6_{1,5}\}$. The set $\mathcal{B}_1$ is the union of these three sets, as $\mathcal{R}_1$ does not share any bottles with $\mathcal{R}_3$. Given $V = \{v_2\}$, then $\mathcal{B}_1(V) = \{b^2_{1,2}, b^2_{1,4}\}$.*

Bottles are used to indicate the priority order between robots over shared cells. For instance, if the bottle $b^k_{m,n}$ is currently held by robot $\mathcal{R}_m$, then $\mathcal{R}_m$ has a higher priority than $\mathcal{R}_n$ over the shared cell v_k. Note that, this order is dynamic as bottles are sent back and forth. However, as long as a philosopher is drinking, it would not send any of the bottles in its current drinking session. Then, collisions can be prevented simply by the following rule: *"to occupy a shared cell v_k, the robot $\mathcal{R}_n$ must be drinking from all the bottles in $B_n(v_k)$."* Upon arriving at a free cell, a drinking robot would become tranquil. If $\mathcal{R}_n$ is drinking from all the bottles in $B_n(v_k)$, it has a higher priority than all other robots over v_k. Moreover, $\mathcal{R}_n$ would keep all of the bottles in its current drinking session and would be the only robot allowed to occupy v_k until it stops drinking. Therefore, the aforementioned rule prevents collisions. However, this is not sufficient to ensure that all robots reach their final cells. Without the introduction of further rules, robots might end up in a *deadlock.* We formally define deadlocks as follows:

Definition 5.1. *A **deadlock** is any configuration where a subset of robots, which have not reached their final cell, wait cyclically and choose $STOP$ action indefinitely.*

To exemplify the insufficiency of the aforementioned rule, imagine the scenario shown in Fig. 5.1. Robots $\mathcal{R}_1$ and $\mathcal{R}_4$ traverse the neighboring cells v_1 and v_2 in the opposite order. Assume $\mathcal{R}_4$ is at v_4 and wants to proceed into v_2, and, at the same time, $\mathcal{R}_1$ wants to move into v_1. Using the aforementioned rule, robots must be drinking from the associated bottles in order to move. Since they wish to drink from different bottles, both robots would be allowed to start drinking. After arriving at v_1, $\mathcal{R}_1$ has to start drinking from $B_1(v_2)$ in order to progress any further. However, $\mathcal{R}_4$ is currently drinking from $b_{1,4}^2 \in B_1(v_2)$ and cannot stop drinking before leaving v_2. Similarly, $\mathcal{R}_4$ cannot progress, as $\mathcal{R}_1$ cannot release $b_{1,4}^1$ before leaving v_1. Consequently, robots would not be able to make any further progress, and would stay in drinking state forever.

5.3.1. Naive Formulation

We now show that deadlocks can be avoided by constructing the drinking sessions carefully. For the correctness of DrPP solutions, all drinking sessions must end in finite time. If drinking sessions are set such that a robot entering a shared cell is free to move until it reaches a free cell without requiring additional bottles along the way, then all drinking sessions would end in finite time. That is, if a robot is about to enter a segment which consists only of consecutive shared cells, it is required to acquire not only bottles associated with the first cell, but also all the bottles on that segment. To formally state this requirement, let $\mathcal{S}_n(t)$ denote the drinking session associated with the cell π_n^t for the robot $\mathcal{R}_n$. That is, to occupy π_n^t, the robot $\mathcal{R}_n$ should be drinking from all the bottles in $\mathcal{B}_n(\mathcal{S}_n(t))$. Now set

$$\mathcal{S}_n(t) = \{\pi_n^t, \dots \pi_n^{t'}\} \tag{5.1}$$

where $\pi_n^k \in \mathcal{V}_{shared}$ for all $k \in [t, t']$ and $\pi_n^{t'+1} \in \mathcal{V}_{free}$ is the first free cell after π_n^t. No other robot could occupy any of the cells in $\mathcal{S}_n(t)$ once $\mathcal{R}_n$ starts drinking. Constantly choosing the action GO, $\mathcal{R}_n$ would eventually reach the free cell $\pi_n^{t'+1}$ and stop drinking in finite time. If the drinking sessions are constructed as in (5.1), any existing DrPP solution, such as [15, 31, 112], can be used to design the control policies that solve Problem 5.1.

However, the control policies resulting from the aforementioned approach are conservative and lead to poor performance in terms of both makespan and flowtime. To illustrate, imagine the scenario shown in Fig. 5.1. To be able to move into v_1, $\mathcal{R}_1$ must be drinking from all the bottles associated with cells $\mathcal{B}_1(\{v_1, v_2, v_4, v_6\})$. Assume that $\mathcal{R}_1$ starts drinking and moves to v_1. If at this point in time, $\mathcal{R}_5$ wants to move into v_6, it would not be allowed to do so since $b_{1,5}^6 \in \mathcal{B}_1(\{v_1, v_2, v_4, v_6\})$ is held by $\mathcal{R}_1$. Note that, this is a conservative action as $\mathcal{R}_5$ cannot cause a deadlock by moving to v_6, as it moves to a free cell right after. To allow more concurrency, we propose the following modifications.

5.3.2. New Drinking State and New Rules

In this subsection, we propose a method based on Algorithm 5.1. In particular, we introduce a new drinking state for the philosophers, namely *insatiable*. This new state is used when robot moves from a shared cell to another shared cell. We also add an additional rule regarding this new state and modify the existing rule $R5$ of Algorithm 5.1:

```
R7: becoming insatiable with session $\mathcal{S}$
    become $insatiable$
    for each bottle $b \in \mathcal{S}$ do $need_p(b) \leftarrow true$
    for all other bottles $b$ do $need_p(b) \leftarrow false$

R'5: receiving a request from $r$, and resolving a conflict
    upon reception of $(req_b, s\_num_r, id_r)$ do
    $hold_p(req_b) \leftarrow true$;
    $max\_rec_p \leftarrow \max(max\_rec_p, s\_num_r)$
    if
      1. $\neg need_p(b)$ or,
      2. $p$ is $thirsty$ and
          a) $r$ is $thirsty$ and $(s\_num_r, id_r) < (s\_num_p, id_p)\big)$ or,
          b) $r$ is $insatiable$,
      3. $(p$ is $insatiable$ and $b \notin \mathcal{S}_p(curr(\mathcal{R}_p))$ and $(s\_num_r, id_r) < (s\_num_p, id_p)$
         $[Send(b); hold_p(b) \leftarrow false]$
```

In the naive formulation, drinking sessions are set such that a robot entering a shared cell is free to move until it reaches a free cell, without requiring additional bottles along the way. The insatiable state is intended to soften this constraint. Assume robot $\mathcal{R}_n$ wants to move to shared cell π_n^t, and the first free cell after π_n^t is $\pi_n^{t'+1}$ for some arbitrary $t' > t$, all the cells in between are shared. If $\mathcal{R}_n$ enters the first shared cell without acquiring all the bottles until $\pi_n^{t'+1}$, it would need to acquire those bottles at some point along the way. If $\mathcal{R}_n$ becomes thirsty to acquire those bottles, it risks losing the bottles it currently holds. If another robot $\mathcal{R}_m$ with a higher priority needs and receives the bottles associated with the cell $\mathcal{R}_n$ currently occupies, two robots might collide.

Insatiable state allows a robot to request new bottles without risking to lose any of the bottles it currently holds. In this state, the robot does not hold all the bottles it needs to start drinking, similar to thirsty state. The difference between two states is that an insatiable philosopher always has a higher priority than a thirsty philosopher regardless of their session numbers, and does not release any of the needed bottles under any circumstance.

The insatiable state and the rule $R7$ regarding its operation might lead to deadlocks without careful construction of drinking sessions. We now explain how to construct drinking sessions to avoid deadlocks.

5.3.3. Constructing Drinking Sessions

To compute drinking sessions, we first need to define a new concept called *Path-Graph*:

Definition 5.2. *The **Path-Graph** induced by the collection* $\Pi = \{\pi_1, \dots \pi_N\}$ *of paths is a directed edge-colored multigraph* $G_\Pi = (\mathcal{V}, \mathcal{E}_\Pi, \mathcal{C})$ *where* $\mathcal{V}$ *is a set of nodes, one* *per each cell in* Π, $\mathcal{E}_\Pi = \{(\pi_n t, c_n, \pi_n t+1) \mid \pi_n \in \Pi\}$ *is the set of edges, representing transitions of each path, and* $\mathcal{C} = \{c_1, \ldots, c_N\}$ *is the set of colors, one per each path (i.e., one per each robot).*

A Path-Graph is a graphical representation of a collection of paths, overlayed on top of each other. The nodes of this graph correspond to discrete cells that partition the workspace, and edges illustrate the transitions between them. Color coding of edges indicate which robot is responsible from a particular transition. In other words, if π_n has a transition from u to v, then there exists a c_n colored edge from u to v in G_Π, i.e., $(u, c_n, v) \in \mathcal{E}_\Pi$.

Path-Graphs are useful to detect possible deadlock configurations. Intuitively, deadlocks occur when a subset of robots wait cyclically for each other. We first show that such configurations correspond to a *rainbow cycle* in the corresponding Path-Graph. A rainbow cycle is a closed walk where no color is repeated. Let Π be a collection of paths and G_Π be the Path-Graph induced by it. Assume that a subset $\{\mathcal{R}_1, \ldots, \mathcal{R}_K\} \subseteq \mathcal{R}$ of robots are in a deadlock configuration such that $\mathcal{R}_n$ waits for $\mathcal{R}_{n+1}$ for all $n \in \{1, \ldots, K\}$ where $\mathcal{R}_{K+1} = \mathcal{R}_1$. That is, $\mathcal{R}_n$ cannot move any further, because it wants to move to the cell that is currently occupied by $\mathcal{R}_{n+1}$. Let v_n denote the current cell of $\mathcal{R}_n$. Since $\mathcal{R}_n$ wants to move from v_n to v_{n+1}, we have $e_n = (v_n, c_n, v_{n+1}) \in \mathcal{E}_\Pi$. Then, $\omega = \{(v_1, c_1, v_2), \ldots, (v_K, c_K, v_1)\}$ is a rainbow cycle of G_Π. For instance, there are two rainbow cycles in Fig. 5.1: $\omega_1 = \{(v_1, c_1, v_2), (v_2, c_4, v_1)\}$ and $\omega_2 = \{(v_2, c_1, v_4), (v_4, c_4, v_2)\}$.

The first idea that follows from this observation is to limit the number of robots

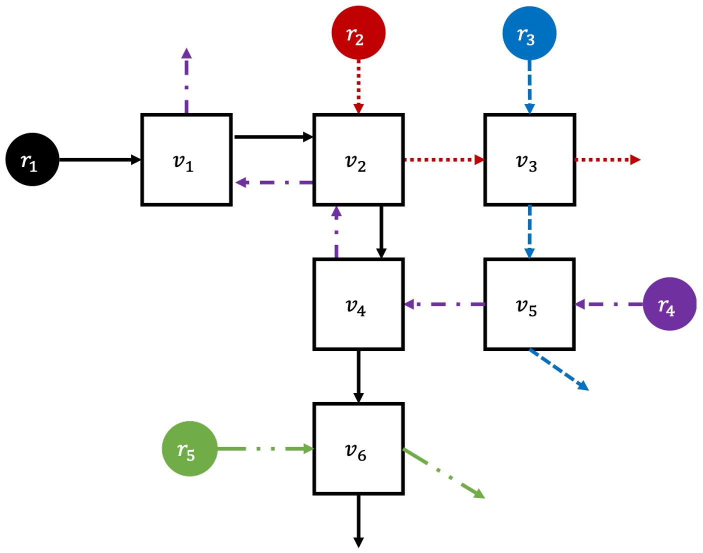


Figure 5.1: An illustrative example for multirobot path execution problem. Robots, each assigned a unique color/pattern pair, are initialized on free cells that are drawn as solid circles. Shared cells are shown as hollow black rectangles. Each path eventually reaches a free cell that is not shown for the sake of simplicity.

in each rainbow cycle to avoid deadlocks. However, this is not enough as rainbow cycles can intersect with each other and robots might end up waiting for each other to avoid eventual deadlocks. For instance, in the scenario illustrated in Fig. 5.1, let $\mathcal{R}_1$ and $\mathcal{R}_4$ be at v_1 and v_4, respectively. The number of robots in each rainbow cycles is limited to one, nonetheless, this configuration will eventually lead to a deadlock.

We propose Algorithm 5.2 to construct the drinking sessions, which are used to prevent such deadlocks. Given a collection Π of paths let $G_\Pi = (\mathcal{V}, \mathcal{E}_\Pi, \mathcal{C})$ denote its Path-Graph. We first define equivalence relation $\sim$ on $\mathcal{V}$ such that each node is equivalent only to itself. We then find all rainbow cycles in G_Π. Let $\mathcal{W}$ denote the set of all rainbow cycles. For each rainbow cycle $W \in \mathcal{W}$, we expand the equivalence relation $\sim$ by declaring all nodes in W to be equivalent. That is, if u and v are two nodes of the rainbow cycle W, we add the pair (u, v) to the equivalence relation $\sim$.

