HIGH-LEVEL COLLABORATIVE TASK PLANNING FOR HETEROGENEOUS MULTI-ROBOT SYSTEMS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Amy Fang

August 2024

HIGH-LEVEL COLLABORATIVE TASK PLANNING FOR HETEROGENEOUS MULTI-ROBOT SYSTEMS

Amy Fang, Ph.D.

Cornell University 2024

Temporal logics serve as a framework for expressing complex, temporally-extended specifications in a mathematical precise manner. Using formal synthesis techniques, these specifications can be automatically translated into high-level, correct-by-construction controllers for robots to execute. This enables the provision of robust guarantees regarding robot behavior and task feasibility. The diverse expressivity of different logics enables them to be used in a wide variety of robotic systems.

This dissertation focuses on synthesizing high-level controllers for heterogeneous robots accomplishing a global task. First, we formulate the autonomous participation problem for multi-robot systems. Using Linear Temporal Logic (LTL), robots autonomously distribute new sub-tasks while still ensuring the satisfaction of their current tasks. Each robot evaluates its ability to satisfy both its current task and the new sub-tasks, then resynthesizes its behavior accordingly. We then present a novel task grammar that extends LTL to increase its expressivity for formulating collaborative tasks in a multi-robot context. Current approaches often require users to specify the numbers and types of robots for the tasks; in contrast, our task grammar focuses on the actions required and how those relate to the robot executing them (e.g. "the same robot that picked up the package must drop it off"). We also provide a synthesis framework and synchronization policies for the robots to collaborate with each other when re-

quired. The work in this dissertation provides approaches for both discrete and continuous actions.

To increase robustness, this dissertation also includes a method for robots to replan and resynthesize their behavior in response to modifications in individual robot capabilities during execution. The replanning approach maintains task satisfaction while minimizing changes at both the global team assignment and local behavior levels.

Finally, the dissertation presents a decentralized, context-based, on-board planning algorithm for Earth-Observation (EO) satellite systems. Each satellite first decides whether it can participate, then if it should participate, and finally either formally verifies a potential team, or synthesizes an optimal team for the mission.

BIOGRAPHICAL SKETCH

Amy Fang is a Ph.D. candidate in the Verifiable Robotics Research Group (VRRG) in the Sibley School of Mechanical and Aerospace Engineering at Cornell University, as well as a National Defense Engineering Graduate (NDSEG) Fellow. In 2019, she received her B.S. in Mechanical Engineering and Minor in Business Analytics at the Massachusetts Institute of Technology.

Dedicated to my family

ACKNOWLEDGEMENTS

First off, thank you Hadas, for being not just an advisor but also an outstanding mentor. You have helped me become a better researcher, a more eloquent communicator, and a more confident engineer. I owe you more cookies than I can count.

Thank you to my committee members, Mark Campbell and Kirstin Petersen, for your guidance and feedback throughout my PhD. I am also grateful to the Verifiable Robotics Research Group for always providing insightful suggestions and directions with my research. A special shout-out to Himani - thanks for being my ride-or-die (and there was a lot of dying!) over the past five years. I couldn't have asked for a better friend to go on this journey with.

Lastly, I want to express my gratitude to my family. Mom, Dad, Allen - your endless love and support mean the world to me. None of this would have been possible without you.

This work is supported by DARPA-PA-19-03-01 and the National Defense Science & Engineering Graduate Fellowship (NDSEG) Program.

TABLE OF CONTENTS

Biographical Sketch . . . iii
Dedication . . . iv
Acknowledgements . . . v
Table of Contents . . . vi
List of Tables . . . x
List of Figures . . . xi

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Thesis Outline . . . 4

2 Background **6**
2.1 Formal Synthesis for Multi-Robot Systems . . . 6
2.2 Linear Temporal Logic (LTL) . . . 8
2.2.1 Syntax . . . 8
2.2.2 Semantics . . . 8
2.3 Büchi Automaton . . . 9
2.4 Automata-based Synthesis . . . 10

3 Summary of Included Papers **13**
3.1 Paper 1 . . . 13
3.1.1 Abstract . . . 13
3.1.2 Contributions . . . 14
3.2 Paper 2 . . . 15
3.2.1 Abstract . . . 15
3.2.2 Contributions . . . 16
3.3 Paper 3 . . . 17
3.3.1 Abstract . . . 17
3.3.2 Contributions . . . 18
3.4 Paper 4 . . . 19
3.4.1 Abstract . . . 19
3.4.2 Contributions . . . 20
3.5 Paper 5 . . . 21
3.5.1 Abstract . . . 21
3.5.2 Contributions . . . 22
3.5.3 Contributions of the Author . . . 24

4 Discussion and Conclusion **25**
4.1 Concluding Remarks . . . 25
4.2 Broader Impact . . . 26
4.3 Limitations and Future Work . . . 27

A Automated Task Updates of Temporal Logic Specifications for Heterogeneous Robots **29**
A.1 Introduction . . . 29
A.2 Preliminaries . . . 32
A.2.1 Linear Temporal Logic . . . 32
A.2.2 Büchi Automata . . . 33
A.3 Problem Setup . . . 34
A.3.1 Task Specification . . . 34
A.3.2 Robot Model . . . 35
A.3.3 Robot Behavior . . . 38
A.4 Problem Statement . . . 40
A.4.1 Example . . . 40
A.5 Approach . . . 42
A.5.1 Synthesis of Robot Behavior . . . 42
A.5.2 Task Allocation . . . 44
A.6 Results and Evaluation . . . 45
A.6.1 Simulation of Robot Behavior . . . 46
A.6.2 Task Allocation Performance . . . 47
A.7 Conclusion . . . 50

B High-Level, Collaborative Task Planning Grammar and Execution for Heterogeneous Agents **51**
B.1 Introduction . . . 51
B.2 Preliminaries . . . 56
B.2.1 Linear Temporal Logic . . . 56
B.2.2 Büchi Automata . . . 56
B.2.3 Agent Model . . . 57
B.3 Task Grammar - LTL$^{\psi}$ . . . 58
B.4 Control Synthesis for LTL$^{\psi}$ . . . 61
B.5 Approach . . . 63
B.5.1 Büchi Automaton for an LTL$^{\psi}$ Formula . . . 64
B.5.2 Agent Behavior for an LTL$^{\psi}$ Specification . . . 67
B.5.3 Finding Possible Individual Agent Bindings . . . 69
B.5.4 Agent Team Assignment . . . 70
B.5.5 Synthesis and Execution of Control and Synchronization Policies . . . 72
B.6 Results and Discussion . . . 73
B.7 Conclusion . . . 77

C Continuous Execution of High-Level Collaborative Tasks for Heterogeneous Robot Teams **79**
C.1 Introduction . . . 79
C.1.1 Related Work . . . 80
C.2 Preliminaries . . . 84

C.2.1 Linear Temporal Logic . . . 84
C.2.2 Büchi Automata . . . 85
C.3 Definitions . . . 85
C.3.1 Actions . . . 85
C.3.2 Robot Model . . . 86
C.3.3 Task Grammar - LTL$^{\psi}$ . . . 87
C.4 Problem Statement . . . 91
C.5 Approach . . . 92
C.5.1 Büchi Automaton for an LTL$^{\psi}$ Formula . . . 93
C.5.2 Constructing the Product Automaton . . . 95
C.5.3 Adding Intermediate Transitions to the Büchi Automaton 98
C.5.4 Updating the Product Automaton . . . 105
C.5.5 Robot Team Behavior . . . 107
C.5.6 Synthesis and Execution of Control and Synchronization Policies . . . 109
C.6 Results . . . 111
C.6.1 Demonstrations . . . 111
C.6.2 Example: Task 1 . . . 112
C.6.3 Example: Task 2 . . . 114
C.6.4 Computational Performance . . . 116
C.7 Conclusion . . . 118

D Online Resynthesis of High-Level Collaborative Tasks for Robots with Changing Capabilities 120
D.1 Introduction . . . 120
D.2 Task Grammar: LTL$^{\psi}$ . . . 123
D.3 Prior work - Robot Model and Büchi Automaton . . . 126
D.3.1 Robot Model . . . 126
D.3.2 Büchi Automaton for an LTL$^{\psi}$ Formula . . . 127
D.4 Behavior Synthesis . . . 127
D.5 Problem Setup . . . 129
D.5.1 Modifications . . . 129
D.5.2 Problem Statement . . . 130
D.5.3 Example . . . 131
D.6 Approach: Resynthesis Framework . . . 132
D.6.1 Evaluating Modified Robot's Behavior . . . 134
D.6.2 Updating the Product Automaton . . . 135
D.6.3 Binding (Re)Allocation . . . 139
D.7 Demonstration and Evaluation . . . 141
D.7.1 Mod 1: Adding Transitions . . . 142
D.7.2 Mod 2: Removing Transitions without Reallocation . . . 142
D.7.3 Mod 3: Removing Transitions with Reallocation . . . 143
D.8 Conclusion . . . 143

E Decentralized Context-Based Planning for Earth Observation Missions **145**
E.1 Introduction . . . 145
E.2 Methodology . . . 154
E.2.1 Overview . . . 154
E.2.2 Knowledge representation . . . 154
E.2.3 Reasoning over KG . . . 158
E.2.4 Sensor Planning Framework . . . 161
E.2.5 Verification and Synthesis of Team Assignment . . . 166
E.2.6 Teaming Algorithm . . . 168
E.3 Results . . . 168
E.3.1 Results from Knowledge Representation . . . 170
E.3.2 Results from Reasoning over KG . . . 171
E.3.3 Results from Sensor Planning Framework . . . 172
E.3.4 Results from Synthesis of Teaming Assignment . . . 173
E.4 Conclusion . . . 175

Bibliography **178**

LIST OF TABLES

B.1 Example teaming assignment with 20 robots 74

E.1 Benchmark Team From NASA and the computed probabilities of event detection from the sensors in each satellite 174

LIST OF FIGURES

2.1 Example Kripke structure for a 2D grid world 11

A.1 Example of a motion model (a), a capability (b), and robot model (c) 36
A.2 Example of non-additive cost. The robot, in blue, is tasked to go to points A and B. Performing the tasks at the same time costs less than doing them separately: $c_1 + c_3 < c_1 + c_2$. 39
A.3 Updated behavior for Robot 1. The robot starts in room 2, and its original trajectory for the current task is drawn in orange. It receives the new task at the star and updates its behavior based on the sub-tasks it assigned itself. The new behavior is shown in blue. The icons indicate the actions the robot takes. 47
A.4 Comparison of overall cost (a) and computation time (b) between our algorithm and the optimal algorithm when varying the number of robots. The error bars represent the min/max values of the simulations. 48
A.5 Comparison of computation time (a) and cost (b) between our algorithm and the optimal algorithm when varying the number of new tasks. The error bars represent the min/max values of the simulations. 49

B.1 Agent partial model: (a) λ_{area} (b) λ_{arm} (c) A_{green} 58
B.2 Agriculture environment and initial agent states. The green, blue, and pink agents are stationary; the orientation of their sensors are indicated by the colored boxes. 63
B.3 $\mathcal{B}$ for φ^{ψ} (Eq. B.4). The purple transitions illustrate a possible accepting trace. 66
B.4 A small portion of $\mathcal{G}_{green}$ 68
B.5 The final step in the synchronized behavior of the agent team with their corresponding actions. 74
B.6 Computation time as the number of agents (a) and bindings (b) increases. The error bars show min/max values. 76

C.1 Partial model of robot A_{blue} (see Sec. C.4): (a) λ_{beep} (b) λ_{motion} (c) A_{blue} 88
C.2 Büchi automaton for Task 1 90
C.3 Environment and robot setup 92
C.4 Updated Büchi automaton for Case 1 (e_1 is a self-transition and e_2 is not). The blue represents the original transitions, the pink represents the added intermediate states and transitions 103
C.5 Initial setup on the physical system 112
C.6 Robots executing task 1. The boxes on the right in each frame represents the pink and green robots' cameras. 113

C.7 Robots executing task 2 115
C.8 Comparing computation times between the instantaneous and non-instantaneous actions frameworks as the number of robots increases (a) and the number of bindings increases (b). The error bars represent min/max values. 117

D.1 Modifications to λ_{mot} in which (a) the robot can no longer move between rooms G and H (Δ_{mot}^{rem} is the set of red transitions), and (b) the robot can now move between rooms B and G (Δ_{mot}^{add} is the set of green transitions). 130
D.2 Environment and robot setup 131
D.3 Buchi Automaton $\mathcal{B}$ for the example in Sec. D.5.3. The highlighted transitions is β, the trace that the team of robots are collectively traversing to satisfy the task. 132
D.4 Overview of resynthesis process 133

E.1 System flowchart, showcasing the interactions between the components in our approach 155
E.2 Different types of knowledge in the probabilistic knowledge base, with their sources and their associated uncertainty 156
E.3 Probability of mission success as a function of mean number of satellites used per day for various teaming strategies. Results demonstrate that our approach is capable of creating teams and daily assignments out of those teams that compete and even improve upon expert-designed teams. 169
E.4 Excerpts from the Graph and Triples databases, showing information on the satellites and sensors currently in orbit 171
E.5 Visualization of a Principal Component Analysis of Embedding Vectors, showing that UniKER is capable of distinguishing useful satellites better than TransE. 173
E.6 Pareto front for a mission to monitor any eruptions of the Kīlauea volcano over two weeks, given a list of potential satellites. Each point on the plot corresponds to a different synthesized teaming plan. 175

CHAPTER 1

INTRODUCTION

1.1 Motivation

Heterogeneous multi-robot systems consist of robots with different capabilities often working together to achieve a common goal. This can be useful in a wide variety of applications, including warehouse automation and disaster relief. As new emergencies arise, the user may need to assign new tasks while the robots are already executing existing tasks. Robots interleaving these tasks rather than performing them sequentially can allow for more efficient execution. In addition, these tasks may also require collaboration across robots depending on what the tasks require and each robot's respective capabilities.

There is a wealth of literature for multi-agent task allocation and coalition formation [38, 44, 52]. Some common approaches involve optimization methods, where the system is represented using mathematical models, such as the vehicle routing problem [6,87] or constraint programming [5,46,49]. To increase efficiency, researchers have also proposed metaheuristic algorithms [67,81,113], such as genetic algorithms [79] and particle swarm optimization [115], which iteratively search to find an approximate solution. Game-theoretic models model systems as stochastic games through the use of Markov Decision Processes (MDPs) [82, 84, 97, 109]. Although these approaches demonstrate convergence to a locally optimal policy, they face scalability challenges when dealing with a large number of agents due to the exponential growth of the state space as more agents are added. The research presented in [82] tackles scalability issues in the multi-agent MDP problem by leveraging specific dependence structures among

robots.

Distributed approaches have also been proposed, such as contract net protocols [47,122] and auction-based algorithms [24,114], which typically require a set of robots to be designated as auctioneers that act as mediators for the group. Other methods with a flat hierarchy include token-passing [118] and swarm approaches for collective robot behavior [2, 93, 95], such as aggregation or shape formation.

Recently, researchers have been interested in using formal method techniques, such as correct-by-construction synthesis and temporal logics, to encode and solve multi-agent tasks [18,51,56,57,101]. Tasks written in temporal logics, including Linear Temporal Logic (LTL) and Signal Temporal Logic (STL), allow users to write tasks that may require complex action sequences or temporally extended constraints in a mathematically rigorous way.

This dissertation proposes frameworks to synthesize high-level controllers that allow a heterogeneous team of robots to satisfy interleaving tasks and actions written in temporal logic. We outline a distributed approach for robots to satisfy new tasks as they currently execute an existing task, resulting in the satisfaction of both tasks. The framework allows for the interleaving of the two tasks, increasing the efficiency of the robot behavior. Robots determine the cost associated with accomplishing each new sub-task, and pass around a task token to assign themselves to the tasks in a near-optimal way. All of these tasks are encoded in LTL.

To increase task expressivity, we also propose a task grammar that is an extension of LTL, called LTL$^{\psi}$, that can capture information about collaborative

tasks for multi-robot systems. It enables the interleaving of robot actions, alleviating the need for explicit task decomposition in order to assign agents to parts of the task. This task grammar is inspired by [73] and its concept of induced propositions, in which each proposition includes information about the number and type of agents, as well as a connector term that binds together the truth values of specific atomic propositions. Our task grammar LTL$^{\psi}$ also includes information about how the atomic propositions are related to one another, which represents the overall relationship between agents and task requirements. However, unlike [73], which considers navigation tasks where the same agents of a certain type may need to visit different regions, we generalize these tasks to any type of abstract action an robot may be able to perform. Another key assumption we relax is that agents do not need to only be categorized as one type, i.e. agents may have overlapping capabilities.

The task grammar eliminates the need to predetermine the number of robots required for a task. Using the concept of *bindings*, the task provides constraints on the relationship between robot assignments and team behavior; we can require certain parts of the task to be satisfied by the same agent without assigning the exact robot or type of robot *a priori*. (e.g. "the same robot(s) that picks up the package must also scan it"). In this way, we can treat this similar to a task allocation problem, where the goal is to assign robots to bindings. However, we remove the need of decomposed independent sub-tasks, which allows the user to express a richer set of tasks, particularly ones that need collaboration. Given these types of tasks and their capabilities, robots autonomously form teams based on the respective parts of the task they can execute such that the team can collectively satisfy the task. This work also considers automatically abstracting continuous actions and synthesizing high-level controllers that are

already correct-by-construction for continuous execution.

Multi-robot tasks often necessitate synchronized collaboration among robots. Similar to approaches in [19, 57, 100], our framework ensures that actions are executed in the correct sequence by incorporating synchronization constraints while the robots are executing their behavior. This involves robots waiting before executing actions concurrently. In our work, the execution of synchronous behavior for each agent is decentralized; robots independently execute their plans and only communicate with each other when synchronization is required. We also describe an approach for robots to react to changes in their capabilities, specifically when transitions in a capability are modified. Robots resynthesize during execution such that they maintain satisfaction of the overall task.

All the aforementioned work can be applied to a variety of multi-agent systems; our work done with satellite teaming for EO missions illustrates one such application. The proposed method allows satellites to autonomously form teams to satisfy Earth Observation (EO) missions (e.g. monitor a volcano) based on contextual information and their probability of success. This alleviates the need for ground-based planning, which can be slow and require expert input.

1.2 Thesis Outline

Chapter 2 provides a brief background for this dissertation, which includes an overview of Linear Temporal Logic, the system abstraction, and automata-based synthesis. Chapter 3 provides summaries of the papers included in this dissertation. The first paper addresses the problem of how to automatically syn-

thesize high-level controllers when new tasks are introduced during execution of existing tasks. The subsequent three papers provide a framework for encoding and synthesizing controllers for collaborative tasks. These papers introduce LTL$^{\psi}$, our novel task grammar for encoding collaborative tasks, provide a synthesis framework to satisfy an LTL$^{\psi}$ task for both discrete actions and ones that take varying duration, and a resynthesis approach when a robot's capability is modified during execution. The last paper explores context-aware teaming, in which satellites receive a task and autonomously decide whether or not they can and should participate in such a mission based on their current state and contextual information. Chapter 4 discusses the conclusions, limitations, and potential future work that can be done based on the work presented in this dissertation. The appendix includes the fully published papers, which provide details regarding definitions, algorithms, and analysis for all the work in this dissertation.

CHAPTER 2

BACKGROUND

This chapter reviews the background material necessary for this dissertation, including the abstraction of the system and the general approach of automata-based synthesis for LTL specifications.

2.1 Formal Synthesis for Multi-Robot Systems

High-level planning problem for homogeneous multi-robot teams or large swarms often address formation and navigation tasks [14, 18, 59]. For heterogeneous robots satisfying temporal logic specifications, a common approach is either to or assigned to agents *a priori* [34, 104], or to decompose them into independent sub-tasks before automatically allocating them to the robots. For instance, [33, 89] first automatically decompose a global automaton representing the task into independent sub-tasks. Then, to mitigate the challenges posed by state-space explosion, the authors synthesize parallel plans, then sequentially interconnect the individual automata using switch transitions.

Other methods extend existing temporal logics in order to provide a more expressive way to encode tasks that require multiple robots. These new grammars often enforce explicit constraints on the number or types of robots for each part of the task. For instance, [86] provides constraints on the number of robots necessary in regions using counting LTL. The authors of [64] use Capability Temporal Logic, an extension of STL, to encode both the number and capabilities necessary at specified locations, then finds an optimal teaming strategy by formulating the system as a Mixed-Integer Linear Program (MILP). In [73],

the authors introduce the concept of induced propositions, where each atomic proposition encodes information about the type, number of agents, and target regions, as well as connectors relating the truth values of certain proposition to each other.

To ensure the satisfaction of multi-robot temporal logic specifications in continuous time, where actions take varying duration to complete, a common approach is to encode the task in STL, which allows for discrete-time continuous signals [13, 40]. Other approaches include constructing hybrid controllers for LTL tasks [83], where actions with timing constraints are abstracted with the use of initiation and completion propositions. Authors in [76] propose iterative synthesize-then-check approaches to remove unsafe transitions.

Common to many frameworks is that the task allocation happens offline prior to execution; resynthesis during execution is not considered. However, in real-world situations, robots may face sensor failures or external disturbances.. Existing work considers robots that adapt their motion plan online to explore or navigate a partially known or uncertain workspace [11, 42, 54, 80]. Other work include scalable partially-observable Markov decision process (POMDP) planning [62] and risk-based planning [68,102] to account for dynamic environments.

Several works also discuss ways to increase robustness in the system to account for these changes. Authors of [123] provide a behavior tree framework to account for pre-defined categories of failures. The work in [121] considers synthesizing controllers for robustness against small perturbations. [50] propose a reactive planning approach that adapts to robots when they fail and can longer participate while still satisfying the global LTL task. The work presented in this

dissertation considers capability modifications in which a transition within a specific capability of a robot either fails or new transitions are introduced.

2.2 Linear Temporal Logic (LTL)

LTL is a discrete logic that can encode temporally extended tasks, such as surveillance or monitoring. LTL formulas are constructed from atomic propositions AP, where $\pi \in AP$ is a Boolean variable [29]. Each atomic proposition represents an abstraction of a robot action. For example, *scan* captures a robot scanning a package.

2.2.1 Syntax

LTL formulas are defined using the following syntax:

$$\varphi ::= \pi \mid \neg\varphi \mid \varphi \vee \varphi \mid \bigcirc \varphi \mid \varphi \,\mathcal{U}\, \varphi$$

where $\neg$ and $\vee$ are Boolean operators ("not" and "or", respectively), and $\bigcirc$ and $\mathcal{U}$ ("next" and "until", respectively) are temporal operators. Using these, we can also define conjunction $\varphi \wedge \varphi$, implication $\varphi \Rightarrow \varphi$, eventually $\Diamond\varphi$ = True $\mathcal{U}\, \varphi$, and always $\Box\varphi = \neg\Diamond\neg\varphi$.

2.2.2 Semantics

The semantics of an LTL formula φ are defined over an infinite trace $\sigma = \sigma(0)\sigma(1)\sigma(2)...$, where $\sigma(i)$ is the set of AP that are true at step i. We use $\sigma \models \varphi$ to

denote that the trace σ satisfies LTL formula φ.

Whether or not a trace σ satsifies an LTL formula φ is defined recursively [29]:

- $\sigma(i) \models \pi$ iff $\pi \in \sigma(i)$
- $\sigma(i) \models \neg\varphi$ iff $\varphi \notin \sigma(i)$
- $\sigma(i) \models \varphi_1 \vee \varphi_2$ iff $\sigma(i) \models \varphi_1$ or $\sigma(i) \models \varphi_2$
- $\sigma(i) \models \bigcirc\varphi$ iff $\sigma(i+1) \in \varphi$
- $\sigma(i) \models \varphi_1 \mathcal{U} \varphi_2$ iff $\exists k \geq i$ such that $\sigma(k) \models \varphi_2$ and $\forall i \leq j < k, \sigma(j) \models \varphi_1$

Intuitively, $\bigcirc\varphi$ is satisfied if φ is satisfied in the next step of the sequence; $\diamond\varphi$ is satisfied if there exists a step in the sequence in which φ is true; $\square\varphi$ is satisfied if φ is true at every step in σ; $\varphi_1 \ \mathcal{U} \ \varphi_2$ is satisfied if φ_1 remains true until φ_2 becomes true.

2.3 Büchi Automaton

An LTL formula φ can be translated into a Nondeterministic Büchi Automaton that accepts infinite traces if and only if they satisfy φ. A Büchi automaton is a tuple $\mathcal{B} = (Z, z_0, \Sigma_{\mathcal{B}}, \delta_{\mathcal{B}}, F)$, where

- Z is the set of states
- $z_0 \in Z$ is the initial state
- $\Sigma_{\mathcal{B}}$ is the input alphabet

- $\delta_{\mathcal{B}} : Z \times \Sigma_{\mathcal{B}} \times Z$ is the transition relation
- $F \subseteq Z$ is a set of accepting states

An infinite run of $\mathcal{B}$ over a word $\sigma = \sigma_1\sigma_2\sigma_3... \in \Sigma_{\mathcal{B}}$ is an infinite sequence of states $z = z_0z_1z_2...$ such that $(z_{i-1}, \sigma_i, z_i) \in \delta_{\mathcal{B}}$. A run is accepting if and only if $\text{Inf}(z) \cap F \neq \emptyset$, where $\text{Inf}(z)$ is the set of states that appear in z infinitely often [4]. In this dissertation, we use the tool Spot [28] to automatically transform an LTL specification into its Büchi automaton equivalent.

2.4 Automata-based Synthesis

Formal synthesis methods take a task specification and a model of the robot as inputs and outputs high-level controllers for the robot to satisfy the task (or informs the user that the specification is unrealizable). A common approach is to model the robot as a Kripke structure $\mathcal{K} = (S, s_0, \Delta, L)$, where

- S is the finite set of states
- s_0 is the set of initial states
- $\Delta \subseteq S \times S$ is the transition relation, where $\forall s \in S$, there exists a state $s' \in S$ such that $(s, s') \in \Delta$
- $L : S \rightarrow 2^{AP}$ is the labeling function, where $L(s) \subseteq AP$ is the set of propositions that are true over a state s

An example of a Kripke structure representing a robot's ability to navigate a 2D grid environment is shown in Fig. 2.1.

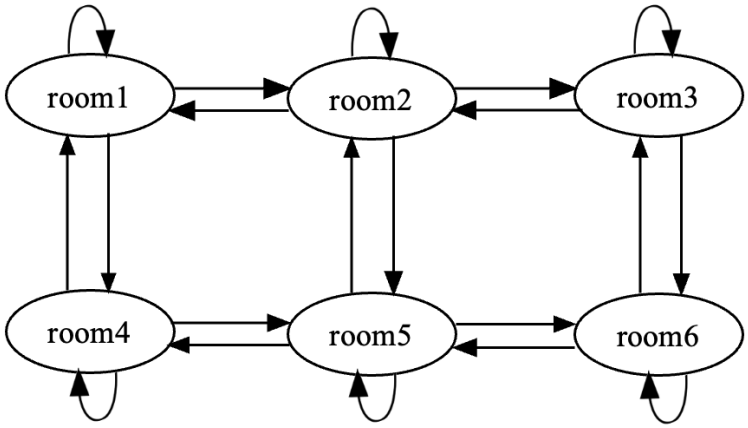


Figure 2.1: Example Kripke structure for a 2D grid world

A common approach to synthesizing a sequence of high-level controllers for the robot to satisfy φ is to find a trace through the product automaton. Formally, given a robot model $\mathcal{K} = (S, s_0, \Delta, L)$ and a desired task specification φ captured by the Büchi automaton $\mathcal{B} = (Z, z_0, \Sigma_{\mathcal{B}}, \delta_{\mathcal{B}}, F)$, we can synthesize a sequence of high-level controllers for the robot to satisfy φ by finding an accepting trace in the product automaton $\mathcal{G} = \mathcal{K} \times \mathcal{B} = (Q, q_0, AP_j, \delta_{\mathcal{G}}, L_{\mathcal{G}}, W_{\mathcal{G}}, F_{\mathcal{G}})$, where

- $Q = S \times Z$ is a finite set of states
- $q_0 = (s_0, z_0) \in Q$ is the initial state
- $L_{\mathcal{G}}$ is the labeling function such that $L^{\mathcal{G}}((s, z)) = L(s)$
- $\delta_{\mathcal{G}}$ is the transition function, where $\delta((s, z), (s', z')) = \Delta(s, s') \times \delta(z, z')$
- $W_{\mathcal{G}} : \gamma \to \mathbb{R}_{\geq 0}$ is the cost function such that for $q = (s, z)$ and $q' = (s', z')$, $W_{\mathcal{G}}((q, q')) = W((s, s'))$
- $F_{\mathcal{G}} = S \times F$ is a set of accepting states

We can find an infinite sequence $q = q_0 \to q_1 \to q_2 \to ...$ in $\mathcal{G}$ that visits the states in $F_{\mathcal{G}}$ infinitely often. The path is composed of a prefix – a finite trace – and a

suffix – a cycle that repeats. Then, a synthesized behavior b_i of a robot is defined as the labels produced by q: $b_i = L_{\mathcal{G}}(q_0)L_{\mathcal{G}}(q_1)L_{\mathcal{G}}(q_2)\ldots$.

CHAPTER 3

SUMMARY OF INCLUDED PAPERS

3.1 Paper 1

Automated Task Updates of Temporal Logic Specifications for Heterogeneous Robots

Amy Fang and Hadas Kress-Gazit

IEEE International Conference on Robotics (ICRA), 2022

3.1.1 Abstract

Given a heterogeneous group of robots executing a complex task represented in Linear Temporal Logic, and a new set of tasks for the group, we define the task update problem and propose a framework for automatically updating individual robot tasks given their respective existing tasks and capabilities. Our heuristic, token-based, conflict resolution task allocation algorithm generates a near-optimal assignment for the new task. We demonstrate the scalability of our approach through simulations of multi-robot tasks.

3.1.2 Contributions

In this paper, we propose a method for heterogeneous robots to respond to a new task given their capabilities and respective ongoing tasks while providing guarantees on task feasibility. The contributions are as follows:

1. a mathematical formulation of the new task distribution problem
2. a framework for robots to automatically update their behavior based on both the new task that is introduced and the progress within their current one, allowing them to perform both tasks
3. a heuristic, token-based task allocation algorithm to determine the final task allocation assignment for the new task while minimizing overall cost

3.2 Paper 2

High-Level, Collaborative Task Planning Grammar and Execution for Heterogeneous Agents

Amy Fang and Hadas Kress-Gazit

Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems, AAMAS '24

3.2.1 Abstract

We propose a new multi-agent task grammar to encode collaborative tasks for a team of heterogeneous agents that can have overlapping capabilities. The grammar allows users to specify the relationship between agents and parts of the task without providing explicit assignments or constraints on the number of agents required. We develop a method to automatically find a team of agents and synthesize correct-by-construction control with synchronization policies to satisfy the task. We demonstrate the scalability of our approach through simulation and compare our method to existing task grammars that encode multi-agent tasks.

3.2.2 Contributions

We propose a task description and control synthesis framework for heterogeneous agents to satisfy collaborative tasks. Specifically, we present a new, LTL-based task grammar for the formulation of collaborative tasks, and provide a framework to form a team of agents and synthesize control and synchronization policies to guarantee the team satisfies the task. We demonstrate our approach in simulated precision agriculture scenarios.

3.3 Paper 3

Continuous Execution of High-Level Collaborative Tasks for Heterogeneous Robot Teams

Amy Fang, Tenny Yin, Jiawei Lin and Hadas Kress-Gazit

IEEE Transactions on Robotics (T-RO), under review

3.3.1 Abstract

We propose a control synthesis framework for a heterogeneous multi-robot system to satisfy collaborative tasks, where actions may take varying duration of time to complete. We encode tasks using the discrete logic LTL$^{\psi}$, which uses the concept of bindings to interleave robot actions and express information about relationship between specific task requirements and robot assignments. We present a synthesis approach to automatically generate a teaming assignment and corresponding discrete behavior that is correct-by-construction for continuous execution, while also implementing synchronization policies to ensure collaborative portions of the task are satisfied. We demonstrate our approach on a physical multi-robot system.

3.3.2 Contributions

In this paper, we propose a framework for control synthesis for continuous execution of collaborative tasks, where actions may take varying duration of time to complete. Based on [31], we use LTL$^{\psi}$ to encode such tasks and present a synthesis approach to automatically generate a teaming assignment and corresponding symbolic behavior that is correct-by-construction during continuous execution, while also ensuring synchronization requirements are satisfied for collaborative portions of the task. We demonstrate our approach on a physical multi-robot system.

3.4 Paper 4

Online Resynthesis of High-Level Collaborative Tasks for Robots with Changing Capabilities

Amy Fang, Tenny Yin and Hadas Kress-Gazit

IEEE Robotics and Automation Letters, under review

3.4.1 Abstract

Given a collaborative high-level task and a team of heterogeneous robots and behaviors to satisfy it, this work focuses on the challenge of automatically, at runtime, adjusting the individual robot behaviors such that the task is still satisfied, when robots encounter changes to their abilities–either failures or additional actions they can perform. We consider tasks encoded in LTL$^{\psi}$and minimize global teaming reassignments (and as a result, local resynthesis) when robots' capabilities change. We also increase the expressivity of LTL$^{\psi}$ by including additional types of constraints on the overall teaming assignment that the user can specify, such as the minimum number of robots required for each assignment. We demonstrate the framework in a simulated warehouse scenario.

3.4.2 Contributions

In the context of synthesizing team and robot control from a high-level specification given in LTL$^{\psi}$, we 1) increase the specification expressivity by allowing the user to provide constraints regarding the binding assignments (i.e. the minimum number of robots assigned to each binding and which bindings are allowed to be assigned together), and 2) propose a resynthesis framework for online adaptation to changes in robot capabilities. We demonstrate our approach in a simulated warehouse scenario.

3.5 Paper 5

Decentralized Context-Based On-Board Planning for Earth Observation Missions

Antoni Viros Martin, Kewei Cheng, Amy Fang, Zhaoliang Zheng, Hadas Kress-Gazit, Ankur Mehta, Daniel Selva and Yizhou Sun

AIAA Scitech 2021 Forum

3.5.1 Abstract

Currently, the observation plans for most Earth observation satellites are static and periodic, and centrally uploaded from ground control. When there is an event of interest that requires changing the plan, new plans are manually patched and uploaded to the spacecraft. This approach is limited in that: 1) it may lead to missing short-lived phenomena, and 2) it does not scale well to very large constellations of satellites. New mission concepts are starting to leverage Artificial Intelligence to make autonomous planning decisions in reaction to short-lived events, which has spurred interest in on board autonomous scheduling. In addition, emerging mission concepts rely more on distributed systems (clusters, constellations) of heterogeneous satellites, in some cases with the ability to communicate through cross-links to coordinate their operation. Motivated by these recent trends towards increased system complexity, decentralization and on-board autonomy and advances in sensor fusion theory, this paper explores a significantly different observing paradigm and corresponding

problem in which satellites receive a mission (observations of a target region with a certain set of required attributes and time constraints) from mission control and autonomously decide whether or not they can and should participate in such a mission based on their current state and contextual information. Specifically, we propose a new approach based on three main steps: first, each satellite determines if it *can* participate in the mission given the mission specification and its sensor characteristics by reasoning over a knowledge graph; second, each satellite assesses if it *should* participate in the mission by estimating the probability that it can measure the observable of interest using a decentralized Kalman filter. Third, a central node can then verify the emerging plan using a probabilistic temporal logic and synthesize a better plan if desired. The new approach is described in detail and evaluated for a case study concerning the detection of volcano eruptions. In order to validate our methodology, we compare a set of teams generated by our method for the problem of volcano eruption detection to a team that is currently being used at NASA to perform the same task, with positive results. Finally, we discuss the limitations of the current method and outline a future path for the method as a whole as well as the individual steps.

3.5.2 Contributions

The problem we are trying to solve is formulated as follows: given a **mission** specified by 1) a **geophysical parameter** of the Earth to observe such as sea surface height or soil moisture, or an **event** such as a volcano eruption, 2) a **region** (e.g. the Arctic, the Mediterranean basin, or the Pacific Ring of Fire), and 3) some **time-related requirements** (mission duration of a week or month, revisit

time of two measurements a day, or a maximum gap between measurements of 24 hours), find **a selection of teams of satellites** that will perform the required observations to successfully carry out the mission. Optionally, the mission statement can also include other parameters such as spatial resolution, accuracy, etc. Of note, relevant requirements for many parameters can be readily obtained from the online World Meteorological Organization (WMO) OSCAR database[1].

We propose a new approach for the decentralized planning problem for heterogeneous, distributed (and potentially federated) EO systems. In this approach, agents independently choose whether and how to respond to a new mission through the use of contextual information. A preliminary offline step of our approach is the mining of such contextual information. Through this paper, we use an extensive definition of contextual information, including facts ranging from the capabilities of a satellite and its sensors such as accuracies, power, image resolution, fields of view, and other characteristics, as well as information about their current status including orbital information and whether sensors are operational or not. This contextual information also includes knowledge about the physics of the Earth system and remote sensing, such as how different so-called Level-1 measurements (e.g., TIR radiances or radar back-scatter cross-sections in L-band) relate to geophysical parameters or Level-2 products (e.g., land surface temperature or soil moisture) and the relation between a mission specification and a set of geophysical parameters to observe. This context could optionally include information about other satellites and their capabilities.

[1] `https://www.wmo-sat.info/oscar/`

3.5.3 Contributions of the Author

In our system, the aim of the verification step is to provide information about the likelihood of mission success given a teaming assignment and sensor properties. Individual agents decide whether to participate or not, but a centralized decision needs to be made to ensure that the mission constraints are met at a team level. For the verification step, the contributions are:

1. Formalizing the satellite mission as a specification encoded in Probabilistic Computation Tree Logic (PCTL)
2. Abstracting the system based on which agents and sensors are used, as well as which observations are being measured
3. Proposing a framework to verify a given team of agents or to synthesize an optimal one
4. Analyzing the output of the overall framework by generate a Pareto front for the multi-objective problem of maximizing the probability of mission success while minimizing the number of satellites for all viable teams measured

CHAPTER 4

DISCUSSION AND CONCLUSION

4.1 Concluding Remarks

The work in this dissertation focuses on synthesizing behavior for teams of robots to interleave their actions to accomplish a task, such as interleaving multiple tasks as they are introduced, or collaborative tasks that require coordination across multiple robots. We presented a distributed framework to allocate new tasks to robots as they are executing existing ones, as well as a new task grammar LTL$^{\psi}$ that extends the expressivity of LTL to heterogeneous multi-robot scenarios. We present methods to synthesize high-level correct-by-construction controllers for robots to satsify LTL$^{\psi}$ tasks for both discrete and continuous actions. We also provide a framework for robots to react to capability modifications. Throughout the included publications, we present simulated and physical demonstrations of multi-robot systems autonomously accomplishing LTL$^{\psi}$ specifications. We also analyze the computational complexity of each approach. Finally, we introduced a framework for satellites to autonomously join new EO missions based on their own set of sensors and likelihood to succeed. We use probabilistic model-checking techniques to both verify a team of satellites and synthesize new teams that are able to accomplish the mission above a predefined probability threshold.

4.2 Broader Impact

While this dissertation primarily focuses on autonomous robotic systems, it can be applied to any high-level, multi-agent planning problem as long as the actions and agents can be abstracted in the ways proposed in this work.

The distributed synthesis framework outlined in paper 1 for interleaving tasks has applications in disaster response; as new emergencies arise during these situations, robots are able to autonomously switch between tasks without human input. For example, autonomous warehouse robots can quickly respond to an emergency fire by directing people towards exits or close doors to stop the fire from spreading.

Our novel task grammar, LTL$^{\psi}$, allows users to express a new type of task that require team of heterogeneous agents to collaboratively satisfy. This grammar, along with the associated control synthesis techniques, eliminates the need to determine the exact number of agents required for a task beforehand. Instead, users can focus on describing the actions required to achieve the task.

LTL$^{\psi}$ synthesis can be used in any multi-agent application, such as warehouse automation, precision agriculture, or multi-robot navigation, where the user may not want or need to provide information about the number of agents necessary to accomplish a task. Additionally, this research extends the approach to synthesizing control for continuous actions and adapting to dynamic changes in robot capabilities, increasing its applicability in complex real-world scenarios.

In the last paper, we specifically address the problem of multi-agent teaming

in the satellite EO-mission domain. Rather than centrally upload satellite teaming plans from ground control, the proposed approach allows heterogeneous satellites to autonomously decide whether or not to participate in the new mission. This allows for more efficient and scalable approach in forming satellite teams.

4.3 Limitations and Future Work

A large assumption the work in this dissertation makes is that the robots are deterministic and their environment is known. Existing temporal logic literature proposes methods for introducing uncertainty in both the robot dynamics [104] and sensing and perception [56, 102]; a possible avenue for future work is to incorporate uncertainty in similar ways. By doing so, we can analyze how it impacts the robustness of the robot behavior, which may challenge the existing guarantees regarding task satisfiability.

Secondly, this dissertation relies on the assumption of seamless, all-to-all communication among the robots. This assumption is particularly critical in the context of distributed task allocation algorithms and synchronization protocols outlined in the respective papers. However, there are many practical limitations of communication systems, such as limited range or potential faults in communication links. Exploring strategies for maintaining satisfiability under these types of constraints or failures could increase the robustness of the algorithms proposed in this dissertation.

In addition, while optimization metrics are considered in the work outlined in papers 1 and 5, the remaining papers addressing synthesis for LTL$^{\psi}$ tasks do

not. Although individual robots do prioritize cost minimization when synthesizing their behavior, the absence of optimization metrics in team assignment decisions presents an area for future work. This could include incorporating optimization metrics regarding cost or assignment distribution during the team assignment phase.

The issue of tractability of the presented frameworks is another area for future research. As with many temporal logic methodologies, the scalability of the approach becomes increasingly challenging as the size of the robots' product automata expands; this occurs when the task complexity increases (thus the Büchi automaton grows), or when the number/complexity of the capabilities increase (e.g. the discretized workspace becomes larger).