Algorithm 5.2 find_equivalence_classes

```
    Input $G_\Pi$ return $\tilde{G}_\Pi$
 1: $\sim \leftarrow \emptyset$
 2: for $u \in G_\Pi$ do
 3:     expand $\sim$ such that $(u, u) \in \sim$
 4: $\mathcal{W} \leftarrow$ find_rainbow_cycles($G_\Pi$)
 5: if $\mathcal{W} = \emptyset$ then
 6:     $\tilde{G}_\Pi \leftarrow G_\Pi$
 7:     return
 8: else
 9:     for $W \in \mathcal{W}$ do
10:         for $u, v \in W$ do
11:             expand $\sim$ such that $(u, v) \in \sim$
12:     $find\_equivalence\_classes(\tilde{G}_\Pi)$
```

Note that, due to transitivity of the equivalence relation, nodes of two intersecting rainbow cycles would belong to the same equivalence class. The relation $\sim$ partitions $\mathcal{V}$ by grouping the intersecting rainbow cycles together. We then find the quotient set $\mathcal{V}/\sim$ and define a new graph $\tilde{G}_\Pi = (\mathcal{V}/\sim, \tilde{\mathcal{E}}_\Pi, \mathcal{C})$ where $([u], c_m, [v]) \in \tilde{\mathcal{E}}_\Pi$ if $[u] \neq [v]$, and there exists $\alpha \in [u]$, $\beta \in [v]$ such that $(\alpha, c_m, \beta) \in \mathcal{E}_\Pi$. That is, we create a node for each equivalence class. We then add a c_m colored edge to $\tilde{G}_\Pi$ between the nodes corresponding $[u]$ and $[v]$ if there is a c_m colored edge in G_Π from a node in $[u]$ to a node in $[v]$. We repeat the same process with $\tilde{G}_\Pi$ in a recursive manner until no more rainbow cycles are found.

Proposition 5.1. *Algorithm 5.2 terminates in finite steps.*

Proof. Since all paths are finite, the number of nodes in the Path-Graph G_Π, $|\mathcal{V}|$, is finite. At each iteration, Algorithm 5.2 either finds a new graph $\tilde{G}_\Pi$ which has a smaller number of nodes, or returns G_Π. Therefore, Algorithm 5.2 is guaranteed to terminate at most in $|\mathcal{V}|$ steps. □

Remark 5.1. *Algorithm 5.2 needs to find all rainbow cycles of an edge-colored multi-graph at each iteration, which can be done in the following way. Given $G = (\mathcal{V}, \mathcal{E}, \mathcal{C})$, obtain $E \in \mathcal{V} \times \mathcal{V}$ from $\mathcal{E}$ by removing the coloring and replacing multiple edges between the same two nodes with a single edge. Then, find all simple cycles in the graph $(\mathcal{V}, E)$. Finally, check if these cycles can be colored as a rainbow cycle. As for the complexity of these steps, finding all simple cycles up to length N can be done $\mathcal{O}(N\mathcal{V}E)$ time [3], and deciding if a cycle can be rainbow colored can be posed as an exact set cover problem, which is NP-complete. This is essentially due to the fact that, in the worst-case, the number of cycles in a multi-graph can be exponential in the number of colors compared to the corresponding directed graph. However, the number of nodes decrease at each iteration of Algorithm 5.2, making computations easier. Moreover, while the worst-case complexity is high, these operations can usually be performed efficiently in practice.*

When the Algorithm 5.2 finds the fixed point, we set

$$\tilde{\mathcal{S}}_n(t) \doteq \mathcal{S}_n(t) \cap [\pi_n^t] \tag{5.2}$$

where $\mathcal{S}_n(t)$ is defined as in (5.1) and $[\pi_n^t]$ is the equivalence class of π_n^t. That is, $\mathcal{R}_n$ must be drinking from all the bottles in $\mathcal{B}_n(\tilde{\mathcal{S}}_n(t))$ to be able to occupy π_n^t. If $\tilde{\mathcal{S}}_n(t) = \emptyset$, $\mathcal{R}_n$ is allowed to occupy π_n^t regardless of its drinking state, as robots in free cells cannot lead to collisions or deadlocks.

Example 5.2. *Let G_Π be given as in Figure 5.1. After the first recursion of Algorithm 5.2, $[v_1] = \{v_1, v_2, v_4\}$ and $[v_i] = \{v_i\}$ for $i \in \{3, 5, 6\}$. After the second recur-*

sion, $[v_1] = \{v_1, v_2, v_3, v_4, v_5\}$ and $[v_6] = \{v_6\}$. No rainbow cycles are found after the second recursion, therefore, $\tilde{\mathcal{S}}_1(1) = \{v_1, v_2, v_4, v_6\} \cap \{v_1, v_2, v_3, v_4, v_5\} = \{v_1, v_2, v_4\}$.

Remark 5.2. *Sessions constructed by* (5.2) *are always contained in the sessions constructed by* (5.1)*. That is, when drinking sessions are found as in* (5.2)*, robots would need fewer bottles to move, and the resulting control policies would be more permissive.*

We now propose a control policy that prevents collisions and deadlocks when drinking sessions are constructed as in (5.2).

5.3.4. Control Strategy

We propose Algorithm 5.3 as a control policy to solve Problem 5.1. We first briefly explain the flow of the control policy, which is illustrated in Figure 5.2, and then provide more details. All robots are initialized in tranquil state. If the final cell is reached, *STOP* action is chosen as the robot accomplished its task. Otherwise, if a robot is in either tranquil or drinking state, the control policy chooses the action *GO* until the robot reaches to the next cell. When a robot moves from a free cell to a shared cell, it first becomes thirsty and the control policy issues the action *STOP* until the robot starts drinking. When moving between shared cells, a robot becomes insatiable if it needs to acquire additional bottles, and *STOP* action is chosen until the robot starts drinking again. When a robot's path terminates at a shared cell, it must be careful not arrive early and block others from progressing. Therefore, when a robot is about to move to a segment of consecutive shared cells which includes its final cell, it needs to wait for others to clear its final cell.

All robots are initialized in tranquil state. Let $\mathcal{R}_n$ be an arbitrary robot. Lines $1-2$ of Algorithm 5.3 ensure that $\mathcal{R}_n$ does not move after reaching its final cell. Otherwise, let π_n^t denote the next cell on $\mathcal{R}_n$'s path. If π_n^t is a free cell, the control policy chooses the *GO* action until the robot reaches π_n^{t+1} (lines $3-9$). When π_n^t is a shared cell, there are two possible options: (i) If there is no free cell between the next cell and the final cell of $\mathcal{R}_n$, i.e., $\pi_n^{end} \in \mathcal{S}_n(t)$ where $\mathcal{S}_n(t)$ is defined as in (5.1), the robot must wait for all other robots to clear this cell (lines $10-14$). This wait is needed, otherwise, $\mathcal{R}_n$ might block others by arriving and staying indefinitely at its final cell. When all others clear its final state, $\mathcal{R}_n$ can start moving again. (ii) If the final cell is not included in the drinking session, $\mathcal{R}_n$ checks its drinking state. If tranquil, $\mathcal{R}_n$ becomes thirsty with the drinking session $\tilde{\mathcal{S}}_n(t)$ and waits until it starts drinking to move to the next cell (lines $15-18$). When the robot starts drinking, it is allowed to move until it reaches π_n^t (lines $19-23$). Upon reachingπ_n^t, the robot checks π_n^{t+1}. If it is a shared cell, the robot becomes insatiable with $\mathcal{B}_n(\tilde{\mathcal{S}}_n(t) \cup \tilde{\mathcal{S}}_n(t+1))$ (lines $24-25$) and waits until it starts drinking again. Otherwise, robot moves until reaching π_n^{t+1} and updates its drinking state as tranquil (lines $26-31$).

We now show the correctness of Algorithm 5.3.

Theorem 5.1. *Given an instance of Problem 5.1, using Algorithm 5.3 as a control policy solves Problem 5.1 if*

1. *Initial drinking sessions are disjoint for each robot, i.e., $\tilde{\mathcal{S}}_m(0) \cap \tilde{\mathcal{S}}_n(0) = \emptyset$ for all m, n and*

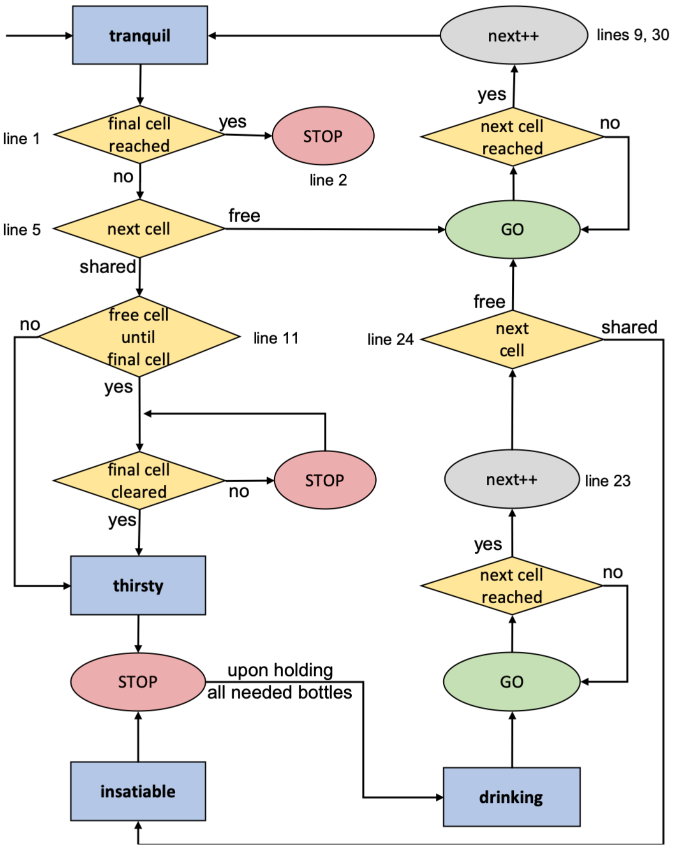


Figure 5.2: Flowchart of Algorithm 5.3.

2. *Final drinking sessions are disjoint for each robot, i.e.,* $\mathcal{S}_m(end) \cap \mathcal{S}_n(end) = \emptyset$ *for all* m, n *and*

3. *There exists at least one free cell in each* π_n*.*

As mentioned in Remark 5.2, constructing drinking sessions as in (5.1) leads to more conservative control policies. Furthermore, doing so also imposes stricter assumptions on the collection of paths due to the conditions (1) and (2) of Theorem 5.1. Due to larger drinking sessions, fewer collections would satisfy the condition that the initial drinking sessions must be disjoint for each robot.

Remark 5.3. *The control policy given in Algorithm 5.3 can be implemented by the robots in a distributed manner. In order to achieve this, we require the communication*

Algorithm 5.3 Control policy for $\mathcal{R}_n$

```
1:  if $\mathcal{R}_n$.is_final_cell_reached then
2:      $\mathcal{R}_n.STOP$
3:  else
4:      $t \leftarrow next(\mathcal{R}_n)$
5:      if is_free($\pi_n^t$) then
6:          while $\neg\mathcal{R}_n$.is_reached($\pi_n^t$) do
7:              $\mathcal{R}_n.GO$
8:          $next(\mathcal{R}_n) \leftarrow next(\mathcal{R}_n) + 1$
9:      else
10:         if $\pi_n^{end} \in \mathcal{S}_n(t)$ then
11:             while $\neg$cleared($\pi_n^{end}$) do
12:                 $\mathcal{R}_n.STOP$
13:         else if $\mathcal{R}_n$.is_tranquil then
14:             $\mathcal{R}_n$.get_thirsty($\mathcal{S}_n(t)$)
15:         else if $\mathcal{R}_n$.is_thirsty or $\mathcal{R}_n$.is_insatiable then
16:             $\mathcal{R}_n.STOP$
17:         else if $\mathcal{R}_n$.is_drinking then
18:             while $\neg\mathcal{R}_n$.is_reached($\pi_n^t$) do
19:                 $\mathcal{R}_n.GO$
20:             $next(\mathcal{R}_n) \leftarrow next(\mathcal{R}_n) + 1$
21:             if is_shared($\pi_n^{t+1}$) then
22:                 $\mathcal{R}_n$.get_insatiable($\mathcal{B}_n(\mathcal{S}_n(t) \cup \mathcal{S}_n(t+1))$)
23:             else
24:                 while $\neg\mathcal{R}_n$.is_reached($\pi_n^t$) do
25:                     $\mathcal{R}_n.GO$
26:                 $next(\mathcal{R}_n) \leftarrow next(\mathcal{R}_n) + 1$
27:                 $\mathcal{R}_n$.get_tranquil()
```

graph to be identical to the resource dependency graph. That is, if two robots visit a common cell, there must be a communication channel between them.

5.4. Examples

In this section, we compare our method, which is explained in Sections 5.3.2-5.3.4 and referred to as *Rainbow Cycle*, with the *Minimal Communication Policy (MCP)* of [58] using identical paths. To judge the improvement in the amount of concurrency better, we also provide comparisons with the *Naive* method which is explained in Section 5.3.1. Our implementation can be accessed from `https://github.com/sahiny/philosophers`.

To explain briefly, MCP prevents collisions and deadlocks by maintaining a fixed visiting order for each cell. A robot is allowed to enter a cell only if all the other robots, which are planned to visit the said cell earlier, have already visited and left the said state. It is shown that, under mild conditions on the collection of the paths, keeping this fixed order prevents collisions and deadlocks. We refer the reader to [58] for more details.

To capture the uncertainty in the robot motions, each robot is assigned a *delay probability*. When the action *GO* is chosen, a robot either stays in its current cell with this probability, or completes its transition to the next cell before the next time step.

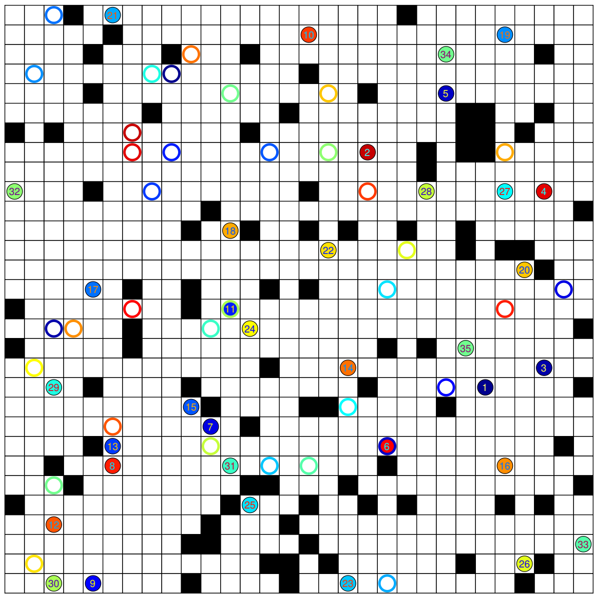


Figure 5.3: Randomly generated example. This example is named random1 and consists of 35 robots on a 30×30 grid with 10% blocked cells. Blocked cells are shown in black. Initial and final cells are marked with a solid and a hollow circle of a unique color, respectively.

Randomly Generated Examples

There are 10 MRPE instances in [58], labelled random 1-10, where 35 robots navigate in 4-connected grids of size 30×30. In each example, randomly generated obstacles block 10% of the cells, and robots are assigned random but unique initial and final locations. All control policies use the same paths generated by the Approximate Minimization in Expectation algorithm of [58]. Delay probabilities of robots are sampled from the range $(0, 1 - 1/t_{max})$. Note that, higher delay probabilities can be sampled as t_{max} increase, resulting in *slow moving* robots. Figure 5.4 reports the makespan and flowtime statistics averaged over 1000 runs for varying t_{max} values. The delay probabilities are sampled randomly for each run, but kept identical over different control policies. As expected, both makespan and flowtime statistics increase with t_{max}, as higher delay probabilities result in slower robots.

From Fig. 5.4, we first observe that the Rainbow Cycle DrPP based control policy always performs better than the Naive method. This is expected as drinking sessions for the Naive method, which are computed by (5.1), are always larger than the ones of Rainbow Cycle methods, which are computed by (5.2). Consequently, robots need more bottles to move, and thus, wait more. Moreover, Naive method requires stronger assumptions to hold for a collection of paths. For instance, only one random example satisfy the the assumptions in Theorem 5.1 for the Naive method, whereas this number increases to four for the Rainbow Cycle method. The example illustrated in Fig. 5.3 originally violates the assumptions, but this is fixed for both drinking based methods by adding a single cell into a robot's path. We here note that, the set of valid paths for MCP and DrPP algorithms are non-comparable. There are paths that satisfy the assumptions of one algorithm and violate the other, and vice versa.

We also observe that makespan values are quite similar for Rainbow Cycle and MCP methods, although MCP often performs slightly better in this regard. Given a collection of paths, the makespan is largely determined by the *"slowest"* robot, a robot with a long path and/or a high delay probability, regardless of the control policies. The makespan statistics do not necessarily reflect the amount of concurrency allowed by the control policies. Ideally, in the case of a slow moving robot, we want the control policies not to stop or slow down other robots unnecessarily, but to allow them move freely. The flowtime statistics reflect these properties better. From Figure 5.4, we see that flowtime values increase more significantly with t_{max} for MCP, compared to Rainbow Cycle policy. This trend can be explained with how priority

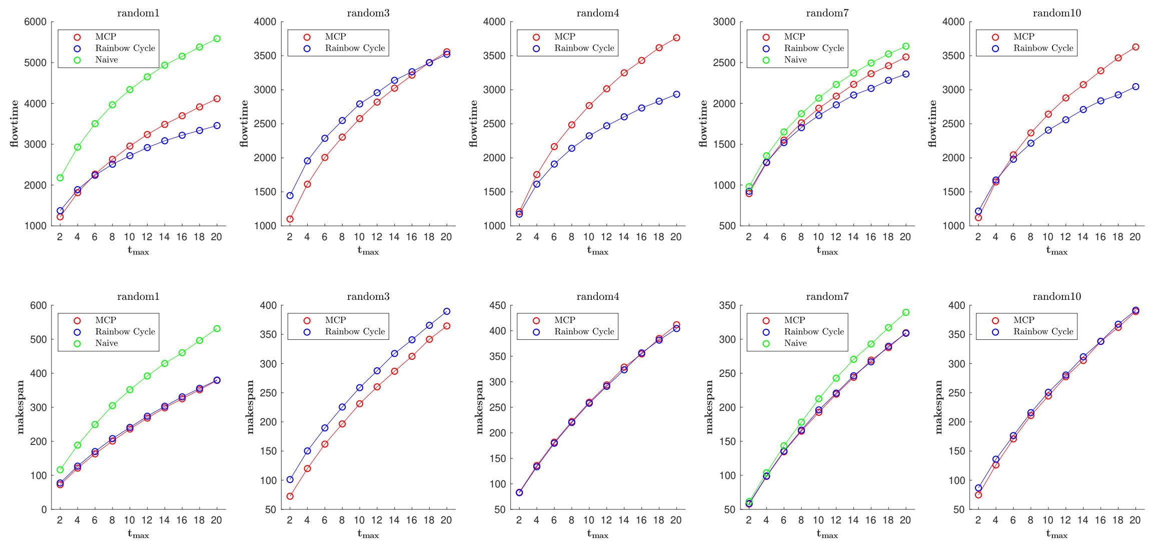


Figure 5.4: Makespan and flowtime statistics averaged over 1000 runs for the warehouse environment under varying t_{max} values. DrPP based method cannot be used in environments where the collection of paths violate the conditions in Theorem 5.1. Out of 10 randomly generated instances, Naive and Rainbow Cycle DrPP based methods can solve 2 and 5 instances, respectively.

orders are maintained in each of the algorithms. As the delay probabilities increase, there is more uncertainty in the motion of robots. MCP keeps a fixed priority order between robots, which might lead to robots waiting for each other unnecessarily. On the other hand, Rainbow Cycle dynamically adjusts this order, which leads to more concurrent behavior, hence the smaller flowtime values. The following illustrates this phenomenon with a simple example.

Makespan versus Flowtime

As mentioned earlier, [58] assumes that delay probabilities are known a priori, and computes paths to minimize the expected makespan. Once the paths are computed, the priority order between robots is fixed to ensure MCP policies are collision and

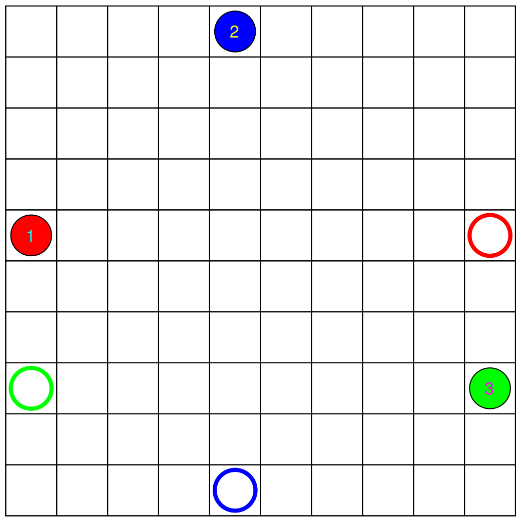


Figure 5.5: A simple example to show effects of a slow moving robot on makespan and flowtime. Robots $\mathcal{R}_1$, $\mathcal{R}_2$ and $\mathcal{R}_3$ are colored in red, blue and green, respectively. Initial and final cells of the robots are marked with solid and hollow circles of their unique color, respectively.

deadlock-free. We now provide a simple example to illustrate the effect of using inaccurate delay probabilities in the path planning process. Imagine 3 robots are sharing a 10 by 10 grid environment as shown in Figure 5.5. Assume that the delay probabilites for robots $\mathcal{R}_1$, $\mathcal{R}_2$ and $\mathcal{R}_3$ are known to be $\{0, 0.4, 0.8\}$, respectively. If we compute paths to minimize the expected makespan, resulting paths are straight lines for each robot. Paths π_1 and π_2 intersect at a single cell, for which $\mathcal{R}_1$ has a priority over $\mathcal{R}_2$. Similarly π_2 and π_3 also intersect at a single cell, for which $\mathcal{R}_2$ has a priority over $\mathcal{R}_3$. We run this example using inaccurate delay probabilities $\{0.8, 0.4, 0\}$ to see how the makespan and flowtime statistics are affected.

Over 1000 runs, makespan values are found to be 48.30 and 45.77 steps for MCP and Rainbow Cycle implementations, respectively. The makespan values are close because of the slow moving $\mathcal{R}_1$, which becomes the bottleneck of the system. Therefore, it is not possible to improve the makespan statistics by employing different control policies.

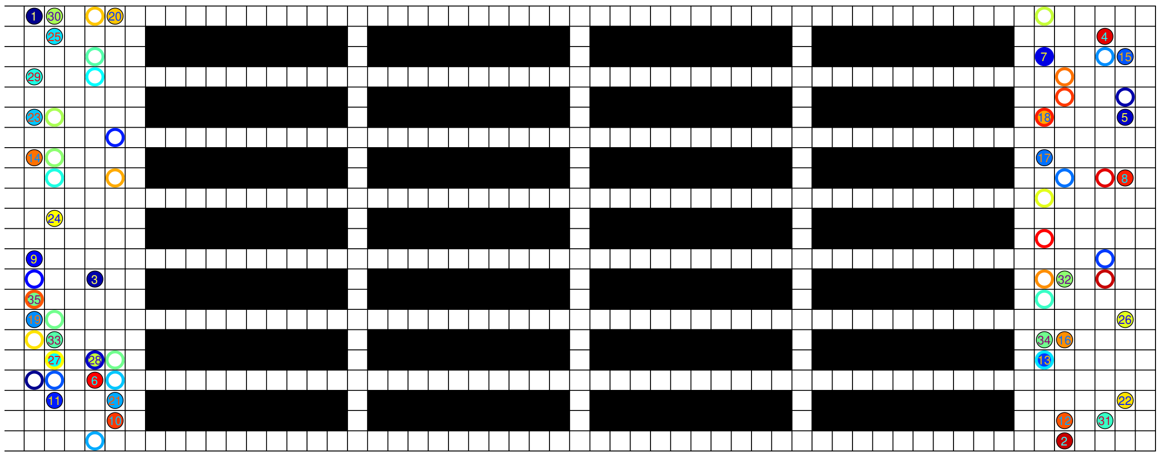


Figure 5.6: Illustration of a warehouse example. The workspace is gridded into 22×57 cells. Blocked cells are shown in black. Initial and final cells are marked with a solid and a hollow circle of a unique color, respectively.

However, the flowtime statistics are found as 128.78 and 77.78 steps for MCP and Rainbow Cycle implementations, respectively. Significant difference is the result of how a slow moving robot is treated by each policy. For the MCP implementation, $\mathcal{R}_2$ (resp. $\mathcal{R}_3$) needs to wait for $\mathcal{R}_1$ (resp. $\mathcal{R}_2$) unnecessarily, since the priority order is fixed at the path planning phase. On the other hand, Rainbow Cycle implementation allows robots to modify the priority order at run-time, resulting in improved flowtime statistics.