To address these scalability concerns, there are several possible approaches. One involves exploring methods for sampling the product automata during the synthesis process rather than constructing it in its entirety. Utilizing greedy heuristic techniques in the synthesis process presents another avenue for managing scalability challenges. Leveraging these techniques to search for accepting traces within the product automata can streamline the synthesis process and improving overall scalability.

APPENDIX A

AUTOMATED TASK UPDATES OF TEMPORAL LOGIC SPECIFICATIONS FOR HETEROGENEOUS ROBOTS

A.1 Introduction

Heterogeneous multi-robot systems consist of robots with different capabilities and are often created with a specific task in mind. However, if the task is changed during execution, especially if more requirements are added to the previous ones, there is a need for automated techniques that would allocate the task to the robots while maintaining the previous task and minimizing cost. For example, in humanitarian aid or disaster response situations, as new emergencies arise and timing is critical, automating the process for robots to interleave new tasks into their existing tasks without human input would 1) increase efficiency in the assignment process 2) ensure that the teams are responding quickly.

In this paper, we address the problem of automatically updating robot behaviors given tasks encoded in Linear Temporal Logic (LTL) over an abstraction of the robot motion and capabilities. We assume each sub-task can be accomplished by a single robot, and that all tasks are defined over controllable atomic propositions. We also assume there exist continuous controllers that can implement the abstract behaviors in a way that ensures collision avoidance between the robots and guarantees that the continuous behaviors implement the abstract ones [61].

There exists a rich literature in addressing automated multi-robot task allocation and coalition formation [38, 41, 108]. Researchers have developed architec-

tures to model robot capabilities and interactions, such as social networks [58]. Game-theoretic models represent systems as stochastic games by using Markov Decision Processes (MDPs) or partially observable MDPs [109]. While these methods can be shown to converge to a locally optimal policy, they do not scale for large numbers of agents, since the state space grows exponentially as more agents are introduced. The work in [82] addresses scalability for the multi-agent MDP problem, but does so by assuming special dependence structures among robots.

Dynamic coalition formation has been of particular interest, where autonomous robots cooperate to perform emerging time-varying tasks. Current methods include greedy approximate algorithms [92], particle swarm optimization [115], and evolutionary algorithms [67]. Another method is to use market-based algorithms for robots to form teams based on the bids they make for the specified task [24, 114]. This requires a leader that acts as a mediator for the group. To maintain a flat hierarchy while still allocating tasks in a near-optimal manner, [116] proposes a token-based framework, where tasks and resources are abstracted as tokens and passed locally among agents. Each agent decides whether to keep the token or to pass it to other agents. This decision requires information about how the token has been passed around to other agents. In this paper, we develop a token-based scheme for task allocation that is able to maintain near-optimal results without any token history information.

To increase the complexity of possible tasks, in recent years there has been a growing interest in multi-robot planning for task specifications written in temporal logics, which enables users to specify temporally extended tasks. In [77], the authors use Time Window Temporal Logic to address multi-robot planning

with synchronization requirements. The authors of [53] encode Signal Temporal Logic specifications as mixed integer linear constraints to generate a plan for a heterogeneous team. The work in [89] presents an algorithm that allocates tasks to robots while simultaneously planning their actions. To avoid state-space explosion in their centralized planning, the authors sequentially link each robot model using switch transitions. In [88], the authors use reinforcement learning to synthesize plans over Markov decision processes under Linear Temporal Logic, and an auction-based algorithm assigns tasks to robots. All of these approaches assume that new tasks can only be directed towards unassigned robots.

There also exists literature that addresses the problem of rescheduling in response to unexpected disturbances. The authors of [106] propose a bargaining game approach to generate a real-time scheduling scheme for an Internet of Things-enabled job shop. The work in [111] presents an online hybrid contract-net negotiation protocol in response to unexpected disturbances in a shop. Agents place bids of their earliest finishing time, and a coordinator creates a new schedule accordingly. While these papers address task reallocation due to unexpected environment changes, to our knowledge, no work has been done to tackle the problem of multi-robot task distribution within the context of temporal logics, where new tasks are introduced to robots that are performing existing tasks.

Contributions: In this paper, we propose a method for heterogeneous robots to respond to a new task given their capabilities and respective ongoing tasks while providing guarantees on task feasibility. The contributions are as follows: 1) a mathematical formulation of the new task distribution problem, 2) a frame-

work for robots to automatically update their behavior based on both the new task that is introduced and the progress within their current one, allowing them to perform both tasks, and 3) a heuristic, token-based task allocation algorithm to determine the final task allocation assignment for the new task while minimizing overall cost.

A.2 Preliminaries

A.2.1 Linear Temporal Logic

Let AP be a set of atomic propositions such that $\pi \in AP$ is a Boolean variable. We use these propositions to capture robot capabilities. For example, *pick_up* can correspond to the robot performing a pick up action.

Syntax: An LTL formula [4] is defined recursively from atomic propositions $\pi \in AP$ using the following grammar:

$$\varphi ::= \pi \mid \neg\varphi \mid \varphi \vee \varphi \mid \bigcirc \varphi \mid \varphi \,\mathcal{U}\, \varphi$$

where $\neg$ ("not") and $\vee$ ("or") are Boolean operators. $\bigcirc$ and $\mathcal{U}$ are the temporal operators "next" and "until", respectively. Using these basic operators, we can construct the additional logical operators conjunction $\varphi \wedge \varphi$, implication $\varphi \Rightarrow \varphi$, and equivalence $\varphi \Leftrightarrow \varphi$, as well as the temporal operators eventually $\diamond\varphi$ and always $\square\varphi$.

Semantics: The semantics of an LTL formula φ are defined over a trace σ, where $\sigma = \sigma_0\sigma_1\sigma_2...$ is an infinite sequence, and σ_p represents the set of AP that

are True at position p. We denote that σ satisfies LTL formula φ as $\sigma \models \varphi$.

Intuitively, $\sigma \models \bigcirc\varphi$ if φ is True at the next state in the trace. To satisfy $\varphi^1 \,\mathcal{U}\, \varphi^2$, φ^1 must stay True until φ^2 becomes True. The formula $\Box\varphi$ is satisfied if φ is True at every position in σ, and $\sigma \models \Diamond\varphi$ if there exists a step in σ where φ is True. For a complete definition of the semantics of LTL, see [4].

A.2.2 Büchi Automata

A nondeterministic Büchi automaton can be constructed from an LTL formula such that an infinite trace is accepted by the Büchi automaton if and only if it satisfies the LTL formula [28]. A Büchi automaton is defined as a tuple $B = (\Sigma, Z, z_0, \delta, F)$, where Σ is the alphabet of B, Z is a finite set of states, $z_0 \in Z$ is the initial state, $\delta : Z \times \Sigma \to 2^Z$ is a transition function, and $F \subseteq Z$ is a set of accepting states. A run of a Büchi automaton on an infinite word $w = w_1 w_2 w_3 ...$ is an infinite sequence of states $z = z_0 z_1 z_2 ...$ such that $\forall i, w_i \in \Sigma$ and $(z_{i-1}, w_i, z_i) \in \delta$. A run is accepting if and only if $\inf(z) \cap F \neq \emptyset$, where $\inf(z)$ is defined as the set of states that are visited infinitely often in z [4].

Büchi intersection: Given two LTL formulas, φ^1 and φ^2 over AP, the intersection of their respective Büchi automata, B^1 and B^2, represents traces that satisfy both φ^1 and φ^2.

Let $B^1 = (\Sigma, Z_1, z_1^0, \delta_1, F_1), B^2 = (\Sigma, Z_2, z_2^0, \delta_2, F_2)$. Their intersection is defined as $B^1 \cap B^2 = (\Sigma, Z_1 \times Z_2 \times \{0,1,2\}, \{z_1^0, z_2^0, 0\}, \delta', Z_1 \times Z_2 \times \{2\})$. There is a transition on $a \in \Sigma$, $(\langle r, q, x\rangle, a, \langle r', q', x'\rangle) \in \delta'$ if and only if $(r, a, r') \in \delta_1$ and $(q, a, q') \in \delta_2$. The components $x, x' \in \{0,1,2\}$ are determined by F_1 and F_2, the accepting con-

ditions of the Büchi automata. It is insufficient to simply define the accepting states of $B^1 \cap B^2$ as $F_1 \times F_2$ – even if the accepting states from both automata appear individually infinitely often, they may appear together only finitely many times. Thus, $x, x' \in \{0, 1, 2\}$ ensures that accepting states from both B^1 and B^2 appear infinitely often together. For a detailed explanation on how to construct the intersection of two Büchi automata, see [25].

A.3 Problem Setup

A.3.1 Task Specification

A task is a set of LTL formulas $\Phi = \{\varphi^1, \varphi^2, ..., \varphi^m\}$ for which the following properties hold:

- ***Non-conflicting***: There exists a σ such that $\forall j, k \;\; \sigma \models \varphi^j \wedge \varphi^k$. Intuitively, this means that the satisfaction of one sub-task must not violate any other sub-task.
- ***Non-collaborative***: Every φ^j can be satisfied by a single robot.

Example. The task "pick up a box from room 2 and drop it off in room 3, pull the lever in room 3, and repeatedly scan and take a picture in room 1" can be encoded in LTL and decomposed into three sub-tasks:

$$\begin{aligned} \varphi^1 = (\neg \mathit{drop\_off}\ \mathcal{U}\ (\mathit{room}_2 \wedge \mathit{pick\_up})) \\ \wedge (\neg \mathit{drop\_off}\ \mathcal{U}\ (\mathit{room}_3 \wedge \mathit{drop\_off})) \end{aligned} \tag{A.1}$$

$$\varphi^2 = \Diamond(room_3 \wedge pull\_lever) \tag{A.2}$$

$$\varphi^3 = \Box\Diamond(room_1 \wedge scan \wedge use\_camera) \tag{A.3}$$

A.3.2 Robot Model

Each robot in the group has a set of capabilities related to the required tasks. We define a *capability* as a weighted transition system defined as a tuple $\lambda = (AP, S, s_0, R, L, W)$, where

- AP is a set of atomic propositions
- S is a finite set of states
- $s_0 \in S$ is the initial state
- $R \subseteq S \times S$ is a transition relation where for all $s \in S$ there exists $s' \in S$ such that $(s, s') \in R$
- $L : S \rightarrow 2^{AP}$ is the labeling function such that $L(s) \subseteq AP$ is the set of AP that are true in state s
- $W : R \rightarrow \mathbb{R}_{\geq 0}$ is the cost function

Let there be a set of v capabilities, $\Lambda = \{\lambda_1, \lambda_2, ...\lambda_v\}$ where $\lambda_k = (AP_k, S_k, s_{0,k}, R_k, L_k, W_k)$. We consider robots that are heterogeneous, where each robot has its own capability set $\Lambda_i \subseteq \Lambda$.

We assume all robots are moving in a shared workspace that is partitioned into a set of regions. We describe the possible motion of a robot as a motion

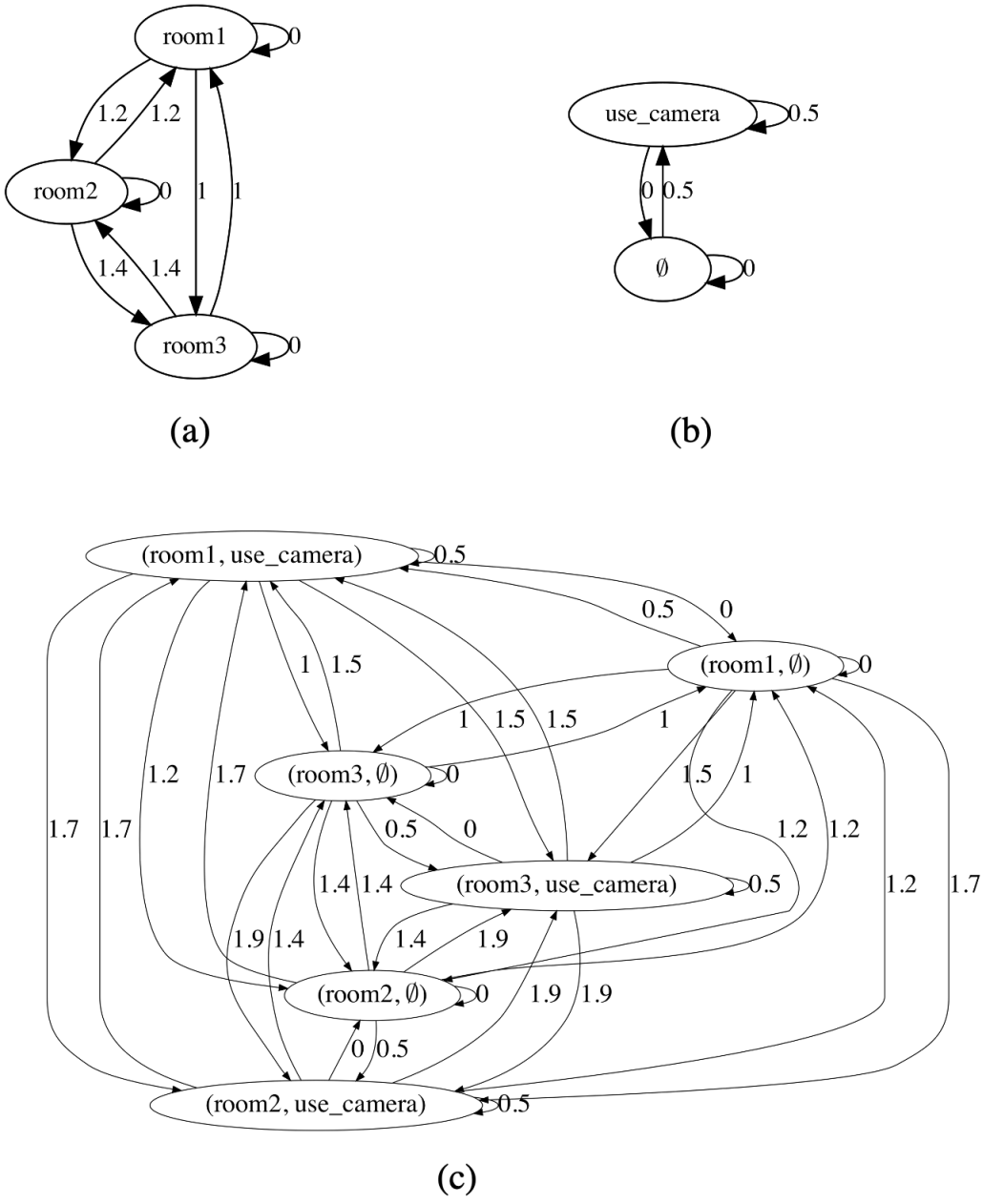


Figure A.1: Example of a motion model (a), a capability (b), and robot model (c)

capability λ_{Mot}, where AP is the set of region labels, and $(s, s') \in R$ if and only if a robot can directly move from the region labeled with $L(s)$ to the region labeled with $L(s')$.

A robot model A_i is given by the product of its motion model and $u = |\Lambda_i|$ capabilities: $A_i = \lambda_{Mot} \times \lambda_1 \times ... \times \lambda_u$ such that $A_i = (AP, S, s_0, R, L, W)$, where

- $AP = \bigcup_{k=1}^{u} AP_k$
- $S = S_{Mot} \times S_1 \times S_2 \times ... \times S_u$
- $s_0 = (s_{0,Mot}, s_{0,1}, s_{0,2}, ..., s_{0,u})$
- $R \subseteq S \times S$ is a transition relation where for all $s = (s_{Mot}, s_1, s_2, ..., s_u)$ in S, there exists $s' = (s'_{Mot}, s'_1, s'_2, ..., s'_u)$ in S such that $(s_{Mot}, s'_{Mot}) \in R_{Mot}, (s_1, s'_1) \in R_1, (s_2, s'_2) \in R_2, ..., (s_u, s'_u) \in R_u$.
- L is the labeling function such that $L(s) = (L_{Mot}(s_{Mot}), L_1(s_1), L_2(s_2), ..., L_k(s_k))$
- $W : R \to \mathbb{R}_{\geq 0}$ is the cost function

Let $s = (s_{Mot}, s_1, s_2, ..., s_u)$ and $s' = (s'_{Mot}, s'_1, s'_2, ..., s'_u)$ be two states in A_i for which the transition (s, s') exists. Then, the cost $W((s, s'))$ is the sum over $W((s_{Mot}, s'_{Mot}))$ and all $W((s_k, s'_k))$. See Fig. A.1 for a simple example. In this motion model, the cost for transition $(room_1, room_2)$ is $W_{Mot}((room_1, room_2)) = 1.2$, and the cost for the transition $(\emptyset, use\_camera)$ in the capability is $W_1((\emptyset, use\_camera)) = 0.5$. Then, in the robot model, the cost of the transition between states $(room_1, \emptyset)$ and $(room_2, use\_camera)$ is 1.7.

A.3.3 Robot Behavior

Given a robot model A_i and a desired behavior φ captured using B^φ, we can synthesize a behavior for the robot such that it satisfies φ by choosing an accepting trace in $\mathcal{G} = A_i \times B^\varphi$ [4]. Let $A_i = (AP^i, S, s_0, R, L, W)$ and $B^\varphi = (\Sigma, Z, z_0, \delta, F)$ such that $\Sigma = 2^{AP^\varphi}$ is created from AP^φ, the propositions in φ. Note that it is not necessary for AP^i to be equivalent to AP^φ. $AP^i \nsubseteq AP^\varphi$ when a robot has additional capabilities that are not required for task φ. Similarly, $AP^\varphi \nsubseteq AP^i$ when a robot does not have all the necessary capabilities required for task φ.

The product $\mathcal{G} = (\Sigma, AP^i, Q, q_0, L^\mathcal{G}, \Delta, W^\mathcal{G}, F^\mathcal{G})$, where

- $Q = S \times Z$ is a finite set of states
- $q_0 = (s_0, z_0)$ is the initial state
- $L^\mathcal{G}$ is the labeling function such that $L^\mathcal{G}((s, z)) = L(s) \cap AP^\varphi$
- Δ is the transition function, where a transition exists from (s, z) to (s', z') if and only if $(s, s') \in R$, and $\exists\, \sigma \in \Sigma$ such that $\sigma = L^\mathcal{G}(s')$ and $z' \in \delta(z, \sigma)$
- $W^\mathcal{G} : R \to \mathbb{R}_{\geq 0}$ is the cost function such that for $q = (s, z)$ and $q' = (s', z')$, $W^\mathcal{G}((q, q')) = W((s, s'))$
- $F^\mathcal{G} = S \times F$ is a set of accepting states

Let path $q = q_0 \to q_1 \to q_2 \to ... \to q_\ell \to q_{\ell+1}...$ be an infinite sequence in $\mathcal{G}$ that visits the states in $F^\mathcal{G}$ infinitely often. The path is composed of a prefix – a finite trace – and a suffix – a cycle that repeats. A behavior b_i of a robot is defined as the labels produced by q: $b_i = L^\mathcal{G}(q_0)L^\mathcal{G}(q_1)L^\mathcal{G}(q_2)...L^\mathcal{G}(q_\ell)L^\mathcal{G}(q_{\ell+1})...$. Given a

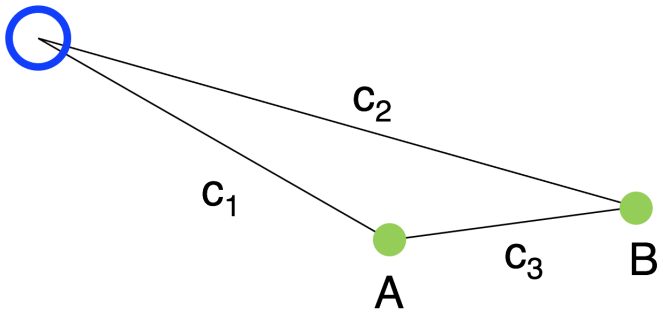


Figure A.2: Example of non-additive cost. The robot, in blue, is tasked to go to points A and B. Performing the tasks at the same time costs less than doing them separately: $c_1 + c_3 < c_1 + c_2$.

prefix of length ℓ, we define the cost of b_i as:

$$c_i(b_i) = \sum_{r=0}^{\ell-1} W^{\mathcal{G}}((q_r, q_{r+1})) \tag{A.4}$$

We allow the cost for each sub-task to be non-additive. That is, the sum of the costs of the behaviors for satisfying two individual sub-tasks may be more than the cost of the behavior for satisfying both. For a simple illustration, see Fig. A.2.

We define P to be a partition of Φ^{new} such that the following properties hold for $p_x \in P$:

$$\begin{aligned} p_x \cap p_y &= \emptyset, \quad \forall x \neq y \\ \bigcup_x p_x &= P \end{aligned} \tag{A.5}$$

Given a set of new sub-tasks $\Phi^{new} = \{\varphi^1, \varphi^2, ..., \varphi^m\}$, we define for each robot A_i the corresponding cost and satisfiability structures:

Cost Γ_i is a function that maps $\varphi_i^{curr} \bigwedge_{j \in p_i} \Phi^{new}[j]$ to its corresponding cost, $c_i(b_i^{new})$,

where $p_i \in P$ is the set of sub-tasks assigned to A_i.

Satisfiability ζ_i denotes which sub-tasks A_i is able to perform:

$$\zeta_i[j] = \begin{cases} 1 & \exists\, b_i \text{ such that } b_i \models \varphi^j \\ 0 & \text{otherwise} \end{cases}$$

A.4 Problem Statement

Let there be n heterogeneous robots $A = \{A_1, A_2, ..., A_n\}$. Each robot A_i has behavior b_i^{curr} that satisfies its existing task specification φ_i^{curr}.

Given a new task $\Phi^{new} = \{\varphi^1, \varphi^2, ..., \varphi^m\}$ that is introduced during the robots' execution of their current tasks, find a partition P, as defined in Eq. A.5, to assign robots to new sub-tasks so that $b_i^{new} \models \varphi_i^{curr} \bigwedge_{j \in p_i} \varphi^j$, subject to the following optimization criteria:

$$\arg\min_{p_i \in P} \sum_{i=1}^{n} c_i(b_i^{new}), \tag{A.6}$$

We make the following assumptions about the system: collision avoidance is taken care of by low-level controllers; the sub-tasks are nonreactive, meaning that the robot behavior does not depend on external events; each sub-task can be satisfied by a single robot and does not require robot collaboration, as outlined in A.3.1.

A.4.1 Example

Consider a 2D environment with five rooms containing three robots. The set of all capabilities is $\Lambda = \{\lambda_{Mot}, \lambda_{arm}, \lambda_{scan}, \lambda_{camera}\}$. AP_{arm} is an abstraction of a phys-

ical robot manipulator that is capable of grasping objects, such as boxes and levers. AP_{scan} represents a robot's ability to scan barcodes. Similarly, we abstract a robot's camera as AP_{camera}, which denotes whether or not the robot is taking a picture.

Each robot has the following capabilities and tasks:

- Robot 1: "Scan in room 4, then scan in room 1"

$$\varphi_1^{curr} = (\neg(room_1 \wedge scan)\ \mathcal{U}\ (room_4 \wedge scan)) \wedge \Diamond(room_1 \wedge scan) \quad \text{(A.7)}$$

$$\Lambda_1 = \{\lambda_{Mot}, \lambda_{arm}, \lambda_{scan}\}, s_0 = (room_2, \emptyset, \emptyset)$$

- Robot 2: "Repeatedly travel between rooms 2 and 5 and scan in those rooms"

$$\varphi_2^{curr} = \Box\Diamond(room_2 \wedge scan) \wedge \Box\Diamond(room_5 \wedge scan) \quad \text{(A.8)}$$

$$\Lambda_2 = \{\lambda_{Mot}, \lambda_{scan}, \lambda_{camera}\}, s_0 = (room_1, \emptyset, \emptyset)$$

- Robot 3: "Take a picture in room 1 and pick up a box in room 4, in any order"

$$\varphi_3^{curr} = \Diamond(room_1 \wedge use\_camera) \wedge \Diamond(room_4 \wedge pickup) \quad \text{(A.9)}$$

$$\Lambda_3 = \{\lambda_{Mot}, \lambda_{arm}, \lambda_{camera}\}, s_0 = (room_2, \emptyset, \emptyset)$$

The new task Φ^{new}, provided in Eq. A.1 - A.3, is introduced while the robots are executing these tasks.

A.5 Approach

The approach is as follows: each robot determines how much of the current task it has already accomplished. There are two reasons for this: 1) so that the robot does not repeat completed portions of the task (thus reducing cost), and 2) so that if the new task conflicts only with completed portions of the current task, the robot does not deem the new task as impossible to achieve. The robot then synthesizes the corresponding behavior for the new sub-tasks based on its capabilities and the remaining current task. It calculates the cost of performing different feasible combinations of sub-tasks (Sec. A.5.1). To determine the assignment of tasks that minimizes the overall cost for the robots, we develop a token-based, conflict resolution task allocation algorithm. Robots pass around an assignment token and assign themselves to tasks based on the cost of the corresponding behavior (Sec. A.5.2).

A.5.1 Synthesis of Robot Behavior

Algorithm 1: Synthesize Behavior

Input : A_i, z_i, B_i^{curr}, φ^k
Output: b_i^{new}, $c_i(b_i^{new})$

1 B^k := LTL2BUCHI(φ^k)
2 B_i^{curr} := FIND_REACHABLE_BUCHI(z_i, B_i^{curr})
3 B_i^k := CREATE_BUCHI_INTERSECT(B^k, B_i^{curr})
4 $\mathcal{G} = A_i \times B_i^k$
// Let $F^{\mathcal{G}}$:= be the set of accepting states in $\mathcal{G}$
// Let q_i := initial state of $\mathcal{G}$
5 $b_i^{new}, c_i(b_i^{new})$:= DIJKSTRA($\mathcal{G}, q_i, F^{\mathcal{G}}$)
6 **if** $b_i^{new} = \emptyset$ **then**
7 $\quad c_i(b_i^{new}) := \infty$

Given a new task, each robot runs Alg. 1 to automatically synthesize a new behavior. We transform the LTL formula φ^k into Büchi automaton B^k using

Spot [28] (line 1).

To synthesize a behavior for a robot that would cause it to perform both its current task and φ^k, we first determine what the robot needs to do to complete its current task. To do so, we calculate the reachable portion of B_i^{curr} from the state the robot is in when the new task is introduced, denoted as z_i. To generate the reachable portion of B_i^{curr}, the function FIND_REACHABLE_BUCHI calculates the forward reachable set [1] and removes any non-reachable states.

The function CREATE_BUCHI_INTERSECT finds the intersection of B_i^{curr}, the current task remaining, and B^k representing the new task (line 3). The alphabets $\Sigma_i^{curr} = 2^{AP_i^{curr}}, \Sigma^k = 2^{AP^k}$ of the respective Büchi automata might not be equivalent, since the task specifications φ may require different capabilities and thus be defined over different APs.

Borrowing from the definition in Sec. A.2.2, the Büchi intersection B_i^k has the alphabet $\Sigma_i^k = 2^{AP_i^{curr} \cup AP^k}$. Given $\sigma \in \Sigma_i^k$, a transition $(\langle r, q, x\rangle, \sigma, \langle r', q', x'\rangle) \in \delta_i^k$ if and only if $(r, \sigma \cap AP_i^{curr}, r') \in \delta_i^{curr}$ and $(q, \sigma \cap AP^k, q') \in \delta^k$. All other elements in the tuple B_i^k remain the same as defined in Sec. A.2.2.

In line 5, the robot calculates the minimum cost behavior b_i^{new} by using Dijkstra's shortest path algorithm to find the minimum cost path through $A_i \times B_i^k$ to an accepting state and an accepting cycle. $b_i^{new} = \emptyset$ when the robot is unable to perform φ^k. This occurs either if the robot does not have the capabilities to satisfy the new task, or if the current remaining task and the new task conflict with each other.

Given b_i^{new}, each robot calculates its satisfiability ζ_i and cost Γ_i, as outlined in Alg. 2. In lines 2-4, the robot determines if it can perform both its current task

and the j^{th} new sub-task. If it can, we set $\boldsymbol{\zeta_i}[j] = 1$ and include the corresponding cost in Γ_i.

After calculating $\boldsymbol{\zeta_i}$, the robot synthesizes the behavior for each combination of sub-tasks it can do (lines 7-11). It does this because the cost for each sub-task is non-additive (as explained in Sec. A.3.3). The combinations of sub-tasks are determined based on $\boldsymbol{\zeta_i}$.

Algorithm 2: Compute Satisfiability and Cost

Input : A_i, z_i, B_i^{curr}, Φ^{new}
Output: $\boldsymbol{\zeta_i}, \Gamma_i$

1 $\boldsymbol{\zeta_i} := \mathbf{0}, \Gamma_i := \emptyset$
`// for individual sub-tasks`
2 **for** $j \in |\Phi^{new}|$ **do**
3 $\quad b_i^{new}, c_i(b_i^{new}) :=$ SYNTHESIZE_BEHAVIOR$(A_i, z_i, B_i^{curr}, \Phi^{new}[j])$
4 $\quad \Gamma_i[j] := c_i(b_i^{new})$
5 $\quad$ **if** $b_i^{new} \neq \emptyset$ **then**
6 $\quad\quad \boldsymbol{\zeta_i}[j] := 1$
7 `// for combinations of sub-tasks` $Y_i = \{j \mid \zeta_i[j] = 1\}$
8 **for** $k \in 2^{Y_i}$ **do**
9 $\quad$ **if** $|k| > 1$ **then**
10 $\quad\quad \varphi^k := \bigwedge_\ell \Phi^{new}[k_\ell]$
11 $\quad\quad b_i^{new}, c_i(b_i^{new}) :=$ SYNTHESIZE_BEHAVIOR$(A_i, z_i, B_i^{curr}, \varphi^k)$
12 $\quad\quad \Gamma_i[k] := c_i(b_i^{new})$

A.5.2 Task Allocation

We introduce a token-based heuristic algorithm to determine a near-optimal allocation for the new task, as shown in Alg. 3. The token is $\boldsymbol{\alpha}$, the global assignment vector of length m where α_j corresponds to the robot that has been assigned to $\Phi^{new}[j]$. $\boldsymbol{\alpha}$ is initialized to be a zero vector.

Each robot A_i assigns itself to the sub-tasks that it can perform and have not yet been assigned to any other robot (lines 3-5). The algorithm includes a

conflict resolution scheme when the sub-tasks that A_i can perform have already been assigned (lines 6-16). In this case, for each robot A_k with conflicts, the algorithm looks at the overlap between the tasks already assigned to A_k and the tasks A_i can do, which is provided by its satisfiability vector $\boldsymbol{\zeta_i}$ (lines 10-15). The function UPDATE_ASSIGNMENT iterates through every combination of overlapping assignments for the A_i and A_k, finds the one with the minimum cost, and updates $\boldsymbol{\alpha}$.

In this algorithm, at each iteration of the conflict resolution, a robot only compares possible conflicting assignments with one other robot. Although we cannot guarantee optimality of the final task allocation assignment, it significantly reduces the computation time. Our algorithm has complexity $O(2^m n)$, compared to the optimal algorithm, which checks every $\binom{n}{m}$ combinations and has complexity $O(m^n)$. In addition, because the token is passed to every robot, we can guarantee that the algorithm will find an assignment for every sub-task if one exists.

A.6 Results and Evaluation

We demonstrate the effectiveness of our synthesis framework by showing the changes to the robot behaviors for the example in Sec. A.4.1. Furthermore, we compare the results of our token-based algorithm to the optimal assignment for different team and task sizes.

Algorithm 3: Task allocation for Φ^{new}

Input : $\boldsymbol{\zeta_1}, \boldsymbol{\zeta_2}, ..., \boldsymbol{\zeta_n}, \Gamma_1, \Gamma_2, ..., \Gamma_n, m := |\Phi^{new}|$
Output: $\boldsymbol{\alpha}$

$\boldsymbol{\alpha} := \mathbf{0}$
for $i \in \{1, ..., n\}$ **do**
  for $j \in \{1, ..., m\}$ **do**
    if $\boldsymbol{\zeta}_i[j] = 1$ and $\boldsymbol{\alpha}[j] = 0$ **then**
      $\boldsymbol{\alpha}[j] = i$
  `// Conflict resolution`
  $compared = \emptyset$
  $satis_i := \{p \mid \zeta_i[p] = 1\}$
  for $j \in \{1, ..., m\}$ **do**
    if $\boldsymbol{\zeta}_i[j] = 1$ and $\boldsymbol{\alpha}[j] \neq i$ **then**
      $k = \boldsymbol{\alpha}[j]$
      if $k \notin compared$ **then**
        $assigned_i := \{p \mid \boldsymbol{\alpha}[p] = i\}$
        $assigned_k := \{p \mid \boldsymbol{\alpha}[p] = k\}$
        $conflicts := satis_i \cap assigned_k$
        $\boldsymbol{\alpha}$:= UPDATE_ASSIGNMENT($assign_i, assign_k, satis_i, conflicts, \Gamma_i, \Gamma_k$)
        $compared = compared \cup \{k\}$

A.6.1 Simulation of Robot Behavior

For the example in A.4.1, the final task allocation assignment is $\boldsymbol{\alpha} = [1, 1, 2]$, meaning that Robot 1 is tasked to complete the first two sub-tasks (Eq. A.1, A.2), and Robot 2 is tasked to complete the third sub-task (Eq. A.3). Fig. A.3 shows the updated behavior of Robot 1 after being assigned, mid-execution, the new sub-tasks. For reference, its current task is to scan first in room 4 and then scan in room 1 (Eq. A.7). Observe that the robot interleaves the current and new tasks rather than performing them sequentially - before completing its current task by scanning in room 1, it performs part of the new task by picking up a box in room 2.

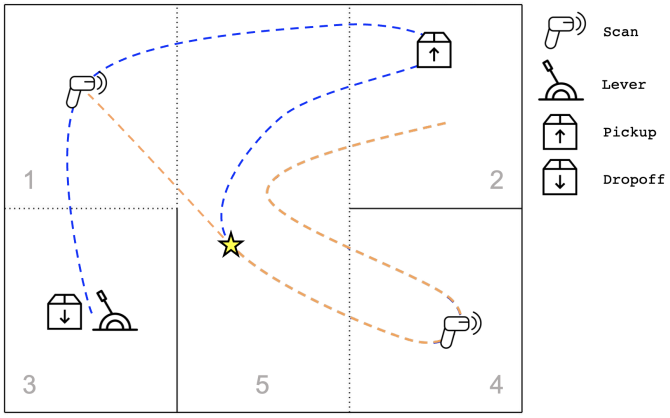


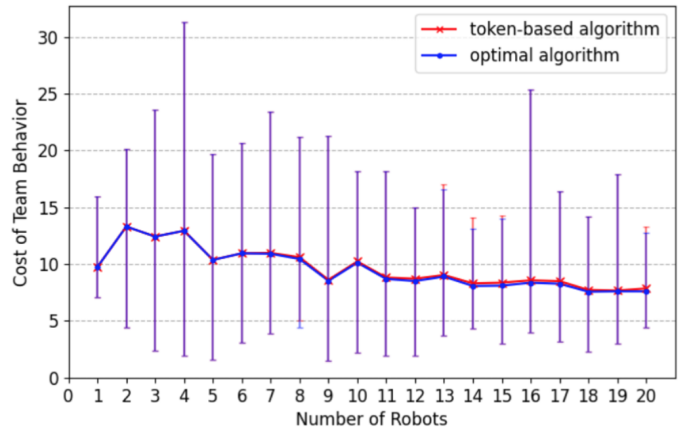
Figure A.3: Updated behavior for Robot 1. The robot starts in room 2, and its original trajectory for the current task is drawn in orange. It receives the new task at the star and updates its behavior based on the sub-tasks it assigned itself. The new behavior is shown in blue. The icons indicate the actions the robot takes.

A.6.2 Task Allocation Performance

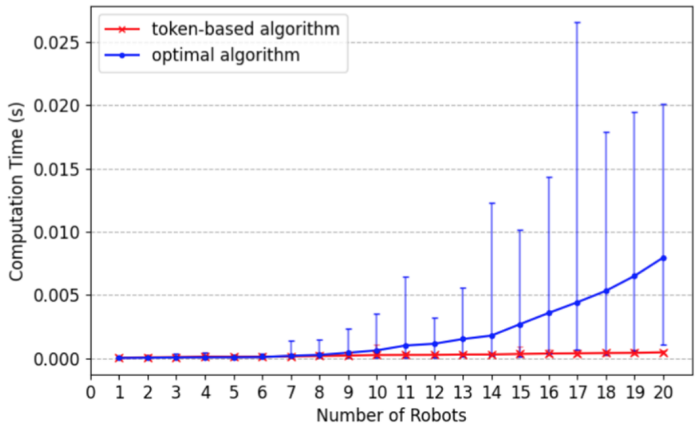
We compare our token-based allocation scheme with the optimal algorithm, which produces the task assignment with the minimum cost by checking $\binom{n}{m}$ different assignments. We show the cost of the behavior of the final assignment and the computation time of the two algorithms.

We varied the number of robots from 1 to 20 with 10 fixed new sub-tasks (Fig. A.4). Similarly, we varied the number of new sub-tasks from 1 to 8 with 5 robots (Fig. A.5). For each of these scenarios, we ran 30 simulations, randomizing the robots' capabilities and current tasks each time. The simulations ran on a 2.5 GHz quad-core Intel Core i7 CPU.

In both cases, the optimal algorithm's computation time grows exponentially. The computation time of our task allocation algorithm grows much slower while also maintaining little to no sub-optimality in the final allocation results.

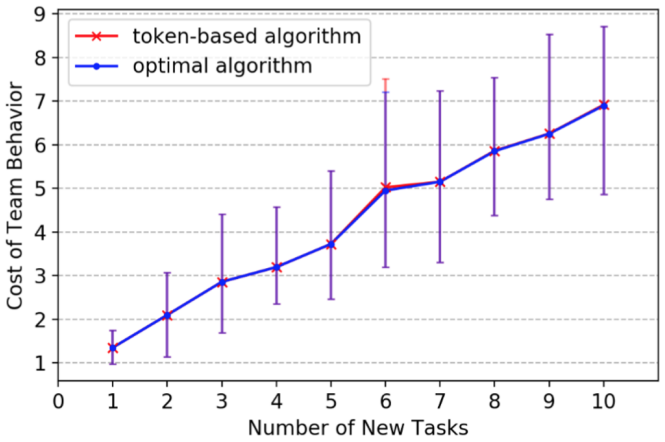


(a)

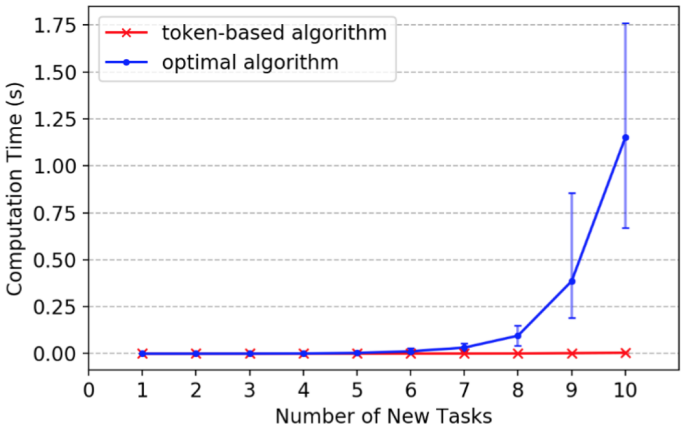


(b)

Figure A.4: Comparison of overall cost (a) and computation time (b) between our algorithm and the optimal algorithm when varying the number of robots. The error bars represent the min/max values of the simulations.



(a)



(b)

Figure A.5: Comparison of computation time (a) and cost (b) between our algorithm and the optimal algorithm when varying the number of new tasks. The error bars represent the min/max values of the simulations.

A.7 Conclusion

We present an approach for robots to automatically distribute a new task while still satisfying their current tasks. Each robot determines if it can satisfy both its current task and the new sub-tasks and resynthesizes its behavior accordingly. We provide a heuristic token-based task distribution algorithm to determine the final task assignment for the new task. The algorithm is scalable and provides a near-optimal assignment that minimizes overall cost.

In future work, we will consider new tasks that are reactive, which will require robots to be able to adapt their behavior at runtime, and a method to model the dynamic and possibly adversarial environment. We also plan to introduce new tasks that require collaboration between robots, which will add complexity to both the synthesis of new behaviors and the allocation of tasks.

APPENDIX B

HIGH-LEVEL, COLLABORATIVE TASK PLANNING GRAMMAR AND EXECUTION FOR HETEROGENEOUS AGENTS

B.1 Introduction

Agents working together to achieve common goals have a variety of applications, such as warehouse automation or disaster response. Multi-agent tasks have been defined in different ways in the scheduling and planning literature. For example, in multi-agent task allocation [38, 52, 60] and coalition formation [67, 115], each task is a single goal with an associated utility. Individual agents or agent teams then automatically assign themselves to a task based on some optimization metric. Swarm approaches [90, 105] consider emergent behavior of an agent collective as the task, for example, aggregation or shape formation.

Recently, formal methods, such as temporal logics for task specifications and correct-by-construction synthesis, have been used to solve different types of multi-agent planning tasks [18, 89, 101]. Tasks written in temporal logic, such as Linear Temporal Logic (LTL), allow users to capture complex tasks with temporal constraints. Existing work has extended LTL [73, 86] and Signal Temporal Logic [64] to encode tasks that require multiple agents.

In this paper, we consider tasks that a team of heterogeneous agents are required to collaboratively satisfy. For instance, consider a precision agriculture scenario in which a farm contains agents with different on-board sensors to monitor crop health. The user may want to take a moisture measurement

in one region, and then take a soil sample of a different region. Depending on the agents' sensors and sensing range, the agents may decide to collaborate to satisfy the task. For example, one agent may perform the entire task on its own if it has both a moisture sensor and an arm mechanism to pick up a soil sample and can move between the two regions. However, another possible solution is for two agents to team up so that one takes a moisture measurement and the other picks up the soil. Existing task grammars [64, 73, 86] capture tasks such as the above by providing explicit constraints on the types or number of agents for each part of the task, i.e. the task must explicitly encode whether it should be one agent, two agents, or either of these options. In this paper, we create a task grammar and associated control synthesis that removes the need to a priori decide on the number of agents necessary to accomplish a task, allowing users to focus solely on the actions required to achieve the task (e.g. "take a moisture measurement and then pick up a soil sample, irrespective of which or how many agents perform which actions").