Warehouse Example

We also compare the performance of the control policies in a more structured warehouse-like environment. This warehouse example is taken from [58], and it has 35 robots as shown in Figure 5.6. The makespan and flowtime statistics are reported in Figure 5.7, which are averaged over 1000 runs for varying t_{max} values. Due to stronger assumptions on the collection of paths, the Naive DrPP based method is not able to handle

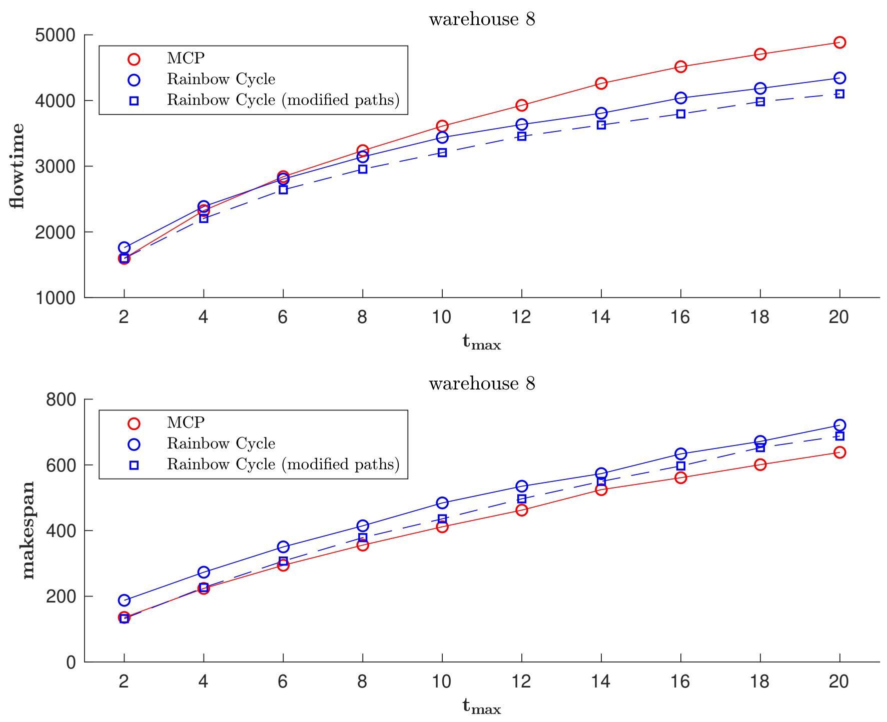


Figure 5.7: Makespan and flowtime comparisons. Statistics are averaged over 1000 runs for the warehouse environment under varying t_{max} values. Dashed lines show the improvement obtained by modifying the paths to decrease the number of rainbow cycles. Naive DrPP based method cannot solve this instance as the collection of paths violate the conditions in Theorem 5.1

this example. Similar to Section 5.4, we observe that makespan values are better for MCP, but Rainbow Cycle method scales better with t_{max} for flowtime statistics. Upon closer inspection, we see that robots moving in narrow corridors in opposite directions lead to many rainbow cycles. By enforcing a one-way policy in each corridor, similar to [19], many of these rainbow cycles can be eliminated and the performance of our method can be improved. Indeed, Figure 5.7 reports the results when paths are modified such that no horizontal corridor has robots moving in opposing directions.

5.5. Summary

In this chapter, we presented a method to solve the MRPE problem. Our method is based on a reformulation of the MRPE problem as an instance of DrPP. We showed that the existing solutions to the DrPP can be used to solve instances of MRPE problems if drinking sessions are constructed carefully. However, such an approach leads to conservative control policies. To improve the system performance, we provided a less conservative approach where we modified an existing DrPP solution. We provided conditions under which our control policies are shown to be collision and deadlock-free. We further demonstrated the efficacy of this method by comparing it with existing work. We observed that our method provides similar makespan performance to [58] while outperforming it in flowtime statistics, especially as uncertainty in robots' motion increase. This improvement can be explained mainly by our method's ability to change the priority order between robots during run-time, as opposed to keeping a fixed order.

Chapter 6.

Summary and Future Work

In this chapter we summarize the results of the previous chapters and discusses directions for future research.

6.1. Summary

In this thesis, we provided a framework for multirobot coordination to achieve complex tasks autonomously. This framework consists of (i) a formalism to specify multirobot tasks, (ii) algorithms to synthesize paths that collectively satisfy these tasks, and (iii) methods to deal with synchronization errors.

As the first step, we introduced counting logic cLTL+ in Chapter 2, which allows one to specify multirobot tasks concisely. We then introduced a notion of robust satisfaction for counting constraints. This notion allows us to analyze the effects of synchronization errors on particular solutions. We further introduced a fragment of cLTL+, namely cLTL, which results in permutation invariant tasks. We provided complexity analysis for this fragment and showed that it is PSPACE-complete.

In Chapter 3, we assumed that robots move synchronously and provided optimization-based methods to generate paths that collectively satisfy the specifications given in cLTL+. We showed that our method is sound and complete. We also provided an alternative method for the particular case where specifications are given in cLTL, and robots have identical dynamics. For this alternative method, solution times do not depend on the number of robots. Thus, hundreds of robots could be coordinated, as shown in Section 3.7.

In Chapter 4, we relaxed the synchrony assumption and discussed how to generate multirobot paths that are robust to synchronization errors. In particular, we showed that the generated paths could be asynchronously executed while preserving the satisfaction of the cLTL+ specification, if the asynchrony between robots is bounded. We further characterized the conditions under which this approach is complete.

In Chapter 5, we studied the multi-robot path execution problem where a group of robots move on predefined paths from their initial to target positions while avoiding collisions and deadlocks in the face of asynchrony. We first reformulated this problem as a well-known conflict resolution problem, namely Drinking Philosophers Problem (DrPP). We showed that by careful construction of the drinking sessions, any existing solutions to DrPP could be used to design distributed control policies that are collectively collision and deadlock-free. We then proposed modifications to an existing DrPP algorithm to allow more concurrent behavior and characterized the conditions under which our method is deadlock-free. We demonstrated the efficacy of our method on simulation examples by comparing it against the state-of-the-art.

6.2. Future Work

6.2.1. Decentralized and Reactive Controller Synthesis

In this thesis we provided centralized methods to generate paths that collectively satisfy the specifications given in cLTL+. As the number of robots, the complexity of the tasks and the size of the environment increase, such centralized methods could easily get intractable. We tried to address this issue by a hierarchical approach in Chapter 4. One interesting research direction is to develop decentralized algorithms to address the same issue.

The methods proposed in this thesis also require the environment to be static and known. Typically multirobot teams are expected to work in dynamic environments and interact with other vehicles, objects, or humans. It might not be possible to have a priori knowledge of the environment, such as the case of emergency response. There are also other uncertainties, such as modelling errors, which might limit the applicability of multirobot systems in practice. A potential research direction is to develop reactive controllers to handle such uncertainties. However, the reactive synthesis problem is known to be hard even for single-robot systems under LTL constraints. GR(1) fragment of LTL is shown to have an efficient polynomial-time synthesis algorithm [78]. Whether such fragments exists for cLTL+ is left for future work.

6.2.2. Robustness Against Non-deterministic Transitions

In Chapter 4 and Chapter 5, we studied robustness against a certain type of uncertainty, namely synchronization errors. We assumed that the robots can follow their nominal paths without tracking errors. We use a discrete representation, namely transition systems, to model the dynamics of robots which operate in the continuous domain. If the robot dynamics satisfy certain properties and a large enough discretization step is used, robots can indeed follow their discrete paths perfectly by using local feedback controllers. However, this assumption may not always be true. Non-deterministic transition systems could be used to capture such uncertainties.

When it is not possible to track a nominal path perfectly, one would still like to ensure the satisfaction of the specifications if possible, or the system performance to degrade gracefully. In [60], authors provide a notion of robustness for reachability properties such that the error from the target set can be bounded for bounded disturbances. However, it is also important to bound the errors from the nominal path along the way. In continuous domain, ideas such as LQR trees [104], funnel libraries [59], control contraction metrics [61], or Hamilton Jacobi reachability based methods [35] are used to find a region of attraction around a nominal trajectory in which the system is guaranteed to stay. An interesting research direction is to find discrete analogues of these concepts in the discrete domain.

6.2.3. Realistic Communication Constraints

We investigated the multirobot path execution problem in Chapter 5 and presented an algorithm to prevent collisions and deadlocks. The resulting control policies generated by this algorithm could be implemented by the robots in a distributed manner. To be able to do so, we require the communication graph to be identical to the resource dependency graph. That is, if two robots have a potential conflict, communication channel between them must be available at any given time. This connectivity requirement might restrict the mobility of robots to maintain proximity. Moreover, we assumed that the communication channel is lossless. An interesting future research direction is to relax these assumptions and study the case where the communication range is limited and channel is lossy.

Appendix A.

Supplements to Chapter 4

A.1. Proof of Theorem 4.1

First of all, note that $r_n^\phi(t) = 1$ if and only if $z_n^\phi(t+k) = 1$ for all $k \in [0, \tau]$ due to (4.2). That is, $r_n^\phi(t) = 1$ implies that robot $\mathcal{R}_n$ satisfies the inner formula ϕ for $\tau + 1$ consecutive steps, starting from time t. By the restriction of formulas to PNF, it is enough to prove the soundness for the operators in (4.1) and we do so recursively, starting with temporal counting propositions.

tcp: Let $\mu = [\phi, m] \in \Phi \times \mathbb{N}$ and a collection $\Pi = \{\pi_1, \dots, \pi_N\}$ of trajectories be given. We first show that $y^\mu(t) = 1$ implies that Π τ-robustly satisfies μ at anchor time t. Assume $m > 1$ and $y^\mu(t) = 1$. Then $\sum_{n=1}^N r_n^\phi(t) \geq m$ due to (4.3). Without loss of generality, assume that robots are enumerated such that the first m robots robustly satisfy ϕ at time step t, i.e., $r_n^\phi(t) = 1$ for all $n \in [m]$. Then $z_n^\phi(t+k) = 1$ for all $n \in [m]$ and for all $k \in [0, \tau]$ due to equation (4.2). Now let

$K \in \mathcal{K}_N(\tau)$ be an arbitrary τ-bounded execution and T be an arbitrary time step with anchor time t, i.e., $b_K(T) = t$. By definition of τ-bounded executions, local times are restricted to $t \leq k_n(T) \leq t + \tau$. Then, $\sum_{n=1}^{N} z_n^{\phi}(k_n(T)) \geq \sum_{n=1}^{m} z_n^{\phi}(k_n(T)) = m$. Hence, $(\Pi, K), T \models_\tau \mu$. Note that this is true for all for all $K \in \mathcal{K}_N(\tau)$ and for all $T \in b_K^{-1}(t)$. Thus, $\Pi, t \models_\tau \mu$ by definition of robust satisfaction.

Now assume $m = 1$ and $y^{\mu}(t) = 1$. Due to (4.4), either $\sum_{n=1}^{N} r_n^{\phi}(t) \geq 1$ or $\sum_{n=1}^{N} z_n^{\phi}(t) = N$. If the former is true, earlier arguments apply. Then, assume the latter is true, that is, $z_n^{\phi}(t) = 1$ for all n. Let $K = [k_1 \dots k_N]^T \in \mathcal{K}_N(\tau)$ be arbitrary. At anchor time t, there exists at least one robot such that $k_n(T) = t$. Without loss of generality assume $k_1(T) = t$. Then $\sum_{n=1}^{N} z_n^{\phi}(k_n(T)) \geq z_n^{\phi}(k_1(T)) = z_n^{\phi}(t) = 1$, hence $\Pi, t \models_\tau \mu$. These arguments hold for any t, including $t = 0$, hence the modified encodings in (4.3)-(4.4) are sound for temporal counting propositions.

conjunction: Showing soundness for conjunction is straightforward. Assume $\mu = \bigwedge \mu_i$ and for a collection $\Pi = \{\pi_1, \dots, \pi_N\}$, $y^{\mu}(t) = 1$ for some t. Then $y^{\mu_i}(t) = 1$ for all i, implying that $\Pi, t \models_\tau \mu_i$. In other words, for all $K \in \mathcal{K}_N(\tau)$ and for all $T \in b_K^{-1}(t)$; $(\Pi, K), T \models \mu_i$ for all i. Hence $\Pi, t \models_\tau \mu$.

disjunction: Let $\mu = \bigvee_i \mu_i$ and $y^{\mu}(t) = 1$ for some t and some collection $\Pi = \{\pi_1, \dots, \pi_N\}$. Since disjunction is associative and commutative, we can rewrite $\mu = \mu_{tcp} \vee \mu_o$ where $\mu_{tcp} = \bigvee_i [\phi_i, m_i]$ is conjunction of *tcp* and μ_o is the disjunction of the rest of the clauses that are not *tcp*. We first show that encoding of disjunction of temporal counting propositions is sound. If $y^{\mu}(t) = 1$, then either $y^{\mu_i}(t) = 1$ for some i, or $\sum_{n=1}^{N} r_n^{(\bigvee_i \phi_i)}(t) > \sum_i (m_i - 1)$. If it is the former, $y^{\mu_i}(t) = 1$ for some $\mu_i = [\phi_i, m_i]$, then it follows from the soundness of *tcp* encodings that $\Pi, t \models_\tau \mu_i$.

Thus $\Pi, t \models_\tau \mu$. Now assume $y^{\mu_i}(t) = 0$ for all i and $\sum_{n=1}^{N} r_n^{(\bigvee_i \phi_i)}(t) > \sum_i (m_i - 1)$. Note that $r_n^{(\bigvee_i \phi_i)}(t) = 1$ implies that for each $k \in [0, \tau]$, there exists at least one ϕ_i such that $\pi_n, t + k \models \phi_i$. Now for arbitrary set of local indices $\{k_n(T)\}$ such that $t \leq k_n(T) \leq t + \tau$, let $\tilde{m}_i$ be the number of robots who satisfy ϕ_i, i.e., $\tilde{m}_i \doteq |\{n \mid z_n^{\phi_i}(k_n(T)) = 1\}|$. Then $\sum_i \tilde{m}_i \geq \sum_{n=1}^{N} r_n^{(\bigvee_i \phi_i)}(t) > \sum_i (m_i - 1)$. Note that if $\tilde{m}_i < m_i$ for all i, the last inequality cannot be true. Hence, there exists at least one $\tilde{m}_i \geq m_i$. As a result, $\Pi, t \models \mu_i$ for at least one μ_i and $\Pi, t \models \mu$.