Our task grammar has several unique aspects. First, this grammar enables the interleaving of agent actions, alleviating the need for explicit task decomposition in order to assign agents to parts of the task. Second, rather than providing explicit constraints on the types or number of agents for each part of the task, the task encodes, using the concept of *bindings* (inspired by [73]), the overall relationship between agent assignments and team behavior; we can require certain parts of the task to be satisfied by the same agent without assigning the exact agent or type of agent *a priori*. Lastly, the grammar allows users to make the distinction between the requirements "for all agents" and "at least one agent". Given these types of tasks, agents autonomously determine, based on their capabilities, which parts of the task they can and should do for the team

to satisfy the task.

Tasks may require collaboration between different agents. Similar to [19, 57, 100], to ensure the actions are performed in the correct order, our framework takes the corresponding synchronization constraints into account while synthesizing agent behavior; agents must wait to execute the actions together. In our approach, execution of the synchronous behavior for each agent is decentralized; agents carry out their plan and communicate with one another when synchronization is necessary.

Depending on the task and the available agents, there might be different teams (i.e., subsets of the agent set) that can carry out the task; our algorithm for assigning a team and synthesizing behavior for the agents finds the largest team of agents that satisfies the task. This means that the team may have redundancies, i.e. agents can be removed while still ensuring the overall task is satisfied. This is beneficial both for robustness and optimality; the user can choose a subset of the team (provided that all the required bindings are still assigned) to optimize different metrics, such as cost or overall number of agents.

Related work: One way to encode tasks is to first decompose them into independent sub-tasks and then allocate them to the agents. For example, [33, 89] address finite-horizon tasks for multi-agent teams. The authors first automatically decompose a global automaton representing the task into independent sub-tasks. To synthesize control policies, the authors build product automata for each heterogeneous agent. Each automaton is then sequentially linked using switch transitions to reduce state-space explosion in synthesizing parallel plans. In our prior work [30], we address infinite-horizon tasks that have already been decomposed into sub-tasks. Given a new task, we proposed a decentralized

framework for agents to automatically update their behavior based on a new task and their existing tasks, allowing agents to interleave the tasks.

The works discussed above make the critical assumption that tasks are independent, i.e. agents do not collaborate with one another. One approach to including collaborative actions is to explicitly encode the agent assignments in the tasks. To synthesize agent control for these types of tasks, in [100], the authors construct a reduced product automaton in which the agents only synchronize when cooperative actions are required. The work in [56] proposes a sampling-based method that approximates the product automaton of the team by building trees incrementally while maintaining probabilistic completeness. In this paper, we consider the more general setting in which agents may need to collaborate with each other, but are not given explicit task assignments *a priori*.

Rather than providing predetermined task assignments, another approach for defining collaborative tasks is to capture information about the number and type of agents needed for parts of the specification. For example, [86] imposes constraints on the number of agents necessary in regions using counting LTL. [64] uses Capability Temporal Logic to encode both the number and capabilities necessary in certain abstracted locations in the environment and then formulates the problem as a MILP to find an optimal teaming strategy. The authors of [73] introduce the concept of induced propositions, where each atomic proposition not only encodes information about the number, type of agents, and target regions, but also has a connector that binds the truth of certain atomic propositions together. To synthesize behavior for the agents, they propose a hierarchical approach that first constructs the automaton representing the task and then decomposes the task into possible sub-tasks. The temporal order of

these sub-tasks is captured using partially ordered sets and are used in the task allocation problem, which is formulated as a MILP.

Inspired by [73] and the concept of induced propositions, we create a task grammar that includes information about how the atomic propositions are related to one another, which represents the overall relationship between agents and task requirements. Unlike [73], which considers navigation tasks in which the same set of agents of a certain type may need to visit different regions, we generalize these tasks to any type of abstract action an agent may be able to perform. In addition, a key assumption we relax is that we do not require each agent to be only categorized as one type. As a result, agents can have overlapping capabilities. To our knowledge, no other grammars have been proposed for these generalized types of multi-agent collaborative tasks.

Contributions: We propose a task description and control synthesis framework for heterogeneous agents to satisfy collaborative tasks. Specifically, we present a new, LTL-based task grammar for the formulation of collaborative tasks, and provide a framework to form a team of agents and synthesize control and synchronization policies to guarantee the team satisfies the task. We demonstrate our approach in simulated precision agriculture scenarios.

B.2 Preliminaries

B.2.1 Linear Temporal Logic

LTL formulas are defined over a set of atomic propositions AP, where $\pi \in AP$ are Boolean variables [29]. We abstract agent actions as atomic propositions. For example, UV captures an agent taking UV measurement.

Syntax: An LTL formula is defined as:

$$\varphi ::= \pi \mid \neg\varphi \mid \varphi \vee \varphi \mid \bigcirc \varphi \mid \varphi \, \mathcal{U} \, \varphi$$

where $\neg$ ("not") and $\vee$ ("or") are Boolean operators, and $\bigcirc$ ("next") and $\mathcal{U}$ ("until") are temporal operators. From these operators, we can define: conjunction $\varphi \wedge \varphi$, implication $\varphi \Rightarrow \varphi$, eventually $\Diamond\varphi$ = True $\mathcal{U}\, \varphi$, and always $\Box\varphi = \neg\Diamond\neg\varphi$.

Semantics: The semantics of an LTL formula φ are defined over an infinite trace $\sigma = \sigma(0)\sigma(1)\sigma(2)...$, where $\sigma(i)$ is the set of true AP at position i. We denote that σ satisfies LTL formula φ as $\sigma \models \varphi$.

Intuitively, $\Diamond\varphi$ is satisfied if there exists a $\sigma(i)$ in which φ is true. $\Box\varphi$ is satisfied if φ is true at every position in σ. To satisfy $\varphi_1 \, \mathcal{U} \, \varphi_2$, φ_1 must remain true until φ_2 becomes true. See [29] for the full semantics.

B.2.2 Büchi Automata

An LTL formula φ can be translated into a Nondeterministic Büchi Automaton that accepts infinite traces if and only if they satisfy φ. A Büchi automaton is a

tuple $\mathcal{B} = (Z, z_0, \Sigma_\mathcal{B}, \delta_\mathcal{B}, F)$, where Z is the set of states, $z_0 \in Z$ is the initial state, $\Sigma_\mathcal{B}$ is the input alphabet, $\delta_\mathcal{B} : Z \times \Sigma_\mathcal{B} \times Z$ is the transition relation, and $F \subseteq Z$ is a set of accepting states. An infinite run of $\mathcal{B}$ over a word $w = w_1 w_2 w_3 ..., w_i \in \Sigma_\mathcal{B}$ is an infinite sequence of states $z = z_0 z_1 z_2 ...$ such that $(z_{i-1}, w_i, z_i) \in \delta_\mathcal{B}$. A run is accepting if and only if $\text{Inf}(z) \cap F \neq \emptyset$, where $\text{Inf}(z)$ is the set of states that appear in z infinitely often [4].

B.2.3 Agent Model

Following [30], we create an abstract model for each agent based on its set of capabilities. A capability is a weighted transition system $\lambda = (S, s_0, AP, \Delta, L, W)$, where S is a finite set of states, $s_0 \in S$ is the initial state, AP is the set of atomic propositions, $\Delta \subseteq S \times S$ is a transition relation where for all $s \in S$, $\exists s' \in S$ such that $(s, s') \in \Delta$, $L : S \rightarrow 2^{AP}$ is the labeling function such that $L(s)$ is the set of propositions that are true in state s, and $W : \Delta \rightarrow \mathbb{R}_{\geq 0}$ is the cost function assigning a weight to each transition. Since we are considering a group of heterogeneous agents, agent j has its own set of k capabilities $\Lambda_j = \{\lambda_1, ..., \lambda_k\}$.

An agent model A_j is the product of its capabilities: $A_j = \lambda_1 \times ... \times \lambda_k$ such that $A_j = (S, s_0, AP_j, \gamma, L, W)$, where $S = S_1 \times ... \times S_k$ is the set of states, $s_0 \in S$ is the initial state, $AP_j = \bigcup_{i=1}^{k} AP_i$ is the set of propositions, $\gamma \subseteq S \times S$ is the transition relation such that $(s, s') \in \gamma$, where $s = (s_1, ..., s_k)$, $s' = (s'_1, ..., s'_k)$, if and only if for all $i = \{1, ..., k\}, (s_i, s'_i) \in \Delta_i$, $L : S \rightarrow 2^{AP_j}$ is the labeling function where $L(s) = \bigcup_{i=1}^{k} L_i(s_i)$, and $W : \gamma \rightarrow \mathbb{R}_{\geq 0}$ is the cost function that combines the costs of the capabilities. Fig. B.1c depicts a snippet of an agent model where we treat the cost as additive. Fig. B.1a represents the agent's sensing area λ_{area}; the agent

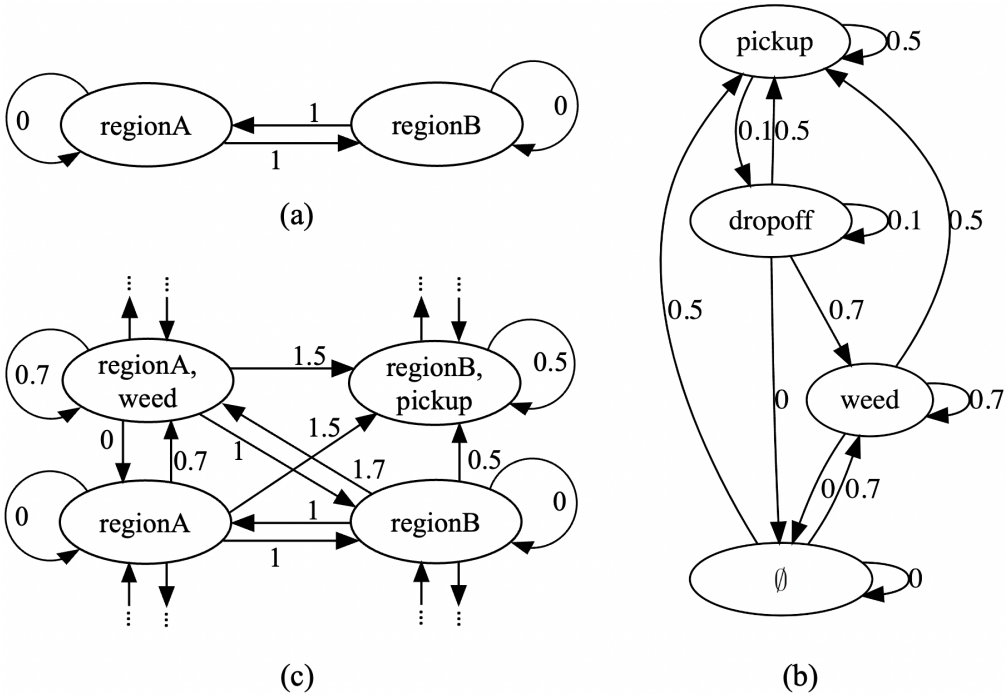


Figure B.1: Agent partial model: (a) λ_{area} (b) λ_{arm} (c) A_{green}

can orient its sensors to take measurements in different regions of a partitioned workspace (in this case, regions A and B). Fig. B.1b represents the agent's robot manipulator, which can pick up and drop off soil samples, as well as pull weeds.

B.3 Task Grammar - LTL$^{\psi}$

We define the task grammar LTL$^{\psi}$ that includes atomic propositions that abstract agent action, logical and temporal operators, as in LTL, and bindings that connect actions to specific agents; any action labeled with the same binding must be satisfied by the same agent(s) (the actual value of the binding is not

important). We define a task recursively over LTL and binding formulas.

$$\psi := \rho \mid \psi_1 \vee \psi_2 \mid \psi_1 \wedge \psi_2 \tag{B.1}$$

$$\varphi := \pi \mid \neg\varphi \mid \varphi \vee \varphi \mid \bigcirc \varphi \mid \varphi \, \mathcal{U} \, \varphi \tag{B.2}$$

$$\varphi^{\psi} := \varphi^{\psi} \mid \neg(\varphi^{\psi}) \mid \varphi_1^{\psi_1} \wedge \varphi_2^{\psi_2} \mid \varphi_1^{\psi_1} \vee \varphi_2^{\psi_2} \mid \bigcirc \varphi^{\psi} \mid \varphi_1^{\psi_1} \mathcal{U} \varphi_2^{\psi_2} \mid \Box \varphi^{\psi} \tag{B.3}$$

where ψ, the *binding formula*, is a Boolean formula excluding negation over $\rho \in AP_{\psi}$, and φ is an LTL formula. An LTL$^{\psi}$ formula consists of conjunction, disjunction, and temporal operators; we define eventually as $\diamond\varphi^{\psi} = \text{True} \, \mathcal{U} \, \varphi^{\psi}$. An example of an LTL$^{\psi}$ formula is shown in Eq. B.4.

Semantics: The semantics of an LTL$^{\psi}$ formula φ^{ψ} are defined over σ and R; $\sigma = \sigma_1\sigma_2...\sigma_n$ is the team trace where σ_j is agent j's trace, and $\forall i, \sigma(i) = \sigma_1(i)\sigma_2(i)...\sigma_n(i)$. $R = \{r_1, r_2, ..., r_n\}$ is the set of binding assignments, where $r_j \in R$ is the set of AP_{ψ} that are assigned to agent j. Once a team is established, R is constant, i.e. an agent's binding assignment does not change throughout the task execution. For example, $r_1 = \{2, 3\}, r_2 = \{1\}$ denotes that agent 1 is assigned bindings 2 and 3, and agent 2 is assigned binding 1.

Given n agents and a set of binding propositions AP_{ψ}, we define the function $\zeta : \psi \to 2^{2^{AP_{\psi}}}$ such that $\zeta(\psi)$ is the set of all possible combinations of ρ that satisfy ψ. For example, $\zeta((1 \vee 2) \wedge 3) = \{\{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$.

The semantics of LTL$^{\psi}$ are:

- $(\sigma(i), R) \models \varphi^{\psi}$ iff $\exists K \in \zeta(\psi)$ s.t. $(K \subseteq \bigcup_{p=1}^{n} r_p)$ and $(\forall j$ s.t. $K \cap r_j \neq \emptyset, \sigma_j(i) \models \varphi)$
- $(\sigma(i), R) \models (\neg\varphi)^{\psi}$ iff $\exists K \in \zeta(\psi)$ s.t. $(K \subseteq \bigcup_{p=1}^{n} r_p)$ and $(\forall j$ s.t. $K \cap r_j \neq \emptyset, \sigma_j(i) \not\models \varphi)$
- $(\sigma(i), R) \models \neg(\varphi^{\psi})$ iff $\exists K \in \zeta(\psi)$ s.t. $(K \subseteq \bigcup_{p=1}^{n} r_p)$ and $(\exists j$ s.t. $K \cap r_j \neq \emptyset, \sigma_j(i) \not\models \varphi)$

- $(\sigma(i),R) \models \varphi_1^{\psi_1} \wedge \varphi_2^{\psi_2}$ iff $(\sigma(i),R) \models \varphi_1^{\psi_1}$ and $(\sigma(i),R) \models \varphi_2^{\psi_2}$
- $(\sigma(i),R) \models \varphi_1^{\psi_1} \vee \varphi_2^{\psi_2}$ iff $(\sigma(i),R) \models \varphi_1^{\psi_1}$ or $(\sigma(i),R) \models \varphi_2^{\psi_2}$
- $(\sigma(i),R) \models \bigcirc \varphi^{\psi}$ iff $\sigma(i+1),R \models \varphi^{\psi}$
- $(\sigma(i),R) \models \varphi_1^{\psi_1} \mathcal{U} \varphi_2^{\psi_2}$ iff $\exists \ell \geq i$ s.t. $(\sigma(\ell),R) \models \varphi_2^{\psi_2}$ and $\forall i \leq k < \ell, (\sigma(k),R) \models \varphi_1^{\psi_1}$
- $(\sigma(i),R) \models \square \varphi^{\psi}$ iff $\forall \ell > i, (\sigma(\ell),R) \models \varphi^{\psi}$

Intuitively, the behavior of an agent team and their respective binding assignments satisfy φ^{ψ} if there exists a possible binding assignment in $\zeta(\psi)$ in which all the bindings are assigned to (at least one) agent, and the behavior of all agents with a relevant binding assignment satisfy φ. An agent can be assigned more than one binding, and a binding can be assigned to more than one agent.

Remark 1. *For the sake of clarity in notation, $\neg\varphi^{\psi}$ is equivalent to $(\neg\varphi)^{\psi}$. For example, $\neg pickup^1 \triangleq (\neg pickup)^1$.*

Remark 2. *Note the subtle but important difference between $(\neg\varphi)^{\psi}$ and $\neg(\varphi^{\psi})$. Informally, the former requires all agents with binding assignments that satisfy ψ to satisfy $\neg\varphi$; the latter requires the formula φ^{ψ} to be violated, meaning that at least one agent's trace violates φ, i.e. satisfies $\neg\varphi$.*

Remark 3. *Unique to LTL$^{\psi}$ is the ability to encode both tasks that include constraints on all agents or on at least one agent; "For all agents" is captured by φ^{ψ}; "at least one agent" is encoded as $\neg((\neg\varphi)^{\psi})$, which captures "at least one agent assigned a binding in $K \in \zeta(\psi)$ satisfies φ". This allows for multiple agents to be assigned the same binding, but only one of those agents is necessary to satisfy φ. This can be particularly useful in tasks with safety constraints; for example, we can write $\neg(\neg region_A^1) \Rightarrow (region_A \wedge visual)^2$, which says "if any agent assigned binding 1 is in region A, all agents assigned binding 2 must take a picture of the region."*

Example. Let $AP_\psi = \{1, 2, 3\}$, $AP_\varphi = \{region_A, region_B, pickup,$ $thermal, visual, moisture, UV\}$, and $\varphi^\psi = \varphi_1^\psi \wedge \varphi_2^\psi$, where

$$\varphi_1^\psi = \Diamond((region_B \wedge moisture \wedge UV)^{2\wedge 3} \wedge (region_A \wedge pickup)^1) \tag{B.4a}$$

$$\varphi_2^\psi = \neg pickup^1 \; \mathcal{U} \; (region_A \wedge ((thermal \vee visual) \wedge \neg(thermal \wedge visual)))^2 \tag{B.4b}$$

φ_1^ψ captures "Agent(s) assigned bindings 2 and 3 must take a moisture and UV measurement in region B at the same time that agent(s) assigned binding 1 picks up a soil sample in region A." φ_2^ψ says "Before the soil sample is picked up, agent(s) assigned binding 2 must take a thermal or a visual image (but not both) of region A."

Since multiple bindings can be assigned to the same agent, an agent can be assigned both bindings 2 and 3, provided it has the capabilities to satisfy the corresponding parts of the formula. In addition, depending on the final assignments, the agents may need to synchronize with one another to perform parts of the task. For example, to satisfy φ_1^ψ, agents assigned with any subset of bindings $\{1, 2, 3\}$ need to synchronize their respective actions.

B.4 Control Synthesis for LTL$^\psi$

Problem statement: Given n heterogeneous agents $A = \{A_1, ..., A_n\}$ and a task φ^ψ in LTL$^\psi$, find a team of agents $\hat{A} \subseteq A$, their binding assignments $R_{\hat{A}}$, and synthesize behavior σ_j for each agent such that $(\sigma(0), R_{\hat{A}}) \models \varphi^\psi$. This behavior includes synchronization constraints for agents to satisfy the necessary collaborative actions. We assume that each agent is able to wait in any state (i.e. every state in

the agent model has a self-transition).

Example. Consider a group of four agents $A = \{A_{green}, A_{blue}, A_{orange}, A_{pink}\}$ in a precision agriculture environment composed of 5 regions, as illustrated in Fig. B.2. A_{orange} is a mobile robot manipulator, such as Harvest Automation's HV-100, while the other agents are stationary with different onboard sensing capabilities. The set of all capabilities is $\Lambda = \{\lambda_{area\_j}, \lambda_{motion}, \lambda_{arm}, \lambda_{UV}, \lambda_{moisture}, \lambda_{visual}, \lambda_{thermal}\}$, where $\forall j = \{green, blue, pink\}$, λ_{area_j} is agent j's sensing area model. The green agent can orient its arm to reach either region A or B. The blue agent can orient its sensors to see one of three regions, B, C, or D; in order to reorient its sensors from regions B to D, its sensing range must first pass through region C. Similarly, the pink agent can orient its sensors to see either region A, B, or C, and its sensing range must pass through region B to get from regions A to C. The orange agent's ability to move between adjacent regions is represented by the capability λ_{motion}. Its sensing region is whichever region it is in. $AP_{arm} = \{pickup, dropoff, weed\}$ is an abstraction of a robot manipulator that represents different actions the arm can perform, such as picking up soil samples or pulling weeds. AP_{UV}, $AP_{moisture}$, AP_{visual}, $AP_{thermal}$ all contain a single proposition representing a agent's ability to take UV measurements, soil moisture measurements, visual images, and thermal images, respectively. λ_{arm} has more states (see Fig. B.1b). Each agent may have distinct cost functions corresponding to individual capabilities.

The agent capabilities and label on the initial state are:

$\Lambda_{green} = \{\lambda_{area\_1}, \lambda_{arm}\}, L(s_0) = \{region_B\}$

$\Lambda_{blue} = \{\lambda_{area\_2}, \lambda_{moisture}, \lambda_{UV}\}, L(s_0) = \{region_D\}$

$\Lambda_{orange} = \{\lambda_{motion}, \lambda_{moisture}, \lambda_{UV}, \lambda_{arm}\}, L(s_0) = \{region_E\}$

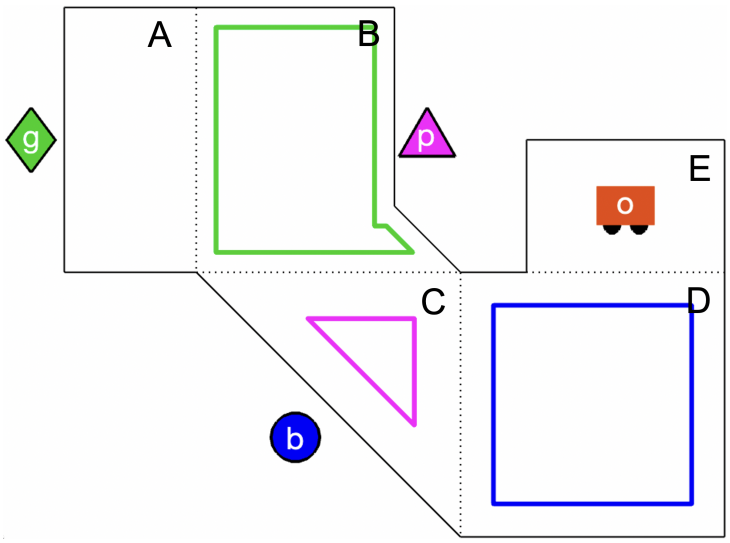


Figure B.2: Agriculture environment and initial agent states. The green, blue, and pink agents are stationary; the orientation of their sensors are indicated by the colored boxes.

$$\Lambda_{pink} = \{\lambda_{area\_4}, \lambda_{thermal}, \lambda_{visual}, \lambda_{moisture}, \lambda_{UV}\},\ L(s_0) = \{region_C\}$$

The team receives the task (Eq. B.4) and must determine a teaming assignment and behavior to satisfy the task. During execution, the agents must also synchronize with each other when necessary.

B.5 Approach

To find a teaming assignment and synthesize the corresponding synchronization and control, we first automatically generate a Büchi automaton $\mathcal{B}$ for the task φ^{ψ} (Sec. B.5.1). Each agent A_j then constructs a product automaton $\mathcal{G}_j = A_j \times \mathcal{B}$ (Sec. B.5.2). For each binding $\rho \in AP_{\psi}$, it checks whether or not it can perform the task associated with that binding by finding a path to an accepting cycle in $\mathcal{G}_j$. Each agent creates a copy of the Büchi automaton $\mathcal{B}_j$ pruned to remove any unreachable transitions and stores information about which com-

binations of binding assignments it can do.

For parts of the task that require collaboration (e.g., when a transition calls for actions with bindings $\{1, 2\}$ and $r_{green} = \{1, 2\}, r_{blue} = \{2\}$), we need agents to synchronize. Thus, we synthesize behavior that allows for parallel execution while also guaranteeing that the team's overall behavior satisfies the global specification.

To find a team of agents that can satisfy the task and their assignments, we need to guarantee that 1) every binding is assigned to at least one agent and 2) the agents synchronize for the collaborative portions of the task. To do so, we first run a depth-first search (DFS) to find a path through the $\mathcal{B}$ to an accepting cycle in which there exists a team of agents such that for every transition in the path, every proposition in AP_ψ is assigned to at least one agent (Sec. B.5.4). Each agent then synthesizes behavior to satisfy this path and communicates to other agents when synchronization is necessary.

B.5.1 Büchi Automaton for an LTL$^\psi$ Formula

When constructing a Büchi automaton for an LTL$^\psi$ specification, we automatically rewrite the specification such that the binding propositions are only over individual atomic proposition $\pi \in AP_\varphi$ (i.e. the formula is composed of π^ρ). For instance, the formula $(\neg pickup\ \mathcal{U}\ region_A)^{1 \vee 2}$ is rewritten as $(\neg pickup^1\ \mathcal{U}\ region_A^1) \vee (\neg pickup^2\ \mathcal{U}\ region_A^2)$.

In our running example, we rewrite the formula in Eq. B.4a as

$$\begin{aligned} &\Diamond(region_B^2 \wedge moisture^2 \wedge UV^2 \\ &\wedge region_B^3 \wedge moisture^3 \wedge UV^3 \wedge region_A^1 \wedge pickup^1) \end{aligned} \tag{B.5}$$

Remark 4. *In rewriting the specification, negation follows bindings in the order of operations. For example,* $\neg pickup^{1\wedge 2} = \neg pickup^1 \wedge \neg pickup^2$*, and* $\neg(pickup^{1\wedge 2}) = \neg(pickup^1 \wedge pickup^2) = \neg(pickup^1) \vee \neg(pickup^2)$.

From AP_φ and AP_ψ, we define the set of propositions AP_φ^ψ, where $\forall \pi \in AP_\varphi$ and $\forall \rho \in AP_\psi$, $\pi^\rho \in AP_\varphi^\psi$. Given AP_φ^ψ, we automatically translate the specification into a Büchi automaton using Spot [28].

To facilitate control synthesis, we transform any transitions in the Büchi automaton labeled with disjunctive formulas into disjunctive normal form (DNF). We then replace the transition labeled with a DNF formula containing ℓ conjunctive clauses with ℓ transitions between the same states, each labeled with a different conjunction of the original label.

In general, when creating a Büchi automaton from an LTL formula φ, $w \in \Sigma_\mathcal{B}$ are Boolean formulas over AP_φ, the atomic propositions that appear in φ, as seen in Fig. B.3. In the following, for creating the product automaton, we use an equivalent representation, where $\Sigma_\mathcal{B} = 2^{AP_\varphi^\psi} \times 2^{AP_\varphi^\psi}$ and $w = (\sigma_T, \sigma_F) \in \Sigma_\mathcal{B}$ contains the set of propositions that must be true, σ_T, and the set of propositions that must be false, σ_F, for the Boolean formula over a transition to evaluate to True. These sets are unique in our case since each transition is labeled with a conjunctive clause (i.e. no disjunction). Note that $\sigma_T \cap \sigma_F = \emptyset$ and $\sigma_T \cup \sigma_F \subseteq AP_\varphi$; propositions that do not appear in w can have any truth value.

Given a Büchi automaton for an LTL$^\psi$ specification $\mathcal{B}$, we define the follow-

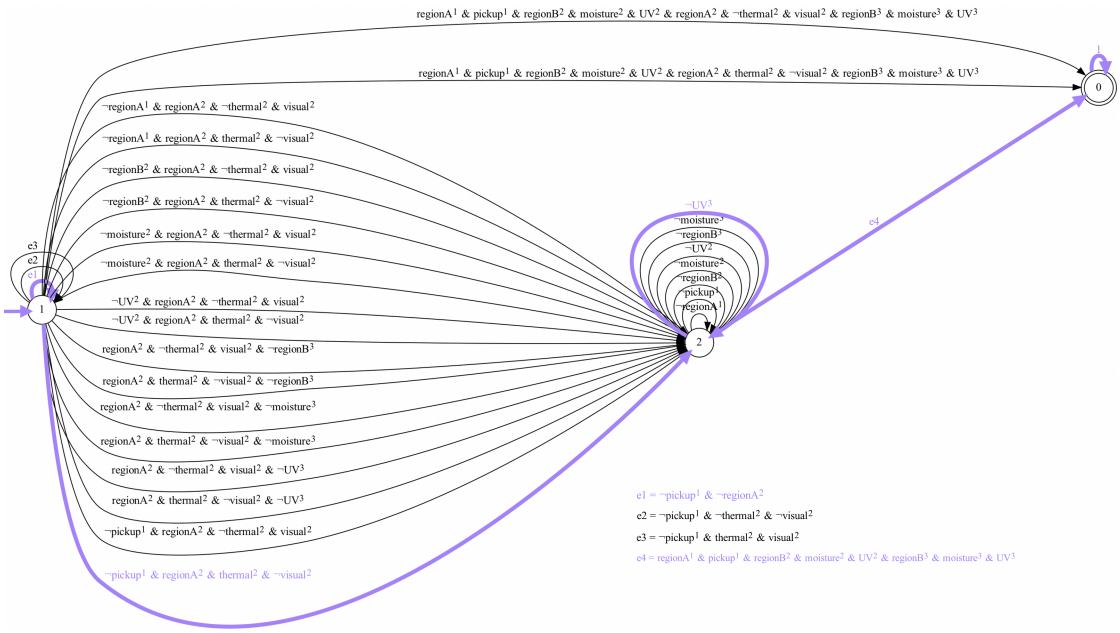


Figure B.3: $\mathcal{B}$ for φ^{ψ} (Eq. B.4). The purple transitions illustrate a possible accepting trace.

ing functions:

Definition 1 (Binding Function). $\mathfrak{B} : \Sigma_{\mathcal{B}} \rightarrow 2^{AP_{\psi}}$ *such that for* $\sigma = (\sigma_T, \sigma_F) \in \Sigma_{\mathcal{B}}$, $\mathfrak{B}(\sigma) \subseteq AP_{\psi}$ *is the set* $\{\rho \in AP_{\psi} \mid \exists \pi^{\rho} \in \sigma_T \cup \sigma_F\}$.

Intuitively, it is the set of bindings that appear in label σ of a Büchi transition.

Definition 2 (Capability Function). $\mathfrak{C} : \Sigma_{\mathcal{B}} \times AP_{\psi} \rightarrow 2^{AP_{\varphi}} \times 2^{AP_{\varphi}}$ *such that for* $\sigma = (\sigma_T, \sigma_F) \in \Sigma_{\mathcal{B}}, \rho \in AP_{\psi}$, $\mathfrak{C}(\sigma, \rho) = (C_T, C_F)$, *where* $C_T = \{\pi \in AP_{\varphi} \mid \exists \pi^{\rho} \in \sigma_T\}$ *and* $C_F = \{\pi \in AP_{\varphi} \mid \exists \pi^{\rho} \in \sigma_F\}$.

Here, C_T and C_F are the sets of action propositions that are True/False and appear with binding ρ in label σ of a Büchi transition.

B.5.2 Agent Behavior for an LTL$^{\psi}$ Specification

To synthesize behavior for an agent, we find an accepting trace in its product automaton $\mathcal{G}_j = A_j \times \mathcal{B}$, where $A_j = (S, s_0, AP_j, \gamma, L, W)$ is the agent model, and $\mathcal{B} = (Z, z_0, \Sigma_{\mathcal{B}}, \delta_{\mathcal{B}}, F)$ is the Büchi automaton.

Since the set of propositions of A_j may not be equivalent to the set of propositions of $\mathcal{B}$, we borrow from the definition of the product automaton in [30]. We first define the following function:

Definition 3 (Binding Assignment Function)**.** *Let* $q = (s, z)$, $q' = (s', z')$, $\sigma = (\sigma_T, \sigma_F) \in \Sigma_{\mathcal{B}}$. *Then* $\Re(q, \sigma, q') = \{r \in 2^{AP_\psi} \setminus \emptyset \mid \forall \rho \in r, (C_T, C_F) = \mathfrak{C}(\sigma, \rho), \bigcup_{\rho \in r} C_T \subseteq L(s')$ *and* $\bigcup_{\rho \in r} C_F \cap L(s') = \emptyset\}$.

Intuitively, $\Re$ outputs all possible combinations of binding propositions that the agent can be assigned for a transition (q, σ, q'). An agent can be assigned ρ if and only if the agent's next state s' is labeled with all the action and motion propositions $\pi \in AP_\varphi$ that appear in σ_T as π^ρ, and all the propositions $\pi \in AP_\varphi$ that appear in σ_F as π^ρ are not part of the state label (i.e. the agent is not performing that action). If a proposition π^ρ is in σ_F and π is not in AP_j (e.g. $scan^1 \in \sigma_F$ and the agent does not have λ_{scan}), the agent may be assigned ρ. Note that r may include any binding propositions that are not in σ, since there are no actions required by those bindings in that transition. For example, if $\sigma = (\{scan^1\}, \{pickup^2\})$ and $AP_\psi = \{1, 2, 3\}$, then $\{3\}$ will be in the set $\Re(q, \sigma, q')$ for all q, q'.

Given A_j and $\mathcal{B}$, we define the product automaton $\mathcal{G}_j = A_j \times \mathcal{B}$:

Definition 4 (Product Automaton)**.** *The product automaton* $\mathcal{G}_j = (Q, q_0, AP_j, \delta_{\mathcal{G}}, L_{\mathcal{G}}, W_{\mathcal{G}}, F_{\mathcal{G}})$, *where*

- $Q = S \times Z$ *is a finite set of states*
- $q_0 = (s_0, z_0) \in Q$ *is the initial state*
- $\delta_{\mathcal{G}} \subseteq Q \times Q$ *is the transition relation, where for* $q = (s, z)$ *and* $q' = (s', z')$, $(q, q') \in \delta_{\mathcal{G}}$ *if and only if* $(s, s') \in \gamma$ *and* $\exists \sigma \in \Sigma_{\mathcal{B}}$ *such that* $(z, \sigma, z') \in \delta_{\mathcal{B}}$ *and* $\Re(q, \sigma, q') \neq \emptyset$
- $L_{\mathcal{G}}$ *is the labeling function s.t. for* $q = (s, z)$, $L_{\mathcal{G}}(q) = L(s) \subseteq AP_j$
- $W_{\mathcal{G}} : \delta_{\mathcal{G}} \to \mathbb{R}_{\geq 0}$ *is the cost function s.t. for* $(q, q') \in \delta_{\mathcal{G}}$, $q = (s, z)$, $q' = (s', z')$, $W_{\mathcal{G}}((q, q')) = W((s, s'))$
- $F_{\mathcal{G}} = S \times F$ *is the set of accepting states*

Example. Fig. B.4 depicts a small portion of $\mathcal{G}_{green}$; for the self-transition in $\mathcal{B}$ that is labeled with $\sigma = (\emptyset, \{pickup^1, region_A^2\})$ (labeled as $e1$ in Fig. B.3), and for states in A_{green} where $L(s_1) = \{region_B\}$, $L(s_2) = \{region_A\}$, $L(s_3) = \{region_A, pickup\}$, then the possible binding assignments are $\Re((s_1, 1), \sigma, (s_1, 2)) = 2^{\{1,2,3\}} \setminus \emptyset$ and $\Re((s_1, 1), \sigma, (s_2, 2)) = \{\{1\}, \{3\}, \{1, 3\}\}$. When the agent is in s_3, it cannot be assigned either bindings 1 or 2, but since no propositions appear with binding 3 in σ, $\Re((s_1, 1), \sigma, (s_3, 2)) = \{\{3\}\}$.

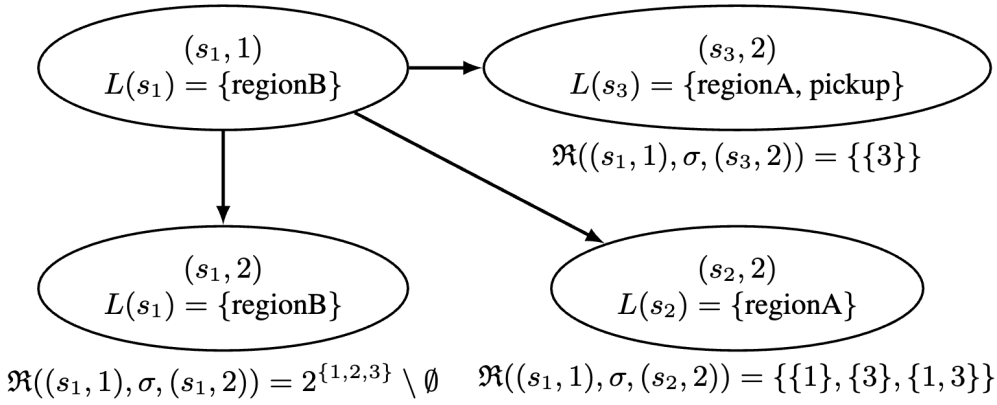


Figure B.4: A small portion of $\mathcal{G}_{green}$

B.5.3 Finding Possible Individual Agent Bindings

To construct a team, we first reason about each agent and the sets of bindings it can perform. For example, for a formula $region_A^1 \wedge region_B^2$, an agent may be assigned $r_j = \{1\}$ or $r_j = \{2\}$ but not $r_j = \{1, 2\}$, since it cannot be in two regions at the same time.

To find the set of possible binding assignments $R_j \subseteq 2^{AP_\psi}$, we search for an accepting trace in $\mathcal{G}_j$ for every binding assignment $r_j \in 2^{AP_\psi}$. We start from the full set of bindings $r_j = AP_\psi$. Given an assignment r_j to check, we find an accepting trace in $\mathcal{G}_j$ such that for all transitions (q, q') in the trace, $r_j \subseteq \Re(q, \sigma, q')$. This ensures that the agent can satisfy its binding assignment for the entirety of its execution (i.e. r_j does not change). Since every subset of a binding assignment r_j is itself a possible binding assignment, if the agent can be assigned all $m = |AP_\psi|$ bindings, then we know it can also be assigned every possible subset of m. If not, we check the $\binom{m}{m-1}$ combinations, and continue iterating until we have determined the agent's ability to perform every combination of the m bindings.

Once an agent determines its possible binding assignments R_j, it creates the Büchi automaton $\mathcal{B}_j$ by removing any transition in $\mathcal{B}$ that cannot be traversed by any assignment in R_j. In our example (Fig. B.3), each agent can be assigned at least one binding over every transition in $\mathcal{B}$. Thus, $\forall j \in \{green, blue, orange, pink\}, \mathcal{B}_j = \mathcal{B}$.

B.5.4 Agent Team Assignment

A team of agents can perform the task if 1) all the bindings are assigned, with each agent maintaining the same binding assignment for the entirety of the task, and 2) the agents satisfy synchronization requirements. For a viable team, the agents' control follows the same path in the Büchi automaton $\mathcal{B}$ to an accepting cycle. We perform DFS over $\mathcal{B}$ to find an accepting trace (Alg. 4), where each tuple in *stack* contains the current edge (z, σ, z'), the current team of agents $R_{\hat{A}}$, and the path traversed so far $\beta_{\hat{A}}$.

We initialize the team with all agents A_j and all possible binding assignments R_j, and each path $\beta_{\hat{A}}$ starts from state z_0 of $\mathcal{B}$. When checking a transition (z, σ, z'), we remove any agent j if $\forall((s, z), (s', z')) \in \delta_{\mathcal{G}_j}$, there are no possible binding assignments it can satisfy. This is done by checking each agent's pruned Büchi automaton $\mathcal{B}_j$ in UPDATE_TEAM (line 8). We want the agent's behavior to satisfy not only the current transition, but also the entire path with a consistent binding assignment. Thus, we update possible bindings (UPDATE_BINDINGS, lines 9-14).

To guarantee the overall team behavior, we need to ensure agents are able to "wait in a state" before they synchronize, as they may reach states at different times. This means that each state in the trace must have a corresponding self-transition. Thus, for every (z, σ, z') that we add to the path in which $z \neq z'$, the next edge to traverse must be a self-transition from z' to itself; the same holds vice-versa. In line 21, we check if the current transition is self-looping or not, and add subsequent transitions into the stack accordingly. If there is no self-transition on z' (i.e. $(z', \sigma, z') \notin \delta_{\mathcal{B}}$), then we do not consider z' to be valid and do not add it to the path.

Once we find a valid path to an accepting cycle, we parse it into β, the path without self-transitions, and δ_{self}, which contains the corresponding self-transition for each state in the path. Fig. B.3 shows a valid path in $\mathcal{B}$ for the example in Sec. B.4 and the corresponding team assignment $\hat{A} = \{A_{green}, A_{blue}, A_{orange}, A_{pink}\}$ and bindings $r_{green} = \{1\}$, $r_{blue} = \{3\}, r_{orange} = \{1\}, r_{pink} = \{2, 3\}$. Note that we find a valid path rather than a globally optimal one. However, the algorithm is complete; it will find a feasible path if one exists.