Showing soundness of (4.5) is straightforward and omitted here. All of these combined together proves the correctness of (4.6) and (4.5).

until: Until encodings are quite close to standard encodings but the modification is needed due to change in disjunction encodings. Let $\eta = \mu_1 \,\mathcal{U}\, \mu_2$ and $\Pi = \{\pi_1, \ldots, \pi_N\}$ be a collection. If $y^{\mu_2}(t) = 1$ for Π and some t, then $\Pi, t \models \mu_2$ and $\Pi, t \models \eta$. Now assume $y^{\mu_2}(t) \neq 1$. The first line in equation (4.7) requires $y^{\mu_1 \vee \mu_2}(t) = 1$ and $y^{\eta}(t + 1) = 1$, for $y^{\eta}(t) = 1$ to hold. Then $\Pi, t \models_\tau \mu_1 \vee \mu_2$ and $\Pi, t + 1 \models_\tau \mu_1 \,\mathcal{U}\, \mu_2$. This implies that $\Pi, t \models_\tau \mu_1 \,\mathcal{U}\, \mu_2$. Similar to standard encodings, auxiliary variables are used to avoid trivial satisfaction and make sure μ_2 is satisfied at some point.

Proving that the "release" operator encodings are also sound is similar to "until" case and omitted here. We showed that outer logic encodings are sound. The soundness of the whole encoding procedure follows as before from soundness of ILP encodings of LTL, which is used for inner logic formulas. $\square$

A.2. Proof of Theorem 4.2

We first give an outline of the proof and then provide details. The proof starts by showing that the modified encodings are complete for the simplest specification, $\mu = [\phi, m]$. We then show that conjunction and next operators preserve completeness. Next, we show that disjunction and until operators are complete for mutually exclusive atomic propositions. That is enough to prove Theorem 4.2 due to the special form of specifications and the second assumption that atomic propositions are mutually exclusive. We now give details of these steps.

tcp: Let $\mu = [\phi, m]$ be a temporal counting proposition and $\Pi = \{\pi_1, \dots, \pi_N\}$ be a collection such that $\Pi, t \models_\tau \mu$ for some t. We are going to show that if (4.3) (or (4.4) for $m = 1$) does not hold for some t, then $\Pi, t \not\models_\tau \mu$. First assume $m > 1$ and $\sum_{n=1}^{N} r_n^\phi(t) < m$. Assume without loss of generality that robots are enumerated such that $r_n^\phi(t) = 0$ at least for the first $N-m+1$ robots. Then, for all $n \in [N-m+1]$, there exist at least one $z_n^\phi(t+k) = 0$ for some $k \in \{0, 1, \dots, \tau\}$. Assume each $\hat{t}_n$ denotes the first instance where $z_n^\phi(\hat{t}_n) = 0$ for $\hat{t}_n \in [t, t+\tau]$ and for all $n \in [N-m+1]$. Then, there exists a τ-bounded execution $K = [k_1 \dots k_N]^T \in \mathcal{K}_N(\tau)$ such that $k_n(T) = \hat{t}_n$ for all $n \in [N-m+1]$ and $k_N(T) = t$ for some T. Note that such a Π violates (2.5) and creates a contradiction. Thus $\sum_{n=1}^{N} r_n^\phi(t) \geq m$ must hold.

In the special case when $m = 1$, further assume that $\sum_{n=1}^{N} z_n^\phi(t) < N$. This implies that, for each $n \in [N]$, $z_n^\phi(k_n(T)) = 0$ for some T where $t \leq k_n(T) \leq t + \tau$ and there exists at least one robot $\tilde{n}$ such that $z_{\tilde{n}}^\phi(t) = 0$. Then choose $k_{\tilde{n}}(T) = t$ and for all other robots choose $k_n(T)$ such that $z_n^\phi(k_n(T)) = 0$. These set of indices have the anchor time t and satisfy the τ-boundedness criteria. Hence, there exists a τ-bounded asynchronous execution such that μ is not satisfied. But this is a contradiction. Thus either $\sum_{n=1}^{N} r_n^{\phi}(t) \geq 1$ or $\sum_{n=1}^{N} z_n^{\phi}(t) = N$ must hold.

disjunction: For the sake of ease, we show that (4.6) is complete for disjunction of two temporal counting propositions. Let $\mu_i = [\phi_i, m_i]$ for $i = 1, 2$ and $\mu = \mu_1 \vee \mu_2$. Assume that (4.6) fails to hold for some t, but that there exists a collection $\Pi = \{\pi_1, \dots, \pi_N\}$ such that $\Pi, t \models_\tau \mu$. This implies that, for all local time permutations with anchor time t, i.e., $k_n(T) \in [t, t+\tau]$ and $\min k_n(T) = t$, we have $\sum_n z_n^{\phi_i}(k_n(T)) \geq m_i$ for either $i = 1$ or $i = 2$. Since (4.6) fails to hold, we have $\sum_{n=1}^{N} r_n^{(\phi_1 \vee \phi_2)}(t) < m_1 + m_2 - 1$ which implies that $\sum_{n=1}^{N} r_n^{\phi_i}(t) < m_i$ for $i = 1, 2$. Now without loss of generality, enumerate robots such that $r_n^{(\phi_1 \vee \phi_2)}(t) = 1$ only for the first n_{12} robots. This implies that, for the rest of the robots, one can choose a local time where both ϕ_1 and ϕ_2 fails to hold. Furthermore, assume that $r_n^{\phi_1}(t)$ holds for the first n_1 robots and that $r_n^{\phi_2}(t)$ holds for the following n_2 robots. Since AP are mutually exclusive, no robot can satisfy ϕ_1 and ϕ_2 at the same time. Then, starting from the $(n_1 + n_2 + 1)^{th}$ robot, choose as local times the first $m_1 - n_1 - 1$ such that $z_n^{\phi_1}(k_n(T)) = 1$. For the rest of the robots, until n_{12}^{th}, choose local times such that $z_n^{\phi_2}(k_n(T)) = 1$. Note that such selection always exists. Then $\sum_n z_n^{\phi_1}(k_n(T)) = m_1 - 1$, and $\sum_n z_n^{\phi_2}(k_n(T)) = r_{12} - \sum_n z_n^{\phi_1}(k_n(T)) = r_{12} - (m_1 - 1) < m_1 + m_2 - 1 - (m_1 - 1) < m_2$.

Note that we can always choose $k_1(T) = t$. This is contradictory to the assumption that μ is τ-robustly satisfied. Thus, we conclude that (4.6) is necessary for μ to be satisfied.

A.3. Proof of Theorem 4.3

First part of the proof is straightforward. Assume an arbitrary T-path π is given. To obtain π^{abs}, assign $\pi^{abs}(t) = \mathcal{R}(\pi_n(t))$. Note that π^{abs} is a valid T^{abs} path since each $(\pi^{abs}(t), \pi^{abs}(t+1)) \in \mathcal{E}^{abs}$ due to (4.13). Furthermore $L^{abs}(\pi^{abs}(t)) = L(\pi(t))$. Thus, π and π^{abs} have the same trace and are stutter trace equivalent.

Conversely, let π^{abs} be an arbitrary T^{abs}-path. Starting from $t = 0$, choose arbitrary $u, v \in V$ such that $u \in \mathcal{R}^{-1}(\pi^{abs}(t))$ and $v \in \mathcal{R}^{-1}(\pi^{abs}(t+1))$. Since $(v_i^{abs}, v_j^{abs}) \in E^{abs}$, there exists $(u', v') \in E$ such that $u' \in \mathcal{R}^{-1}(\pi^{abs}(t))$ and $v' \in \mathcal{R}^{-1}(\pi^{abs}(t+1))$ due to (4.13). Since both $u, u' \in (\pi^{abs}(t))$, there exist a T-path $\pi_{uu'}$ from u to u' due to (4.11). Similarly, there exist another T-path $\pi_{v'v}$ from v' to v. The concatenation $\pi_{uu'}\pi_{v'v}$ of these two paths is a valid T-path from u to v since $(u', v') \in \mathcal{E}$. Note that we can compute such a T-path for each t and obtain π by concatenating them.

By construction of the abstraction $L(u) = L(u')$ for all $u, u' \in (\pi^{abs}(t))$ and $L(\pi^{abs}(t)) = L(u)$. Then $\sigma(\pi_{uu'}\pi_{v'v}) = (L(v_i^{abs}) \dots L(v_i^{abs}))(L(v_j^{abs}) \dots L(v_j^{abs}))$. This implies that $\pi_{uu'}\pi_{v'v}$ is trace equivalent to path segment $\pi^{abs}(t)\pi^{abs}(t+1)$. Since trace equivalence holds for all t, π is stutter trace equivalent to π^{abs}. $\square$

A.4. Proof of Theorem 4.4

Showing $\{\pi_1, ..., \pi_N\}$ are collision-free is straight-forward. Each GMRPP instance generates collision-free T-paths. Concatenation of them would also be collision-free. Furthermore, $\pi_n(0) = S_0(\mathcal{R}_n)$ for all $\mathcal{R}_n \in \mathcal{A}$ due to (4.15).

Next we show $\{\pi_1, ..., \pi_N\} \models_0 \mu$. Let $\Sigma = \{\sigma_{\pi_1}, \dots, \sigma_{\pi_N}\}$ and $\Sigma^{abs} = \{\sigma_{\pi_1^{abs}}, \dots, \sigma_{\pi_N^{abs}}\}$.

Define synchronous execution $K = \{k_1, \dots, k_N\}$ such that $k_n(t) = t$ for all $t \geq 0$ and for all $n \in [N]$. We first show that there exists a 1-bounded asynchronous execution $K^{abs} = \{k_1^{abs}, ..., k_N^{abs}\}$ such that collective traces (Σ, K) and (Σ^{abs}, K^{abs}) are identical.

Note that $\beta_n^t(h) \in \mathcal{R}^{-1}(\pi_n^{abs}(t+1)))$ due to (4.15). Furthermore, $\pi_n(0) = S_0(\mathcal{R}_n) \in \mathcal{R}^{-1}(\pi_n^{abs}(0))$. This implies $L(\beta_n^t(h)) = L(\pi_n(ht)) = L(\pi_n^{abs}(t))$ for all $t \geq 0$ due to (4.13). Also note that, for all $n \in [N]$, there exists a non-negative integer $l_n^t \leq h$ such that $L(\beta_n^t(t)) = L(\beta_n^t(0))$ for all $t \leq l_n^t$ and $L(\beta_n^t(t)) = L(\beta_n^t(h))$ for all $l_n^t < t \leq h$ due to definition of Problem 1, and equations (4.13) and (4.15). Therefore $L(\pi_n(th + \alpha)) = L(\beta_n^t(\alpha)) = L(\pi_n^{abs}(t))$.

Now initialize $k_n^{abs}(0) \doteq 0$ for all $n \in [N]$. Then iteratively define local times for all integers $0 < \alpha \leq h$ and $t \geq 0$ as

$$k_n^{abs}(th + \alpha) \doteq \begin{cases} k_n^{abs}(th + \alpha - 1) & \text{if } \alpha \leq l_n^t \\ k_n^{abs}(th + \alpha - 1) + 1 & \text{if } \alpha > l_n^t \end{cases} \tag{A.1}$$

Note that $k_n^{abs}(t)$ is well-defined for all $t \geq 0$ and $L(\pi_n(t)) = L(\pi_n^{abs}(k_n^{abs}(t)))$ for all t. This implies that (Σ, K) and (Σ^{abs}, K^{abs}) are identical collective traces.

Moreover it is guaranteed that $k_n^{abs}(t + h) = k_n^{abs}(t) + 1$ and $k_n^{abs}(th) = k_m^{abs}(th)$ for all pairs of $n, m \in [N]$ and for all $t \geq 0$. This implies that, the collection $K^{abs} = \{k_1^{abs}, \dots, k_N^{abs}\}$ is a 1-bounded asynchronous execution. Since $\Sigma^{abs} \models_1 \mu$, we have $\Sigma^{abs}, K^{abs} \models \mu$. Thus, $\Sigma \models_0 \mu$. $\square$

A.5. Proof of Theorem 4.5

We first show that collisions would be avoided for any 1-bounded asynchronous execution $K = \{k_1, \ldots, k_N\}$. Note that $|k_n(t) - k_m(t)| \leq 1$. Assume $k_n(t) = k_m(t)$, then $\pi_n(k_n(t)) \neq \pi_m(k_m(t))$ since generated paths satisfy $\pi_n(t) \neq \pi_m(t)$. Similarly assume $k_n(t) = k_m(t) + 1$, then $\pi_n(k_n(t)) \neq \pi_m(k_m(t))$ since generated paths satisfy $\pi_n(t+1) \neq \pi_m(t)$. Since selection of n, m was arbitrary, all collisions would be avoided. Furthermore, $\pi_n(0) = S_0(\mathcal{R}_n)$ for all $\mathcal{R}_n \in \mathcal{A}$ due to (4.15), as before.

Now we show that $\{\pi_1, \ldots, \pi_N\} \models_1 \mu$. Let 1-bounded asynchronous execution $K = [k_1, \ldots, k_N]$ be arbitrary and $\Sigma = \{\sigma(\pi_1), \ldots, \sigma(\pi_N)\}$ and $\Sigma^{abs} = \{\sigma(\pi_1^{abs}), \ldots, \sigma(\pi_N^{abs})\}$. We first show that there exists a 1-bounded asynchronous execution $K^{abs} = \{k_1^{abs}, ..., k_N^{abs}\}$ such that (Σ, K) and (Σ^{abs}, K^{abs}) are identical collective traces.