Algorithm 4: Find Accepting Trace for Agent Team

Input : $A = \{A_1, A_2, ..., A_n\}$, $R = \{R_1, R_2, ..., R_n\}$, $\mathcal{B}$, $\{\mathcal{B}_1, \mathcal{B}_2..., \mathcal{B}_n\}$
Output: β, δ_{self}, $\hat{A} \subseteq A$, $R_{\hat{A}}$

$stack = \emptyset$, $visited = \emptyset$
for $e \in \{(z, \sigma, z') \in \delta_{\mathcal{B}} \mid z = z_0\}$ **do**
  $stack = stack \cup \{(e, R, [e])\}$
while $stack \neq \emptyset$ **do**
  $((z, \sigma, z'), R_{\hat{A}}, \beta_{\hat{A}}) = stack.pop()$
  if $(z, \sigma, z') \notin visited$ **then**
    $visited = visited \cup (z, \sigma, z')$
    $R_{\hat{A}} = \text{UPDATE\_TEAM}((z, \sigma, z'), \{\mathcal{B}_1, ..., \mathcal{B}_n\})$
    for $R_j \in R_{\hat{A}}$ **do**
      $R'_j = \text{UPDATE\_BINDINGS}(R_j, (z, \sigma, z'))$
      if $R'_j = \emptyset$ **then**
        $R_{\hat{A}} = R_{\hat{A}} \setminus R_j$
      else
        $R_{\hat{A}} = (R_{\hat{A}} \setminus R_j) \cup R'_j$
    if $\bigcup_j (R_j \in R_{\hat{A}}) = AP_\psi$ **then**
      if $z' \in F$ **then**
        $\beta, \delta_{self} = \text{PARSE\_PATH}(\beta_{\hat{A}})$
        return $\beta, \delta_{self}, R_{\hat{A}}$
      $E = \{(z', \sigma', z'') \in \delta_{\mathcal{B}}\}$
      for $(z', \sigma', z'') \in E$ **do**
        if ($z = z'$ and $z' \neq z''$) or ($z \neq z'$ and $z' = z''$) **then**
          $stack = stack \cup \{((z', \sigma', z''), R_{\hat{A}}, [\beta_{\hat{A}}\ (z', \sigma', z'')])\}$

B.5.5 Synthesis and Execution of Control and Synchronization Policies

Given an accepting trace β through $\mathcal{B}$ and the corresponding self-transitions δ_{self} that are valid for all agents in $R_{\hat{A}}$, we synthesize control and synchronization for each agent such that the overall team execution satisfies β (Alg. 5). For each transition (z, σ, z') in β, we find $\overline{R}$, which contains the binding assignments of all agents that require synchronization at state z'. Agent j participates in the synchronization step if r_j contains a binding ρ that is required by σ and is not the only agent assigned bindings from σ (line 3).

Subsequently, agent j finds an accepting trace in $\mathcal{G}_j$ that reaches z' with minimum cost, following self-transitions stored in δ_{self} if necessary. As it executes this behavior, it communicates with other agents the tuple p, which contains 1) its ID, 2) the state z' it is currently going to, and 3) if it is ready for synchronization (line 8). If no synchronization is required (line 3), the agent can simply execute the behavior. Otherwise, to guarantee that the behavior does not violate the requirements of the task, the agent executes the synthesized behavior up until the penultimate state, z_{wait}.

When the agent reaches z_{wait}, it signals to other agents that it is ready for synchronization. Since all agents know the overall teaming assignment, the agent continues to wait in state z_{wait} until it receives a signal that all other agents in $\overline{R}$ are ready (line 13). These agents then move to the next state in the behavior simultaneously. Agent j continues synthesizing behavior through β until synchronization is necessary again, and this process is repeated.

Algorithm 5: Synthesize an Agent's Behavior

Input : $\mathcal{G}_j$, r_j, $R_{\hat{A}}$, β, δ_{self}

for $(z, \sigma, z') \in \beta$ **do**
  b_j = FIND_BEHAVIOR($\mathcal{G}_j, r_j, (z, \sigma, z'), \delta_{self}$)
  $\overline{R} = \{r_k \in R_{\hat{A}} \mid r_k \cap \mathfrak{B}(\sigma) \neq \emptyset\}$ **if** $r_j \notin \overline{R}$ or $\overline{R} = \{r_j\}$ **then**
    $p = ()$
    EXECUTE(b_j, p)
  else
    $p = (j, z', 0)$, $\ell = length(b_j)$
    EXECUTE($b_j[1 : \ell - 1], p$)
    $z_{wait} = b_j[\ell - 1]$, $P = \{j\}$
    while $\bigcup_{i \in P}(r_i \in \overline{R}) \neq \mathfrak{B}(\sigma)$ **do**
      $p = (j, z', 1)$
      EXECUTE(z_{wait}, p)
      $P = j \cup \{k \mid (k, z', 1) \in$ RECEIVE()$\}$
    EXECUTE($b_j[\ell]$)

B.6 Results and Discussion

Fig. B.5 shows the final step of the synchronized behavior of the agents for the example in Section B.4, where $\hat{A} = \{A_{green}, A_{blue}, A_{orange}, A_{pink}\}$ with binding assignments $r_{green} = \{1\}$, $r_{blue} = \{3\}, r_{orange} = \{1\}, r_{pink} = \{2, 3\}$. A simulation of the full behavior is shown in the accompanying video[1].

Optimizing teams: Our synthesis algorithm can be seen as a greatest fixpoint computation, where we start with the full set of agents and remove those that cannot contribute to the task. As a result, the team may have redundancies, i.e. agents can be removed while still ensuring the overall task will be completed; this may be beneficial for robustness. Furthermore, we can choose a sub-team to optimize different metrics, as long as the agent bindings assignments still cover all the required bindings. For example, minimizing the number of bindings per agent could result in $\hat{A} = \{A_{green}, A_{blue}, A_{pink}\}$, $r_{green} = \{1\}, r_{blue} = \{3\}, r_{pink} = \{2\}$;

[1] https://youtu.be/Qx14xGza3Bg

j	r_j	$cost$	j	r_j	$cost$	j	r_j	$cost$	j	r_j	$cost$	j	r_j	$cost$
1	1	1.2	5	3	2.75	9	3	2.6	13	3	2.0	17	2,3	3.275
2	3	1.0	6	1	0.95	10	1	2.8	14	1	1.2	18	3	2.55
3	1	1.2	7	1	0.65	11	2,3	0.9	15	3	1.1	19	1	1.9
4	2,3	1.3	8	1	1.0	12	2,3	1.825	16	1	0.775	20	2,3	2.35

Table B.1: Example teaming assignment with 20 robots

minimizing the number of agents results in $\hat{A} = \{A_{green}, A_{pink}\}$, $r_{green} = \{1\}, r_{pink} = \{2, 3\}$.

To illustrate other possible metrics, we consider a set of 20 agents and create a team for the specification in Eq. B.4. Their final binding assignments and costs are shown in Table B.1. Minimizing cost results in a team $\hat{A} = \{A_7, A_{11}\}$. Minimizing cost while requiring each binding to be assigned to two agents results in $\hat{A} = \{A_4, A_7, A_{11}, A_{16}\}$.

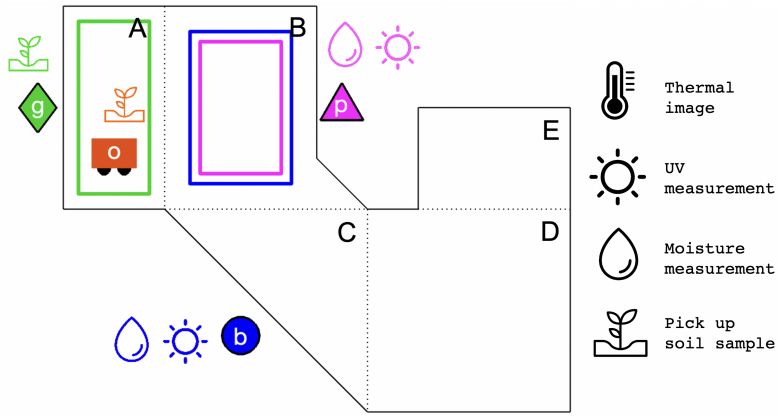


Figure B.5: The final step in the synchronized behavior of the agent team with their corresponding actions.

Computational complexity: The control synthesis algorithm (Alg. 5) is agnostic to the number of agents, since each agent determines its own possible binding assignments and behavior. For the team assignment (Alg. 4), we store the current team and possible binding assignments as we build an accepting trace. Thus, it has both a space and time complexity of $O(|E| * 2^m * n)$, where $|E|$ is

the number of edges in $\mathcal{B}$, m is the number of bindings, and n is the number of agents.

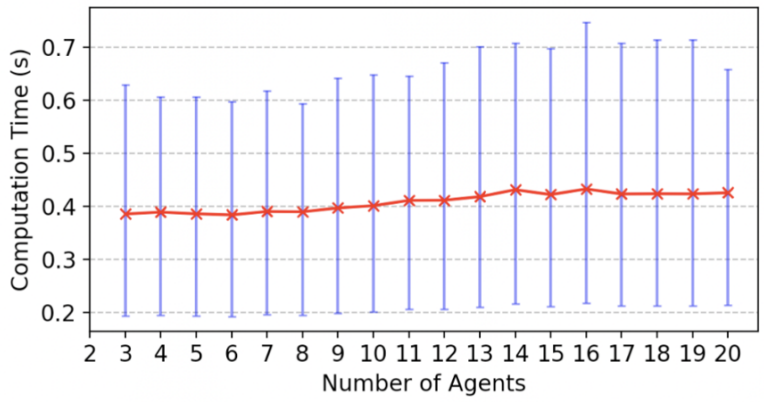
Fig. B.6a shows the computation time of the synthesis framework (Sec. B.5.2 – B.5.4) for simulated agent teams in which we vary the number of agents from 3 to 20, running 30 simulations for each set of agents and randomizing their capabilities. The task for each simulation is the example in Eq. B.4. We also ran simulations in which we increase the number of bindings from 3 to 10 and randomized the capabilities of 4 agents (Fig. B.6b). The variance in computation time is a result of the randomized agent capabilities, which affects the computation time of possible binding assignments (Sec. B.5.3). All simulations ran on a 2.5 GHz quad-core Intel Core i7 CPU.

Task expressivity: We compare LTL$^{\psi}$ with other temporal logic approaches to encode collaborative heterogeneous multi-agent tasks.

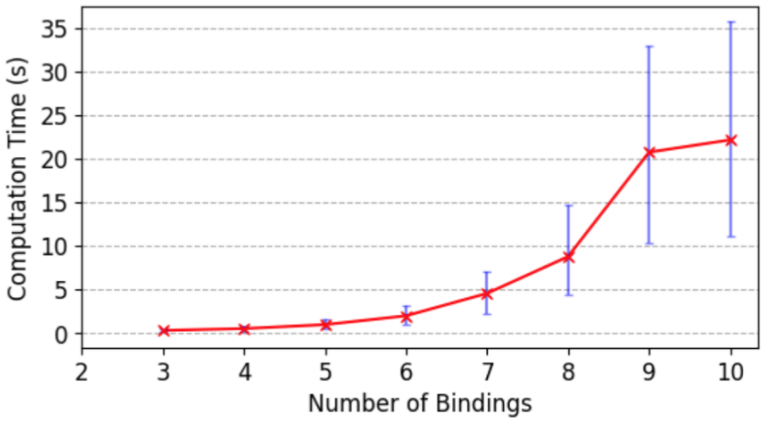
Standard LTL: One approach is to use LTL to express the task by enumerating all possible assignments in the specification. In our example, Eq. B.4a would be rewritten as:

$$\varphi_1^{\psi} = (\Diamond((region_B^g \wedge mois^g \wedge UV^g) \wedge (region_A^b \wedge pickup^b)))$$
$$\vee(\Diamond((region_B^g \wedge mois^g \wedge UV^g) \wedge (region_A^o \wedge pickup^o))) \vee ...$$

where each agent has its own unique set of AP, denoted here by each proposition's superscript. As a result, the number of propositions increases exponentially with the number of agents. The task complexity also increases, as the specification must include all possible agent assignments. Another drawback of using LTL for such tasks is that the specification is not generalizable to any number of agents; it must be rewritten when the set of agents change.



(a)



(b)

Figure B.6: Computation time as the number of agents (a) and bindings (b) increases. The error bars show min/max values.

LTL$^\chi$: In [73], tasks are written in LTL$^\chi$, where proposition $\pi_{i,j}^{k,\chi}$ is true if at least i agents of type j are in region k with binding χ. We can express φ_1^ψ (Eq. B.4a) of our example as $\Diamond(\pi_{1,mois}^{regionB,2} \wedge \pi_{1,UV}^{regionB,2} \wedge \pi_{1,mois}^{regionB,3} \wedge \pi_{1,UV}^{regionB,3} \wedge \pi_{1,arm}^{regionA,1})$. The truth value of $\pi_{i,j}^{k,\chi}$ is not dependent on any particular action an agent might take. LTL$^\chi$ can be extended to action propositions, but since an agent can only be categorized as one type, each type of agent must have non-overlapping capabilities (here, we have written the LTL$^\chi$ formula such that each type of agent only has one capability). In addition, φ_2^ψ (Eq. B.4b) cannot be written in LTL$^\chi$ because the negation defined in our grammar cannot be expressed in LTL$^\chi$. On the other

hand, the negative proposition $\neg\pi_{i,j}^{k,\chi}$ from [73] is equivalent to "less than i agents of type j are in region k", which our logic cannot encode.

Capability Temporal Logic (CaTL): Tasks in CaTL [64] are constructed over tasks $T = (d, \pi, cp_T)$, where d is a time duration, π is a region, $(c_i, m_i) \in cp_T$ denotes that at least m_i agents with capability c_i are required. Similar to our grammar, CaTL allows agents to have multiple capabilities, but each task must specify the number of agents required. Since it is an extension of STL, tasks provide timing requirements, which our logic cannot encode. However, it does not include the concept of binding assignments; in our example φ_1^{ψ} (Eq. B.4a), CaTL cannot express that the same agent that took a UV measurement must also take a thermal image. Ignoring binding assignments and adding timing constraints, φ_1^{ψ} (Eq. B.4a) can be rewritten in CaTL as $\diamond_{[0,10)}(T(0.1, region_B, \{(moisture, 2), (UV, 2)\}) \wedge T(0.5, region_A, \{(arm, 1)\})$. Each capability in CaTL is represented as a sensor and therefore cannot include more complex capabilities, e.g. a robot arm that can perform several different actions. In addition, because CaTL requires the formula to be in positive normal form (i.e. no negation), we cannot express φ_2^{ψ} (Eq. B.4b) in this grammar.

B.7 Conclusion

We define a new task grammar for heterogeneous teams of agents and develop a framework to automatically assign the task to a (sub)team of agents and synthesize correct-by-construction control policies to satisfy the task. We include synchronization constraints to guarantee that the agents perform the necessary collaborations.

In the future, we plan to demonstrate the approach on physical systems where we must ensure that the continuous execution satisfies all safety constraints. We will also explore different notions of optimality when finding a teaming plan, and increase expressivity of the grammar to allow for reactive tasks where agents modify their behavior at runtime in response to environment events.

APPENDIX C

CONTINUOUS EXECUTION OF HIGH-LEVEL COLLABORATIVE TASKS FOR HETEROGENEOUS ROBOT TEAMS

C.1 Introduction

This paper presents a framework to generate decentralized controllers and synchronization signals, when needed, for a team of heterogeneous robots to satisfy a collaborative task. We consider tasks in which the user requires specific actions to be performed, but does not have requirements on the number or type of robots needed to satisfy a task. For example, in a warehouse environment, the user may want to move a pallet and then pick up a package from the pallet. Depending on the number of available robots and their capabilities, the robots may decide to collaborate to satisfy the task, or perform it by themselves. For instance, there may be a mobile manipulator that can perform both actions on its own, or, if it cannot handle a heavy pallet but there is a stationary manipulator that can, the robots will team up so that one moves the pallet and then the other picks up the package.

In our previous work [31], we introduced the logic LTL$^{\psi}$ that allows us to capture such tasks, and proposed a framework that transforms an LTL$^{\psi}$ task to high-level controllers. The logic uses *bindings* to allow users to specify the relationship between robots and parts of the task without providing explicit assignments or constraints on the number of robots required. In [31], the synthesized behavior assumes actions are discrete and treated as instantaneous. However, in physical systems, robots may execute actions with varying duration, such as picking up an object or navigating to an adjacent room. As a result, the syn-

thesized behaviors may violate parts of the specification when executed by the robots. For example, consider a task that requires a surveillance robot to monitor a room only if other robots are present. In the symbolic controller, the two robots can both be outside the room, then immediately be in the room in the next step of their behavior. However, when the robots execute this desired behavior, it is extremely unlikely that they make the transition into the room at the exact same time - any mismatch of timing will cause the task to fail.

In addition, under the instantaneous action assumption of [31], tasks of the form "all robots must enter the room at the same time" and "at least one robot must enter the room" can be solved by assuming all robots with that binding move; however, under the varying action duration assumptions, tasks such as "all robots must enter the room at the same time" are unrealistic and cannot be synthesized. We modify the synthesis algorithms in [31] to explicitly look for solutions of the form "at least one robot" when such specifications are given,

To address these issues, in this paper we define the semantics and synthesize symbolic correct-by-construction controllers for LTL$^{\psi}$ tasks such that the continuous-time behavior of the robots is still guaranteed to satisfy the specification during execution.

C.1.1 Related Work

There is a wealth of literature in task assignment and scheduling for multi-robot systems. Multi-robot task allocation problems are often treated as optimization problems [69, 98, 99], such as the multiple traveling salesman problem [16] or the vehicle routing problem [12]. To mitigate the challenges of finding an exact

solution, common approximate methods are used to solve the problem, such as learning algorithms, heuristics and meta-heuristics [81] [113], and contract net protocols [47] [122]. Coalition formation algorithms allow robots to automatically form teams to execute a task. Approaches include swarm algorithms [115] and multi-stage coalition formation [71], where the authors provide a two-stage approach to prune for a satisfying coalition.

The aforementioned approaches adopt a more generalized representation of tasks that are agnostic to the specific characteristics of each task, with the primary objective being to map agents to tasks. Each task is abstracted with certain constraints associated with it, such as cost or number of agents required. While these methods enable efficient task allocation algorithms, they do not account for the temporal dependencies and sequences of actions within the task itself. For describing temporally extended tasks, temporal logics can be useful; they allow users to rigorously define temporally extended tasks that may require complex action sequences and constraints. These logics define tasks with symbolic abstractions of the given continuous system. For multi-robot systems, existing work has extended Linear Temporal Logic (LTL) [18, 73, 86] and Signal Temporal Logic (STL) [64] [40] to encode tasks that require multiple robots to execute.

To ensure the satisfaction of multi-robot temporal logic specifications in continuous time, a common approach is to encode the task in STL, which allows for discrete-time continuous signals. For example, in [13], the authors synthesize coordinated trajectories of a heterogeneous multi-robot team to satisfy a global task written in STL. They also introduce integral predicates to provide a quantitative metric for the cumulative progress of the tasks. The type of coordi-

nation that can be encoded only includes accumulation (e.g. data collection), as opposed to highly collaborative tasks that require strict synchronization. To address strict synchronization constraints, the authors of [100] propose an event-based synchronization approach in which robots execute local LTL specifications and send synchronization requests when needed. This removes the necessity to synchronize at every discrete step, thus increasing efficiency. However, this work assumes that each robot is assigned a local specification *a priori*, as opposed to collectively satisfying a global task.

In [70], the authors address the problem of synthesizing controllers for a multi-agent system by extending Capability Temporal Logic (CaTL), which is a fragment of STL, to CaTL+. To ensure satisfiability, they encode two layers of logic, one for individual trajectories and one for the team trajectory. The controllers for each agent are then automatically synthesized by solving a centralized two-step optimization problem. For synchronization requirements, each task includes a counting proposition to indicate the timesteps in which synchronization must happen. The task explicitly encodes the number of capabilities required for each part of the task. In our work, we use LTL$^{\psi}$ to define multi-agent tasks, which is an LTL-based grammar we first proposed in [31]. The grammar allows users to define tasks based on actions required instead of the number of agents/capabilities (e.g. "move the pallet and then pick up the package, irrespective of how many agents perform which actions"). In our work, we provide guarantees about the satisfaction of continuous controllers while using a discrete LTL-based logic. In this way, we can maintain the discrete abstraction of actions without any information about their specific timings.

To take into account the fact that actions may have varying durations, the

authors in [83] generate reactive hybrid controllers for LTL tasks such that the continuous behavior of the robots are safe. The authors abstract actions with timing constraints by using initiation and completion propositions, and add constraints in the specification for activating these propositions. The specification also includes constraints to ensure collision avoidance across robots. While this approach accounts for collision avoidance, it does not consider collaborative tasks and therefore also does not account for any necessary synchronization constraints. [76] uses synchronization skeletons to coordinate a homogeneous swarm of robots to collectively satisfy a task. The task contains both a macro specification that describes the behavior of a group of robots, and a micro specification for individual robots to satisfy. To ensure the behavior satisfies the specification in the continuous space (i.e. when robots are between two regions), the authors propose an iterative method of synthesizing labeled transition systems, then removing unsafe transitions and adding safety formulas until a valid labeled transition system is found. Rather than an iterative synthesize-then-check approach, our work aims to automatically abstract continuous actions and synthesize symbolic controllers that are already correct-by-construction for continuous execution.

Contributions: In this paper, we propose a framework for control synthesis for continuous execution of collaborative tasks, where actions may take varying duration of time to complete. Based on [31], we use LTL$^{\#}$to encode such tasks and present a synthesis approach to automatically generate a teaming assignment and corresponding symbolic behavior that is correct-by-construction during continuous execution, while also ensuring synchronization requirements are satisfied for collaborative portions of the task. We demonstrate our approach on a physical multi-robot system.

C.2 Preliminaries

C.2.1 Linear Temporal Logic

LTL formulas are constructed from atomic propositions AP, where $\pi \in AP$ are Boolean variables [29]. The atomic propositions represent an abstraction of robot actions. For example, *camera* captures a robot taking a picture.

Syntax: We use the following syntax to define an LTL formula:

$$\varphi ::= \pi \mid \neg\varphi \mid \varphi \vee \varphi \mid \bigcirc \varphi \mid \varphi \,\mathcal{U}\, \varphi$$

where $\neg$ and $\vee$ are Boolean operators ("not" and "or", respectively), and $\bigcirc$ and $\mathcal{U}$ ("next" and "until", respectively) are temporal operators. Using these, we can also define conjunction $\varphi \wedge \varphi$, implication $\varphi \Rightarrow \varphi$, eventually $\diamond\varphi$ = True $\mathcal{U}\, \varphi$, and always $\square\varphi = \neg\diamond\neg\varphi$.

Semantics: The semantics of an LTL formula φ are defined over an infinite trace $\sigma = \sigma(0)\sigma(1)\sigma(2)...$, where $\sigma(i)$ is the set of AP that are true at step i. We use $\sigma \models \varphi$ to denote that the trace σ satisfies LTL formula φ.

Intuitively, $\bigcirc\varphi$ is satisfied if φ is satisfied in the next step of the sequence; $\diamond\varphi$ is satisfied if φ is true at some step in the sequence; $\square\varphi$ is satisfied if φ is true at every step in σ; $\varphi_1 \,\mathcal{U}\, \varphi_2$ is satisfied if φ_1 remains true until φ_2 becomes true. See [29] for the full semantics.

C.2.2 Büchi Automata

An LTL formula φ can be translated into a Nondeterministic Büchi Automaton that accepts infinite traces if and only if they satisfy φ. A Büchi automaton is a tuple $\mathcal{B} = (Z, z_0, \Sigma_{\mathcal{B}}, \delta_{\mathcal{B}}, F)$, where Z is the set of states, $z_0 \in Z$ is the initial state, $\Sigma_{\mathcal{B}}$ is the input alphabet, $\delta_{\mathcal{B}} : Z \times \Sigma_{\mathcal{B}} \times Z$ is the transition relation, and $F \subseteq Z$ is a set of accepting states. An infinite run of $\mathcal{B}$ over a word $\sigma = \sigma_1\sigma_2\sigma_3... \in \Sigma_{\mathcal{B}}$ is an infinite sequence of states $z = z_0z_1z_2...$ such that $(z_{i-1}, \sigma_i, z_i) \in \delta_{\mathcal{B}}$. A run is accepting if and only if $\text{Inf}(z) \cap F \neq \emptyset$, where $\text{Inf}(z)$ is the set of states that appear in z infinitely often [4].

C.3 Definitions

C.3.1 Actions

We categorize the actions the robots can execute into two types, *instantaneous* and *non-instantaneous*. For example, the action of taking a picture is considered instantaneous; in contrast, the action of moving between rooms is non-instantaneous.

Each action is abstracted as an atomic proposition (AP) π, and its corresponding completion proposition, π_c. π is true at $\sigma(i)$ iff the action associated with π is being executed at position i in the trace; $\pi_c \in AP$ is true at $\sigma(i)$ iff the action associated with π has been completed at position i in the trace. For example, *roomA* indicates that the robot is currently moving towards room A; $roomA_c$ indicates the robot has finished executing the action *roomA* and is currently in room A.

The set of all AP is composed of the subsets AP_{inst} and $AP_{non-inst}$, which are the sets of propositions representing instantaneous actions and non-instantaneous actions, respectively. Formally, an action is instantaneous iff $T(\pi, \pi_c) = 0$, where T is the timing function that outputs the amount of time it takes for the action to be executed; thus, $\pi_c \in AP_{inst}$. Similarly, an action is considered non-instantaneous iff $T(\pi, \pi_c) \neq 0$. For these types of actions, $\pi_c \in AP_{non-inst}$. For example, for the action of beeping, $T(beep, beep_c) = 0$ and therefore $beep_c \in AP_{inst}$. In contrast, $T(roomA, roomA_c) \neq 0$ (i.e. moving into room A takes time), so $roomA_c \in AP_{non-inst}$.

Note that, for synchronization purposes, we assume that robots can coordinate such that they can execute an action simultaneously, so all π are in the set of AP_{inst}. For example, $roomA_c \in AP_{non-inst}$ but $roomA \in AP_{inst}$; $beep, beep_c \in AP_{inst}$ (note that, for instantaneous actions, π and π_c have equivalent meanings).

C.3.2 Robot Model

Based on our prior work [30], each robot j is modeled based on its set of capabilities, $\Lambda_j = \{\lambda_1, ..., \lambda_k\}$. Each capability is a weighted transition system $\lambda = (X, x_0, AP, \Delta, \mathcal{L}, \mathcal{W})$, where X is a finite set of states, $x_0 \in X$ is the initial state, AP is the set of atomic propositions representing the actions the capability can execute, $\Delta \subseteq X \times X$ is a transition relation, $\mathcal{L} : X \to 2^{AP}$ is the labeling function, and $\mathcal{W} : \Delta \to \mathbb{R}_{\geq 0}$ is the cost function assigning a weight to each transition. The cost function is predefined by the user and can represent execution time, power usage, etc.

A robot model A_j is the cross product of its capabilities: $A_j = \lambda_1 \times ... \times \lambda_k$

such that $A_j = (S, s_0, AP_j, \gamma, L, W)$. $S = X_1 \times ... \times X_k$ is the set of states, $s_0 \in S$ is the initial state, $AP_j = \bigcup_{i=1}^{k} AP_i$ is the set of propositions, $\gamma \subseteq S \times S$ is the transition relation, $L : S \to 2^{AP_j}$ is the labeling function, and $W : \gamma \to \mathbb{R}_{\geq 0}$ is the cost function (a function of $\mathcal{W}_i$). More details on the full definition can be found in [30]. See Fig. C.1 for an example of a robot model.

C.3.3 Task Grammar - LTL$^{\psi}$

We first introduced LTL$^{\psi}$in [31]. In this work, because we are removing the assumption of instantaneous behavior, the "next" operator is no longer meaningful therfore we omit it. The task grammar for LTL$^{\psi}$includes atomic propositions that abstract robot action, logical and temporal operators, as in LTL, and bindings that relate actions to specific robots; any action labeled with a certain binding must be satisfied by all the robot(s) assigned that binding.

We define a task φ^{ψ} recursively over LTL and binding formulas.

$$\psi := \rho \mid \psi_1 \vee \psi_2 \mid \psi_1 \wedge \psi_2 \tag{C.1}$$

$$\varphi := \pi \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \, \mathcal{U} \, \varphi \tag{C.2}$$

$$\varphi^{\psi} := \varphi^{\psi} \mid \neg(\varphi^{\psi}) \mid \varphi_1^{\psi_1} \wedge \varphi_2^{\psi_2} \mid \varphi_1^{\psi_1} \vee \varphi_2^{\psi_2} \mid \varphi_1^{\psi_1} \mathcal{U} \varphi_2^{\psi_2} \mid \Box\varphi^{\psi} \tag{C.3}$$

where ψ, the *binding formula*, is a Boolean formula excluding negation over the binding propositions $\rho \in AP_{\psi}$, and φ is an LTL formula consisting of the set of propositions AP_{φ}.

Semantics: The semantics of an LTL$^{\psi}$ formula φ^{ψ} are defined over the team trace $\sigma = \sigma_1\sigma_2...\sigma_n$, where each σ_j is the trace for robot j; and the set of binding assignments $R = \{r_1, r_2, ..., r_n\}$, where $r_j \in R$ is the set of bindings in AP_{ψ} that are

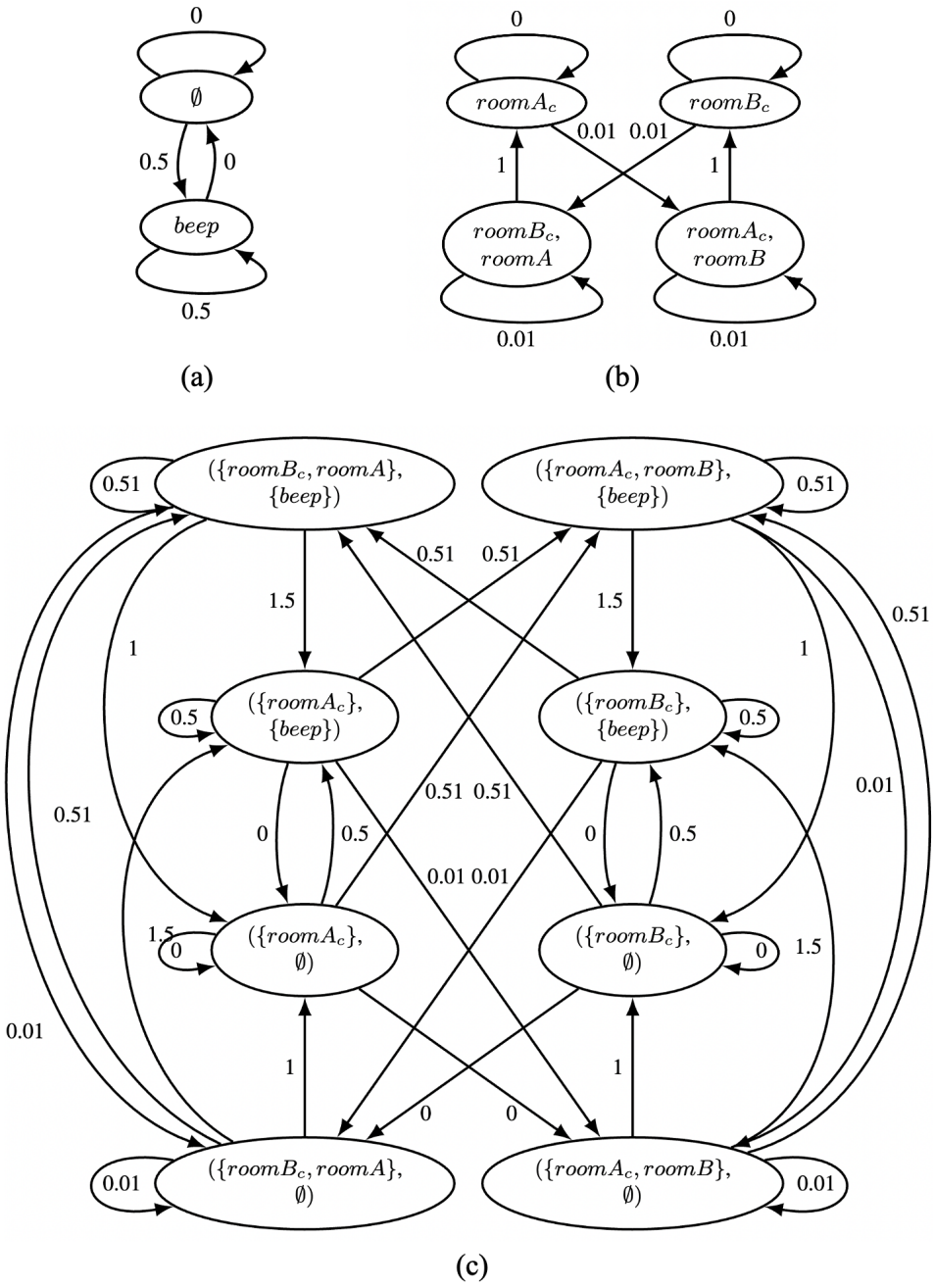


Figure C.1: Partial model of robot A_{blue} (see Sec. C.4): (a) λ_{beep} (b) λ_{motion} (c) A_{blue}

assigned to robot j. A robot's binding assignment does not change throughout the task execution (i.e. R remains the same). For example, $r_1 = \{2\}, r_2 = \{1, 3\}$ indicates that robot 1 is assigned binding 2, and robot 2 is assigned bindings 1 and 3.

Given a set of binding propositions AP_ψ, $\zeta : \psi \rightarrow 2^{2^{AP_\psi}}$ outputs all possible combinations of $\rho \in AP_\psi$ that satisfy ψ. For example, $\zeta((1 \wedge 2) \vee 3) = \{\{1, 2\}, \{3\}, \{1, 2, 3\}\}$. We say that a team of robots satisfies the formula φ^ψ if and only if a binding assignment exists in $\zeta(\psi)$ such that all the bindings are assigned to (at least one) robot, and the behavior of all robots with these binding assignments satisfy φ. A robot may be assigned to multiple bindings, and a binding may be assigned to multiple robots.

Formally, the semantics are defined recursively as follows:

- $(\sigma(i), R) \models \varphi^\psi$ iff $\exists K \in \zeta(\psi)$ s.t. $(K \subseteq \bigcup_{p=1}^{n} r_p)$ and $(\forall j$ s.t. $K \cap r_j \neq \emptyset, \sigma_j(i) \models \varphi)$
- $(\sigma(i), R) \models (\neg\varphi)^\psi$ iff $\exists K \in \zeta(\psi)$ s.t. $(K \subseteq \bigcup_{p=1}^{n} r_p)$ and $(\forall j$ s.t. $K \cap r_j \neq \emptyset, \sigma_j(i) \not\models \varphi)$
- $(\sigma(i), R) \models \neg(\varphi^\psi)$ iff $\exists K \in \zeta(\psi)$ s.t. $(K \subseteq \bigcup_{p=1}^{n} r_p)$ and $(\exists j$ s.t. $K \cap r_j \neq \emptyset, \sigma_j(i) \not\models \varphi)$
- $(\sigma(i),R) \models \varphi_1^{\psi_1} \wedge \varphi_2^{\psi_2}$ iff $(\sigma(i),R) \models \varphi_1^{\psi_1}$ and $(\sigma(i),R) \models \varphi_2^{\psi_2}$
- $(\sigma(i),R) \models \varphi_1^{\psi_1} \vee \varphi_2^{\psi_2}$ iff $(\sigma(i),R) \models \varphi_1^{\psi_1}$ or $(\sigma(i), R) \models \varphi_2^{\psi_2}$
- $(\sigma(i), R) \models \bigcirc\varphi^\psi$ iff $\sigma(i+1), R \models \varphi^\psi$
- $(\sigma(i), R) \models \varphi_1^{\psi_1} \mathcal{U} \varphi_2^{\psi_2}$ iff $\exists \ell \geq i$ s.t. $(\sigma(\ell), R) \models \varphi_2^{\psi_2}$ and $\forall i \leq k < \ell, (\sigma(k), R) \models \varphi_1^{\psi_1}$
- $(\sigma(i), R) \models \Box\varphi^\psi$ iff $\forall \ell > i, (\sigma(\ell), R) \models \varphi^\psi$

Remark 5. *The notation $\neg\varphi^\psi$ and $(\neg\varphi)^\psi$ are equivalent. For example, $\neg roomB^1 \triangleq (\neg roomB)^1$.*

Remark 6. *There is an important distinction between $(\neg\varphi)^{\psi}$ and $\neg(\varphi^{\psi})$. $(\neg\varphi)^{\psi}$ requires that the traces of all robots with binding assignments that satisfy ψ, satisfy $\neg\varphi$. $\neg(\varphi^{\psi})$ requires the formula φ^{ψ} to be violated; at least one robot's trace with a binding assignment that satisfies ψ does not satisfy φ (equivalently, satisfies $\neg\varphi$).*

Remark 7. *Unique to LTL$^{\psi}$ is the ability to encode constraints either on all robots or on at least one robot. φ^{ψ} captures "all robots assigned a binding in $K \in \zeta(\psi)$ must satisfy φ"; $\neg(\neg\varphi^{\psi})$ captures "at least one robot assigned a binding in $K \in \zeta(\psi)$ satisfies φ". This allows the same binding to be assigned to multiple robots, but only one of those robots needs to satisfy φ. This can be especially valuable when encoding safety requirements; for example, the formula $\neg(\neg roomB^1) \Rightarrow (roomB \wedge camera)^2$ captures "if any robot assigned binding 1 is in room B, all robots assigned binding 2 must take a picture of the room."*

Example: Task 1

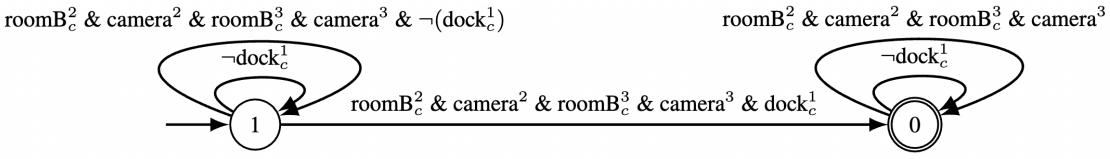


Figure C.2: Büchi automaton for Task 1

$$\varphi^{\psi} = \Diamond(dock_c^1) \wedge \Box(\neg(\neg dock_c^1) \rightarrow (roomB_c \wedge camera)^{2\wedge 3}) \tag{C.4}$$

In English, the specification captures "All robot(s) assigned binding 1 must eventually be at the dock. Anytime one of these robots is at the dock, all robot(s) assigned bindings 2 and 3 must be in room B taking pictures." The second part of the task is a safety constraint that ensures that the dock is monitored anytime a robot assigned binding 1 is in that room.

C.4 Problem Statement

Given n heterogeneous robots $A = \{A_1, ..., A_n\}$ and a task φ^ψ in LTL$^\psi$, find a team of robots $\hat{A} \subseteq A$, their binding assignments $R_{\hat{A}}$, and synthesize behavior σ_j for each robot such that $(\sigma(0), R_{\hat{A}}) \models \varphi^\psi$ and can by design be implemented in continuous execution (i.e., the physical execution of σ_j does not violate any part of φ). Similar to our prior work, this behavior includes synchronization constraints implicitly encoded in the task for robots to satisfy the necessary collaborative actions.

We assume that the system is nonreactive; robots cannot change their behavior at runtime. We also assume that every state in both the Büchi automaton of the task φ^ψ and the robot model has a self-transition (i.e. the robot can wait in any state).

Example: Let there be four robots $A = \{A_{green}, A_{blue}, A_{orange}, A_{pink}\}$ in a warehouse environment (Fig. C.3). The set of all capabilities is $\Lambda = \{\lambda_{mot}, \lambda_{arm}, \lambda_{beep}, \lambda_{cam}, \lambda_{scan}\}$, where λ_{mot} is the motion model representing how the robots can move through the environment. λ_{arm} represents a robotic manipulator; the arm can pick up and drop off packages, as well as push boxes. λ_{beep}, λ_{cam}, and λ_{scan} are a robot's ability to beep, take a picture, and scan, respectively. λ_{mot} and λ_{arm} execute non-instantaneous actions; the remaining capabilities contain only instantaneous actions.

The robots' capabilities and labels on their initial states are:

$\Lambda_{green} = \{\lambda_{mot}, \lambda_{arm}, \lambda_{camera}\}, L(s_0) = \{roomD_c\}$

$\Lambda_{blue} = \{\lambda_{mot}, \lambda_{beep}\}, L(s_0) = \{roomC_c\}$

$\Lambda_{orange} = \{\lambda_{mot}, \lambda_{arm}\}, L(s_0) = \{roomE_c\}$

$\Lambda_{pink} = \{\lambda_{mot}, \lambda_{beep}, \lambda_{cam}, \lambda_{scan}\}, L(s_0) = \{roomE_c\}$

The goal is to construct a team, i.e. a subset of these robots, and corresponding binding assignments for the robots to satisfy the task captured as the LTL$^{\psi}$specification provided in Eq. C.4. We assume that the robots have low-level controllers that ensure collision avoidance and coordination when multiple robots execute a capability together (e.g. pushing a cart).

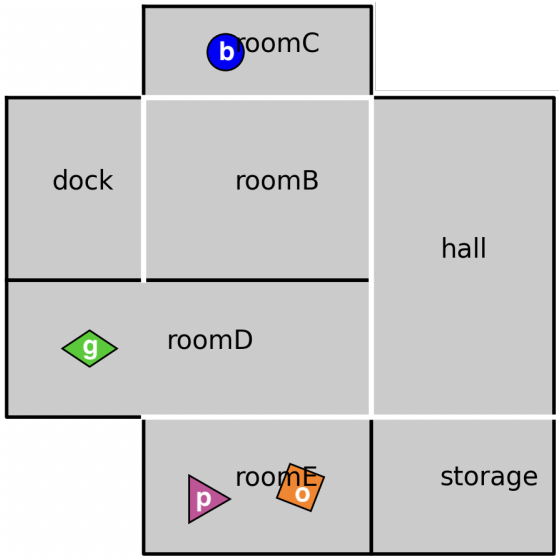


Figure C.3: Environment and robot setup

C.5 Approach

To find a teaming assignment and synthesize the corresponding synchronization and control that is guaranteed to satisfy the task even when actions have varying execution times, we start with an approach similar to our prior work [31]: first, we automatically generate a Büchi automaton $\mathcal{B}$ for the task φ^{ψ} (Sec. C.5.1). Each robot A_j then constructs a product automaton $\mathcal{G}_j = A_j \times \mathcal{B}$

(Sec. C.5.2). The novelty in our current work is our approach to finding a team of robots to satisfy the task. In prior work, we used a depth first search (DFS) algorithm to find both a team of robots and a trace through $\mathcal{B}$ to an accepting cycle. For every transition in the trace, every proposition in AP_ψ is assigned to at least one robot in the team. In this paper, we further ensure that the varying time durations of the actions do not result in violations of any part of the task. To do so, we update the Büchi automaton as the DFS progresses to include intermediate states that enforce the necessary ordering of actions without violating the original specification (Sec. C.5.3). As a result, we also update a robot's product automaton to reflect the changes in the Büchi automaton (Sec. C.5.4). The entire DFS framework is outlined in Sec. C.5.5.

Once an accepting trace and corresponding robot team is found, each robot synthesizes its own behavior that allows for parallel execution. However, some parts of the task may require collaboration, in which case the robots must synchronize. We create a synchronization policy where each robot communicates to others when they need to synchronize and waits for other robots before executing the collaborative portions of their behavior (Sec. C.5.6).

C.5.1 Büchi Automaton for an LTL$^\psi$ Formula

To create a Büchi automaton from an LTL$^\psi$ specification, we first modify the specification to constrain binding propositions solely to individual atomic propositions $\pi \in AP_\varphi$ (i.e. the formula contains only π^ρ). For example, the formula $(\Diamond(pickup \wedge roomA_c))^{1\vee 2}$ is rewritten as $(\Diamond(pickup^1 \wedge roomA_c^1) \vee (\Diamond(pickup^2 \wedge roomA_c^2))$.

Using AP_φ and AP_ψ, we define AP_φ^ψ, where $\forall \pi \in AP_\varphi$ and $\forall \rho \in AP_\psi$, $\pi^\rho \in AP_\varphi^\psi$. We automatically create the Büchi automaton from these propositions using Spot [28].

Prior to control synthesis, we first convert the transitions in the Büchi automaton, labeled with Boolean formulas, into disjunctive normal form (DNF). Subsequently, we substitute the transition labeled with a DNF formula containing ℓ conjunctive clauses with ℓ transitions between the same states. The label for each of these transitions is a distinct conjunction of the original label.

When constructing a Büchi automaton from an LTL formula φ, $\sigma \in \Sigma_\mathcal{B}$ are Boolean formulas over AP_φ, the atomic propositions that appear in φ. In this work, for a Büchi automaton of an LTL$^\psi$ formula, $\Sigma_\mathcal{B} = 2^{AP_\varphi^\psi} \times 2^{AP_\varphi^\psi} \times 2^{AP_\varphi^\psi} \times 2^{AP_\varphi^\psi}$, where $\sigma = (\sigma^T, \sigma^{exT}, \sigma^F, \sigma^{exF}) \in \Sigma_\mathcal{B}$ contains the set of propositions π^ρ that must be true/false for all robots (σ^T and σ^F), called *for all* propositions, and the set of propositions π^ρ that must be true/false for at least one robot (σ^{exT} and σ^{exF}), called *there exists* propositions. σ^{exT} are the set of propositions in which $\neg(\neg\pi^\rho)$ is true; σ^{exF} are the set of propositions in which $\neg(\pi^\rho)$ is true. Any proposition that does not appear in σ can maintain any truth value.

We modify the definition of σ compared to our prior work [31], where we assumed instantaneous robot actions; there, $\sigma = (\sigma_T, \sigma_F)$. While [31] also had *there exists* propositions, in practice the resulting behavior of *there exists* and *for all* propositions was the same. This is because, for a *there exists* proposition π^ρ, only one robot assigned binding ρ needs to execute π^ρ. However, since the robots do not coordinate, if there are multiple robots assigned ρ, they all satisfy π^ρ (satisfying *for all* propositions also satisfies *there exists* propositions). Thus, it was unnecessary in [31] to differentiate between these two types of propositions.

This is not true for time varying actions; there are scenarios in which a *there exists* proposition is satisfied but a *for all* proposition is not.

Example. Consider the task defined in Eq. C.4. Supposed we have two robots with $r_{green} = \{1\}, r_{blue} = \{2, 3\}$. Our prior approach may output a discrete teaming plan in which the blue robot enters the dock at exactly the same time the green robot enters room B. However, the physical execution of this plan may violate the specification; although the robots can begin moving towards their respective areas at the same time, the green robot may enter the dock before the blue robot enters room B, thus violating the safety constraint.

C.5.2 Constructing the Product Automaton

To formally define the product automaton, we first modify the functions introduced in [31]. For illustration, we will use the Büchi automaton for task 1 as our example (Fig. C.2). Let the outer self-transition on state 1 be $\sigma_{11,o}$. $\sigma_{11,o} = (\sigma^T_{11,o}, \sigma^{exT}_{11,o}, \sigma^F_{11,o}, \sigma^{exF}_{11,o}) = (\{roomB^2_c, camera^2, roomB^3_c, camera^3\}, \emptyset, \emptyset, \{dock^1_c\})$.

Definition 5 (Binding Transition Function)**.** $\mathfrak{B}_\sigma : \Sigma_\mathcal{B} \rightarrow 2^{AP_\psi}$ *such that for* $\sigma = (\sigma^T, \sigma^{exT}, \sigma^F, \sigma^{exF}) \in \Sigma_\mathcal{B}, \mathfrak{B}_\sigma(\sigma) \subseteq AP_\psi$ *is the set* $\{\rho \in AP_\psi \mid \exists \pi^\rho \in \sigma^T \cup \sigma^{exT} \cup \sigma^F \cup \sigma^{exF}\}$.

$\mathfrak{B}_\sigma(\sigma)$ outputs the set of bindings that appear in a Büchi transition label σ. For example, $\mathfrak{B}_\sigma(\sigma_{11,o}) = \{1, 2, 3\}$.

Definition 6 (Binding Set Function)**.** $\mathfrak{B}_\Pi : 2^{AP^\psi_\varphi} \rightarrow 2^{AP_\psi}$ *such that for* $\Pi \subseteq AP^\psi_\varphi, \mathfrak{B}_\Pi(\Pi) \subseteq AP_\psi$ *is the set* $\{\rho \in AP_\psi \mid \exists \pi^\rho \in \Pi\}$.

$\mathfrak{B}_\Pi(\Pi)$ outputs the set of bindings in a given set of LTL$^\psi$ propositions Π. For example, for $\Pi = \{roomB_c^2, camera^2, roomB_c^3, camera^3, dock_c^1\}$, $\mathfrak{B}_\Pi(\Pi) = \{1, 2, 3\}$; for $\Pi = \{dock_c^1\}$, $\mathfrak{B}_\Pi(\Pi) = \{1\}$.

Definition 7 (Capability Function)**.** $\mathfrak{C} : \Sigma_\mathcal{B} \times AP_\psi \to 2^{AP_\varphi} \times 2^{AP_\varphi} \times 2^{AP_\varphi} \times 2^{AP_\varphi}$ *such that for* $(\sigma^T, \sigma^{exT}, \sigma^F, \sigma^{exF}) \in \Sigma_\mathcal{B}, \rho \in AP_\psi$, $\mathfrak{C}(\sigma, \rho) = (C_T, C_{exT}, C_F, C_{exF})$*, where for* $k \in \{T, exT, F, exF\}$, $C_k = \{\pi \in AP_\varphi \mid \exists \pi^\rho \in \sigma^k\}$.

C_T and C_F are the sets of *for all* propositions that are True/False and appear with binding ρ in label σ of a Büchi transition and must be True/False for all robots assigned binding ρ; C_{exT} and C_{exF} are defined similarly but for *there exists* propositions with binding ρ. For example, for $\sigma_{11,o} = (\{roomB_c^2, camera^2, roomB_c^3, camera^3\}, \emptyset, \emptyset, \{dock_c^1\})$, $\mathfrak{C}(\sigma_{11,o}, 2) = (\{roomB_c, camera\}, \emptyset, \emptyset, \emptyset)$ (for binding 2, the relevant propositions are $roomB_c$ and *camera*, both of which must be true and are *for all* propositions, i.e. the set σ^T), and $\mathfrak{C}(\sigma_{11,o}, 1) = (\emptyset, \emptyset, \emptyset, \{dock_c\})$ ($dock_c$ is the only proposition with the binding 1 and is in σ^{exF}).

Definition 8 (Binding Assignment Function)**.** *Given a robot model* A_j *and a label* $\sigma = (\sigma^T, \sigma^{exT}, \sigma^F, \sigma^{exF})$, $\mathfrak{R}(s, \sigma, s') = \{r \in 2^{AP_\psi} \setminus \emptyset \mid \forall \rho \in r, (C_T, C_{exT}, C_F, C_{exF}) = \mathfrak{C}(\sigma, \rho), \bigcup_{\rho \in r}(C_T \cup C_{exT}) \subseteq L(s')$ *and* $\bigcup_{\rho \in r}(C_F \cup C_{exF}) \cap L(s') = \emptyset\}$.

To synthesize behavior for a robot, we find the minimum cost accepting trace in its product automaton $\mathcal{G}_j = A_j \times \mathcal{B}$, where $A_j = (S, s_0, AP_j, \gamma, L, W)$ is the robot model, and $\mathcal{B} = (Z, z_0, \Sigma_\mathcal{B}, \delta_\mathcal{B}, F)$ is the Büchi automaton.

Given a transition (s, σ, s'), $\mathfrak{R}$ generates all possible combinations of binding propositions that can be assigned to a robot to satisfy σ. A robot can be assigned binding ρ if and only if two conditions hold on the robot's next state, s': 1) the

state label of s' contains all propositions π that appear in either σ^T or σ^{exT} as π^ρ, and 2) the state label of s' does not contain any propositions π that appear in σ^F or σ^{exF} as π^ρ.

A binding assignment r can contain any binding propositions not in σ, since there are no propositions required by those bindings. In addition, it follows from condition (2) that if a proposition π^ρ is in $\sigma^F \cup \sigma^{exF}$ and π is not in AP_j (e.g. $camera^1 \in \sigma^F$ and the robot does not have the capability λ_{camera}), ρ can still be assigned to robot j.

Given the binding assignment function $\Re$, and as shown in [31], we can now define the product automaton:

Definition 9 (Product Automaton). $\mathcal{G}_j = A_j \times \mathcal{B} = (Q, q_0, AP_j, \delta_\mathcal{G}, L_\mathcal{G}, W_\mathcal{G}, F_\mathcal{G})$, *where*

- $Q = S \times Z$ *is a finite set of states*
- $q_0 = (s_0, z_0) \in Q$ *is the initial state*
- $\delta_\mathcal{G} \subseteq Q \times Q$ *is the transition relation, where for* $q = (s, z)$ *and* $q' = (s', z')$, $(q, q') \in \delta_\mathcal{G}$ *if and only if* $(s, s') \in \gamma$ *and* $\exists \sigma \in \Sigma_\mathcal{B}$ *such that* $(z, \sigma, z') \in \delta_\mathcal{B}$ *and* $\Re(s, \sigma, s') \neq \emptyset$
- $L_\mathcal{G}$ *is the labeling function s.t. for* $q = (s, z)$, $L_\mathcal{G}(q) = L(s) \subseteq AP_j$
- $W_\mathcal{G} : \delta_\mathcal{G} \to \mathbb{R}_{\geq 0}$ *is the cost function s.t. for* $(q, q') \in \delta_\mathcal{G}$, $q = (s, z)$, $q' = (s', z')$, $W_\mathcal{G}((q, q')) = W((s, s'))$
- $F_\mathcal{G} = S \times F$ *is the set of accepting states*

C.5.3 Adding Intermediate Transitions to the Büchi Automaton

To ensure correct team behavior when actions take different time durations, we add intermediate transitions to the Büchi automaton $\mathcal{B}$ to reason about and enforce an ordering on individual robot behaviors (Alg. 6). In doing so, we maintain the symbolic abstractions of actions while providing guarantees for the satisfaction of the task by a physical system.

To motivate the use of intermediate transitions, we look at the transitions $\sigma_{11,i}$ (the inner self-transition along state 1) and σ_{10} of the Büchi automaton in Fig C.2, where $\sigma_{11,i} = (\sigma^T_{11,i}, \sigma^{exT}_{11,i}, \sigma^F_{11,i}, \sigma^{exF}_{11,i}) = (\emptyset, \emptyset, \{dock^1_c\}, \emptyset)$, $\sigma_{10} = (\sigma^T_{10}, \sigma^{exT}_{10}, \sigma^F_{10}, \sigma^{exF}_{10}) = (\{roomB^2_c, camera^2, roomB^3_c, camera^3, dock^1_c\}, \emptyset, \emptyset, \emptyset)$. σ_{10} requires all robots assigned binding 2 and 3 to be in roomB. Since it is unrealistic for multiple robots to enter the room simultaneously, we can leverage the fact that $\sigma_{11,i}$ does not explicitly assign truth values to $roomB^2_c$ and $roomB^3_c$ (i.e. these propositions can have any truth value over $\sigma_{11,i}$) in order to enforce that all the robots assigned bindings 2 and 3 enter room B at some point before executing the transition σ_{10}. For clarity in the remainder of the paper, we define a proposition π^ρ to be assigned an *explicit* truth value over a transition σ if it appears in σ, i.e. $\pi^\rho \in \sigma^T \cup \sigma^{exT} \cup \sigma^F \cup \sigma^{exF}$. If a proposition is not explicitly in σ, it can maintain any truth value over that transition.

To automatically construct intermediate transitions, we first define the intermediate propositions function, $\mathcal{I}(\sigma)$:

Definition 10 (Intermediate Propositions Function)**.** $\mathcal{I}(\sigma) : \Sigma_\mathcal{B} \rightarrow 2^{AP_{non-inst}} \times 2^{AP_{non-inst}} \times 2^{AP_{non-inst}} \times 2^{AP_{non-inst}}$ *such that for* $\sigma = (\sigma^T, \sigma^{exT}, \sigma^F, \sigma^{exF}) \in \Sigma_\mathcal{B}$, $\mathcal{I}(\sigma) = (I_T, I_{exT}, I_F, I_{exF})$, *where* $\forall k \in \{T, exT, F, exF\}, I_k = \{\pi^\rho \in \sigma^k \,|AP_{non-inst} \cap (C_k \in$

$\mathfrak{C}(\sigma, \rho)) \neq \emptyset\}$

I_T, I_{exT}, I_F, and I_{exF} are the sets of LTL$^{\psi}$ propositions corresponding to non-instantaneous actions that are assigned truth values over $\sigma^T, \sigma^{exT}, \sigma^F,$ and σ^{exF}, respectively. For example, in Fig. C.2 where $\sigma_{11,o} = (\sigma^T_{11,o}, \sigma^{exT}_{11,o}, \sigma^F_{11,o}, \sigma^{exF}_{11,o}) = (\{roomB^2_c, camera^2, roomB^3_c, camera^3\}, \emptyset, \emptyset, \{dock^1_c\})$, the intermediate propositions are $\mathcal{I}(\sigma_{11,o}) = (\{roomB^2_c, roomB^3_c\}, \emptyset, \emptyset, \{dock^1_c\})$, since all the motion propositions are non-instantaneous; *camera* is instantaneous and therefore omitted from $\mathcal{I}(\sigma_{11,o})$.

We use the intermediate propositions function to determine which non-instantaneous propositions change truth values across two transitions; we then use these propositions to enforce constraints on the binding assignments and on robot behavior. Let σ, σ' be labels of two transitions in the Büchi automaton, and $\mathcal{I}(\sigma) = (I_T, I_{exT}, I_F, I_{exF}), \mathcal{I}(\sigma') = (I'_T, I'_{exT}, I'_F, I'_{exF})$. To categorize how these propositions change their truth values, we define the following sets (Alg. 6, line 3):

$$\Pi_{FT} = I_T \cap (I'_F \cup I'_{exF}) \tag{C.5}$$

$$\Pi_{TF} = I_F \cap (I'_T \cup I'_{exT}) \tag{C.6}$$

$\Pi_{FT}, \Pi_{TF} \subseteq AP_{non-inst}$ are the sets of non-instantaneous *for all* propositions that explicitly change truth values between σ and σ'.

$$\Pi_{exT} = I_{exT} \cap (I'_F \cup I'_{exF}) \tag{C.7}$$

$$\Pi_{exF} = I_{exF} \cap (I'_T \cup I'_{exT}) \tag{C.8}$$

$\Pi_{exT}, \Pi_{exF} \subseteq AP_{non-inst}$ include any non-instantaneous proposition that is a *there*

exists proposition in σ that explicitly changes truth values between σ and σ'.

$$\Pi_{\emptyset T} = (I'_T \setminus I_T) \tag{C.9}$$

$$\Pi_{\emptyset exT} = (I'_{exT} \setminus I_{exT}) \tag{C.10}$$

$$\Pi_{\emptyset F} = (I'_F \setminus I_F) \tag{C.11}$$

$$\Pi_{\emptyset exF} = (I'_{exF} \setminus I_{exF}) \tag{C.12}$$

$\Pi_{\emptyset T}, \Pi_{\emptyset exT}, \Pi_{\emptyset F}, \Pi_{\emptyset exF} \subseteq AP_{non-inst}$ are the sets of non-instantaneous propositions that are assigned truth values in σ' but not in σ.

Example: Task 1. In Fig. C.2, consider the two transitions $(1, \sigma_{11,i}, 1)$, $(1, \sigma_{10}, 0)$, where $\sigma_{11,i} = (\emptyset, \emptyset, \{dock_c^1\}, \emptyset)$ is the inner self-transition on state 1, and $\sigma_{10} = (\{roomB_c^2, camera^2, roomB_c^3, camera^3, dock_c^1\}, \emptyset, \emptyset, \emptyset)$. Then $\Pi_{TF} = \{dock_c^1\}$, $\Pi_{\emptyset T} = \{roomB_c^2, roomB_c^3\}$; the remaining sets are empty.

Violations of the task may occur when the robots execute a transition in the Büchi automaton $\mathcal{B}$. Thus, to prevent any violations, we check two sequential transitions in $\mathcal{B}$, $e_1 = (z_1, \sigma_{12}, z_2), e_2 = (z_2, \sigma_{23}, z_3)$, and update the Büchi automaton $\mathcal{B}$ if a specific ordering of propositions is necessary. When there is at least one non-instantaneous proposition that changes truth values along e_2, we want to enforce an ordering such that the other non-instantaneous propositions are completed first. Thus, we only need to modify the Büchi automaton if there exists more than one non-instantaneous proposition appears along e_2 ($|I'_T \cup I'_{exT} \cup I'_F \cup I'_{exF}| > 1$) and at least one of them explicitly changes truth values ($|\Pi_{FT} \cup \Pi_{exT} \cup \Pi_{TF} \cup \Pi_{exF}| \geq 1$, line 8).

Binding Constraints (lines 4 - 7): There may be a sequence of transitions in which non-instantaneous propositions change truth values, which may require imposing constraints of the robot binding assignments. For example, looking at

Algorithm 6: Update Büchi Automaton

Input : $\mathcal{B}, e_1 = (z_1, \sigma_{12}, z_2), e_2 = (z_2, \sigma_{23}, z_3)$, where
$\sigma_{12} = (\sigma_{12}^{T}, \sigma_{12}^{exT}, \sigma_{12}^{F}, \sigma_{12}^{exF})$, $\sigma_{23} = (\sigma_{23}^{T}, \sigma_{23}^{exT}, \sigma_{23}^{F}, \sigma_{23}^{exF})$

Output: $\mathcal{B}_{upd}, E_{upd}, (c_{single}, c_{combo})$

$\mathcal{B}_{upd} = \mathcal{B}$, $c_{single} = \emptyset, c_{combo} = \emptyset$

$(I_T, I_{exT}, I_F, I_{exF}) = \mathcal{I}(\sigma_{12}), (I'_T, I'_{exT}, I'_F, I'_{exF}) = \mathcal{I}(\sigma_{23})$

// compute sets of non-inst actions

$\Pi_{FT}, \Pi_{TF}, \Pi_{exT}, \Pi_{exF}, \Pi_{\emptyset T}, \Pi_{\emptyset F}, \Pi_{\emptyset exT}, \Pi_{\emptyset exF} =$
NON_INST_SETS$((I_T, I_{exT}, I_F, I_{exF}), (I'_T, I'_{exT}, I'_F, I'_{exF}))$

// keep track of binding constraints

if $|\Pi_{FT} \cup \Pi_{TF}| \geq 1$ **then**
  $c_{single} = \mathfrak{B}_{\Pi}(\Pi_{FT} \cup \Pi_{TF})$

if $|\Pi_{exT} \cup \Pi_{exF}| \geq 2$ **then**
  $c_{combo} = \mathfrak{B}_{\Pi}(\Pi_{exT} \cup \Pi_{exF})$

// no need to update Büchi

if $|I'_T \cup I'_{exT} \cup I'_F \cup I'_{exF}| \leq 1$ or $|\Pi_{FT} \cup \Pi_{TF} \cup \Pi_{exT} \cup \Pi_{exF}| < 1$ **then**
  return $\mathcal{B}, \emptyset, c_{single}, c_{combo}$

$\sigma_{11^*} = (\sigma_{11^*}^{T}, \sigma_{11^*}^{exT}, \sigma_{11^*}^{F}, \sigma_{11^*}^{exF}) =$
INTERMEDIATE_TRANS$(\sigma_{12}, \Pi_{\emptyset T}, \Pi_{\emptyset exT}, \Pi_{\emptyset F}, \Pi_{\emptyset exF})$ **if** $z_1 = z_2$ **then**
  add $(z_1, \sigma_{12}, z_{1^*})$, $(z_{1^*}, \sigma_{11^*}, z_{1^*})$, $(z_{1^*}, \sigma_{23}, z_3)$ to $\mathcal{B}_{upd}$; remove (z_2, σ_{23}, z_3) from $\mathcal{B}_{upd}$
  $E_{upd} = ((z_1, \sigma_{12}, z_{1^*}), (z_{1^*}, \sigma_{11^*}, z_{1^*}), (z_{1^*}, \sigma_{23}, z_3))$

else
  // modify first transition (non-self transition)
  if $\sigma_{12} \neq \sigma_{11^*}$ **then**
    modify $(z_1, \sigma_{12}, z_2) = (z_1, \sigma_{11^*}, z_2)$ in $\mathcal{B}_{upd}$
    $E_{upd} = ((z_1, \sigma_{11^*}, z_2))$
  else
    $E_{upd} = ()$

return $\mathcal{B}_{upd}, E_{upd}, (c_{single}, c_{combo})$

the same two transitions as before, $(1, \sigma_{11,i}, 1)$ and $(1, \sigma_{10}, 0)$, $\Pi_{FT} \cup \Pi_{TF} = \{dock_c^1\}$. In $\sigma_{11,i}$, we require all robots assigned binding 1 to not be in the dock, and in the next transition σ_{10}, we require all robots assigned binding 1 to be in the dock. If multiple robots are assigned binding 1, this realistically will not happen when executing in continuous time, as it would require all robots to enter the dock at the exact same time. Thus, to guarantee these transitions in the Büchi automaton are satisfied, we constrain the binding assignments such that one and only one robot can be assigned bindings in $\mathfrak{B}(\Pi_{FT} \cup \Pi_{TF})$ (in this case, binding 1). The same constraints apply if multiple bindings are present (e.g. if $\Pi_{FT} \cup \Pi_{TF} = \{dock_c^1, roomB_c^2\}$, then one and only one robot can be assigned both bindings 1 and 2; otherwise, two robots must make the transition at the exact same time, which is not realistic). These combinations of bindings are stored in c_{single}.

Similar constraints apply when $|\Pi_{exT} \cup \Pi_{exF}| \geq 2$, i.e. multiple *there exists* propositions change truth values. In this case, for each proposition π^ρ in $\Pi_{exT} \cup \Pi_{exF}$, we require that at least one robot assigned ρ will change its truth value of π_c across the two transitions. For instance, if $\sigma_{12} = \{\emptyset, \emptyset, \emptyset, \{dock_c^1, push_c^2\}\}$, $\sigma_{23} = \{\{dock_c^1, push_c^2\}, \emptyset, \emptyset, \emptyset\})$, σ_{12} requires at least one robot assigned binding 1 to not be at the dock, and at least one robot assigned binding 2 to not complete the pushing action, while σ_{23} requires all robots assigned binding 1 and 2 to be at the dock and complete pushing, respectively. On a physical system, it is unrealistic for robots to be able to synchronize completing a non-instantaneous action at the same time. As a result, to guarantee satisfiability, we constrain the binding assignment such that if a robot is assigned to any binding in $\mathfrak{B}_\Pi(\Pi_{exT} \cup \Pi_{exF})$, then it must be assigned to all bindings within that set; otherwise, it cannot be assigned to any of them. These binding assignments are stored in c_{combo}. Note that the constraints are not necessary when $|\Pi_{exT} \cup \Pi_{exF}| = 1$; in this case, only one

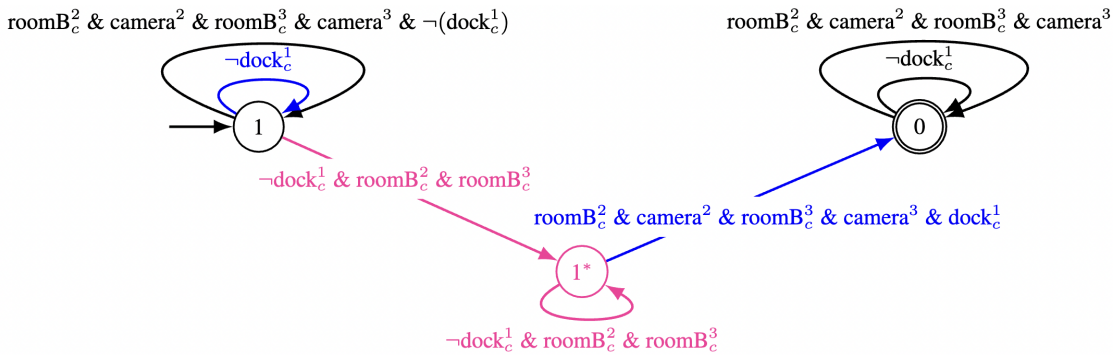


Figure C.4: Updated Büchi automaton for Case 1 (e_1 is a self-transition and e_2 is not). The blue represents the original transitions, the pink represents the added intermediate states and transitions

robot assigned ρ must change its truth value across the two transitions, which can easily be satisfied by a physical system.

If more than one non-instantaneous proposition appears along the second transition e_2, we generate an intermediate transition $\sigma_{11^*} = (\sigma^T_{11^*}, \sigma^{exT}_{11^*}, \sigma^F_{11^*}, \sigma^{exF}_{11^*})$ (line 10), where $\sigma^T_{11^*} = \sigma^T_{12} \cup \Pi_{\emptyset T}$, $\sigma^F_{11^*} = \sigma^F_{12} \cup \Pi_{\emptyset F}$, $\sigma^{exT}_{11^*} = \sigma^{exT}_{12} \cup \Pi_{\emptyset exT}$, $\sigma^{exF}_{11^*} = \sigma^{exF}_{12} \cup \Pi_{\emptyset exF}$; because any proposition that does not appear in σ_{12} can maintain any truth value, the transition σ_{11^*} ensures that all propositions that appear in σ_{23} but not in σ_{12} (i.e. $\Pi_{\emptyset T} \cup \Pi_{\emptyset exT} \cup \Pi_{\emptyset F} \cup \Pi_{\emptyset exF}$) change their truth values, in any order, before any of the propositions that change truth values between the transitions (i.e. $\Pi_{TF} \cup \Pi_{exT} \cup \Pi_{FT} \cup \Pi_{exF}$) do so.

Updating transitions in $\mathcal{B}$: Since we assume that every state in the Büchi automaton $\mathcal{B}$ has a self transition, either e_1 is a self-transition and e_2 is not, or vice-versa; the approach in updating $\mathcal{B}$ differs for the two cases.

Case 1: *e_1 is a self-transition and e_2 is not (lines 10 - 12).* We take advantage of the fact that e_1 is a self-transition and add an intermediate state along with the transitions $(z_1, \sigma_{12}, z_{1^*})$, $(z_{1^*}, \sigma_{11^*}, z_{1^*})$, and $(z_{1^*}, \sigma_{23}, z_3)$. These transitions satisfy

the self-transition e_1 while also including constraints that must be enforced to ensure the robot's continuous time execution of the actions do not violate any part of e_2.

Example: Given the Büchi automaton in Fig. C.2, let $e_1 = (1, \sigma_{11,i}, 1)$, $e_2 = (1, \sigma_{10}, 0)$, where $\sigma_{11,i}$ and σ_{10} are defined earlier in the example. Then $\sigma_{11^*} = (\{roomB_c^2, roomB_c^3\}, \emptyset, \{dock_c^1\}, \emptyset)$. We add an intermediate state 1^* and the corresponding transitions $(1, \sigma_{11,i}, 1^*)$, $(1^*, \sigma_{11^*}, 1^*)$, and $(1^*, \sigma_{10}, 0)$. Fig. C.4 shows the Büchi automaton with these updates.

Case 2: *e_2 is a self-transition and e_1 is not.* Because e_1 is not a self-transition, we cannot add intermediate states between e_1 and e_2. In this case, we modify e_1 from (z_1, σ_{12}, z_2) to $(z_1, \sigma_{11^*}, z_2)$. Intuitively, we are adding all the non-instantaneous propositions explicitly in σ_{23} into σ_{12}. Changing this transition does not affect the satisfiability of the original task, since the modification only adds more constraints on propositions that do not violate the original formula along the transition.

Example: Given the Büchi automaton in Fig. C.2, let $e_1 = (1, \sigma_{10}, 0)$, $e_2 = (0, \sigma_{00}, 0)$, where σ_{10} is defined earlier in the example, and $\sigma_{00} = (\emptyset, \emptyset, \{dock_c^1\}, \emptyset)$. Since both $\Pi_{\emptyset T}$ and $\Pi_{\emptyset F}$ are empty, σ_{11^*} and σ_{10} are equivalent, and no updates to the Büchi automaton are necessary. However, since $\Pi_{FT} \cup \Pi_{TF} = \{dock_c^1\}$, one and only one robot can be assigned binding 1. Thus, $c_{single} = \{1\}$.

Algorithm 6 outputs $\mathcal{B}_{upd}$, the updated Büchi automaton, E_{upd}, the edges that have been modified/added, and (c_{single}, c_{combo}), which include constraints on the bindings in which one and only one robot can be assigned to (i.e. c_{single}), or bindings that a robot must be assigned either to all or none of those bindings

(i.e. c_{combo}).

C.5.4 Updating the Product Automaton

Because we are modifying the Büchi automaton as we search for an accepting trace through it, we also need to modify each robot's product automaton $\mathcal{G}$ to check if the robot can synthesize controllers to satisfy the given trace in the Büchi automaton (Alg. 7). This is done for each transition we check in the Büchi automaton, when constructing the trace.

Algorithm 7: Update Robot j's Product Automaton

Input : $\mathcal{B}, \mathcal{B}_{upd}, E_{upd}, \mathcal{G}_j$
Output: $\mathcal{G}_{j,upd}$

if $\mathcal{B}_{upd} = \mathcal{B}$ **then**
  return $\mathcal{G}_j$
$\mathcal{G}_{j,upd} = \mathcal{G}_j$
if $|E_{upd}| > 1$ **then**
  $(z_1, \sigma_{12}, z_{1^*}), (z_{1^*}, \sigma_{11^*}, z_{1^*}), (z_{1^*}, \sigma_{23}, z_3) = E_{upd}$
  $\mathcal{G}_{j,upd}$ = REMOVE_EDGES($\mathcal{G}_{j,upd}, (z_2, \sigma_{23}, z_3)$)
  $\mathcal{B}'_{upd}$ = GET_SUBBUCHI($\mathcal{B}_{upd}, E_{upd}$)
  A'_j = GET_SUBAGENT($\mathcal{G}_j, (z_1, \sigma_{12}, z_2), (z_2, \sigma_{23}, z_3)$)
  $\mathcal{G}'_j = A'_j \times \mathcal{B}'_{upd}$
  `// join two product automata`
  $\mathcal{G}_{j,upd} = \mathcal{G}_{j,upd} \cup \mathcal{G}'_j$
else if $|E_{upd}| = 1$ **then**
  $(z_1, \sigma_{11^*}, z_2) = E_{upd}$
  `// sub product automaton to check`
  $\mathcal{G}'_j$ = GET_SUBPROD($\mathcal{G}_j, (z_1, \sigma_{12}, z_2)$)
  for $(q, q') \in \delta_{\mathcal{G}'_j}$ **do**
    if not VALID_EDGE($(q, q'), \sigma_{11^*}$) **then**
      $\mathcal{G}_{j,upd}.remove\_edge(q, q')$
return $\mathcal{G}_{j,upd}$

As mentioned in Sec. C.5.3, there are two types of modifications to the Büchi automaton depending on whether the first transition is a self-transition and the

second is not, or vice-versa; as a result, there are two ways to update the product automaton. In both cases, we take advantage of the fact that $\sigma_{12} \subseteq \sigma_{11^*}$ (i.e. the propositions in σ_{11^*} may have constraints on the truth values of propositions in addition to the ones in σ_{12}); rather than reconstructing robot j's product automaton $\mathcal{G}_j$ by taking the entire cross product $A_j \times \mathcal{B}_{upd}$, we modify portions of the existing $\mathcal{G}_j$ based on the changes made to $\mathcal{B}$ to get $\mathcal{G}_j^{upd}$.

Let the two transitions we checked in the original Büchi automaton $\mathcal{B}$ be $e_1 = (z_1, \sigma_{12}, z_2), e_2 = (z_2, \sigma_{23}, z_3)$, where $\sigma_{12} = (\sigma_{12}^T, \sigma_{12}^{exT}, \sigma_{12}^F, \sigma_{12}^{exF})$, $\sigma_{23} = (\sigma_{23}^T, \sigma_{23}^{exT}, \sigma_{23}^F, \sigma_{23}^{exF})$, and the set of updated transitions E_{upd} is outputted by Alg. 6.

Case 1: *e_1 is a self-transition and e_2 is not (i.e. $|E_{upd}| > 1$).* In this case, the transition e_2 has been removed and replaced with intermediate states and associated transitions in the Büchi automaton (Alg. 6).

To update the product automaton, we first remove any transitions of the product automaton $((s, z_2), (s', z_3)) \in \delta_{\mathcal{G}_j}$, since we have also removed e_2 from $\mathcal{B}_{upd}$. Then, we take the cross product only of the modified portions of each graph and add it to $\mathcal{G}_{j,upd}$.

Case 2: *e_2 is a self-transition and e_1 is not (i.e. $|E_{upd}| = 1$).* In this case, $\mathcal{B}_{upd}$ does not have any new transitions, but σ_{12} has been modified to become σ_{11^*}. Thus, we check any transition in $((s, z_1), (s', z_2)) \in \delta_{\mathcal{G}_j}$ to see if it is still valid based on $(z_1, \sigma_{11^*}, z_2)$, the updated transition in $\mathcal{B}_{upd}$.

C.5.5 Robot Team Behavior

To find a team of robots that can satisfy the task, we need to find a trace in the Büchi automaton such that every binding is assigned, with each robot maintaining a consistent set of bindings throughout the entire trace.

In our prior work, we introduced a DFS algorithm to find a trace β in the Büchi automaton, $\beta_{\hat{A}}$, a team of robots $\hat{A}$, and corresponding binding assignments $R_{\hat{A}}$ to collectively execute discrete actions. We extend this framework to continuous actions so that the robots' behaviors are guaranteed to satisfy the original specifications (Alg. 8). To do so, during the DFS, for each transition (z, σ, z') we check, we update the Büchi automaton if intermediate states are necessary (line 12). Each robot updates its product automaton, and determines which bindings it can satisfy (line 16).

We want the robot's behavior to satisfy not only the current transition, but also the entire path with a fixed binding assignment. To ensure this, each robot's assigned binding set r_j is first initialized to be the set of all possible binding assignments, and each robot removes binding assignments that are not possible with each iteration of the DFS algorithm (line 17). It does so by finding a trace through its product automaton following the current trace in the Büchi automaton, $\beta_{\hat{A}} \cup E_{upd}$. The robots also take the binding constraints (c_{single}, c_{combo}) into account, reducing the number of combinations of bindings the robots need to check (e.g. if $c_{combo} = \{1, 2\}$, then a robot assigned binding 1 must also be assigned binding 2; there is no need for the robot to check its ability to satisfy the bindings separately). If $r_j = \emptyset$ (i.e. the robot cannot be assigned any binding), we remove the robot from the team $\hat{A}$ entirely.

Algorithm 8: Find Accepting Trace for Robot Team

Input : $A = \{A_1, A_2, ..., A_n\}$, $\mathcal{B}$, $G = \{\mathcal{G}_1, \mathcal{G}_2 ..., \mathcal{G}_n\}$, AP_ψ
Output: β, δ_{self}, $\hat{A} \subseteq A$, $R_{\hat{A}}$, $G_{\hat{A}}$, $\mathcal{B}_{upd}$

$stack = \emptyset$, $visited = \emptyset$
$R = \{r_k = 2^{AP_\psi} \setminus \emptyset \mid k = \{1, 2, ..., n\}\}$
for $e \in \{(z, \sigma, z') \in \delta_{\mathcal{B}} \mid z = z_0\}$ **do**
  $stack = stack \cup \{(e, R, [e], \mathcal{B}, G, \emptyset)\}$
while $stack \neq \emptyset$ **do**
  $((z, \sigma, z'), R_{\hat{A}}, \beta_{\hat{A}}, \mathcal{B}, G_{\hat{A}}, C) = stack.pop()$
  if $(z, \sigma, z') \notin visited$ **then**
    $visited = visited \cup (z, \sigma, z')$
    // update buchi automaton with intermediate states
    if $|\beta_{\hat{A}}| > 1$ **then**
      $e_1 = \beta_{\hat{A}}[-1]$, $e_2 = (z, \sigma, z')$
      $\mathcal{B}_{upd}, E_{upd}, (c_{all}, c_{ex}) =$ UPDATE_BUCHI$(\mathcal{B}, e_1, e_2)$
    else
      $\mathcal{B}_{upd}, E_{upd}, (c_{all}, c_{ex}) = \mathcal{B}, \emptyset, (\emptyset, \emptyset)$
    for $\mathcal{G}_j \in G_{\hat{A}}$ **do**
      $\mathcal{G}_{j,upd} =$ UPDATE_PRODUCT_AUT$(\mathcal{B}_{upd}, E_{upd}, \mathcal{G}_j)$
    for $r_j \in R_{\hat{A}}$ **do**
      $r'_j =$ UPDATE_BINDINGS$(r_j, \mathcal{G}_{j,upd}, \beta_{\hat{A}} \cup E_{upd}, (c_{all}, c_{ex}))$
      if $r'_j = \emptyset$ **then**
        $R_{\hat{A}} = R_{\hat{A}} \setminus r_j$, $G_{\hat{A}} = G_{\hat{A}} \setminus \mathcal{G}_j$
      else
        $R_{\hat{A}} = (R_{\hat{A}} \setminus r_j) \cup r'_j$
    if $\bigcup_j (R_j \in R_{\hat{A}}) = AP_\psi$ **then**
      if $z' \in F$ **then**
        $\beta, \delta_{self} =$ PARSE_TRACE$(\beta_{\hat{A}})$
        return $\beta, \delta_{self}, R_{\hat{A}}, \mathcal{B}_{upd}, C \cup \{(c_{all}, c_{ex})\}$
      $E = \{(z', \sigma', z'') \in \delta_{\mathcal{B}}\}$
      for $(z', \sigma', z'') \in E$ **do**
        if $(z = z'$ and $z' \neq z'')$ or $(z \neq z'$ and $z' = z'')$ **then**
          $stack = stack \cup \{(z', \sigma', z''), R_{\hat{A}},$
          $\beta_{\hat{A}} \cup E_{upd}, \mathcal{B}_{upd}, G_{\hat{A}}, C \cup \{(c_{all}, c_{ex})\}$

c_{single} is the set of bindings assignments for which exactly one robot must be assigned, but the framework may output a teaming assignment that includes multiple robots assigned those bindings. We keep track of these binding constraints and ask the user to choose one robot for each $c \in c_{single}$.

Because we are using a DFS approach to find a viable team, the final teaming assignment may not be the globally optimal one. However, the framework is sound; it is guaranteed to find an assignment if one exists.

C.5.6 Synthesis and Execution of Control and Synchronization Policies

To guarantee that the behavior of the robots do not violate the original task, the robots must implement synchronization policies to transition to the next non-intermediate state in $\mathcal{B}_{upd}$ at the same time. The algorithm (Alg. 9) is similar to [31]; a robot j first synthesizes its behavior b_j for the entirety of the task that satisfies the collective trace in the Büchi automaton β (function FIND_BEHAVIOR, line 1). For each transition (z, σ, z') in β, the robot parses out $b_j^{zz'}$, the behavior that correlates to (z, σ, z') (function SUB_BEHAVIOR, line 3). Before it executes this behavior, it checks if it is required to participate in the synchronization step. It participates if 1) z' is not an intermediate state, 2) r_j contains a binding ρ required by σ and 3) is not the only robot assigned bindings from σ (line 8). If the robot satisfied these criteria, its binding assignment r_j is added to $\overline{R}$, which contains the binding assignments of every robot that needs to participate in the synchronization step at state z'.