Intuitively, we define the abstract local times such that $k_n^{abs}(t)$ denotes the abstract state of robot $\mathcal{R}_n$ is at time t, i.e., $\pi_n(k_n(t)) \in \mathcal{R}^{-1}(\pi_n^{abs}(k_n^{abs}(t)))$. Since each β_n^t is of length h, $\alpha \leq k_n^{abs}(t) \leq \alpha + 1$ should be satisfied for all $n \in [N]$ and for all t such that $\alpha h \leq k_n(t) \leq (\alpha+1)h$. To do so, initialize $k_n^{abs}(0) \doteq 0$ for all $n \in [N]$. Then, iteratively define local times for all $n \in [N]$ as follows:

$$k_n^{abs}(t) \doteq \begin{cases} k_n^{abs}(t-1) + 1 \text{ if} & L(\pi_n(k_n(t))) \neq L(\pi_n(k_n(t-1))) \text{ or} \\ & L(\pi_n(k_n(t))) = L(\pi_n(k_n(t) - \alpha)) \text{ for all } \alpha \in [h] \\ k_n^{abs}(t-1) & \text{otherwise} \end{cases} \tag{A.2}$$

Note that if $L(\pi_n^{abs}(\alpha)) \neq L(\pi_n^{abs}(\alpha-1))$, there exist a time step t such that $\alpha h \leq k_n(t) \leq (\alpha+1)h$ and $\mathcal{R}_n$ leaves the set of states $\mathcal{R}^{-1}(\pi_n^{abs}(\alpha-1))$ and enters $\mathcal{R}^{-1}(\pi_n^{abs}(\alpha))$. At that time, $L(\pi_n(k_n(t))) \neq L(\pi_n(k_n(t-1)))$. As stated in (A.2), abstract local time is increased by 1 at this time and $\pi_n(k_n(t)) \in \mathcal{R}^{-1}(\pi_n^{abs}(k_n^{abs}(t)))$. On the other hand, if $L(\pi_n^{abs}(t)) = L(\pi_n^{abs}(t-1))$, abstract local time k_n^{abs} increased by 1 after local time k_n is increased h times. As a result, $\alpha \leq k_n^{abs}(t) \leq \alpha+1$ is satisfied for all $n \in [N]$ and for all t such that $\alpha h \leq k_n(t) \leq (\alpha+1)h$. This implies that $K^{abs} = \{k_1^{abs}, \dots, k_N^{abs}\}$ is a 1-bounded asynchronous execution. Moreover, $\pi_n(k_n(t)) \in \mathcal{R}^{-1}(\pi_n^{abs}(k_n^{abs}(t)))$ for all t. Then (Σ, K) and (Σ^{abs}, K^{abs}) are identical collective traces. This implies that $\Sigma \models_1 \mu$ since K was arbitrary and $\Sigma^{abs} \models_1 \mu$. $\square$

Appendix B.

Supplements to Chapter 5

B.1. Proof of Theorem 5.1

We first start by showing that the Algorithm 5.3 is collision-free. Assume that $\mathcal{R}_n$ is currently occupying the shared cell π_n^t. Note that, when a robot is about to move to a shared cell, *GO* action is issued only when $\mathcal{R}_n$ is drinking (lines 3 and $19-22$). Therefore, before reaching π_n^t, $\mathcal{R}_n$ was in drinking state, and thus, was holding all the bottles in $\tilde{\mathcal{S}}_n(t)$. If π_n^{t+1} is a free cell, $\mathcal{R}_n$ would stay in drinking state until reaching π_n^{t+1} (lines $23-29$). Otherwise, it would get insatiable with $\tilde{\mathcal{S}}_n(t) \cup \tilde{\mathcal{S}}_n(t+1)$. In neither of these scenarios, $\mathcal{R}_n$ releases any bottles before reaching to π_n^{t+1}. Note also that, by construction of drinking sessions, $\pi_n^t \subseteq \mathcal{S}_n(t)$. Since bottles are mutually exclusive, none of the other robots could acquire the bottles in $\mathcal{B}_n(\pi_n^t)$ while $\mathcal{R}_n$ is in π_n^t. This implies that collisions are avoided, as no other robot is allowed to occupy π_n^t before $\mathcal{R}_n$ leaves.

We now show that Algorithm 5.3 is deadlock free. As defined in Definition 5.1 deadlock is any configuration where a subset of robots, which have not reached their final cell, choose $STOP$ action indefinitely. As it can be seen from Algorithm 5.3, there are only three cases where a robot chooses the $STOP$ action: (i) when the robot is already in the final cell (line 2), (ii) when there are no free cells from the next cell up to and including the final cell, and the final cell is not yet cleared by all other robots (line 13), (iii) when the robot is in thirsty or insatiable state (line 18). In the following, we show that none of these cases can cause a deadlock.

We start by showing that neither (i) nor (ii) could cause a deadlock. To do so, assume $\mathcal{R}_n$ has reached its final cell and is causing a deadlock by blocking others from progressing. By (3) of Theorem 5.1, we know that there exist at least one free cell in each path. Since we assumed that $\mathcal{R}_n$ is blocking others by waiting in its final cell, π_n^{end} must be a shared cell. Then, there must be at least one free cell before π_n^{end}. Let π_n^t denote the last free cell on π_n. A robot reaching a free cell gets into tranquil state, if its not already in tranquil state, due to line 31 of Algorithm 5.3. Otherwise, if π_n^t is the first cell of π_n, $\mathcal{R}_n$ would be in tranquil state before trying to move forward, since all robots are initialized in tranquil state. According to lines 12 and 13 of Algorithm 5.3, $\mathcal{R}_n$ would wait in π_n^t in tranquil state, until its final cell is cleared by all other robots. Since a tranquil robot does not need any bottles, no other robot could be waiting for $\mathcal{R}_n$. However, this is a contradiction, and it is not possible for a robot to reach its final cell and block others from progressing. Therefore, (i) cannot be a reason for a deadlock. Furthermore, we showed that a robot waiting due to (ii) would stay in tranquil state until all others clear its final cell. As stated, a tranquil robot does not need any bottles, and thus, no other robot could be waiting for $\mathcal{R}_n$. Thus, (ii) cannot cause deadlocks, either.

We now show that (iii) cannot cause deadlocks. To do so, assume that a subset of robots are stuck due to (iii), i.e., they are all in thirsty or insatiable state, and they need additional bottle(s) to move. If there was a robot who does not wait for any other robot, it would start drinking and moving. Therefore, some non-empty subset of these robots must be waiting circularly for each other. Without loss of generality, let $\mathcal{R}_n$ be waiting for $\mathcal{R}_{n+1}$ for $n \in \{1, \dots, K\}$ where $\mathcal{R}_{K+1} = \mathcal{R}_1$. That is, $\mathcal{R}_n$ has some subset of bottles $\mathcal{R}_{n-1}$ needs, and would not release them without acquiring some subset of bottles from $\mathcal{R}_{n+1}$. Note that, there might be other robots choosing the $STOP$ action indefinitely as well, however, the main reason for the deadlock is this circular wait. Once the circular waiting ends, all robots would start moving according to their priority ordering.

For the time being, assume that each robot starts from a free inital cell and moves towards a free cell through an arbitrary number of shared cells in between. We later relax this assumption. Firstly, we know that none of the robots could be in tranquil or drinking state, otherwise they would be moving until reaching the next cell as lines $5-7$ and $26-29$ of Algorithm 5.3. Secondly, we show that, not all robots can be thirsty. Since a strict priority order is maintained between robots at all times, if all of them were thirsty, the robot with the highest priority would acquire all the bottles it needs according to $R'5$ and start drinking. A drinking robot starts moving, therefore cannot be participating in a deadlock. Therefore, there must be at least one robot that is in insatiable state. Thirdly, we show that if there is a deadlock, all robots participating in it must be in insatiable state. To show a contradiction, assume that at least one of the robots participating in the deadlock is thirsty. According to $R'5$, an insatiable robot always has a higher priority than a thirsty robot. Therefore, an insatiable robot cannot be waiting for a thirsty robot. Thus, all robots in a deadlock configuration must in insatiable state.

Let $\tilde{G}_\Pi$ be the graph returned by the Algorithm 5.2 for the input Path-Graph G_Π. We showed that all robots are in insatiable state. Let $\pi_n^{t_n}$ denote the current cell $\mathcal{R}_n$ is occupying, i.e., $curr(\mathcal{R}_n) = t_n$. Lines $24 - 25$ of Algorithm 5.3 show that $\mathcal{R}_n$ must be insatiable with $\mathcal{B}_n(\tilde{\mathcal{S}}_n(t_n) \cup \tilde{\mathcal{S}}_n(t_n + 1))$. That is, $\mathcal{R}_n$ needs all the bottles in $\mathcal{B}_n(\tilde{\mathcal{S}}_n(t_n) \cup \tilde{\mathcal{S}}_n(t_n + 1))$ to start drinking. Since $\mathcal{R}_n$ currently occupies $\pi_n^{t_n}$, it must hold all the bottles in $\mathcal{B}_n(\tilde{\mathcal{S}}_n(t_n))$. Then, $\tilde{\mathcal{S}}_n(t_n) \neq \tilde{\mathcal{S}}_n(t_n + 1)$, and $\mathcal{R}_n$ needs and does not hold some of the bottles in $\mathcal{B}_n(\tilde{\mathcal{S}}_n(t_n + 1))$. Then, by construction of drinking sessions, there must be two nodes in $\tilde{G}_\Pi$, one corresponding to $[\tilde{\mathcal{S}}_n(t_n)]$ and another corresponding to $[\tilde{\mathcal{S}}_n(t_n + 1)]$, and a c_n colored edge from $[\tilde{\mathcal{S}}_1(t_1)]$ to $[\tilde{\mathcal{S}}_1(t_1 + 1)]$ in $\tilde{G}_\Pi$. Similarly, $\mathcal{R}_{n+1}$ holds all the bottles in $\mathcal{B}_{n+1}(\tilde{\mathcal{S}}_{n+1}(t_{n+1}))$ and is missing some of the bottles in $\mathcal{B}_{n+1}(\tilde{\mathcal{S}}_{n+1}(t_{n+1} + 1))$. Since $\mathcal{R}_n$ is waiting for $\mathcal{R}_{n+1}$, either $[\tilde{\mathcal{S}}_n(t_n + 1)] = [\tilde{\mathcal{S}}_{n+1}(t_{n+1})]$ or $[\tilde{\mathcal{S}}_n(t_n + 1)] = [\tilde{\mathcal{S}}_{n+1}(t_{n+1} + 1)]$ must hold. This implies that, there exists a c_n colored edge from $[\tilde{\mathcal{S}}_n(t_n)]$ to either $[\tilde{\mathcal{S}}_{n+1}(t_{n+1})]$ or to $[\tilde{\mathcal{S}}_{n+1}(t_{n+1} + 1)]$. In a similar manner, there exists a c_{n+1} colored edge from $[\tilde{\mathcal{S}}_{n+1}(t_{n+1})]$ to either $[\tilde{\mathcal{S}}_{n+2}(t_{n+2})]$ or to $[\tilde{\mathcal{S}}_{n+2}(t_{n+2} + 1)]$. Repeating the same reasoning, we can find colored edges and show that there exists a rainbow cycle $\{([v_1], c_1, [v_2]), \ldots, ([v_K], c_K, [v_1])\}$ in $\tilde{G}_\Pi$. However, this is a contradiction as such a rainbow cycle would be found by the Algorithm 5.2, and $\tilde{G}_\Pi$ would not be returned. Therefore, such a deadlock configuration cannot be reached and (iii) cannot be a reason for a deadlock.

We now relax the assumption that all robots are initialized at a free cell. To do so, we *"modify"* all paths by appending *virtual* free cell at the beginning. That is, all robots are initialized at a virtual free cell, which does not exist physically, and the next cell in a robot's path is its original initial cell. Theorem 5.1 assumes that initial drinking sessions are disjoint for each robot, i.e., $\mathcal{S}_m(0) \cap \mathcal{S}_n(0) = \emptyset$ for all m, n. Since initial drinking sessions are disjoint, all robots whose initial cell is a shared cell can immediately start drinking. As a result, all of those robots can immediately *"virtually move"* into their original initial cell. All other robots with free initial cells can also move to their original initial cells immediately. Therefore the assumption that all robots are initialized at a free cell is not restricting.

Finally, we relax the assumption that each robot moves towards a free cell. Theorem 5.1 requires each path to have at least one free cell. Then, up until reaching the final free cell, moving towards a free cell assumption is not restrictive. We know under this condition that deadlocks are prevented, therefore all robots are at least guaranteed to reach to the final free cell in their path. Theorem 5.1 also requires that the final drinking sessions are disjoint. Therefore, all robots would eventually be able to start drinking and reach their final location.

Deadlocks occur when a subset of robots, which have not reached their final cell, choose $STOP$ action indefinitely. A robot chooses the $STOP$ action only under three conditions. We showed that none of these conditions can cause a deadlock. Thus, Algorithm 5.3 is deadlock-free. $\square$

Bibliography

[1] H. Ahn and D. Del Vecchio. "Safety verification and control for collision avoidance at road intersections". In: *IEEE Transactions on Automatic Control* 63.3 (2017), pp. 630–642.