Algorithm 9: Synthesize Robot j's Behavior

Input : $\mathcal{G}_j$, r_j, $R_{\hat{A}}$, β, δ_{self}

1 b_j = FIND_BEHAVIOR($\mathcal{G}_j, r_j, \beta, \delta_{self}$)
2 **for** $(z, \sigma, z') \in \beta$ **do**
3 $\quad b_j^{zz'}$ = SUB_BEHAVIOR($b_j, (z, \sigma, z')$)
4 $\quad$ **if** $z' = z^*$ **then**
5 $\quad\quad p = ()$
6 $\quad\quad$ EXECUTE($b_j^{zz'}, p$)
7 $\quad\quad$ **continue**
8 $\quad \overline{R} = \{r_k \in R_{\hat{A}} \mid r_k \cap \mathfrak{B}(\sigma) \neq \emptyset\}$
9 $\quad$ **if** $r_j \notin \overline{R}$ or $\overline{R} = \{r_j\}$ **then**
10 $\quad\quad p = ()$
11 $\quad\quad$ EXECUTE(b_j, p)
12 $\quad$ **else**
13 $\quad\quad p = (j, z', 0)$, $\ell = length(b_j)$
14 $\quad\quad$ EXECUTE($b_j^{zz'}[1 : \ell - 1], p$)
15 $\quad\quad z_{wait} = b_j[\ell - 1]$, $P = \{j\}$
16 $\quad\quad$ **while** $\bigcup_{i \in P}(r_i \in \overline{R}) \neq \mathfrak{B}(\sigma)$ **do**
17 $\quad\quad\quad p = (j, z', 1)$
18 $\quad\quad\quad$ EXECUTE(z_{wait}, p)
19 $\quad\quad\quad P = j \cup \{k \mid (k, z', 1) \in$ RECEIVE$()\}$
20 $\quad\quad \Phi = (j, z', 0)$
21 $\quad\quad$ EXECUTE_NONINST($b_j^{zz'}[\ell], \Phi$)
22 $\quad\quad$ EXECUTE_INST($b_j^{zz'}[\ell], ()$)

One difference of this algorithm compared to our previous approach [31] is that we do not need to execute the synchronization policy if z' is an intermediate state (Alg. 9. lines (4 - 7)), since the synchronization is only required on the transitions of the original Büchi automaton.

The other key difference is in lines 20 - 22 of Alg. 9. Because instantaneous actions may need to synchronize when another action finishes (e.g. a robot must take a camera as soon as another robot enters room B), robots also need to communicate when all of their non-instantaneous actions finish executing (e.g. when *roomBc* is true) by executing the function EXECUTE_NONINST (line 20). As

soon as all the non-instantaneous actions across the robots in $\overline{R}$ are complete, the robots execute their instantaneous actions (function EXECUTE_INST, line 22).

The waiting state for each robot z_{wait} is the penultimate state in its behavior, $b_j[\ell - 1]$. Each robot stays in z_{wait} until all the other robots in $\overline{R}$ are also in their waiting states. Then, the robots execute the last step in their behavior, $b_j[\ell]$, synchronously.

C.6 Results

C.6.1 Demonstrations

We demonstrate our framework through two different collaborative high-level tasks, executed on physical hardware. We use 4 robots - two Hello Robot Stretch RE1, one Hello Robot Stretch 2, and a Clearpath Boxer with a Kinova Gen3 arm. We use an Optitrack motion capture system to track the robots and the environment. The setup is shown in Fig. C.5. Their capabilities are the same as those listed in Sec. C.4:

$\Lambda_{green} = \{\lambda_{mot}, \lambda_{arm}, \lambda_{camera}\}, L(s_0) = \{roomD_c\}$

$\Lambda_{blue} = \{\lambda_{mot}, \lambda_{beep}\}, L(s_0) = \{roomC_c\}$

$\Lambda_{orange} = \{\lambda_{mot}, \lambda_{arm}\}, L(s_0) = \{roomE_c\}$

$\Lambda_{pink} = \{\lambda_{mot}, \lambda_{beep}, \lambda_{cam}, \lambda_{scan}\}, L(s_0) = \{roomE_c\}$

where the green and pink robots are Hello Robot Stretch RE1s, the blue robot is a Hello Robot Stretch 2, and the orange robot is the Clearpath Boxer with a Kinova Gen3 arm. Although all the Hello Robot Stretch robots have arms, we treat the

blue and pink robots as not having those capabilities. The demonstrations of the full behavior is shown in the accompanying video[1].

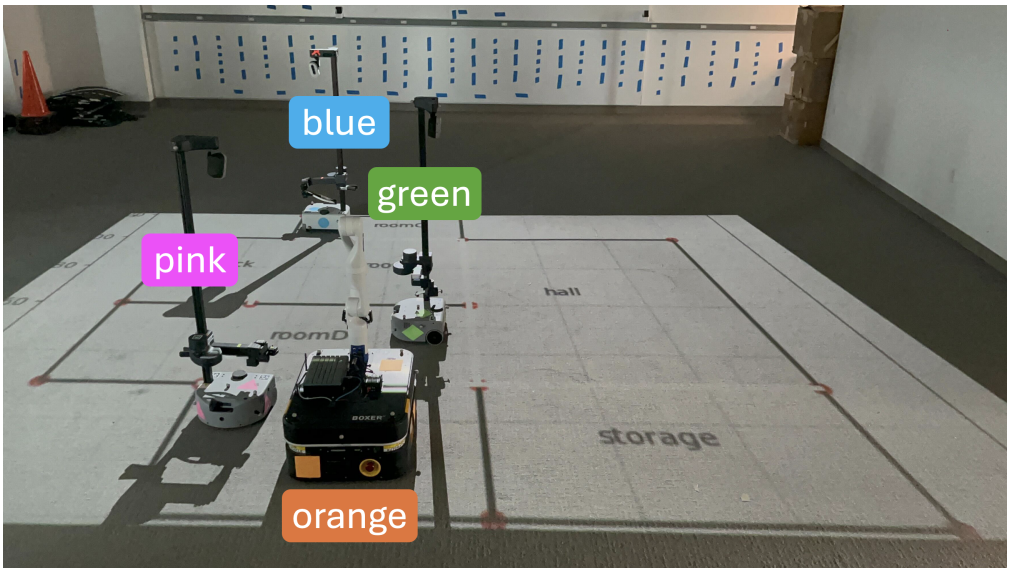


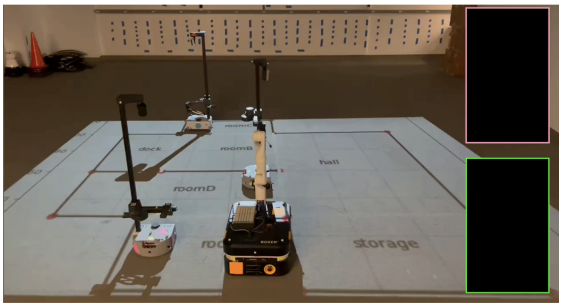
Figure C.5: Initial setup on the physical system

C.6.2 Example: Task 1

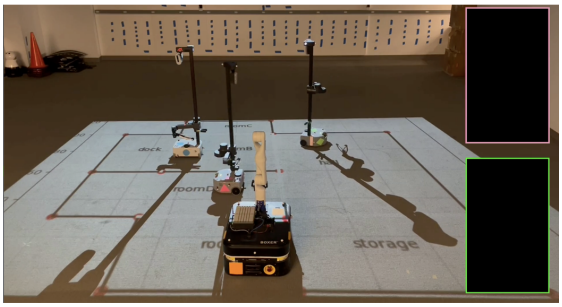
We implement our framework for the team of robots to satisfy the task defined in Eq. C.4. The team created by the framework is $\hat{A} = \{A_{green}, A_{blue}, A_{orange}, A_{pink}\}$ with binding assignments $r_{green} = \{2, 3\}$, $r_{blue} = \{1\}, r_{orange} = \{1\}, r_{pink} = \{2, 3\}$ with the constraints $c_{all} = \{\{1\}\}$. In this case, exactly one robot can be assigned binding 1, so the user chooses between A_{blue} and A_{orange} to be part of the team. If we choose between the two based on minimizing cost, the final team is $\hat{A} = \{A_{green}, A_{pink}, A_{blue}\}$.

Fig. C.6 provides key frames in the robots' execution of the task. The boxes on the right in each frame represents the pink and green robots' cameras. We

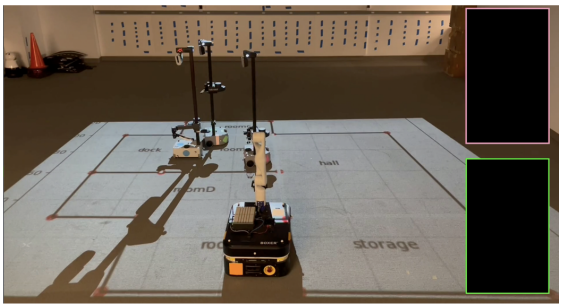
[1] https://youtu.be/DiWjqXHQjmI



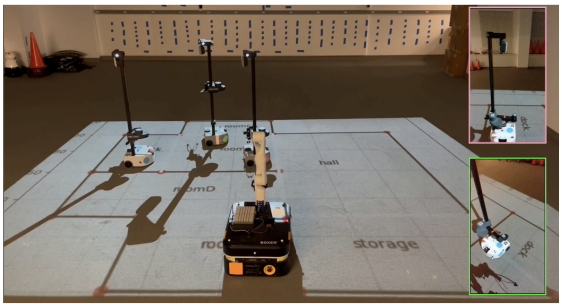
(a)



(b)



(c)



(d)

Figure C.6: Robots executing task 1. The boxes on the right in each frame represents the pink and green robots' cameras.

can see that the blue robot waits outside the dock area while the pink and green robots make their way to roomB (Fig. C.6b). After they are in roomB, the blue

robot moves to the dock area (Fig. C.6c); as soon as the blue robot enters that room, the other robots immediately execute the instantaneous action of taking a picture (Fig. C.6d).

C.6.3 Example: Task 2

For this task, we use the same environment with three available robots, A_{green}, A_{orange}, and A_{pink}. Their capabilities and initial positions are the same as in the previous example.

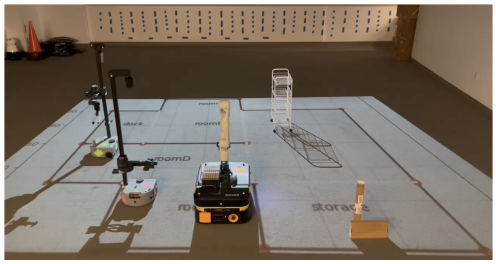
The task is $\varphi^{\psi} = \varphi_1^{\psi} \wedge \varphi_2^{\psi}$, where

$$\varphi_1^{\psi} = \Diamond((beep \wedge dock_c)^{1 \vee 2} \wedge \Diamond(pickup \wedge storage_c)^{1}) \tag{C.13a}$$

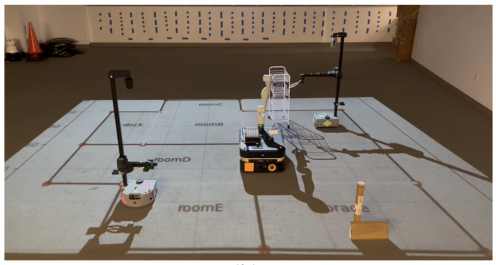
$$\varphi_2^{\psi} = \neg roomB_c^{1 \wedge 2 \wedge 3} \, \mathcal{U} \, (push_c \wedge hall_c)^{3} \tag{C.13b}$$

φ_1^{ψ} captures "robot(s) with bindings 1 or 2 should beep at the dock, then robot(s) with binding 1 should pickup packages at the storage. φ_2^{ψ} enforces "none of the robots can be in room B until the cart in the hallway is pushed out of the way".

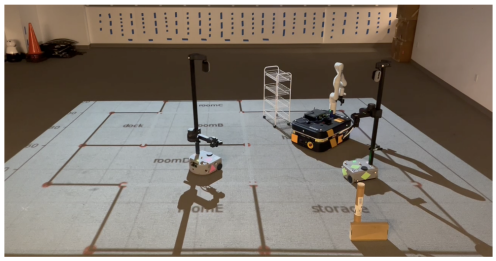
In this example, $push_c$, $pickup_c$, and the location completion propositions make up the set of $AP_{non-inst}$, while $beep$ is an instantaneous action. To guarantee the robots do not violate φ_2^{ψ}, any robot assigned binding 3 must move into the hallway first before any robots can be in room B. If we assume discrete execution of actions, this constraint is not necessary; all the robots can move into the respective areas simultaneously.



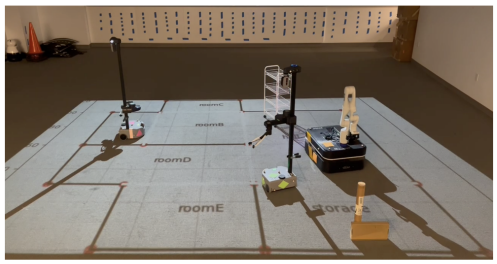
(a)



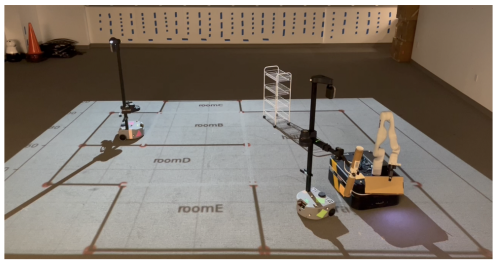
(b)



(c)



(d)



(e)

Figure C.7: Robots executing task 2

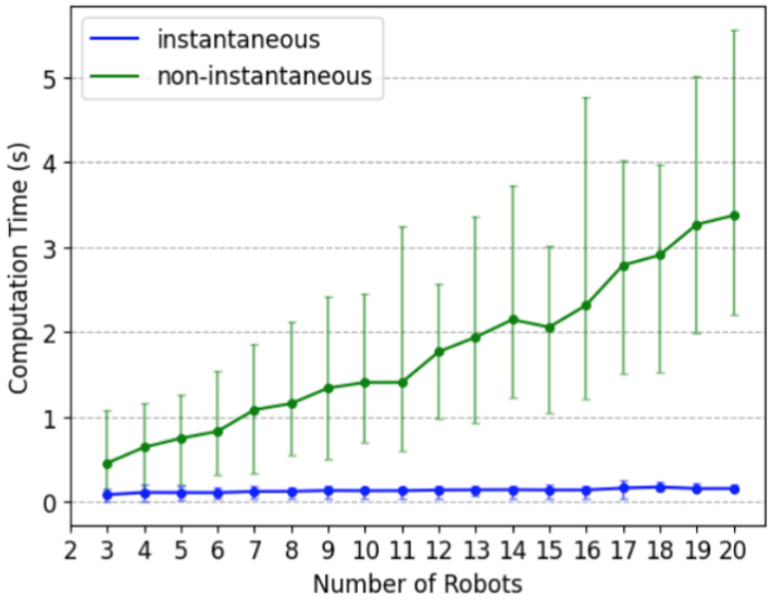
The final team is $\hat{A} = \{A_{green}, A_{orange}, A_{pink}\}$ with binding assignments $r_{green} = \{1, 3\}$, $r_{orange} = \{1, 3\}, r_{pink} = \{2\}$. Snapshots of their behavior is shown in Fig. C.7. We can see that the green and orange robots first move into the hall (Fig. C.7b) to push the cart before the pink robot passes through roomB (Fig. C.7c) to make its way to the dock. The pink robot beeps immediately as it enter the dock (Fig. C.7d). Then, the green and orange robots move to pickup their respective packages at the dock and synchronize the pickup action (Fig. C.7e). We assume the robots have low-level controllers to coordinate executing the *push* and *pickup* actions. The robots begin pushing the cart together when they have both grabbed it. The action *pickup* includes reorienting, reaching, and grasping the packages; the robots begin executing this at the same time (i.e. they begin reorienting simultaneously).

C.6.4 Computational Performance

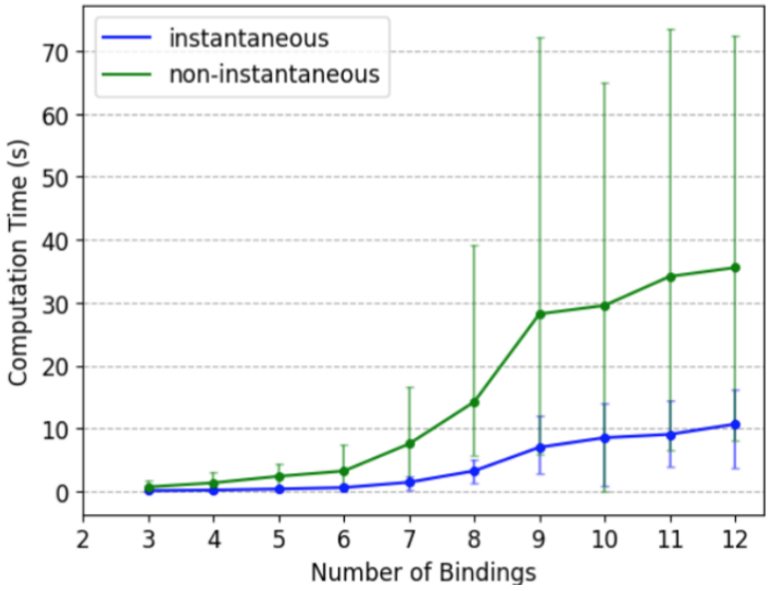
We compare the computation performance (without significant code optimization) of the approach for synthesizing a team of robots and their corresponding binding assignments for instantaneous versus non-instantaneous actions.

Effects of number of robots: We analyze the effect of increasing the number of robots while keeping the task specification the same (task 1, Eq. C.4). This task requires the generation of intermediate states when considering non-instantaneous actions.

The instantaneous approach involves decentralized pre-processing for each robot j's possible combination of binding assignments for each transition in the Büchi automaton before conducting the DFS algorithm to find an overall team-



(a)



(b)

Figure C.8: Comparing computation times between the instantaneous and non-instantaneous actions frameworks as the number of robots increases (a) and the number of bindings increases (b). The error bars represent min/max values.

ing plan. As a result, this framework is largely agnostic to the number of robots.

However, the approach for non-instantaneous actions is centralized, as the DFS

algorithm may need to update each robot's product automaton during each iteration. As a result, while both framework's scale linearly with the number of robots, there is a larger impact on the computational performance of the framework under the assumption of non-instantaneous actions.

Effects of number of bindings: We consider the effect of increasing the number of bindings while keeping the number of robots fixed at 4, but randomizing their capabilities with each simulation. We add more bindings to the task specification through conjunction. By doing so, we increase the number of bindings, but keep the number of edges in the Büchi automaton the same.

In both the instantaneous and non-instantaneous actions frameworks, we need to store all possible binding assignments for the current team of robots as we search for a possible trace in the Büchi automaton. As a result, both the space and time complexity for each approach are exponential with the number of bindings. The non-instantaneous actions framework scales worse with the number of bindings because we determine the possible binding assignments for each robot with every transition in the DFS; in the instantaneous action framework, each robot had performed this computation in a decentralized manner before conducting the DFS algorithm.

C.7 Conclusion

We introduced a method for control synthesis for a heterogeneous multi-robot system to satisfy collaborative tasks in continuous time, where actions may take varying duration of time to complete. We presented a synthesis approach to automatically generate a teaming assignment and corresponding discrete behav-

ior that is correct-by-construction for continuous execution and ensured collaborative portions of the task are satisfied. We implemented our approach on a physical multi-robot system in two warehouse scenarios.

In the future, we plan to explore different notions of optimality when finding a teaming plan, as well as implement methods for robots to respond to capability failures or modifications in real time.

APPENDIX D

ONLINE RESYNTHESIS OF HIGH-LEVEL COLLABORATIVE TASKS FOR ROBOTS WITH CHANGING CAPABILITIES

D.1 Introduction

There is a wealth of literature in planning for multi-robot systems due to its wide variety of applications, such as search and and rescue and warehouse automation. Recently, there has been a growing interest in using formal logic, such as Linear Temporal Logic (LTL), to capture multi-robot tasks with temporally extended tasks, such as surveillance and coverage, in a mathematically precise way (e.g. [18,51,56,57,101]).

During execution, robots may encounter changes to themselves and to their environment; they may experience failures (e.g. a broken gripper) that limit what they can do, or they may acquire additional capabilities, such as a change in the environment (e.g. a new opening) that may allow them to reach previously unreachable areas. When considering multi-robot collaborative behavior, a single robot modification may affect the ability of the overall team to accomplish the task; as a result, other robots' behavior may need to change at runtime in order to successfully accomplish the task.

To account for such changes during execution, we extend our framework from [31,32] that both automatically assigns robots to the task, as well as synthesizes high-level robot behaviors to satisfy a global task encoded in LTL$^{\psi}$. Here we propose a method for the team to autonomously adapt when robot capabilities change in the middle of execution while guaranteeing that the team is

still able to satisfy the task. We aim to minimize the change in the original team assignment and behavior and only locally resynthesize a robot's behavior when possible; our approach only reconstructs the entire team when necessary.

In addition to online resynthesis, we increase the expressivity of LTL$^{\psi}$ to allow users to provide information on 1) the minimum number of robots that must be assigned to a specific subtask (captured through the notion of a *binding*), and 2) which subtasks cannot be assigned to the same robot.

Related Work: Existing work have proposed methods to synthesize behavior for homogeneous multi-robot teams to satisfy temporal logic specifications [56, 57, 119]. For heterogeneous robots, the common approaches are either to decompose the global task into independent sub-tasks [33, 89], or *a priori* task assignment [100, 104]. Approaches for heterogeneous teams to satisfy a global task include [64, 73, 86]. The task is not explicitly decomposed; rather, portions of the task are assigned to robots based on their type or onboard capabilities. In our prior work [31], we proposed an extension of LTL, called LTL$^{\psi}$, in which a user can encode information about the relationships between actions and robots (e.g. the same robot that picks up a package must also drop it off). The synthesis framework was then extended in [32] for LTL$^{\psi}$ tasks to account for actions that take varying time duration to execute. In the aforementioned work, the task allocation happens offline prior to execution; resynthesis during execution is not considered.

In prior work [30], we considered resynthesis in scenarios where robots are already executing existing LTL tasks when new tasks are introduced. The distributed framework allows robots to resynthesize during execution such that they can interleave both tasks rather than perform them sequentially. Another

application in which resynthesis is critical is in partially known or uncertain workspaces. In [42], robots revise their motion plan in real-time. The robots synthesize a preliminary motion plan, then iteratively revise the plan as the robot receives more information about its environment. [54] considers online revisions to robot plans for tackling reach-avoid problems encoded in LTL, specifically when environment is dynamic and uncertain. There also exists work that addresses the issue of robustness. For example, [101] generates plans online that are robust to timing errors. The work in [68] introduces risk predicates to synthesize behavior that reduces the amount of risk in violating spatial temporal logic specifications.

To address resynthesis specifically due to robot failures, the approach in [123] first decomposes the global specification into independent sub-tasks, characterizes the disturbances into four types of failures, and uses a behavior tree to autonomously react to those failures. These failures are specific to quadruped and wheeled robots; they do not generalize to any type of robot. In [50], the authors consider a homogeneous team of robots executing a navigation task encoded in co-safe LTL. The user specifies at most how many robots can fail, and whenever a robot fails, the centralized planner updates the global plan. The authors in [54] consider failures in capabilities, where each capability is a binary variable (either the robot has the capability or it does not). Our work considers a more granular level of failure in which a failure happens within a capability. A robot may no longer be able to execute specific actions within the capability (e.g. picking up an object with a robot manipulator), but other actions can still be executed (e.g. pushing an object with a robot manipulator). This approach allows us to address a broader range of potential failures that a may occur to a robot. In addition, we can consider other types of modifications to a robot's capability,

such as a gripper being added during runtime or a change in the environment that increases the robot's reachable workspace.

Contributions: In the context of synthesizing team and robot control from a high-level specification given in LTL$^{\psi}$, we 1) increase the specification expressivity by allowing the user to provide constraints regarding the binding assignments (i.e. the minimum number of robots assigned to each binding and which bindings are allowed to be assigned together), and 2) propose a resynthesis framework for online adaptation to changes in robot capabilities. We demonstrate our approach in a simulated warehouse scenario.

D.2 Task Grammar: LTL$^{\psi}$

We use LTL$^{\psi}$ [31, 32] as the grammar for writing high-level collaborative tasks. In this work, we extend the grammar to allow additional constraints on the team composition. The task grammar for LTL$^{\psi}$is defined over atomic propositions that abstract robot actions, as well as bindings that relate actions to specific robots; any action associated with a given binding must be satisfied by all the robot(s) assigned that binding. Note that a robot may be assigned to multiple bindings, and a binding may be assigned to multiple robots.

An LTL$^{\psi}$ specification φ^{ψ} is defined recursively as:

$$\psi := \rho \mid \psi_1 \vee \psi_2 \mid \psi_1 \wedge \psi_2 \tag{D.1}$$

$$\varphi := \pi \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \,\mathcal{U}\, \varphi \tag{D.2}$$

$$\varphi^{\psi} := \varphi^{\psi} \mid \neg(\varphi^{\psi}) \mid \varphi_1^{\psi_1} \wedge \varphi_2^{\psi_2} \mid \varphi_1^{\psi_1} \vee \varphi_2^{\psi_2} \mid \varphi_1^{\psi_1} \mathcal{U} \varphi_2^{\psi_2} \mid \Box\varphi^{\psi} \tag{D.3}$$

where ψ, the *binding formula*, is a Boolean formula (excluding negation) over

the binding propositions $\rho \in AP_\psi$, and φ is defined over the action propositions $\pi \in AP_\varphi$.

In this work, we extend the expressivity of LTL$^\psi$ by also defining the semantics over two types of binding constraints the user can now specify: 1) $c_{distinct}$, where $c \in c_{distinct}$ are sets of two or more bindings that cannot be allocated to the same robot (e.g. $\{I, II\} \in c_{distinct}$ enforces "a robot cannot be assigned both bindings I and II"), and 2) the set c_{min}, which contains the tuples (ρ, k); this enforces that at least k robots must be assigned binding ρ.

Semantics: The semantics of an LTL$^\psi$ formula φ^ψ are defined over 1) a team trace $\sigma = \sigma_1\sigma_2 \ldots \sigma_n$, where σ_j is the trace of robot j such that $\sigma_j(i)$ is the set of atomic propositions AP_φ that are true for robot j at time step i; and 2) the team binding assignments $\mathcal{R} = \{r_1, r_2, \ldots, r_n\}$, where $r_j \in \mathcal{R}$ is the set of bindings in AP_ψ that are assigned to robot j. For example, $r_{green} = \{I\}, r_{blue} = \{I, II\}$ indicates that green robot is assigned binding I, and the blue robot is assigned bindings I and II. We also define the function $\zeta : \psi \rightarrow 2^{2^{AP_\psi}}$, which outputs all possible combinations of $\rho \in AP_\psi$ that satisfy ψ. For example, $\zeta(I \vee (II \wedge III)) = \{\{I\}, \{II, III\}, \{I, II, III\}\}$.

Given $c_{distinct}$ and c_{min}, and given the semantics of LTL [4] we define the semantics of LTL$^\psi$ as follows:

- $(\sigma(i), \mathcal{R}) \models \varphi^\psi$ iff $\exists K \in \zeta(\psi)$ s.t. $(K \subseteq \bigcup_{p=1}^{n} r_p)$
  and $(\forall j$ s.t. $K \cap r_j \neq \emptyset, \sigma_j(i) \models \varphi)$
  and $(\forall r_j \in \mathcal{R}, \forall c \in c_{distinct},\ c \nsubseteq r_j)$
  and $(\forall (\rho, k) \in c_{min},\ |\{r_j \in \mathcal{R} \mid \rho \in r_j\}| \geq k)$

- $(\sigma(i), \mathcal{R}) \models (\neg\varphi)^\psi$ iff $\exists K \in \zeta(\psi)$ s.t. $(K \subseteq \bigcup_{p=1}^{n} r_p)$
  and $(\forall j$ s.t. $K \cap r_j \neq \emptyset, \sigma_j(i) \not\models \varphi)$

and $(\forall r_j \in \mathcal{R}, \forall c \in c_{distinct},\ c \nsubseteq r_j)$

and $(\forall (\rho, k) \in c_{min},\ |\{r_j \in \mathcal{R} \mid \rho \in r_j\}| \geq k)$

- $(\sigma(i), \mathcal{R}) \models \neg(\varphi^{\psi})$ iff $\exists K \in \zeta(\psi)$ s.t. $(K \subseteq \bigcup_{p=1}^{n} r_p)$
  and $(\exists j$ s.t. $K \cap r_j \neq \emptyset, \sigma_j(i) \not\models \varphi)$
  and $(\forall r_j \in \mathcal{R}, \forall c \in c_{distinct},\ c \nsubseteq r_j)$
  and $(\forall (\rho, k) \in c_{min},\ |\{r_j \in \mathcal{R} \mid \rho \in r_j\}| \geq k)$

- $(\sigma(i),\mathcal{R}) \models \varphi_1^{\psi_1} \wedge \varphi_2^{\psi_2}$ iff $(\sigma(i),\mathcal{R}) \models \varphi_1^{\psi_1}$ and $(\sigma(i),\mathcal{R}) \models \varphi_2^{\psi_2}$

- $(\sigma(i),\mathcal{R}) \models \varphi_1^{\psi_1} \vee \varphi_2^{\psi_2}$ iff $(\sigma(i),\mathcal{R}) \models \varphi_1^{\psi_1}$ or $(\sigma(i), \mathcal{R}) \models \varphi_2^{\psi_2}$

- $(\sigma(i), \mathcal{R}) \models \varphi_1^{\psi_1}\ \mathcal{U}\ \varphi_2^{\psi_2}$ iff $\exists \ell \geq i$ s.t. $(\sigma(\ell), \mathcal{R}) \models \varphi_2^{\psi_2}$ and $\forall i \leq k < \ell, (\sigma(k), \mathcal{R}) \models \varphi_1^{\psi_1}$

- $(\sigma(i), \mathcal{R}) \models \Box\varphi^{\psi}$ iff $\forall \ell > i, (\sigma(\ell), \mathcal{R}) \models \varphi^{\psi}$

Intuitively, a team of robots satisfies the formula φ^{ψ} if and only if the following conditions hold: 1) there exists a set of bindings $K \in \zeta(\psi)$ for which all the bindings are assigned to (at least one) robot; 2) for all robots assigned these bindings, their traces satisfy φ [4]; 3) none of the robots are assigned any combinations of bindings $c \in c_{distinct}$; any binding not appearing in $c_{distinct}$ must still be assigned to at least one robot; and 4) at least k number of robots are assigned binding ρ for every $(\rho, k) \in c_{min}$.

Example:

$$\begin{aligned} \varphi^{\psi} = {} & \Diamond(beep \wedge storage_c)^{III} \wedge \Diamond dock_c^I \\ & \wedge \Box(dock_c^I \rightarrow (roomB_c \wedge camera)^{II}) \\ & c_{distinct} = \emptyset,\ c_{min} = \{(I, 2)\} \end{aligned} \tag{D.4}$$

In English, the task captures "all robots assigned binding *III* must go to the storage room and beep, and all robots assigned binding *I* must eventually go to the dock. Anytime all the robots assigned binding *I* are in the dock, all robots assigned binding *II* must be in room B and taking a picture. At least two robots must be assigned binding *I*."

D.3 Prior work - Robot Model and Büchi Automaton

D.3.1 Robot Model

Each robot j is modeled according to its set of capabilities, $\Lambda_j = \{\lambda_1, \ldots, \lambda_k\}$ [30]. Each capability is a transition system $\lambda = (X, x_0, AP, \Delta, \mathcal{L}, \mathcal{W})$, where X is a set of states, $x_0 \in X$ is the initial state, AP is the set of atomic propositions that are an abstraction of the actions the capability can execute, $\Delta \subseteq X \times X$ is a transition relation, $\mathcal{L} : X \to 2^{AP}$ is the labeling function, and $\mathcal{W} : \Delta \to \mathbb{R}_{\geq 0}$ is the cost function; each transition is assigned a weight using $\mathcal{W}$.

A robot model A_j [30] is the product of its capabilities: $A_j = \lambda_1 \times \ldots \times \lambda_k$ such that $A_j = (S, s_0, AP_j, \gamma, L, W)$. $S = X_1 \times \ldots \times X_k$ is the finite set of states, $s_0 \in S$ is the initial state, $AP_j = \bigcup_{i=1}^{k} AP_i$ is the set of propositions, $\gamma \subseteq S \times S$ is the transition relation, $L : S \to 2^{AP_j}$ is the labeling function, and $W : \gamma \to \mathbb{R}_{\geq 0}$ is the cost function. The constraints of the workspace the robot operates in are encoded in its motion capability.

D.3.2 Büchi Automaton for an LTL$^{\psi}$ Formula

An LTL formula φ can be translated into a Nondeterministic Büchi automaton $\mathcal{B} = (Z, z_0, \Sigma_{\mathcal{B}}, \delta_{\mathcal{B}}, F)$, where Z is the set of states, $z_0 \in Z$ is the initial state, $\Sigma_{\mathcal{B}}$ is the input alphabet, $\delta_{\mathcal{B}} : Z \times \Sigma_{\mathcal{B}} \times Z$ is the transition relation, and $F \subseteq Z$ is the set of accepting states. An infinite run of $\mathcal{B}$ over a word $\sigma = \sigma_1\sigma_2\sigma_3 \ldots \in \Sigma_{\mathcal{B}}$ is an infinite sequence of states $\mathcal{Z} = z_0 z_1 z_2 \ldots$ such that $(z_{i-1}, \sigma_i, z_i) \in \delta_{\mathcal{B}}$. A run is accepting if and only if an accepting state or set of accepting states appear infinitely often in $\mathcal{Z}$, i.e. $\text{Inf}(\mathcal{Z}) \cap F \neq \emptyset$ [4].

When creating a Büchi automaton for an LTL$^{\psi}$ formula, we first rewrite the formula to include only propositions of the form π^{ρ} [31]; then $\Sigma_{\mathcal{B}} = 2^{AP_{\varphi}^{\psi}} \times 2^{AP_{\varphi}^{\psi}} \times 2^{AP_{\varphi}^{\psi}} \times 2^{AP_{\varphi}^{\psi}}$ and $\sigma = (\sigma^T, \sigma^{exT}, \sigma^F, \sigma^{exF}) \in \Sigma_{\mathcal{B}}$. σ^T and σ^F are the sets of propositions π^{ρ} that are true/false for all robots; σ^{exT} and σ^{exF} are the sets of propositions π^{ρ} that are true/false for at least one robot. The set of $\sigma^T \cup \sigma^F$ are denoted as *for all* propositions, and $\sigma^{exT} \cup \sigma^{exF}$ as *there exists* propositions.

D.4 Behavior Synthesis

To synthesize robot behavior, we take the product of the robot model and the Büchi automaton and find a satisfying trace [32].

Definition 11 (Capability Function)**.** $\mathfrak{C} : \Sigma_{\mathcal{B}} \times AP_{\psi} \to 2^{AP_{\varphi}} \times 2^{AP_{\varphi}} \times 2^{AP_{\varphi}} \times 2^{AP_{\varphi}}$ *such that for* $(\sigma^T, \sigma^{exT}, \sigma^F, \sigma^{exF}) \in \Sigma_{\mathcal{B}}, \rho \in AP_{\psi}$, $\mathfrak{C}(\sigma, \rho) = (C_T, C_{exT}, C_F, C_{exF})$, *where for* $k \in \{T, exT, F, exF\}$, $C_k = \{\pi \in AP_{\varphi} \mid \exists \pi^{\rho} \in \sigma^k\}$.

Given a binding ρ, C_T and C_F are the sets of propositions π in which π^{ρ} is a *for all* proposition that is True/False and appear with binding ρ in label σ of a

Büchi transition; C_{exT} and C_{exF} are defined similarly for *there exists* propositions with binding ρ. For example, the transition between states 3 and 0 of the Büchi automaton in Fig. D.3 is $\sigma = \{\{roomB_c^{II}, camera^{II}, dock_c^I\}, \emptyset, \emptyset, \emptyset\}$. Then, $\mathfrak{C}(\sigma, II) = (\{roomB_c, camera\}, \emptyset, \emptyset, \emptyset)$.

We modify the following definition from [32] to account for the user-specified constraint $c_{distinct}$:

Definition 12 (Binding Assignment Function)**.** *Given two states in the robot model, s and s', and $\sigma = (\sigma^T, \sigma^{exT}, \sigma^F, \sigma^{exF})$, $\Re(s, \sigma, s') = \{r \in 2^{AP_\psi} \setminus \emptyset \mid \forall c \in c_{distinct}, c \nsubseteq r$ and $\forall \rho \in r, \mathfrak{C}(\sigma, \rho) = (C_T, C_{exT}, C_F, C_{exF}), \bigcup_{\rho \in r}(C_T \cup C_{exT}) \subseteq L(s')$ and $\bigcup_{\rho \in r}(C_F \cup C_{exF}) \cap L(s') = \emptyset\}$.*

The output of function $\Re$ is the set of all combinations of binding propositions that can be assigned to a robot over a given transition σ in the Büchi automaton. A robot can be assigned a set of binding propositions r if and only if the following are satisfied: 1) r is not a superset of any set in $c_{distinct}$, which are combinations of bindings that cannot be assigned to the same robot; 2) for all $\rho \in r$, all propositions π that are in $\sigma^T \cup \sigma^{exT}$ as π^ρ also appear in the state label of the next state s'; and 3) for all $\rho \in r$, none of the propositions π that appear in $\sigma^F \cup \sigma^{exF}$ as π^ρ also appear in the state label of s'.

To synthesize behavior for a robot [32], we find the minimum cost accepting trace in its product automaton $\mathcal{G}_j = A_j \times \mathcal{B} = (Q, q_0, AP_j, \delta_\mathcal{G}, L_\mathcal{G}, W_\mathcal{G}, F_\mathcal{G})$, where

- $Q = S \times Z$ is a finite set of states
- $q_0 = (s_0, z_0) \in Q$ is the initial state
- $\delta_\mathcal{G} \subseteq Q \times Q$ is the transition relation, where for $q = (s, z)$ and $q' = (s', z')$,

$(q, q') \in \delta_{\mathcal{G}}$ if and only if $(s, s') \in \gamma$ and $\exists \sigma \in \Sigma_{\mathcal{B}}$ such that $(z, \sigma, z') \in \delta_{\mathcal{B}}$ and $\Re(q, \sigma, q') \neq \emptyset$

- $L_{\mathcal{G}}$ is the labeling function s.t. for $q = (s, z)$, $L_{\mathcal{G}}(q) = L(s) \subseteq AP_j$
- $W_{\mathcal{G}} : \delta_{\mathcal{G}} \rightarrow \mathbb{R}_{\geq 0}$ is the cost function s.t. for $(q, q') \in \delta_{\mathcal{G}}$, $q = (s, z)$, $q' = (s', z')$, $W_{\mathcal{G}}((q, q')) = W((s, s'))$
- $F_{\mathcal{G}} = S \times F$ is the set of accepting states

If a team of robots and their synthesized behavior follow the same trace in the Büchi automaton $\mathcal{B}$ to an accepting cycle, the team is guaranteed to satisfy the task. We denote such a collective trace as β.

D.5 Problem Setup

D.5.1 Modifications

We define a robot capability modification as a change in the capability's transition relation Δ. Specifically, this involves either adding new transitions to Δ or removing existing ones. For robot m, we define Δ_m^{add} and Δ_m^{rem} where $\Delta_m^{add} = \{\Delta_{m,a}^{add}, \Delta_{m,b}^{add}, \ldots\}$, $\Delta_{m,\alpha}^{add}$ is the set of transitions added to capability λ_α; $\Delta_m^{rem} = \{\Delta_{m,a}^{rem}, \Delta_{m,b}^{rem}, \ldots\}$, $\Delta_{m,\alpha}^{rem}$ is the set of transitions that are removed from capability λ_α. The cost function of a capability $\mathcal{W}$ may also change, particularly if transitions are added. We represent the corresponding set of cost functions as $\mathcal{W}_m^{add}$ and $\mathcal{W}_m^{rem}$. Fig. D.1 shows examples of modifications to a robot's motion capability λ_{mot}.

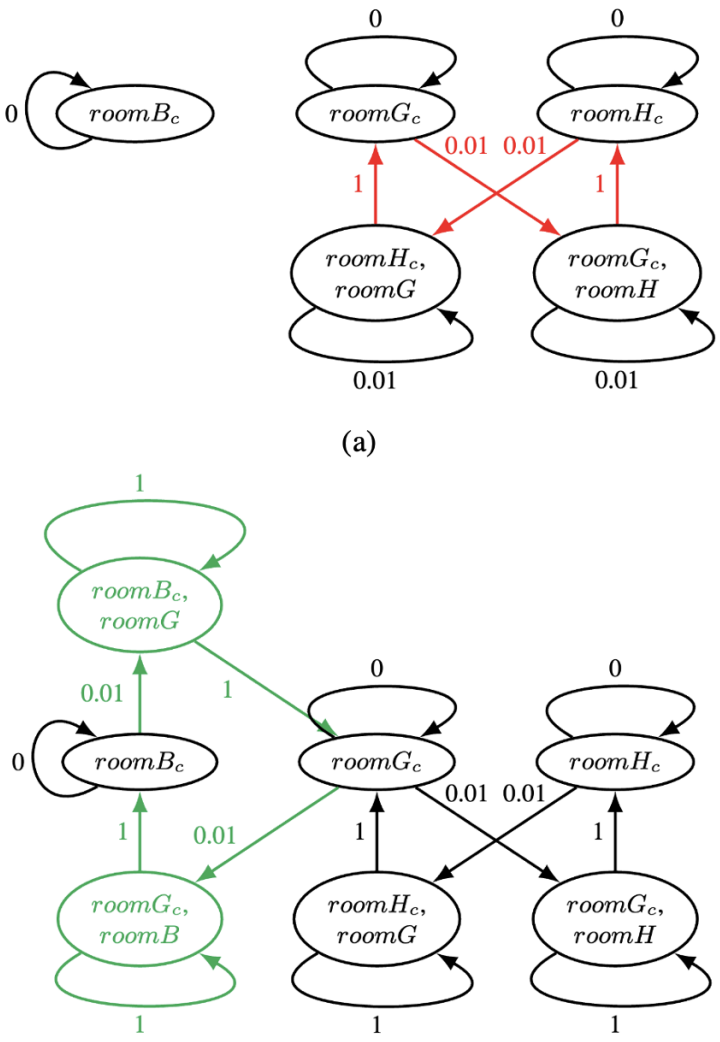


Figure D.1: Modifications to λ_{mot} in which (a) the robot can no longer move between rooms G and H (Δ^{rem}_{mot} is the set of red transitions), and (b) the robot can now move between rooms B and G (Δ^{add}_{mot} is the set of green transitions).