[2] D. Aksaray, K. Leahy, and C. Belta. "Distributed multi-agent persistent surveillance under temporal logic constraints". In: *IFAC-PapersOnLine* 48.22 (2015), pp. 174–179.

[3] N. Alon, R. Yuster, and U. Zwick. "Finding and counting given length cycles". In: *Algorithmica* 17.3 (1997), pp. 209–223.

[4] J. Alonso-Mora, A. Breitenmoser, P. Beardsley, and R. Siegwart. "Reciprocal collision avoidance for multiple car-like robots". In: *2012 IEEE International Conference on Robotics and Automation*. IEEE. 2012, pp. 360–366.

[5] C. Baier and J. Katoen. *Principles of Model Checking*. MIT Press, 1999.

[6] J. Banfi, N. Basilico, and F. Amigoni. "Multirobot Reconnection on Graphs: Problem, Complexity, and Algorithms". In: *IEEE Transactions on Robotics* 34.5 (Oct. 2018), pp. 1299–1314.

[7] D. Bareiss and J. van den Berg. "Generalized reciprocal collision avoidance". In: *The International Journal of Robotics Research* 34.12 (2015), pp. 1501–1514.

[8] J. L. Baxter, E. Burke, J. M. Garibaldi, and M. Norman. "Multi-robot search and rescue: A potential field based approach". In: *Autonomous robots and agents*. Springer, 2007, pp. 9–16.

[9] C. Belta and S. Sadraddini. "Formal methods for control synthesis: An optimization perspective". In: *Annual Review of Control, Robotics, and Autonomous Systems* 2 (2019), pp. 115–140.

[10] C. Belta, B. Yordanov, and E. A. Gol. *Formal Methods for Discrete-Time Dynamical Systems*. Springer, 2017.

[11] A. Biere, K. Heljanko, T. Junttila, T. Latvala, and V. Schuppan. "Linear Encodings of Bounded LTL Model Checking". In: *Logical Methods in Computer Science* 2 (2006), pp. 1–64.

[12] E. Boyarski, A. Felner, R. Stern, G. Sharon, D. Tolpin, O. Betzalel, and E. Shimony. "ICBS: improved conflict-based search algorithm for multi-agent pathfinding". In: *Twenty-Fourth International Joint Conference on Artificial Intelligence*. 2015.

[13] M. Čap, P. Novak, J. Vokrınek, and M. Pěchouček. "Multi-agent RRT: sampling-based cooperative pathfinding". In: *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*. International Foundation for Autonomous Agents and Multiagent Systems. 2013, pp. 1263–1264.

[14] D. Carlino, S. D. Boyles, and P. Stone. "Auction-based autonomous intersection management". In: *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*. IEEE. 2013, pp. 529–534.

[15] K. M. Chandy and J. Misra. "The drinking philosophers problem". In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 6.4 (1984), pp. 632–646.

[16] J. Y. Chen, M. J. Barnes, and M. Harper-Sciarini. "Supervisory control of multiple robots: Human-performance issues and user-interface design". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 41.4 (2011), pp. 435–454.

[17] L. Chen and C. Englund. "Cooperative intersection management: A survey". In: *IEEE Transactions on Intelligent Transportation Systems* 17.2 (2015), pp. 570–586.

[18] Y. Chen, M. Cutler, and J. P. How. "Decoupled multiagent path planning via incremental sequential convex programming". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2015, pp. 5954–5961.

[19] L. Cohen, T. Uras, and S. Koenig. "Feasibility study: Using highways for bounded-suboptimal multi-agent path finding". In: *Eighth Annual Symposium on Combinatorial Search*. 2015.

[20] S. Demri and D. D'Souza. "An automata-theoretic approach to constraint LTL". In: *Information and Computation* 205.3 (2007), pp. 380–415.

[21] S. Demri and R. Gascon. "Verification of qualitative $\mathbb{Z}$ constraints". In: *International Conference on Concurrency Theory*. Springer. 2005, pp. 518–532.

[22] A. Desai, I. Saha, J. Yang, S. Qadeer, and S. A. Seshia. "DRONA: a framework for safe distributed mobile robotics". In: *Proc. of the 8th ICCPS*. ACM. 2017, pp. 239–248.

[23] M. B. Dias, M. Zinck, R. Zlot, and A. Stentz. "Robust multirobot coordination in dynamic environments". In: *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*. Vol. 4. IEEE. 2004, pp. 3435–3442.

[24] E. W. Dijkstra. "Hierarchical ordering of sequential processes". In: *The origin of concurrent programming*. Springer, 1971, pp. 198–227.

[25] A. Donze and O. Maler. "Robust satisfaction of temporal logic over real-valued signals". In: *International Conference on Formal Modeling and Analysis of Timed Systems*. Springer. 2010, pp. 92–106.

[26] K. Dresner and P. Stone. “A multiagent approach to autonomous intersection management”. In: *Journal of artificial intelligence research* 31 (2008), pp. 591–656.

[27] E. Erdem, D. G. Kisa, U. Oztok, and P. Schuller. “A general formal framework for pathfinding problems with multiple agents”. In: *Twenty-Seventh AAAI Conference on Artificial Intelligence*. 2013.

[28] P. Fiorini and Z. Shiller. “Motion planning in dynamic environments using velocity obstacles”. In: *The International Journal of Robotics Research* 17.7 (1998), pp. 760–772.

[29] A. Franchi, P. R. Giordano, C. Secchi, H. I. Son, and H. H. Bulthoff. “A passivity-based decentralized approach for the bilateral teleoperation of a group of UAVs with switching topology”. In: *2011 IEEE International Conference on Robotics and Automation*. IEEE. 2011, pp. 898–905.

[30] H. Fukushima, K. Kon, and F. Matsuno. “Model predictive formation control using branch-and-bound compatible with collision avoidance problems”. In: *IEEE Transactions on Robotics* 29.5 (2013), pp. 1308–1317.

[31] D. Ginat, A. U. Shankar, and A. K. Agrawala. “An efficient solution to the drinking philosophers problem and its extensions”. In: *International Workshop on Distributed Algorithms*. Springer. 1989, pp. 83–93.

[32] A. Gray, Y. Gao, T. Lin, J. K. Hedrick, H. E. Tseng, and F. Borrelli. “Predictive control for agile semi-autonomous ground vehicles using motion primitives”. In: *American Control Conference (ACC), 2012*. IEEE. 2012, pp. 4239–4244.

[33] I. Gurobi Optimization. *Gurobi Optimizer Reference Manual*. 2016. URL: `http://www.gurobi.com`.

[34] R. Hegde and D. Panagou. “Multi-agent motion planning and coordination in polygonal environments using vector fields and model predictive control”. In: *2016 European Control Conference (ECC)*. IEEE. 2016, pp. 1856–1861.

[35] S. L. Herbert, M. Chen, S. Han, S. Bansal, J. F. Fisac, and C. J. Tomlin. “FaS-Track: A modular framework for fast and guaranteed safe motion planning”. In: *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE. 2017, pp. 1517–1522.

[36] E. G. Hernandez-Martınez and E. Aranda-Bricaire. “Convergence and collision avoidance in formation control: A survey of the artificial potential functions approach”. In: *Multi-agent systems-modeling, control, programming, simulations and applications*. IntechOpen, 2011.

[37] G. Hu. “Robust consensus tracking of a class of second-order multi-agent dynamic systems”. In: *Systems & Control Letters* 61.1 (2012), pp. 134–142.

[38] Y. Kantaros and M. M. Zavlanos. “A distributed LTL-based approach for intermittent communication in mobile robot networks”. In: *2016 American Control Conference (ACC)*. IEEE. 2016, pp. 5557–5562.

[39] Y. Kantaros and M. M. Zavlanos. “Sampling-based optimal control synthesis for multi-robot systems under global temporal tasks”. In: *IEEE Transactions on Automatic Control* (2018).

[40] S. Karaman and E. Frazzoli. “Linear temporal logic vehicle routing with applications to multi-UAV mission planning”. In: *International Journal of Robust and Nonlinear Control* 21.12 (2011), pp. 1372–1395.

[41] S. Karaman and E. Frazzoli. “Vehicle routing problem with metric temporal logic specifications”. In: *2008 47th IEEE Conference on Decision and Control*. IEEE. 2008, pp. 3953–3958.

[42] S. Karaman, R. G. Sanfelice, and E. Frazzoli. “Optimal control of mixed logical dynamical systems with linear temporal logic specifications”. In: *Proceedings IEEE CDC*. 2008, pp. 2117–2122.

[43] A. Khasawneh, H. Rogers, J. Bertrand, K. C. Madathil, and A. Gramopadhye. “Human adaptation to latency in teleoperated multi-robot human-agent search and rescue teams”. In: *Automation in Construction* 99 (2019), pp. 265–277.

[44] D.-H. Kim, Y.-J. Kim, K.-C. Kim, J.-H. Kim, and P. Vadakkepat. “Vector field based path planning and Petri-net based role selection mechanism with Q-learning for the soccer robot system”. In: *Intelligent Automation & Soft Computing* 6.1 (2000), pp. 75–87.

[45] M. Kloetzer and C. Belta. “Automatic deployment of distributed teams of robots from temporal logic motion specifications”. In: *IEEE Transactions on Robotics* 26.1 (2010), pp. 48–61.

[46] M. Kloetzer and C. Belta. “Temporal logic planning and control of robotic swarms by hierarchical abstractions”. In: *IEEE Trans. Robotics* 23.2 (2007), pp. 320–330.

[47] A. Kolling and S. Carpin. “Multi-robot surveillance: an improved algorithm for the graph-clear problem”. In: *2008 IEEE International Conference on Robotics and Automation*. IEEE. 2008, pp. 2360–2365.

[48] A. Krontiris, Q. Sajid, and K. E. Bekris. “Towards using discrete multiagent pathfinding to address continuous problems”. In: *Workshops at the Twenty-Sixth AAAI Conference on Artificial Intelligence*. 2012.

[49] S. M. LaValle. *Planning algorithms*. Cambridge Univ. Press, 2006.

[50] D. Le and E. Plaku. “Cooperative multi-robot sampling-based motion planning with dynamics”. In: *Twenty-Seventh International Conference on Automated Planning and Scheduling*. 2017.

[51] J. Liu and N. Ozay. “Finite abstractions with robustness margins for temporal logic-based control synthesis”. In: *Nonlinear Analysis: Hybrid Systems* 22 (2016), pp. 1–15.

[52] Z. Liu, J. Dai, B. Wu, and H. Lin. “Communication-aware motion planning for multi-agent systems from signal temporal logic specifications”. In: *2017 American Control Conference (ACC)*. IEEE. 2017, pp. 2516–2521.

[53] Z. Liu, B. Wu, J. Dai, and H. Lin. “Distributed communication-aware motion planning for multi-agent systems from stl and spatel specifications”. In: *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE. 2017, pp. 4452–4457.

[54] J. Lofberg. “YALMIP: A toolbox for modeling and optimization in MATLAB”. In: *2004 IEEE international conference on robotics and automation (IEEE Cat. No. 04CH37508)*. IEEE. 2004, pp. 284–289.

[55] S. G. Loizou and K. J. Kyriakopoulos. “Automatic synthesis of multi-agent motion tasks based on ltl specifications”. In: *2004 43rd IEEE Conference on Decision and Control (CDC)(IEEE Cat. No. 04CH37601)*. Vol. 1. IEEE. 2004, pp. 153–158.

[56] R. J. Luna and K. E. Bekris. “Push and swap: Fast cooperative path-finding with completeness guarantees”. In: *Twenty-Second International Joint Conference on Artificial Intelligence*. 2011.

[57] H. Ma, W. Hönig, T. S. Kumar, N. Ayanian, and S. Koenig. “Lifelong path planning with kinematic constraints for multi-agent pickup and delivery”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 2019, pp. 7651–7658.

[58] H. Ma, T. S. Kumar, and S. Koenig. “Multi-Agent Path Finding with Delay Probabilities.” In: *AAAI*. 2017, pp. 3605–3612.

[59] A. Majumdar and R. Tedrake. “Funnel libraries for real-time robust feedback motion planning”. In: *The International Journal of Robotics Research* 36.8 (2017), pp. 947–982.

[60] R. Majumdar, E. Render, and P. Tabuada. “A theory of robust omega-regular software synthesis”. In: *ACM Transactions on Embedded Computing Systems (TECS)* 13.3 (2013), pp. 1–27.

[61] I. R. Manchester and J.-J. E. Slotine. “Control contraction metrics: Convex and intrinsic criteria for nonlinear feedback design”. In: *IEEE Transactions on Automatic Control* 62.6 (2017), pp. 3046–3053.

[62] D. Mellinger and V. Kumar. “Minimum snap trajectory generation and control for quadrotors”. In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE. 2011, pp. 2520–2525.

[63] Y. Meng and J. Gan. “A distributed swarm intelligence based algorithm for a cooperative multi-robot construction task”. In: *2008 IEEE Swarm Intelligence Symposium*. IEEE. 2008, pp. 1–6.

[64] M. Mesbahi and M. Egerstedt. *Graph theoretic methods in multiagent networks*. Princeton University Press, 2010.

[65] S. Moarref and H. Kress-Gazit. “Decentralized control of robotic swarms from high-level temporal logic specifications”. In: *Multi-Robot and Multi-Agent Systems (MRS), 2017 International Symposium on*. IEEE. 2017, pp. 17–23.

[66] K. Nagatani, S. Kiribayashi, Y. Okada, K. Otake, K. Yoshida, S. Tadokoro, T. Nishimura, T. Yoshida, E. Koyanagi, M. Fukushima, et al. "Emergency response to the nuclear accident at the Fukushima Daiichi Nuclear Power Plants using mobile rescue robots". In: *Journal of Field Robotics* 30.1 (2013), pp. 44–63.

[67] P. Nilsson and N. Ozay. "Control Synthesis for Large Collections of Systems with Mode-Counting Constraints". In: *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control.* ACM. 2016, pp. 205–214.

[68] P. Nilsson and N. Ozay. "Control synthesis for permutation-symmetric high-dimensional systems with counting constraints". In: *IEEE Transactions on Automatic Control* (2019).

[69] I. R. Nourbakhsh, K. Sycara, M. Koes, M. Yong, M. Lewis, and S. Burion. "Human-robot teaming for search and rescue". In: *IEEE Pervasive Computing* 4.1 (2005), pp. 72–79.

[70] K.-K. Oh, M.-C. Park, and H.-S. Ahn. "A survey of multi-agent formation control". In: *Automatica* 53 (2015), pp. 424–440.

[71] N. Ozay and P. Tabuada. "Guest editorial: special issue on formal methods in control". In: *Discrete Event Dynamic Systems* 27.2 (2017), pp. 205–208.

[72] D. Panagou. "Motion planning and collision avoidance using navigation vector fields". In: *2014 IEEE International Conference on Robotics and Automation (ICRA).* IEEE. 2014, pp. 2513–2518.

[73] A. A. Paranjape, K. C. Meier, X. Shi, S.-J. Chung, and S. Hutchinson. "Motion primitives and 3D path planning for fast flight through a forest". In: *The International Journal of Robotics Research* 34.3 (2015), pp. 357–377.

[74] H. Park and S. A. Hutchinson. "Fault-tolerant rendezvous of multirobot systems". In: *IEEE transactions on robotics* 33.3 (2017), pp. 565–582.

[75] L. E. Parker. "ALLIANCE: An architecture for fault tolerant multirobot cooperation". In: *IEEE transactions on robotics and automation* 14.2 (1998), pp. 220–240.

[76] K. Petersen, R. Nagpal, and J. Werfel. "Termes: An autonomous robotic system for three-dimensional collective construction". In: *Proceedings Robotics: Science & Systems VII* (2011).

[77] D. Pickem, P. Glotfelter, L. Wang, M. Mote, A. Ames, E. Feron, and M. Egerstedt. "The Robotarium: A remotely accessible swarm robotics research testbed". In: *Robotics and Automation (ICRA), 2017 IEEE International Conference on.* IEEE. 2017, pp. 1699–1706.

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

[79] G. Pola, A. Girard, and P. Tabuada. "Approximately bisimilar symbolic models for nonlinear control systems". In: *Automatica* 44.10 (2008), pp. 2508–2516.

[80] D. Portugal and R. P. Rocha. "Distributed multi-robot patrol: A scalable and fault-tolerant framework". In: *Robotics and Autonomous Systems* 61.12 (2013), pp. 1572–1587.

[81] S. Reveliotis and E. Roszkowska. "Conflict resolution in multi-vehicle systems: A resource allocation paradigm". In: *2008 IEEE International Conference on Automation Science and Engineering*. IEEE. 2008, pp. 115–121.

[82] A. Richards and J. P. How. "Aircraft trajectory planning with collision avoidance using mixed integer linear programming". In: *Proceedings of the 2002 American Control Conference (IEEE Cat. No. CH37301)*. Vol. 3. IEEE. 2002, pp. 1936–1941.

[83] E. J. Rodriguez-Seda, J. J. Troy, C. A. Erignac, P. Murray, D. M. Stipanovic, and M. W. Spong. "Bilateral teleoperation of multiple mobile agents: Coordinated motion and collision avoidance". In: *IEEE Transactions on Control Systems Technology* 18.4 (2010), pp. 984–992.

[84] E. Roszkowska and S. Reveliotis. "A distributed protocol for motion coordination in free-range vehicular systems". In: *Automatica* 49.6 (2013), pp. 1639–1653.

[85] I. Saha, R. Ramaithitima, V. Kumar, G. J. Pappas, and S. A. Seshia. "Implan: Scalable Incremental Motion Planning for Multi-Robot Systems". In: *2016 ACM/IEEE 7th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE. 2016, pp. 1–10.

[86] Y. E. Sahin, N. Ozay, and S. Tripakis. "Multi-Agent Coordination Subject to Counting Constraints: A Hierarchical Approach". In: *Proceedings of the 14th International Symp. on Distributed Autonomous Robotic Systems (DARS)*. 2018.

[87] Y. E. Sahin, P. Nilsson, and N. Ozay. "Multirobot coordination with counting temporal logics". In: *IEEE Transactions on Robotics* (2019).

[88] Y. E. Sahin, P. Nilsson, and N. Ozay. "Provably-correct coordination of large collections of agents with counting temporal logic constraints". In: *Proceedings of the 8th International Conference on Cyber-Physical Systems*. ACM. 2017, pp. 249–258.

[89] Y. E. Sahin, P. Nilsson, and N. Ozay. "Synchronous and Asynchronous Multi-agent Coordination with cLTL+ Constraints". In: *Decision and Control (CDC), 2017 IEEE 56th Annual Conference on*. IEEE. 2017.

[90] Y. E. Sahin and N. Ozay. "From Drinking Philosophers to Wandering Robots". In: *arXiv preprint arXiv:2001.00440* (2020).

[91] B. Şenbaşlar, W. Honig, and N. Ayanian. "Robust trajectory execution for multi-robot teams using distributed real-time replanning". In: *Distributed Autonomous Robotic Systems*. Springer, 2019, pp. 167–181.

[92] T. Setter, A. Fouraker, M. Egerstedt, and H. Kawashima. “Haptic interactions with multi-robot swarms using manipulability”. In: *Journal of Human-Robot Interaction* 4.1 (2015), pp. 60–74.

[93] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant. “Conflict-based search for optimal multi-agent pathfinding”. In: *Artificial Intelligence* 219 (2015), pp. 40–66.

[94] G. Sharon, R. Stern, M. Goldenberg, and A. Felner. “The increasing cost tree search for optimal multi-agent pathfinding”. In: *Artificial Intelligence* 195 (2013), pp. 470–495.

[95] Y. Shoukry, P. Nuzzo, A. Balkan, I. Saha, A. L. Sangiovanni-Vincentelli, S. A. Seshia, G. J. Pappas, and P. Tabuada. “Linear temporal logic motion planning for teams of underactuated robots using satisfiability modulo convex programming”. In: *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE. 2017, pp. 1132–1137.

[96] D. Silver. “Cooperative Pathfinding.” In: *AIIDE* 1 (2005), pp. 117–122.

[97] S. L. Smith, M. Schwager, and D. Rus. “Persistent robotic tasks: Monitoring and sweeping in changing environments”. In: *IEEE Transactions on Robotics* 28.2 (2011), pp. 410–426.

[98] K. Solovey, O. Salzman, and D. Halperin. “Finding a needle in an exponential haystack: Discrete RRT for exploration of implicit roadmaps in multi-robot motion planning”. In: *Algorithmic Foundations of Robotics XI*. Springer, 2015, pp. 591–607.

[99] D. E. Soltero, S. L. Smith, and D. Rus. “Collision avoidance for persistent monitoring in multi-robot systems with intersecting trajectories”. In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2011, pp. 3645–3652.

[100] P. Surynek. “A novel approach to path planning for multiple robots in bi-connected graphs”. In: *2009 IEEE International Conference on Robotics and Automation*. IEEE. 2009, pp. 3613–3619.

[101] P. Surynek. “An Optimization Variant of Multi-Robot Path Planning Is Intractable.” In: *AAAI*. 2010.

[102] P. Surynek. “Towards optimal cooperative path planning in hard setups through satisfiability solving”. In: *Pacific Rim International Conference on Artificial Intelligence*. Springer. 2012, pp. 564–576.

[103] P. Surynek, A. Felner, R. Stern, and E. Boyarski. “Efficient SAT approach to multi-agent path finding under the sum of costs objective”. In: *Proceedings of the Twenty-second European Conference on Artificial Intelligence*. IOS Press. 2016, pp. 810–818.

[104] R. Tedrake, I. R. Manchester, M. Tobenkin, and J. W. Roberts. “LQR-trees: Feedback motion planning via sums-of-squares verification”. In: *The International Journal of Robotics Research* 29.8 (2010), pp. 1038–1052.

[105] J. Tumova and D. V. Dimarogonas. “Multi-agent planning under local LTL specifications and event-based synchronization”. In: *Automatica* 70 (2016), pp. 239–248.

[106] A. Ulusoy, S. L. Smith, X. C. Ding, C. Belta, and D. Rus. “Optimality and robustness in multi-robot path planning with temporal logic constraints”. In: *The International Journal of Robotics Research* 32.8 (2013), pp. 889–911.

[107] J. Van Den Berg, S. J. Guy, M. Lin, and D. Manocha. “Reciprocal n-body collision avoidance”. In: *Robotics research*. Springer, 2011, pp. 3–19.

[108] J. Van den Berg, M. Lin, and D. Manocha. “Reciprocal velocity obstacles for real-time multi-agent navigation”. In: *2008 IEEE International Conference on Robotics and Automation*. IEEE. 2008, pp. 1928–1935.

[109] G. Wagner and H. Choset. “M*: A complete multirobot path planning algorithm with performance bounds”. In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2011, pp. 3260–3267.

[110] G. Wagner and H. Choset. “Subdimensional expansion for multirobot path planning”. In: *Artificial Intelligence* 219 (2015), pp. 1–24.

[111] L. Wang, A. D. Ames, and M. Egerstedt. “Safety Barrier Certificates for Collisions-Free Multirobot Systems”. In: *IEEE Transactions on Robotics* 33.3 (June 2017), pp. 661–674.

[112] J. L. Welch and N. A. Lynch. “A modular drinking philosophers algorithm”. In: *Distributed Computing* 6.4 (1993), pp. 233–244.

[113] B. de Wilde, A. W. ter Mors, and C. Witteveen. “Push and rotate: cooperative multi-agent path planning”. In: *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*. International Foundation for Autonomous Agents and Multiagent Systems. 2013, pp. 87–94.

[114] T. Wongpiromsarn. “Formal methods for design and verification of embedded control systems: application to an autonomous vehicle”. PhD thesis. Citeseer, 2010.

[115] P. R. Wurman, R. D’Andrea, and M. Mountz. “Coordinating hundreds of cooperative, autonomous vehicles in warehouses”. In: *AI magazine* 29.1 (2008), p. 9.

[116] Z. Xu and A. A. Julius. “Census Signal Temporal Logic Inference for Multiagent Group Behavior Analysis”. In: *IEEE Transactions on Automation Science and Engineering* PP.99 (2016), pp. 1–14.

[117] L. Yang and N. Ozay. “Fault-tolerant output-feedback path planning with temporal logic constraints”. In: *2018 IEEE Conference on Decision and Control (CDC)*. IEEE. 2018, pp. 4032–4039.

[118] J. Yu and S. M. LaValle. “Optimal Multirobot Path Planning on Graphs: Complete Algorithms and Effective Heuristics”. In: *IEEE Transactions on Robotics* 32.5 (Oct. 2016), pp. 1163–1177.

[119] J. Yu and S. M. LaValle. “Planning optimal paths for multiple robots on graphs”. In: *Robotics and Automation (ICRA), 2013 IEEE International Conference on.* IEEE. 2013, pp. 3612–3617.

[120] J. Yu and S. M. LaValle. “Structure and intractability of optimal multi-robot path planning on graphs”. In: *Twenty-Seventh AAAI Conference on Artificial Intelligence.* 2013.

[121] D. Zhou, Z. Wang, S. Bandyopadhyay, and M. Schwager. “Fast, on-line collision avoidance for dynamic vehicles using buffered voronoi cells”. In: *IEEE Robotics and Automation Letters* 2.2 (2017), pp. 1047–1054.

[122] Y. Zhou, H. Hu, Y. Liu, and Z. Ding. “Collision and deadlock avoidance in multirobot systems: a distributed approach”. In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 47.7 (2017), pp. 1712–1726.

[123] Y. Zhou, H. Hu, Y. Liu, S.-W. Lin, and Z. Ding. “A distributed approach to robust control of multi-robot systems”. In: *Automatica* 98 (2018), pp. 1–13.

[124] Y. Zhou, H. Hu, Y. Liu, S.-W. Lin, and Z. Ding. “A distributed method to avoid higher-order deadlocks in multi-robot systems”. In: *Automatica* 112 (2020), p. 108706.

[125] I. H. Zohdy and H. A. Rakha. “Intersection management via vehicle connectivity: The intersection cooperative adaptive cruise control system concept”. In: *Journal of Intelligent Transportation Systems* 20.1 (2016), pp. 17–32.