D.5.2 Problem Statement

Given a team of heterogeneous robots A executing φ^{ψ} with binding assignments $\mathcal{R}_A$, and given the sets of capability modifications to robot m, $\Delta^{add}_m, \Delta^{rem}_m$, and the corresponding set of cost functions $\mathcal{W}^{add}_m, \mathcal{W}^{rem}_m$, find a (possibly) new assignment $\mathcal{R}^{mod}_A$ and trace σ^{mod} such that $(\sigma^{mod}, \mathcal{R}^{mod}_A) \models \varphi^{\psi}$.

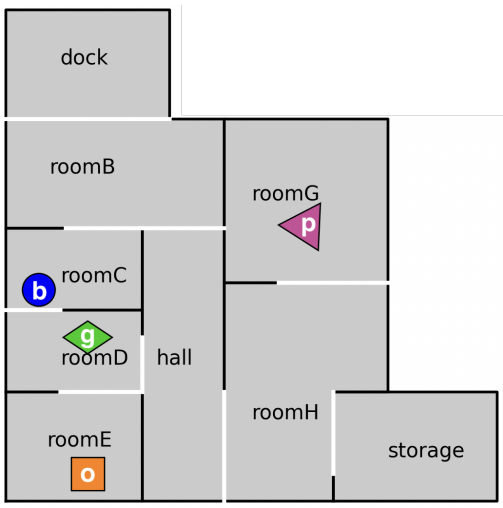


Figure D.2: Environment and robot setup

We assume that at any instance, only one robot is modified. We also assume each robot is aware of its modifications when they happen, and that robots have all-to-all communication with one another; this is to facilitate the binding reallocation and synchronization processes.

D.5.3 Example

Consider a team of robots $A = \{A_{green}, A_{blue}, A_{orange}, A_{pink}\}$ in a warehouse environment shown in Fig. D.2. The robots' capabilities and labels on their initial state are:

$$\Lambda_{green} = \{\lambda_{mot}, \lambda_{camera}\} \qquad L(s_0) = \{roomD_c\}$$

$$\Lambda_{blue} = \{\lambda_{mot}, \lambda_{beep}\} \qquad L(s_0) = \{roomC_c\}$$

$$\Lambda_{orange} = \{\lambda_{mot}\} \qquad L(s_0) = \{roomE_c\}$$

$$\Lambda_{pink} = \{\lambda_{mot}, \lambda_{beep}, \lambda_{cam}, \lambda_{scan}\} \qquad L(s_0) = \{roomG_c\}$$

The robots are currently executing the task in Eq. D.4 with binding assign-

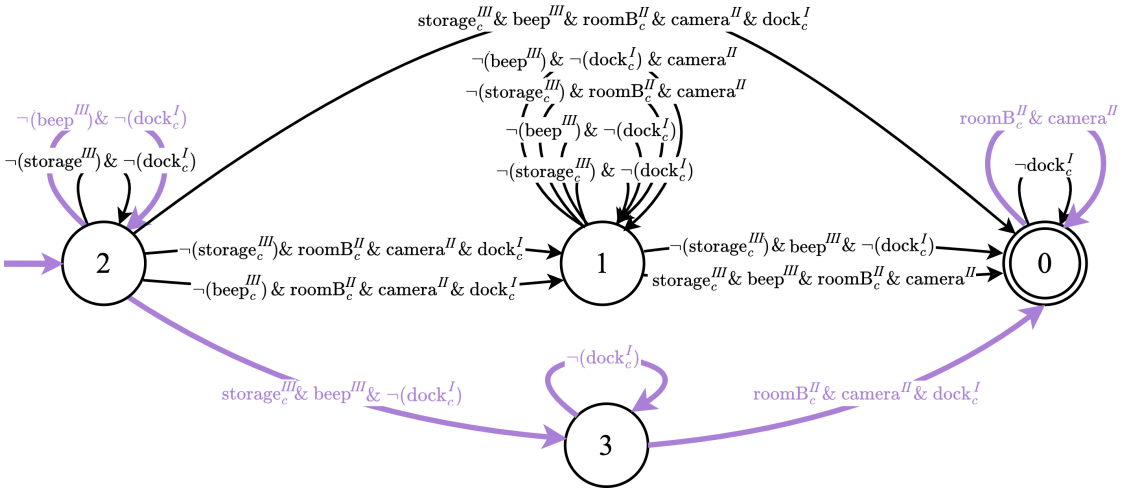


Figure D.3: Buchi Automaton $\mathcal{B}$ for the example in Sec. D.5.3. The highlighted transitions is β, the trace that the team of robots are collectively traversing to satisfy the task.

ments $\mathcal{R}_A = \{r_{green}, r_{blue}, r_{orange}, r_{pink}\}$, where $r_{green} = \{I\}$, $r_{blue} = \{I, III\}$, $r_{orange} = \{I\}$, $r_{pink} = \{II, III\}$. The trace β in the Büchi automaton that the robots collectively satisfy is shown in purple in Fig. D.3. $\mathcal{R}_A$ and β are automatically found using the method proposed in [32].

During execution, modifications to the robots' capabilities occur. In the following sections, we illustrate how the robots conduct online resynthesis such that the overall team still satisfies the task.

D.6 Approach: Resynthesis Framework

We update the behavior of the robots on the fly when capability modifications occur. A summary of the overall framework is shown in Fig. D.4. We only synthesize new behavior when necessary; Section D.6.1 describes how we determine whether resynthesis is needed. Section D.6.2 outlines the method for a robot to update its product automaton. Section D.6.3 outlines the binding reas-

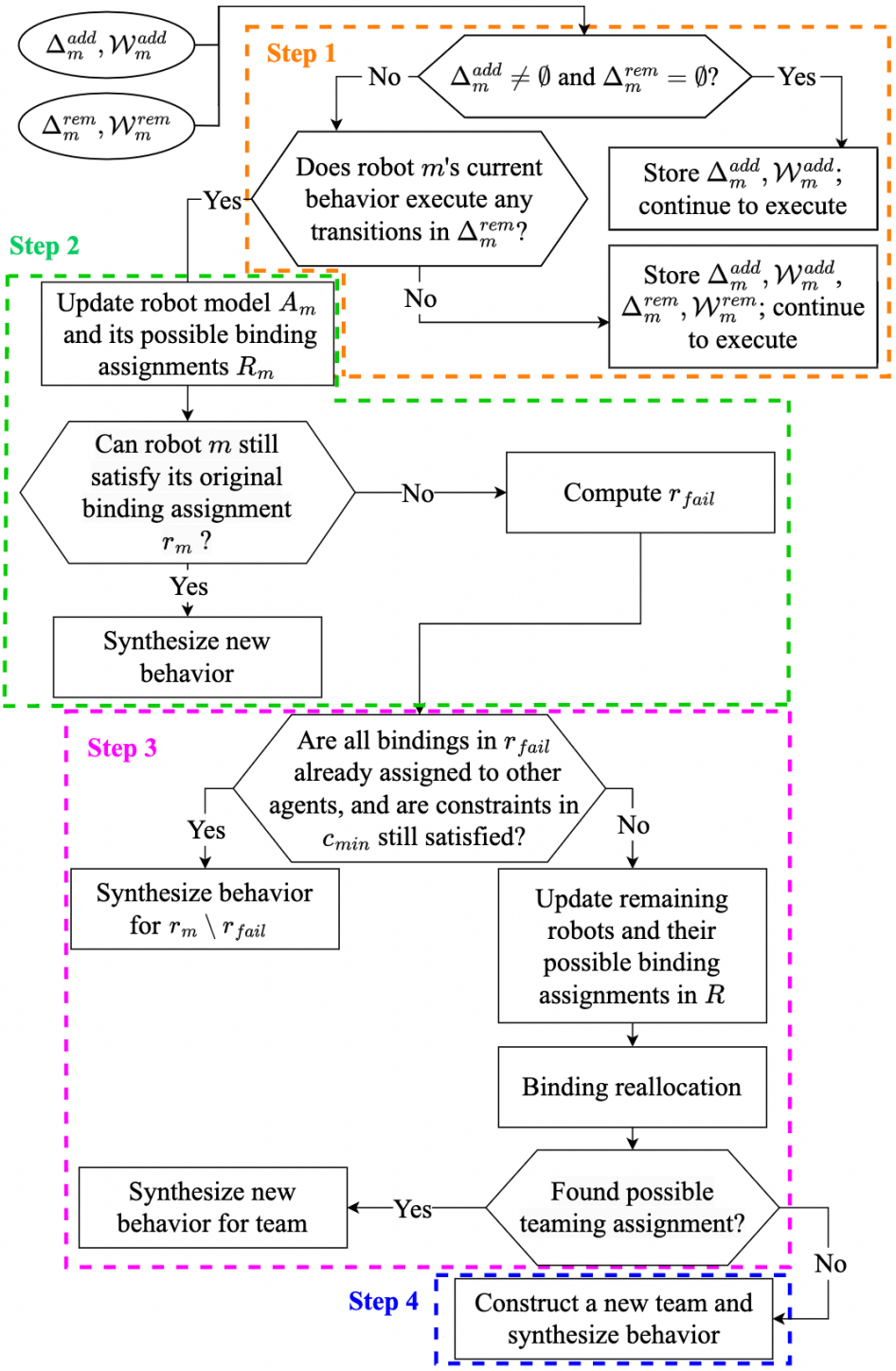


Figure D.4: Overview of resynthesis process

signment process.

D.6.1 Evaluating Modified Robot's Behavior

When robot m is modified (i.e. $\Delta_m^{add} \cup \Delta_m^{rem} \neq \emptyset$), it first determines whether its modifications will affect the ability of the team to successfully execute the task. It does so by executing the following steps:

Step 1: First, the robot checks if $\Delta_m^{add} \neq \emptyset$ (the robot has expanded its capabilities) and $\Delta_m^{rem} = \emptyset$ (the robot has not lost any of its capabilities). In this case, the robot is still able to execute its original behavior. Thus, this information is stored and then incorporated when binding reassignment is necessary; the robot continues executing its original behavior. When $\Delta_m^{rem} \neq \emptyset$, i.e. transitions are removed from a robot's capability, the robot checks if its original behavior contains any of those transitions. If not, the robot continues to execute its original behavior and stores these modifications.

Step 2: The robot proceeds to step 2 if its original behavior includes a transition it can no longer perform, i.e. for behavior $b_m = q_1 q_2 \ldots$, where $q_i = (s_i, z_i)$, $\exists i \in \{2, \ldots, |b_m|\}$ and $\exists (x, x') \in \bigcup_a \Delta_{j,a}^{rem}$ s.t. $x \in s_{i-1}$, $x' \in s_i\}$. It updates its model and product automaton to reflect the modifications (Section D.6.2).

Based on its updated robot model and product automaton, it determines if there are any bindings it was originally assigned, r_m, that it can no longer do. These bindings are stored in r_{fail}.

The team of robots may have overlapping binding assignments. Thus, if all bindings in r_{fail} are already assigned to other robots and the user-specified con-

straints regarding the minimum number of robots assigned to each binding c_{min} are still satisfied, only robot m needs to resynthesize a behavior for the remaining bindings it can do ($r_m \setminus r_{fail}$); the rest of the team continues executing their original behavior.

Step 3: If there are bindings in r_{fail} that are not already assigned to other robots, then binding reallocation is necessary. Before this can happen, each robot j updates R_j, the set of all possible binding combinations it can do. To do so, each robot updates its product automaton based on any modifications that have been previously stored (see Sec. D.6.2). Updating of each robot's model and product automaton is done in a distributed manner; only the binding reallocation algorithm is centralized. During reallocation, we minimize the number of robots that change assignments, since every reassignment requires the robot to resynthesize its behavior.

Step 4: In the worst case scenario, if there is no possible binding reallocation such that the robots can satisfy the task, we synthesize a new team using the framework proposed in [32] to find another collective trace in the Büchi automaton.

D.6.2 Updating the Product Automaton

Before resynthesizing their behavior, each robot j updates its model and product automaton based on current and past modifications, if any. We update the product automaton $\mathcal{G}_j$ based on the sets of transitions to be added Δ_j^{add} and removed Δ_j^{rem}. To do so, we construct $\mathcal{B}_\beta^{j,t}$, which represents the portion of the collective trace β that has not yet been executed by robot j at the step t when

the modification is introduced, and A_j^{mod}, which contains only the affected transitions of the robot model. This way, we can remove or add the transitions to the product automaton based on $\mathcal{G}_j^{mod} = \mathcal{B}_\beta^{j,t} \times A_j^{mod}$, rather than reconstructing the entire product automaton of a robot $\mathcal{G}_j = \mathcal{B} \times A_j$ from scratch.

Constructing $\mathcal{B}_\beta^{j,t}$: Since we know the trace β in the Büchi automaton that the team is traversing, we only need to check how the capability modification affects the transitions in β.

Let the modification to robot m occur when it is at $q_m^t = (s_m^t, z_m^t)$ in its behavior. Then, for each robot j, we define $\mathcal{B}_\beta^{j,t}$ as the reachable portion of β from z_j^t. Note that $\mathcal{B}_\beta^{j,t}$ may not be equivalent $\mathcal{B}_\beta^{m,t}$; that is, given a robot j and the state it is at when the modification occurs, $q_j^t = (s_j^t, z_j^t)$, z_j^t may not be equal to z_m^t for any robot $j \neq m$ due to the synchronization policy each robot executes, as outlined in [31]: For each transition $(z, \sigma, z') \in \beta$, where $z \neq z'$, robots that are assigned bindings that appear in σ must wait to execute their transition $((s, z), (s', z'))$ until all other relevant robots are also ready. Thus, any robots whose assigned bindings do not appear in σ are not involved in the synchronization requirement. For example, given the Büchi automaton shown in Fig. D.3, if a robot's assigned bindings are $r = \{II\}$, then it does not need to wait to synchronize for transition $(2, \sigma, 3) \in \beta$, where $\sigma = \{\{storage_c^{III}, beep^{III}\}, \emptyset, \emptyset, \{dock_c^I\}\}$, since binding II does not appear on any proposition in σ.

Although the robots may be at different states in β when the modification occurs, we want to ensure that the entirety of β is still satisfied when the robots resynthesize new behavior. To do so, let $\textsc{Index}(\beta, q_j^t) = \{i \in \{1, \ldots, |\beta|\} \mid \beta[i] = z_j^t\}$, where $q_j^t = (s_j^t, z_j^t)$. This outputs the index of state z_j^t in trace β.

If $\text{INDEX}(\beta, q_j^t) > \text{INDEX}(\beta, q_m^t)$, then robot j is "ahead" of robot m in the trace β, implying that it did not have to participate in any synchronization policies between states z_m^t and z_j^t in β. This means that robot j can be in any state in its robot model without violating any transitions (and states) between z_m^t and z_j^t. Thus, to guarantee that the entire trace β is satisfied, we take the conservative approach and move robot j "back" to z_m^t ; i.e. we modify the state from $q_j^t = (s_j^t, z_j^t)$ to $q_j^t = (s_j^t, z_m^t)$ and $\mathcal{B}_\beta^{j,t} = \mathcal{B}_\beta^{m,t}$. Conversely, $\text{INDEX}(\beta, q_m^t) \geq \text{INDEX}(\beta, q_j^t)$ indicates that the modified robot m is "ahead" of robot j in β and can maintain any truth value between z_j^t and z_m^t. Thus, q_j^t and $\mathcal{B}_\beta^{j,t}$ remain unchanged.

Constructing A_j^{mod}: The approach to constructing A_j^{mod}, the robot model with the modified transitions, differs depending on if the transitions in the robot capabilities need to be removed (Δ_j^{rem}) or added (Δ_j^{add}).

Constructing A_j^{mod} with Δ_j^{rem}

$\Delta_{j,a}^{rem} \in \Delta_j^{rem}$ is the set of transitions (x, x') that are no longer valid in the robot's capability λ_a, and we need to remove the transitions $((s, z), (s', z')) \in \delta_{\mathcal{G}}$ in the robot's product automaton $\mathcal{G}_j$ that are no longer valid. To do so, we first construct the robot model with the modified transitions, $A_j^{mod} = (S_j^{rem}, s_j^t, AP_j, \gamma_j^{rem}, L, W)$, where $S_j^{rem} \subseteq S, \gamma_j^{rem} \subseteq \gamma$ are defined as

$$\gamma_j^{rem} = \{(s, s') \in \gamma_j | \exists (x, x') \in \bigcup_a \Delta_{j,a}^{rem} \text{ s.t. } x \in s, x' \in s'\} \tag{D.5}$$

$$S_j^{rem} = \bigcup_{(s,s') \in \gamma_j^{rem}} s \cup s' \tag{D.6}$$

Constructing A_j^{mod} with Δ_j^{add}

$\Delta_{j,a}^{add} \in \Delta_j^{add}$ is the set of transitions (x, x') to be added to the robot's existing capability λ_a, and $\mathcal{W}_{j,a}^{add}$ is the set of cost functions that assigns a weight to the added transitions. Note that x or x' might be new states in the capability.

Let the current robot model be $A_j = \lambda_a \times \cdots \times \lambda_k$. Without loss of generality, let $\Delta_{j,a}^{add}$, $\mathcal{W}_{j,a}^{add}$ be the set of transitions and cost function, respectively, in capability λ_a that is being added, where $\lambda_a = (X_a, x_a^t, AP_a, \Delta_a, \mathcal{L}_a, \mathcal{W}_a)$. Then, $\lambda_a^{add} = (X_a^{add}, x_a^t, AP_a, \Delta_{j,a}^{add}, \mathcal{L}_a, \mathcal{W}_{j,a}^{add})$, where $X_a^{add} = \bigcup_{(x,x') \in \Delta_{j,a}^{add}} \{x, x'\}$. The added portion of the robot model is $A_j^{mod} = \lambda_a^{add} \times \cdots \times \lambda_k$.

Constructing $\mathcal{G}_j^{mod}$: Using $\mathcal{B}_\beta^{j,t}$ and A_j^{mod}, we construct the affected product automaton $\mathcal{G}_j^{mod} = A_j^{mod} \times \mathcal{B}_\beta^{j,t}$.

If we are considering Δ_j^{rem}, then the portion of the product automaton that is to be removed is $\mathcal{G}_j^{mod} = \mathcal{B}_\beta^{j,t} \times A_j^{mod}$. We modify the original product automaton by removing the transitions $\delta_\mathcal{G}^{mod}$, i.e. $\delta_\mathcal{G} = \delta_\mathcal{G} \setminus \delta_\mathcal{G}^{mod}$.

If we are considering Δ_j^{add}, then we add $\mathcal{G}_j^{mod}$ to the original product automaton, i.e. $\delta_\mathcal{G} = \delta_\mathcal{G} \cup \delta_\mathcal{G}^{mod}$, $Q = Q \cup Q^{mod}$, and

$$W_\mathcal{G}((x, x')) = \begin{cases} W_\mathcal{G}((x, x')) & (x, x') \in \delta_\mathcal{G} \setminus \delta_\mathcal{G}^{mod} \\ W_\mathcal{G}^{mod}((x, x')) & (x, x') \in \delta_\mathcal{G}^{mod} \end{cases}$$

where $\delta_\mathcal{G}^{mod}$, Q^{mod}, $W_\mathcal{G}^{mod}$ are the transitions, states, and cost function, respectively, in $\mathcal{G}_j^{mod}$.

D.6.3 Binding (Re)Allocation

We modify the binding allocation framework proposed in [31] such that it 1) allows users to provide constraints on the minimum number of robots that must be assigned to a specific binding or which bindings are not allowed to be assigned to the same robot (Section D.2, shown in green in Alg. 10), and 2) to reallocate robots to bindings in response to modifications such that we minimize the number of robots that are reassigned different bindings (shown in blue in Alg. 10).

Given the set of robots and R, where $R_j \in R$ is set of all possible binding assignments robot j can do, the goal is to assign each binding to the minimum number of robots it requires (Alg. 10). We intialize the set of unassigned bindings, *unassigned*, to be the set of all bindings $\mathbf{b}$. For each round of binding allocation, the framework selects the robot j to be assigned based on the following ordering:

1. Given the set of unassigned robots, we first choose the robot j that has a possible binding assignment containing at least one unique binding, i.e. robot j contains at least one binding that can only be assigned to it. If multiple robots qualify, one is selected at random, and its unique bindings are stored in r_j^* (line 3). During the original allocation process, the final assignment for robot j, r_j^{new}, is the largest set of bindings it can do that contains the bindings in r_j^*; for reallocation, preference is given to the original robot assignment r_j.
2. If none of the unassigned robots satisfy the previous criteria, then we find the set R', where robot j's possible binding assignment set R_j is in R' if

and only if robot j can be assigned to at least one binding that is currently unassigned (line 8). If multiple robots qualify, the robot with the least number of elements in its possible binding assignment set R_j (i.e. the robot with the least flexibility in its assignment) is chosen (line 10). We do this to ensure that the robot with the most flexibility in its assignment (i.e. has the most binding assignment options) will not be chosen first. Similar to before, during reallocation, preference is given to the original robot assignment r_j; otherwise, the final assignment for robot j, r_j^{new}, is the largest set of bindings it can do that contains a binding in the set of unassigned bindings (line 12).

3. If all bindings have been assigned, robot j is chosen at random. During the original allocation process, it is assigned the maximally-sized set of bindings (line 14); during reallocation, preference is given to the original robot assignment r_j.

For the constraints c_{min}, after a robot is assigned a set of bindings r_j^{new}, we update c_{min} to be the set $\{(\rho, k-1) \mid (\rho, k) \in c_{min}, \rho \in r_j^{new}, k-1 > 0\}$ (line 16). Intuitively, for each binding $\rho \in r_j^{new}$, we decrement the corresponding value of k, which represents the minimum number of robots that are still required to be assigned ρ. If $k - 1 \leq 0$ (i.e. at least k number of robots have now already been assigned to ρ), we remove (ρ, k) from c_{min} and remove ρ from the set of bindings that have not been assigned yet (line 17).

Because the robot is assigned the largest set of bindings, the team may have overlapping assignments, i.e. robots can be removed while still ensuring the overall task will be completed, which is beneficial for robustness. However, the teaming assignment may vary depending on the ordering in which the robots

are assigned. Thus, the reassignment may not be the globally optimal solution.

Algorithm 10: Binding (Re)allocation

Input : $\mathbf{b}, R = \{R_1, ..., R_n\}, \mathcal{R}_A = \{r_1, ..., r_n\}, c_{min}$
Output: $\mathcal{R}_A^{new}$

$unassigned = \mathbf{b}$
while $R \neq \emptyset$ **do**
  $r_j^* = \text{GET\_UNIQUE\_R}(R)$
  if $r_j^* \neq \emptyset$ **then**
    $R_j^* = \{r \in R_j \mid r_j^* \subseteq r\}$
    $r_j^{new} = \begin{cases} r_j & \text{if } r_j \neq \emptyset, r_j \in R_j^* \\ \text{RANDOF}(\text{argmax}_{r \in R_j^*} \lvert r \rvert) & \text{otherwise} \end{cases}$
  else
    $R' = \{R_j \in R \mid \exists r \in R_j \text{ s.t. } r \cap unassigned \neq \emptyset\}$
    if $R' \neq \emptyset$ **then**
      $R_j = \text{argmin}_{\hat{R} \in R'} \lvert \hat{R} \rvert$
      $R_j' = \{r \in R_j \mid r \cap unassigned \neq \emptyset\}$
      $r_j^{new} = \begin{cases} r_j & \text{if } r_j \neq \emptyset, r_j \in R_j' \\ \text{RANDOF}(\text{argmax}_{r \in R_j'} \lvert r \rvert) & \text{otherwise} \end{cases}$
    else
      $r_j^{new} = \begin{cases} r_j & \text{if } r_j \neq \emptyset, r_j \in R_j \\ \text{RANDOF}(\text{argmax}_{r \in R_j} \lvert r \rvert) & \text{otherwise} \end{cases}$
  $R \setminus R_j, \mathcal{R}_A^{new} \cup \{r_j^{new}\}$
  $c_{min} = \text{UPDATE\_C}(c_{min}, r_j^{new})$
  $unassigned = \text{UPDATE\_UNASSIGNED}(unassigned, c_{min}, r_j^{new})$
if $unassigned = \emptyset$ **then**
  return $\mathcal{R}_A^{new}$
else
  return $\emptyset$

D.7 Demonstration and Evaluation

We illustrate the modification resynthesis framework using the example in Sec. D.5.3. The behavior of the robots as modifications occur in simulation is shown in the accompanying video.

D.7.1 Mod 1: Adding Transitions

During execution, the blue robot gains the ability to move between rooms B and G (Figure D.1(b)), $\Delta^{add}_{blue} = \{(\{roomG_c\}, \{roomG_c, roomB\}), (\{roomG_c, roomB\}, \{roomB_c\}), (\{roomB_c\}, \{roomB_c, roomG\}), (\{roomB_c, roomG\}, \{roomG_c\})\}$. On a physical system, this could represent a door opening or a ramp being introduced between the two rooms. Because adding transitions does not violate the current behavior of the robot, the blue robot stores this modification and continues executing its original behavior. The overall time for this modification was 0.00715 ms.

D.7.2 Mod 2: Removing Transitions without Reallocation

The orange robot can no longer move between room D and the hall, $\Delta^{rem}_{orange} = \{(\{roomD_c\}, \{roomD_c, hall\}), (\{roomD_c, hall\}, \{hall_c\}), (\{hall_c\}, \{hall_c, roomD\}), (\{hall_c, roomD\}, \{roomD_c\})\}$. This could represent a door closing, or the size of the entrance changing such that the robot is no longer able to move through it.

The orange robot's original behavior included transitioning from room D to the hall in order to get to room B. Thus, it updates its model and product automaton and checks if it can still satisfy its original binding assignment $r_{orange} = \{I\}$. Since it can still reach room B by going through rooms D and C, the robot can still satisfy binding I. Thus, the orange robot resynthesizes its behavior, and no other robots are affected. The overall time for this modification was 18.79 ms.

D.7.3 Mod 3: Removing Transitions with Reallocation

During execution, the pink robot's camera fails, $\Delta^{rem}_{pink} = \{(\emptyset, \{camera\}), (\{camera\}, \{camera\})\}$. In this scenario, the robot is unable to perform its original binding assignment $r_{pink} = \{II, III\}$; since it no longer has a camera, it cannot satisfy binding II, and therefore $r_{fail} = \{II\}$. If any other robots were already assigned binding II, then reallocation is not required. However, this is not the case in the original assignment. Thus, the robots go through the reallocation process. After each robot updates their individual models and product automata, their possible binding assignments are $R_{green} = \{(I), (II)\}$, $R_{blue} = \{(I)\}$, $R_{orange} = \{(I)\}$, $R_{pink} = \{(I), (III), (I, III)\}$.

During reallocation, the green robot is reassigned from binding I to binding II and the pink robot is reassigned to bindings I and III; all other robots maintain their original assignment and therefore do not resynthesize their behavior. The overall time for this modification was 196.03 ms. The time for the pink robot to update its model and product automata and find r_{fail} was 109.0 ms; subsequently, the time for the remaining robots to update their models and product automata was 21.35 ms; the time for the task reallocation was 0.0699 ms; the time for the robots to resynthesize their behavior was 37.39 ms.

D.8 Conclusion

We introduced a hierarchical method for a team of heterogeneous robots to react to modifications in their capabilities during execution of a LTL$^{\psi}$ specification. We also increase the expressivity of the LTL$^{\psi}$ grammar by allowing the user to

require a minimum number of robots for a binding, as well as constrain which bindings cannot be assigned to the same robot. We implemented our approach in simulation in a warehouse scenario.

In the future, we plan to extend the resynthesis framework to other aspects of reactivity, such as reacting to external events. We also plan to explore ways to incorporate optimality (e.g. minimizing cost) when finding a teaming assignment, as well as relaxing all-to-all communication constraints as they synchronize their behavior.

APPENDIX E

DECENTRALIZED CONTEXT-BASED PLANNING FOR EARTH OBSERVATION MISSIONS

E.1 Introduction

This paper introduces a new decentralized methodology for autonomous onboard planning of Earth Observation (EO) missions using contextual information. As requirements for EO missions grow tighter, re-purposing available EO satellites currently on orbit is increasingly considered as a valid alternative to launching new missions, especially for missions that require high revisit times only attainable through the use of constellations [39]. For example, NASA's Technology Taxonomy[1] (a compendium of technologies that are of interest to advancing NASA's mission) identifies technologies that foster reasoning, acting, collaboration, and interaction as priorities for development in space autonomous systems in the next years.

For the most part, current EO satellites are similar to silos as far as planning goes: they have a simple static imaging concept (e.g., cross-track scanning) that implicitly defines an observation schedule before the beginning of the mission and that schedule never or rarely changes. Some newer imaging satellites actually have pointing capabilities, like the Sentinel constellation [8], and they can modify those static plans to capture events of interest. It is quite rare to find systems for which the schedule is fully defined in near real time, but when this happens (e.g., the Planet constellation [9]) the scheduling problem is solved on the ground and plans uploaded to individual satellites. In these few cases the

[1] https://www.nasa.gov/offices/oct/taxonomy/index.html

system is always homogeneous and centrally owned and operated by one actor. Many approaches to scheduling and planning for EO missions have been developed assuming this centralized, on-ground paradigm, including approaches based on Constraint Satisfaction [103], Genetic Algorithms [110], or Integer Programming [36].

One limitation of the ground-based planning approach is that it potentially misses short-lived events due to the time incurred between the observation and the data processing on the ground (which includes the time from observation to downlink plus the time for data distribution and processing). Stochastic urgent events require fast revisit times, but the average revisit time for many of the most commonly used satellites (which are in Low Earth Orbit) is on the order of a day at best (such as the observations of MODIS from the Aqua and Terra satellites) and up to two weeks (e.g. Landsat satellites), and that is only if the satellites have no other constraints, such as VNIR sensors requiring cloud-free observations. This revisit time may be too long for short-lived or short-notice events such as cloud formation or volcano eruptions.

A limitation of the centralized planning approach it does not scale well to very large constellations of satellites (which are gaining popularity) due to requiring expert input. These experts have to be knowledgeable on contextual information about the mission and available assets to make a decision, and that leaves no time for evaluating many different options, which may lead to a solution that is not optimal.

Both of these limitations are particularly problematic given the trends in EO system architectures. Experts in the field have postulated for decades that the future of EO is the Sensor Web [27], i.e., decentralized networks of intelli-

gent nodes carrying heterogeneous sensors with on-board decision making and cross-link capabilities. However, nothing close to this has been actually flown. In terms of a having a large constellation of heterogeneous systems actually flown, QB50 [78] is the closest system. QB50 is a constellation of CubeSats designed and launched by different universities with heterogeneous and complementary sensors, but it lacks the advanced on-board decision making and full cross-link capabilities. These technologies however have been demonstrated independently – satellite cross-links by Iridium for example [35] and on-board decision making by EO-1 [21].

Another proposed (but not demonstrated) concept that illustrates the trend towards decentralization and on-board decision-making is that of federated systems [39], in which satellites owned and operated by different organizations come together opportunistically and temporarily to work on emerging tasks.

Our approach is particularly suited for such future EO system concepts, as the different actors involved in a federation or large distributed system like QB50 may have used different onboard "dictionaries" to define and reason about their satellites' capabilities. Our approach's context mining and reasoning are suited to deal with different sources of data to harmonize them and get some level of interoperability between these satellites.

All these new approaches to EO –the Sensor Web, federated systems, the QB50 constellation– require new decentralized planning algorithms. Other than the approach described in this paper, in recent years there has been a constant stream of research in on-board scheduling for EO satellites. In [66], Li et al. introduce an autonomous online approach to satellite scheduling that can handle urgent tasks arriving stochastically during the planning by using two heuristics

to decide when to schedule and how to do so. In more recent work, Li [65] expands their work to encompass a whole distributed satellite system, and bases their algorithm on the asynchronous version of the consensus-based bundle algorithm (ACBBA) [23]. Another example, also based on CBBA, is the approach by Gallud and Selva described in [37] for decentralized planning in distributed and federated EO systems.

As an example of a reentative problem and corresponding distributed EO system, consider the case of detecting volcano eruptions. NASA has developed two parallel approaches for this purpose. On the one hand, it has been using data from both the Aqua and Terra satellites's MODIS sensor to create MODVOLC [112], a detector for eruptions around the Earth with a revisit frequency between 24 and 48h, which may not be fast enough in the case of explosive eruptions. The other, more recent, approach is described in detail in [22]. It combines in-situ and space observations to monitor when an eruption happens and trigger special observations for some satellites and ground assets. In the case of ESA, they developed the Geohazards Exploitation Platform (GEP)[2], which combines images and radar information from the Sentinel 1 and 2 constellations to detect natural disasters, including volcano eruptions. In the three cases, pointing satellites to take extra images of a region of interest requires a complicated process involving humans and approval processes on the ground.

In this paper, we propose a new approach for the decentralized planning problem for heterogeneous, distributed (and potentially federated) EO systems. In this approach, agents independently choose whether and how to respond to a new mission through the use of contextual information. A preliminary offline step of our approach is the mining of such contextual information. Through this

[2]https://geohazards-tep.eu/

paper, we use an extensive definition of contextual information, including facts ranging from the capabilities of a satellite and its sensors such as accuracies, power, image resolution, fields of view, and other characteristics, as well as information about their current status including orbital information and whether sensors are operational or not. This contextual information also includes knowledge about the physics of the Earth system and remote sensing, such as how different so-called Level-1 measurements (e.g., TIR radiances or radar back-scatter cross-sections in L-band) relate to geophysical parameters or Level-2 products (e.g., land surface temperature or soil moisture) and the relation between a mission specification and a set of geophysical parameters to observe. This context could optionally include information about other satellites and their capabilities. Going back to the volcano eruption example, this means the context has information on volcano eruptions generating heat, land displacements, and ash plumes, and we also know that Thermal Infrared (TIR) and Short-Wave Infrared (SWIR) radiance can be used to measure temperature at different ranges, and SAR interferometry can provide ground surface motions (gradients of velocity vectors). Note the wealth of information that is fed to the planning algorithm. In comparison, the approaches mentioned before only take into account a much more narrowly defined state of the satellite as needed to schedule their missions. This comprehensive context information is intended to help with planning by eliminating the need for an EO expert at different steps of the process. While a classic approach requires an expert to realize that TIR sensors can be used to measure the temperature increase in the surface of a volcano during an eruption and then consider the available TIR sensors, our proposed method can derive this piece of information by itself and conduct those steps autonomously. Once this is done, the algorithm can use each satellite and sensor status information

to decide what are the best out of all these to participate in a mission to monitor a specific volcano. In order to extract all the necessary information about sensors, observables, etc, we have used unstructured text mining techniques to fill a probabilistic knowledge graph (KG), which is a set of tuples consisting of a fact reented as a triplet (concept C_1, relation R, concept C_2) and a probability p of that fact being true. This KG is then coupled with a mixed probabilistic- and logic-based reasoning system in order to infer more relations and reason about which satellites can participate in a given mission. Since such a general problem has never been considered before in EO, no other approach in the literature uses a similar reasoning system to start the planing procedure.

Once we have the contextual information for each agent, our approach has three steps, each adding significant novelty to the way scheduling is done for EO systems: first, each satellite decides whether it *can* participate in the mission by comparing its capabilities with the mission specification (e.g., a satellite with a VNIR or TIR imager can participate in a volcano detection mission) and checking if it has potential visibility of the target region in the specified period of time. This step is done by reasoning over a knowledge graph (partially mined from scientific papers) containing relations between different sensor types, geophysical parameters and observables. While this first step provides a very computationally efficient down-selection of the agents that need to consider the mission, it is essentially a binary check that only considers the capabilities of the satellite as opposed to the expected performance. Hence, in the second step, each satellite that has decided it can participate rates its potential to add information and thus value to the mission by considering more detailed sensor specifications (e.g., sensor accuracy) and the specific processes involved, in order to decide if it *should* participate. In the literature, these scores are usually calculated semi-

qualitatively. For example, in the VASSAR methodology [91], scores are computed through a set of rules that take into account sensor characteristics such as spatial resolution, revisit time, accuracy, etc. Often, this number comes from subjective numbers based on expert judgment. Rather than relying on subjective assessments of the value of different combinations of parameters, our approach is based on the physics and mathematics of sensing. Because computing a full error budget for the parameters of interest onboard is impossible based on the information available, we propose a new simple but powerful theoretical framework to estimate the uncertainty in the measurements of each sensor based on a decentralized Kalman filter. In this paper, we consider the case of an event detection mission (specifically volcanic eruption detection) as opposed to other types of missions. Hence, the result of the Kalman filter is fed to a classical binary hypothesis testing framework to assess the probability that the sensor can detect the event of interest. In a completely decentralized system, each satellite could simply decide to act based on the output of this step. However, in practice, there is likely value in considering some coordination between the satellites. In this paper, we consider the case where a central node synthesizes a teaming plan based on the input from each sensor and formally verifies that it can satisfy the mission using probabilistic temporal logic. While the approach does do some search for a good team, it does not find the optimal team as it is a computationally intractable problem. The formal methods approach to the verification of the decentralized plan based on probabilistic model checking is very new for EO and provides a rigorous theoretical framework to synethesize and reason about plans including complex constraints, e.g. related to scarce on board resources. Our method is applicable for space-based observation of any geophysical parameter, although it is most useful for remote sensing of short-

lived or short-notice processes and phenomena by constellations of satellites with heterogeneous sensors (e.g. QB 50) or federated systems.

More generally, the problem we are trying to solve is formulated as follows: given a **mission** specified by 1) a **geophysical parameter** of the Earth to observe such as sea surface height or soil moisture, or an **event** such as a volcano eruption, 2) a **region** (e.g. the Arctic, the Mediterranean basin, or the Pacific Ring of Fire), and 3) some **time-related requirements** (mission duration of a week or month, revisit time of two measurements a day, or a maximum gap between measurements of 24 hours), find **a selection of teams of satellites** that will perform the required observations to successfully carry out the mission. Optionally, the mission statement can also include other parameters such as spatial resolution, accuracy, etc. Of note, relevant requirements for many parameters can be readily obtained from the online World Meteorological Organization (WMO) OSCAR database[3].

The main performance metric in this paper is in the form of a probability of success in the mission. Since teams with more satellites have higher probabilities of success, the output of the system is a Pareto frontier of teams generated by our decentralized approach, with a probability of the mission succeeding for each team vs the number of satellites involved. This probability can be calculated a priori without seeing any dataset. If a real dataset is provided, a probability for each of the teams successfully carrying out the mission in the context of the data can be computed for verification.

The set of available agents can be the set of all Earth observing satellites currently in space, which can be extracted from the CEOS database[4]. This database

[3] `https://www.wmo-sat.info/oscar/`
[4] `http://database.eohandbook.com/`

contains information about all the current and planned Earth observation missions and their sensors.

For example, if the mission is to measure soil moisture in the Mediterranean basin daily for one month, one would expect that the Soil Moisture and Ocean Salinity (SMOS) and Soil Moisture Active Passive (SMAP) satellites would both volunteer to participate in the mission, but more sensors (e.g., C-band radar from Sentinel) may be required to achieve the desired mission attributes. In order to compare against a team that is already in the literature to validate our approach, we formulate a simple but general and relevant test problem consisting of detecting a volcano eruption based on remote sensing of land surface temperature, aerosol plumes (especially SO2), and land surface displacements in one out of the 10 most active volcanic areas on Earth, but the problem is readily generalized to any other event that can be linked to one or more geophysical parameters.

In order to ensure our planning algorithm works, we compare our set of teams to the team used in [22] to perform exactly the same task. Their team is comprised of the following satellites: Aqua, Terra, the Sentinel 1 constellation, the Sentinel 2 constellation, and all active GOES satellites. Given that a similar metric to the probabilities outputted by our algorithm for that team is unknown, we have processed the team through the same pipeline as our teams in order to ensure a fair comparison.

E.2 Methodology

E.2.1 Overview

An outline of the system can be seen in Figure E.1. The outline described in the flowchart is the approach with a centralized decision node (reented by the optimization algorithm). In the case of the decentralized approach, the optimization algorithm changes to a threshold-based approach where each satellite decides to participate based on its own perceived usefulness to the mission. The next subsections describe the subsystems in the flowchart, as well as the teaming algorithms.

E.2.2 Knowledge representation

The knowledge representation framework requires different types of knowledge: 1) domain knowledge, which includes knowledge about: a) events of interest and related observable physical phenomena (e.g., volcano eruptions generate heat and SO_2 aerosols), and b) sensor types and their capabilities to measure various geophysical parameters (e.g., a short-wave infrared sensor can measure aerosols, thermal infrared sensors can measure heat); 2) knowledge about the mission at hand (e.g., to detect volcano eruptions in Hawaii); 3) knowledge about the agents and their sensors that are available to perform missions (e.g., a particular agent can measure TIR radiance with 5% accuracy); 4) knowledge about the state of each agent (e.g., is sensor operational, is the target in visibility of the sensor).

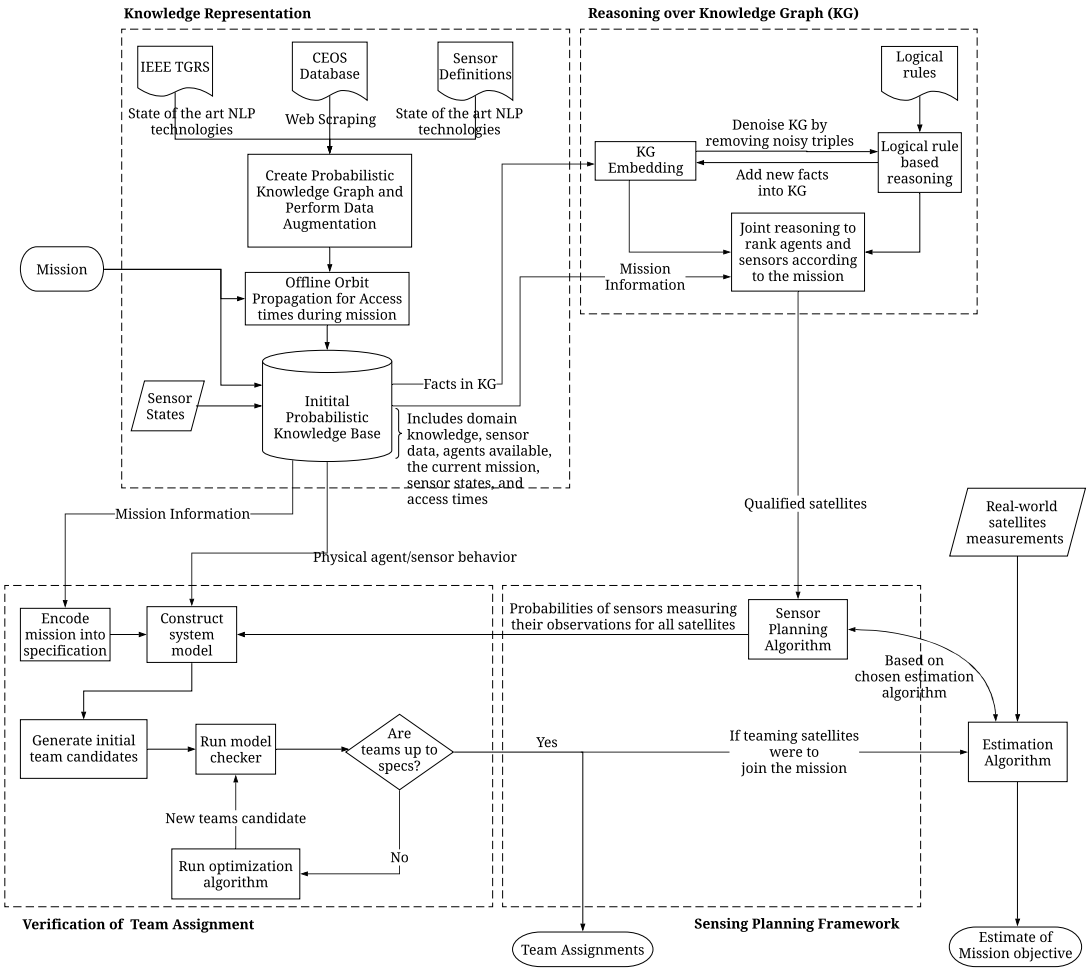


Figure E.1: System flowchart, showcasing the interactions between the components in our approach

These different types of knowledge can all be reented with a probabilistic knowledge base, as seen in Figure E.2. Each agent has a copy of this knowledge base, which can either contain only relevant information to the agent or all the information for the system. Part of the knowledge is intrinsically probabilistic, as seen in the figure.

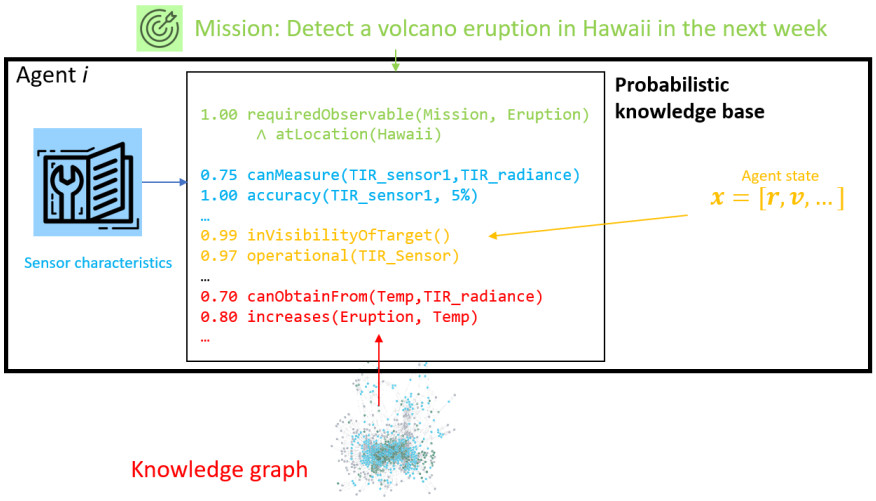


Figure E.2: Different types of knowledge in the probabilistic knowledge base, with their sources and their associated uncertainty

While the sources of knowledge for the mission, the agent state, and the sensor characteristics are relatively clear, finding the domain knowledge linking the sensor capabilities and characteristics to the mission at hand is a challenge. The basic idea is that one must find a path in the KG from the geophysical parameters observable by the sensor to the target observable event required by the mission. This path may in general require multiple steps (e.g. volcano eruption $\rightarrow$heat $\rightarrow$thermal infrared (TIR) radiation $\rightarrow$TIR sensor). If the path already exists in the KG, then one can proceed with inference. However, if such a path does not exist in the KG (e.g., the agent does not currently know that heat is related to TIR radiation), the agent will then attempt to find new information linking those entities from other existing corpuses. In the case an Internet con-

nection is available, the agent will source the information from there. This being said, most satellites in space lack access to the Internet, so they might have to look for such information in local copies stored in each agent. This requires extracting information from unstructured text sources such as databases of papers, using techniques such as Named Entity Recognition (NER) [117] and Relation Extraction (RE) [96]. For the approach described here, we have built a Natural Language Pipeline (NLP) to extract the links between different observations, sensors, and physical phenomena from papers published in the Transactions on Geoscience and Remote Sensing journal[5]. Specifically, our pipeline includes an NER pipe based on the Scibert Transformer [7], followed by an Entity Linker (EL) to match the extracted entities to out KG. The EL is based on the models in SpaCy 2 [48], and retrained for our KG. The last part of the pipeline is an RE engine that we have built for this project that creates new relationships in the KG between the entities that have been found in the previous steps.

Two longer term research challenges related to both knowledge representation and reasoning in our system are: 1) modeling the different types of uncertainty that exist in the problem, and 2) how the knowledge and its uncertainty change over time.

We define two sources of uncertainty. It may be aleatory (e.g., random noise in the sensor) or epistemic (e.g., lack of information about a sensor, observable, mission, or relations between them). Aleatory uncertainty can be modeled as usual with probability theory. However, for epistemic uncertainty, KGs lack the capacity to easily reent uncertain facts. We are exploring approaches combining probability and logic, such as Probabilistic Soft Logic [3] and Markov Logic Networks [74]. In Markov Logic Networks (MLN), rules can have weights related

[5] https://ieeexplore.ieee.org/xpl/RecentIssue.jsp?punumber=36

to the probability of the rule being true, but the evidence is still Boolean. Soft logic approaches allow for uncertain evidence. Regardless of the implementation, the requirement is to have a probabilistic knowledge base where both the evidence and rules have an associated probability of being true. How we obtain these probabilities for the case of epistemic uncertainty is another challenge that can be addressed, for example, with embedding-based methods, which will be explained in more detail in the next section.

In the case of the time dependency of both knowledge and uncertainty, we have to take into account all the different time-dependent sources of information. For example, the mission might change at any time, as well as the agent state, with sensors being taken down or up depending on system status. While the sensor characteristics are relatively stable, the domain knowledge is continuously being updated with the publication of new information, and a successful system will require keeping up to date with the latest advances in the field in order to stay competitive. Such changes might add new relationships to the KG or change the uncertainty on existing ones.

E.2.3 Reasoning over KG

Notation A knowledge graph, denoted by $\mathcal{G} = \{E, R, O\}$, consists of a set of entities E, a set of relations R, and a set of observed facts O. Each fact in O is reented by a triple (e_i, r_k, e_j), where $e_i \in E$, $e_j \in E$, and $r_k \in R$ denote subject entity, object entity, and relation, respectively.

The KG reasoning module is responsible for deciding whether a satellite can participate in a given mission. Given a KG containing both common sense

knowledge about the field and knowledge about the sensors we have, and a set of logical rules specified by domain experts, we aim to predict which sensors can participate in the mission by leveraging the relations between geophysical parameters observable by the sensors and the objective events required by the mission.

There are two main directions for solving the posed problem, which is also known as KG inference: logical rule reasoning and knowledge graph embeddings (KGE). Traditional logical inference aims to find unobserved facts given a set of rules and the currently observed facts, which results in maximizing the number of rules that can be satisfied; on the other hand, KGE aims to capture the similarity of entities by embedding entities and relations into continuous low-dimensional vectors. With the learned embeddings, the KGE approach is able to compute a score for each triple in a KG that indicates the plausibility of the triple. We denote the score of a triple (e_i, r_k, e_j) computed by KGE as $f_{r_k}(e_i, e_j)$. Logical rules and KGE can mutually enhance the reasoning ability of each other in an iterative process [43]. To better model the interactions between embedding learning and logical inference, we will use a unified framework to integrate embeddings and First Order Logic Horn rules in an iterative manner called UniKER [20] developed by authors Cheng and Sun, referred as Reasoning over Knowledge Graph in Figure E.1. In particular, UniKER shows that the knowledge contained in Horn rules can be exploited and completely transferred them into the embeddings without the need for sampling ground rules or hidden triples in KGs. Additionally, UniKER can tolerate erroneous data and thus shows robustness to noise (extraneous facts) and errors (incorrect facts) in the KGs, which previous methods cannot cope with [85, 107].

The basic idea of the algorithm is as follows: we leverage the reasoning power of Horn rules by deriving a satisfying truth assignment $\mathbf{v}_H^{T^*}$ and $\mathbf{v}_H^{F^*}$ by a forward chaining algorithm, where $\mathbf{v}_H^{T^*} = \{r_k(e_i, e_j) = 1 \mid r_k(e_i, e_j) \in \mathbf{v}_H\}$ and $\mathbf{v}_H^{F^*} = \{r_k(e_i, e_j) = 0 \mid r_k(e_i, e_j) \in \mathbf{v}_H\}$. Knowledge contained in Horn rules is guaranteed to be fully exploited by taking $\mathbf{v}_H^{T^*}$ and $\mathbf{v}_H^{F^*}$ as guidance to optimize the KGE model. We summarize the iterative learning procedure as follows. We start by training a KGE model over the observed facts O. Next, following the forward chaining algorithm, we derive the triples set $\mathbf{v}_H^{T^*}$. In particular, each iteration consist of the following steps:

1. Considering that potential noise could affect the performance of forward chaining, we regard $\theta\%$ triples with lowest prediction scores $f_{r_k}(e_i, e_j)$ as noise and remove them from observed facts O to have a cleaner KG.
2. To relieve the data sparsity issue in real-world KGs, we also include potential useful hidden triples to enhance the reasoning ability of Horn rules. Then, we classify all triples in Δ and add the positive triples to O.
3. After that, by looking for the ground rules whose bodies are satisfied in O, we obtain a subset $\mathbf{v}_H^{T\,i*}$ of the satisfying truth assignment $\mathbf{v}_H^{T^*}$.
4. To reduce some of the uncertainty brought by logical rules, we utilize the trained KGE model to check the correctness of the inferred triples in $\mathbf{v}_H^{T\,i*}$ and exclude $\theta\%$ triples with lowest prediction scores $f_{r_k}(e_i, e_j)$ from $\mathbf{v}_H^{T\,i*}$.
5. After that, we continually train the KGE model over $\mathbf{v}_H^{T\,i*}$ and add $\mathbf{v}_H^{T\,i*}$ to KGs by setting $O = O \cup \mathbf{v}_H^{T\,i*}$.

The final triples obtained through UniKER are used to create a list of satellites and sensors within them that can participate in a given mission. This list

of satellites is then sent to the Sensor Planning Framework to compute whether they should participate in a mission or not.

E.2.4 Sensor Planning Framework

The Sensor Planning Framework is an analytic algorithm to compute an a priori error probability on sensing objectives (e.g., the probability of one sensor successfully detecting an event, or the probability of successfully detecting an event if multiple sensors join the mission) based on the contextual knowledge each agent has. We have developed a sensor planning algorithm to calculate the error probability of each participating sensor based on the specific estimation algorithm chosen. Given agents that decide to participate in the mission, the estimation algorithm computes an estimate of the mission objective (e.g. probability of an event $P(Event_t)$) using the agents' sensor capabilities.

To be more specific, it provides the certainty that our sensors are going to tell the truth about their own observation, i.e. the probability that a sensor successfully detects the corresponding observation of an event given the event happened, as well as the probabilities of a false positive, true negative, and false negative. The sensor planning algorithm assumes the process is linear and estimation of mission objectives are done through the estimation algorithm. In our case, we use a Kalman Filter series estimation algorithm such as the ones in [55] [120] [17], given their track record in satellites and other vehicles. Kalman filters can also be applied to multi-sensor data fusion [94] [15].

Thus, our method takes the following matrices as input for each sensor from the KG: Process Model:=A, B, Observation Model:= H, Covariance of

Measurement noise:=R, Covariance of undetermined/initial process noise:=Q. Then, the algorithm outputs the error bound of the estimation algorithm Σ and $P(Detected|Event)$ over time. The sensor teaming planning algorithm is run before observing the real-world data feed. The idea behind this algorithm is to gather information in order to create a team before it commits to taking the measurements for the estimation algorithm, thus avoiding resources and time waste.

Given observation data from a formed team (e.g. temperature, aerosols, and land displacements close to a volcano), the estimation part of the algorithm will compute the optimal estimated states of the sensors (e.g. near-optimal estimate of real land surface temperature). The estimation algorithm takes real-world measurements: observations of sensors z_t, system matrices of the sensors as mentioned in the previous paragraph and confidence factor w_i of each sensor as inputs, and outputs the probability of an event happening over time $P(Event_t)$, but this time with the real data behind it instead of just an apriori estimation.

The formal definition of the problem is given below:

Problem statement

Given A, B, H, Q, R of a sensor, we use this particular sensor to observe physical quantities for a long period of time (e.g. t = 100 days). Then, we assume that on the next time step $t + 1$, the sensor outputs either $u_t = 0$ *or* 1 (event does not happen or event happens). First, compute the probability that the sensor decides $u_t = 1$ given the fact that $u_t = 1$ (true positive). Then, compute the probability that the sensor decides $u_t = 0$ given the fact that $u_t = 1$ (false positive). Further,

compute the probability that the sensor decides $u_t = 0$ given the fact that $u_t = 0$ (true negative). Finally, compute the probability that the agent decides $u_t = 1$ given the fact that $u_t = 0$ (false negative).

Problem Clarification

The probability that we calculate here is actually a testing in a binary situation. Assuming that we have two hypotheses:

- $H_0 : u_t = 0$ (no events happen)
- $H_1 : u_t = 1$ (events happen)

There are four probabilities that need to be computed:

- H_1 is true, decide H_1 : given the H_1 is true hypothesis, decide H1 is in force: $Prob\{decide H_1 | H_1 true\} = P_D$ (probability of detection)
- H_0 is true, decide H_1 : given the H0 is true hypothesis, decide H_1 is in force: $Prob\{decide H_1 | H_0 true\} = P_{FA}$ (probability of false alarm)
- H_1 is true, decide H_0 : $Prob\{decide H_0 \mid H_1 true\} = P_M = 1 - P_D$ (probability of missing)
- H_0 is true, decide H_0: $Prob\{decide H_0 \mid H_0 true\} = P_{CR} = 1 - P_{FA}$ (probability of correct rejection)

Method and derivation

Consider the system model and the observation model:

- Prior: $x_t \mid z_{0:t}, u_{0:t-1} \sim N(\mu_{t|t}, \Sigma_{t|t})$
- System model: $x_{t+1} = Ax_t + Bu_t + w_t, w_t \sim N(0, Q)$, where $x_{t+1}|x_t \sim N(Ax_t + Bu_t, Q)$
- Observation model: $z_{t+1} = Hx_{t+1} + v_t, v_t \sim N(0, R)$, where $z_t|x_t \sim N(Hx_t, R)$

Using a Kalman filter as a linear quadratic optimal estimator, the Kalman prediction step to estimate state x and its likelihood is defined as follows:

- $\mu_{t+1|t} = A\mu_{t|t} + Bu_t$
- $\Sigma_{t+1|t} = A\Sigma_{t|t}A^T + Q$

Then, we define the update step as follows:

- $\mu_{t+1|t+1} = \mu_{t+1|t} + K_{t+1|t}(z_{t+1} - H\mu_{t+1|t})$
- $\Sigma_{t+1|t+1} = (I - K_{t+1|t}H)\Sigma_{t+1|t}$

where the Kalman gain is: $K_{t+1|t} = \Sigma_{t+1|t}H^T(H\Sigma_{t+1|t}H^T + R)^{-1}$

In the Kalman filter, the objective is to calculate:

$$p_{t+1|t+1}(x) = \frac{p(z_{t+1}|x)p_{t+1|t}(x)}{p(z_{t+1}|z_{0:t}, u_{0:t})}$$

In our method, the objective is to calculate uncertainty of the input in the last time step u_t. More specifically, given a binary candidate input sequences $u_t^j = \{0, 1\}, j \in \{1, 2\}$, evaluate the likelihood of each input over time given only observations $z_{0:t+1}$. Therefore, the error probability function that we try to compute is $pdf(u_t|z_{0:t+1}, x_{0:t}, u_{0:t-1})$, denote it as $pdf(U)$.

$$pdf(u_t|z_{0:t+1}, x_{0:t}, u_{0:t-1}) = pdf(U) = z_{t+1} - HA\mu_{t|t} = z_{t+1} - H\mu_{t+1|t}$$

In the Kalman filter, $pdf(U)$ is also Gaussian under each hypothesis and its distribution is as follows:

$$U(z_{t+1}) \sim \begin{cases} N(\mu_{u^1}, \Sigma_{u^1}) & under \quad H_0 \\ N(\mu_{u^2}, \Sigma_{u^2}) & under \quad H_1 \end{cases}$$

where,

$\mu_{u_t} = \mathbf{E}[z_{t+1} - H\mu_{t+1|t}] = \mathbf{E}[Hx_{t+1} + v_t - H\mu_{t+1|t}] = \mathbf{E}[HA\mu_{t|t} + v_t - H(A\mu_{t|t} + Bu_t)] = \mathbf{E}[v_t + HBu_t] = HB\mathbf{E}[u_t]$

$\Sigma_{u_t} = \mathbf{Var}[z_{t+1} - H\mu_{t+1|t}] \overset{indepen}{=} \mathbf{Var}[z_{t+1}] = \mathbf{Var}[Hx_{t+1} + v_t] = H\Sigma_{t+1|t}H^{-1} + R = H(A\Sigma_{t|t}A^T + Q)H^{-1} + R$

Therefore, the error probabilities P_D, P_{FA}, P_M, P_{CR} can be calculated using the following equations:

$P_D = \int_\lambda^\infty pdf_{H_1}(U) = \int_\lambda^\infty \mathbf{N}(\mu_{u^2}, \Sigma_{u^2})$

$P_{FA} = \int_\lambda^\infty pdf_{H_0}(U) = \int_\lambda^\infty \mathbf{N}(\mu_{u^1}, \Sigma_{u^1})$

$P_M = 1 - P_D$

$P_{CR} = 1 - P_{FA}$

where, λ is when $\mathbf{N}(\mu_{u^2}, \Sigma_{u^2}) = \mathbf{N}(\mu_{u^1}, \Sigma_{u^1})$

For each single sensor that observes a single geophysical property, we can use the above method to calculate their error probabilities respectively. In the case of a sensor with more than one measurement, we assume it can be split into multiple sensors so the former is always valid.

The results of this portion of our approach are used to inform the creation of

teams by either the fully decentralized approach or the team synthesis approach described in the following section. (E.2.6).

E.2.5 Verification and Synthesis of Team Assignment

Formal verification has been used in a wide range of applications to verify the safety and correctness of systems with respect to given specifications [10] [75] [72]. In our system, the aim of the verification step is to provide information about the likelihood of mission success given a teaming assignment and sensor properties. Individual agents decide whether to participate or not, but a centralized decision needs to be made to ensure that the mission constraints are met at a team level. If it is determined that the mission is likely to fail, the Sensor Planning Framework then assigns a new team of agents.

Alternatively, the centralized node can also create a teaming plan using synthesis. Formal synthesis provides a framework for automatically translating high-level specifications into correct-by-construction controllers. The user is able to reason about the task specification rather than the actual implementation [61]. The aim of the synthesis step is to create an optimal teaming plan given a list of possible agents and their capabilities. This method provides a globally optimal teaming solution.

There are three main steps in performing synthesis for the mission:

1. **Encoding the mission as a formal specification.** Specifications are often written using temporal logic [26], which allows us to reason about requirements and constraints that change over time. In particular, we use prob-

abilistic computation tree logic (PCTL) [45] to write the specification. The specification can also include constraints on the system (e.g. sensor interference).

2. **Creating a system model.** The assigned team is modeled as a Markov Decision Process (MDP). Each state within the MDP contains information about which agents and sensors are used, as well as which observations are measured.

   The uncertainty in the sensor readings are reented as probabilities, which change over time as the Sensor Planning Framework receives more measurements to improve the sensors' state estimations. From these probabilities, we calculate the transitions between states in the MDP. The transitions reent the probabilities that a specific combination of sensors measure the necessary observations are measured at a given timestep.

   Also encoded in the MDP is the physical behavior of the agents. For example, in our use case, the target volcano is only visible to the satellites periodically.

3. **Verifying the decentralized team.** Once the mission specification and system model is created, we use PRISM [63], a probabilistic model checker, to calculate the probability of success of the decentralized teaming plan. If this probability falls below a designated threshold, the verification step requests a new team assignment from the Sensor Planning Framework.

4. **Synthesizing the optimal team.** Optionally, the centralized node can synthesize a globally optimal team from a list of potential agents. To do so, the model checker maximizes the probabilities of the MDP state transitions, while taking into account any constraints given in the specification. If the mission contains multiple objectives (e.g. maximize the probabil-

ity of mission success while minimizing the number of satellites), PRISM outputs the Pareto front for all viable teams.

E.2.6 Teaming Algorithm

The teaming decision, as explained in the introduction, can be done through two approaches, depending on whether centralization of decisions is acceptable. Both of them start with mission control sending a mission to all satellites. Each satellite and sensor comes up with a probability of completing the mission successfully (aka detecting the event in our case), as described in Section E.2.4. In the decentralized case, if they clear a predetermined threshold they decide to participate. In the centralized approach, a central node receives the probabilities from each satellite and performs a greedy optimization based on the success probability for the chosen team, creating a Pareto front of teams of different sizes and their respective probability of success.

E.3 Results

In order to validate our complete approach to the planning problem, we compare the teaming results of our approach to the team described in [22]. This comparison can be seen in Figure E.3, where each dot is a team with their daily assignment of satellites (e.g., Day 1: Sentinel-1, EO-1, Day 2: GOES-13) (which we will call **Teaming Strategies** going forward), as outputted by the Verification module . The X axis reents the mean number of satellites that have to be used per day, and the Y axis is the maximum computed probability of mission success

based off the results of our system. In order to ensure that the scores are fair to each team, each dot is computed as the average of the teaming strategy applied to 50 random missions generated by a Monte-Carlo approach. Thus, the blue dots reent the best teaming strategies by tasking the whole Earth Observation constellation currently on orbit to figure out a plan for our random missions, while the red dots are the best teaming strategies our approach returns when given the choice of only using the satellites in the benchmark team for the same random missions.

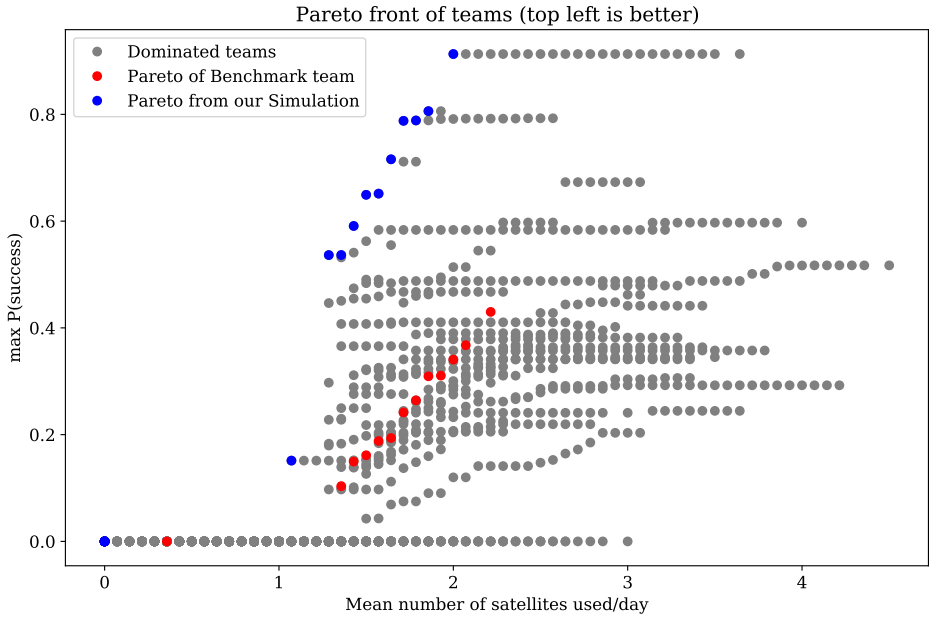


Figure E.3: Probability of mission success as a function of mean number of satellites used per day for various teaming strategies. Results demonstrate that our approach is capable of creating teams and daily assignments out of those teams that compete and even improve upon expert-designed teams.

The mission assigned to our teams and the benchmark team is the same, and that is to observe at least 2 out of 4 parameters (MWIR, TIR, SAR, and Cloud Cover) every day for two weeks in a random location of Earth with high volcanic activity.

We observe that the best teaming strategies –in terms of the trade-off between probability of success and satellite usage– generated by our approach (the blue dots) dominate the teaming strategies generated from the benchmark team at NASA we described (the red dots). Therefore, our approach appears to create better teams and teaming strategies than the benchmark team that is already in use – according to our own metrics, which may be different from the ones used by NASA to make this decision.

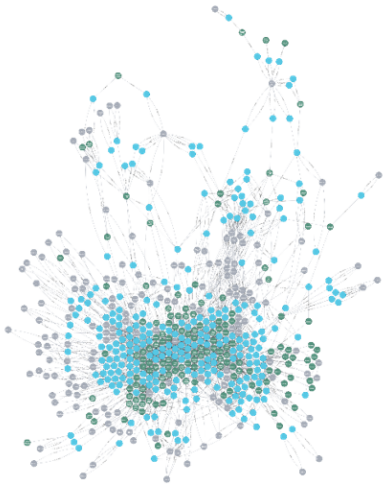
Additionally, we provide some intermediate results showing the outputs of the different subsystems in this method.

E.3.1 Results from Knowledge Representation

The main result of this subsystem is an initial Probabilistic Knowledge Base, which can be reented either as triples or as part of a Graph Database, as seen in Figure E.4. The knowledge base has a total of 2460 nodes and 12500 relationships. Specifically, there are 635 satellites, 927 sensors, 168 Measurements, 519 Types of Sensors, and 169 Agencies, among others. The relationships are between sensors, satellites, measurements, events, with some examples being what types of sensors can measure what properties (with 3403 of those), and which measurements can be used to create higher level measurements and data products (a count of 519), as well as relations between events and their observable properties. For example, it finds that TIR Imaging multi-spectral radiometers can generally measure Earth's surface reflectance.

If we include the results from the knowledge extraction pipeline, there is an increase of nodes in the knowledge graph from 2460 to 3063 (an increase of

24.5%, with new measurements and properties leading the increase), while the number of relationships between nodes has increased from 12500 to 14500 (an increase of 16%, mostly in the field of what sensors can measure what).



```
JPSS-1|VIIRS    isInstanceOf    VIIRS
JPSS-1  HOSTS   JPSS-1|ATMS
JPSS-1|ATMS isInstanceOf    ATMS
JPSS-1  HOSTS   JPSS-1|CERES
JPSS-1|CERES    isInstanceOf    CERES
JPSS-1  HOSTS   JPSS-1|CrIS
JPSS-1|CrIS isInstanceOf    CrIS
JPSS-1  HOSTS   JPSS-1|OMPS
JPSS-1|OMPS isInstanceOf    OMPS
MODIS   OBSERVES    Earth surface albedo
MODIS   OBSERVES    Cloud optical depth
HiRI    OBSERVES    Land surface imagery
CSG SAR OBSERVES    Snow cover
IVM OBSERVES    Ion Density, Drift Velocity, and Temperature
IASI    OBSERVES    Ozone profile
VIIRS   OBSERVES    Cloud top height
Vegetation  OBSERVES    Vegetation type
```

Figure E.4: Excerpts from the Graph and Triples databases, showing information on the satellites and sensors currently in orbit

E.3.2 Results from Reasoning over KG

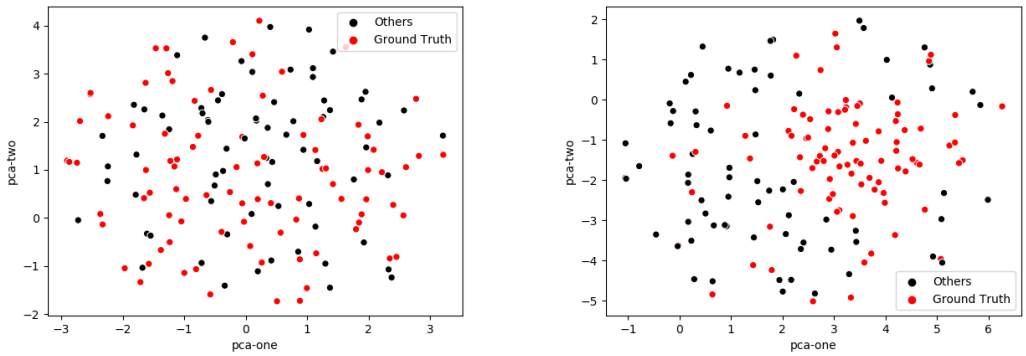
The main result of this subsystem is a list of candidate satellites that can participate, i.e., that are potentially useful for the given mission. There are 161 active EO satellites orbiting Earth right now in the CEOS database. For the given mission "Active Volcano Monitoring", UniKER gives a total of 113 satellites among them as candidate satellites as far as sensor capabilities go. To validate our prediction, we compare our prediction with a handcrafted ground truth based on results from the CEOS database and the accesses of the satellite over the Kīlauea volcano. 89 satellites are identified in this ground truth as potentially useful satellites for our example mission from those in the CEOS database, according

to their status as currently active, a path in the KG connecting their sensors to the observable properties (e.g. a TIR sensor can measure TIR Radiance - TIR radiance is generated by heat - a volcano eruption generates heat), as well as flying over the volcano at the right times. UniKER's logic prediction is able to find all 89 satellites given that all relationships linking measurements and events are ent in the KG.

In addition, as our approach combines embedding and logical rules for better KG inference, in order to show the enhancement of logical rules over embeddings we compare our proposed framework with TransE. TransE is the most reentative knowledge graph embedding model, which learns graph embeddings without the incorporation of logical rules. As shown in Figure E.5, we have plotted the two principal components of the learned embeddings for all satellites. In particular, the red nodes reent the potential useful satellites given in our ground truth while the black nodes reent other satellites in the KG. We can observe that with only the embedding model, it is difficult to distinguish the useful satellites from the others, while with the help of logical rules, the quality of embeddings is improved dramatically.

E.3.3 Results from Sensor Planning Framework

The sensor planning framework outputs the probability of each participating satellite successfully detecting an event given the event happened (true positives), as well as the probabilities of a false positive, true negative, and false negative. To show an example of this intermediate result, we use the sensor planning algorithm to calculate the $p(Detected|EventHappens)$ for NASA's



(a) Result for TransE

(b) Result for Our Proposed Framework

Figure E.5: Visualization of a Principal Component Analysis of Embedding Vectors, showing that UniKER is capable of distinguishing useful satellites better than TransE.

benchmark team in Table E.1. We can see that, as we would expect from pieces of technology that usually cost millions of dollars, most sensors are good at what they do. This seems to prove that our system can accurately predict the probabilities we have mentioned.

E.3.4 Results from Synthesis of Teaming Assignment

The synthesis subsystem outputs the probability of mission success and the optimal team given information from the Sensor Planning Framework and the KG. We generate a Pareto front of two objectives: minimizing the number of satellites and maximizing the max probability of mission success, where the mission is to monitor any eruptions of the Kīlauea volcano over a two-week time frame. To reduce the computation time, the process is parallelized to synthesize a Pareto front for each timestep, which is then aggregated into an overall plot (see Fig. E.6). To cover the entire length of the mission, we generate a sample

Table E.1: Benchmark Team From NASA and the computed probabilities of event detection from the sensors in each satellite

Satellites	Sensors	Observation	$p(Detected\|EventHappens)$
Aqua	AIRS	Land surface temperature	0.99899652
	MODIS	Land surface temperature	0.99899652
		Fire temperature	0.99999966
	AMSU-A	Land surface temperature	0.99899652
		Cloud type	0.84130652
Terra	ASTER	Land surface topography	1.
		Land surface temperature	0.99899652
		Cloud type	0.99899652
	MODIS	Land surface temperature	0.99899652
		Fire temperature	0.99999966
Sentinel-1 A	C-Band SAR	Land surface topography	0.99899652
Sentinel-1 B	C-Band SAR	Land surface topography	0.99899652
GOES-15	Imager	Land surface temperature	0.99122158
		Fire temperature	0.99999882
GOES-17	ABI	Fire temperature	0.99999966
		Land surface temperature	0.99899652
		Cloud type	0.84130652

of all possible teaming combinations of the parallelized daily results, which are shown in gray. The red points reent the approximate Pareto front found from the sample.

The Pareto front provides a way to compare the benchmark team to the final team our system outputs. While the objective is to maximize probability, the other objective of the Pareto front can change depending on the priorities of the mission (e.g. minimizing energy usage instead of number of satellites, or some more complicated utility function).

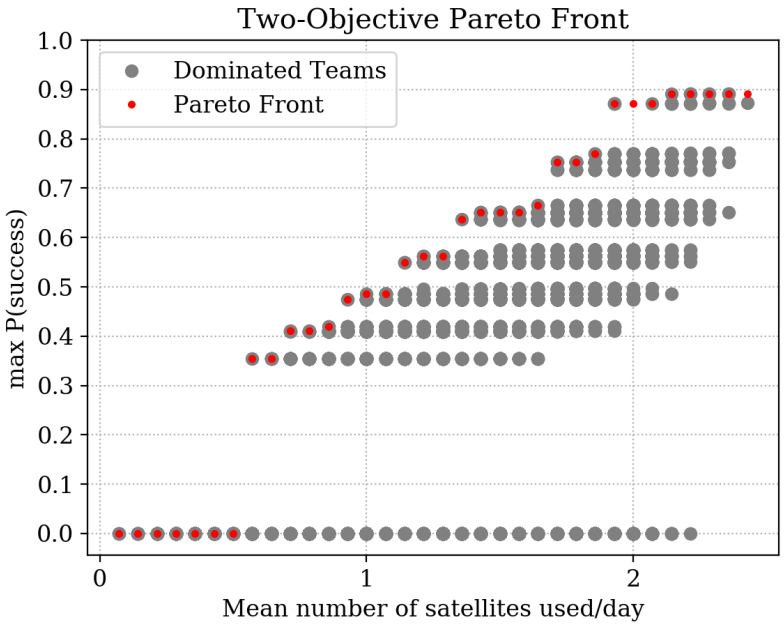


Figure E.6: Pareto front for a mission to monitor any eruptions of the Kīlauea volcano over two weeks, given a list of potential satellites. Each point on the plot corresponds to a different synthesized teaming plan.

E.4 Conclusion

This paper described a new decentralized, context-based, on-board planning algorithm for future EO systems such as sensor webs, smart heterogeneous constellations, or federated systems. We explained in detail the three steps of our method, where each satellite first decides whether it can participate, then if it should participate, and finally a central node creates and formally verifies a good (potentially optimal) team for the mission. Two different teaming algorithms were described, based on the desire for more optimal centralized synthesis. Initial results for each step as well as the whole method were provided. These initial results show that our approach can create teams with a higher probability of success while using less satellites daily to perform a volcano monitoring mission than currently available systems that perform the same task.

The main limitation of this paper is the lack of more valid benchmarks and comparison systems to characterize the performance of the proposed approach. Although the comparison described in the Results section is a good first step, a more exhaustive performance benchmark is necessary. In order to further validate our system, we plan to create a benchmark platform with missions and benchmark teams against which our approach can be compared to the approaches mentioned in the introduction. We are particularly interested in comparing our system to other decentralized approaches such as the ACBBA-based approach described in [65], as well as the modified CBBA approach in [37].

A further limitation of the approach is that when it checks whether a satellite should participate in a mission, it does not check what that would imply for the current mission being performed by that satellite. We plan to extend the approach to a multi-mission framework in the future in order to address this limitation, as well as allow for different types of mission, such as multiple event detection, instead of the current single event detection approach. For example, we would like to test the system with a meta-mission to detect multiple natural disasters instead of just limiting it to a single one, like a Volcano Eruption. This requires expansions in all parts of our approach, among other things to support the testing of multiple simultaneous hypotheses.

In addition, in order to validate the a priori probabilities of success that are given as an output of our approach, we plan to enhance our current Monte-Carlo simulation system to include timelines for each of the physical observations for each random event and check whether the system's is capable of detecting the correct percentage of events (i.e. if the approach says a certain team will detect an event 90% of the time, it should detect 9/10 events).

Finally, we plan on further expanding the text mining capabilities to keep increasing the size of the KG, as well as incorporating more complex constraints on the mission specification that involve spacecraft resources, which are assumed as infinite and perfect right now. Besides, we plan on adding a capability for the system to understand and exploit sensor synergies. Furthermore, our team synthesis approach has two steps where a random algorithm is used due to the time complexity of the optimal approach. These will be improved upon by using algorithms currently available in the literature such as Tournament Selection and Genetic Algorithms.

BIBLIOGRAPHY

[1] R. Alur, T.A. Henzinger, G. Lafferriere, and G.J. Pappas. Discrete abstractions of hybrid systems. *Proceedings of the IEEE*, 88(7):971–984, 2000.

[2] Hong an Yang, Yuhua Li, Xin Duan, Gaopan Shen, and Shaohua Zhang. A parallel shape formation method for swarm robotics. *Robotics and Autonomous Systems*, 151:104043, 2022.

[3] Stephen H. Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. Hinge-loss markov random fields and probabilistic soft logic. *J. Mach. Learn. Res.*, 18(1):3846–3912, January 2017.

[4] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[5] Jan Kristof Behrens, Ralph Lange, and Masoumeh Mansouri. A constraint programming approach to simultaneous task allocation and motion scheduling for industrial dual-arm manipulation tasks. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8705–8711. IEEE, 2019.

[6] Tolga Bektas. The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega*, 34(3):209–219, 2006.

[7] Iz Beltagy, Kyle Lo, and Arman Cohan. Scibert: Pretrained language model for scientific text. In *EMNLP*, 2019.

[8] Michael Berger, Jose Moreno, Johnny A. Johannessen, Pieternel F. Levelt, and Ramon F. Hanssen. Esa's sentinel missions in support of earth system science. *Remote Sensing of Environment*, 120:84 – 90, 2012. The Sentinel Missions - New Opportunities for Science.

[9] Christopher Boshuizen, James Mason, Pete Klupar, and Shannon Spanhake. Results from the planet labs flock constellation. 2014.

[10] A. Boteanu, T. Howard, J. Arkin, and H. Kress-Gazit. A model for verifiable grounding and execution of complex natural language instructions. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2649–2654, 2016.

[11] Sara Bouraine, Thierry Fraichard, and Hassen Salhi. Provably safe navigation for mobile robots with limited field-of-views in unknown dynamic environments. In *2012 IEEE International Conference on Robotics and Automation*, pages 174–179, 2012.

[12] Kris Braekers, Katrien Ramaekers, and Inneke Van Nieuwenhuyse. The vehicle routing problem: State of the art classification and review. *Computers & Industrial Engineering*, 99:300–313, 2016.

[13] Ali Tevfik Buyukkocak, Derya Aksaray, and Yasin Yazıcıoğlu. Planning of heterogeneous multi-agent systems under signal temporal logic specifications with integral predicates. *IEEE Robotics and Automation Letters*, 6(2):1375–1382, 2021.

[14] Gustavo A Cardona, Kevin Leahy, and Cristian-Ioan Vasile. Temporal logic swarm control with splitting and merging. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 12423–12429. IEEE, 2023.

[15] Tsang-Kai Chang and Ankur Mehta. Control-theoretical and topological analysis of covariance intersection based distributed kalman filter. *IEEE Control Systems Letters*, 2(4):665–670, oct 2018.

[16] Omar Cheikhrouhou and Ines Khoufi. A comprehensive survey on the multiple traveling salesman problem: Applications, approaches and taxonomy. *Computer Science Review*, 40:100369, 2021.

[17] Guanrong Chen, Jianrong Wang, and Leang S Shieh. Interval kalman filtering. *IEEE Transactions on Aerospace and electronic Systems*, 33(1):250–259, 1997.

[18] Ji Chen, Ruojia Sun, and Hadas Kress-Gazit. Distributed control of robotic swarms from reactive high-level specifications. In *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, pages 1247–1254, 2021.

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

[20] Kewei Cheng, Ziqing Yang, Ming Zhang, and Yizhou Sun. Uniker: A

unified framework for combining embedding and horn rules for knowledge graph inference. In *Proceedings of the Graph Representation Learning and Beyond Workshop (ICML GRL+ 2020)*, 2020.

[21] Steve Chien, Rob Sherwood, Daniel Tran, Benjamin Cichy, Gregg Rabideau, Rebecca Castano, Ashley Davies, Rachel Lee, Dan Mandl, Stuart Frye, et al. The eo-1 autonomous science agent. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 420–427. Citeseer, 2004.

[22] Steve A. Chien, Ashley G. Davies, Joshua Doubleday, Daniel Q. Tran, David Mclaren, Wayne Chi, and Adrien Maillard. Automated volcano monitoring using multiple space and ground sensors. *Journal of Aerospace Information Systems*, 17(4):214–228, 2020.

[23] Han-Lim Choi, Luc Brunet, and Jonathan P How. Consensus-based decentralized auctions for robust task allocation. *IEEE transactions on robotics*, 25(4):912–926, 2009.

[24] Han-Lim Choi, Andrew K. Whitten, and Jonathan P. How. Decentralized task allocation for heterogeneous teams with cooperation constraints. In *Proceedings of the 2010 American Control Conference*, pages 3057–3062, 2010.

[25] E Clarke, O Grumberg, and D Peled. *Model Checking*. The MIT Press, 2000.

[26] Edmund Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT Press: Cambridge, Massachusetts, 01 1999.

[27] L. Di, K. Moe, and T. L. van Zyl. Earth observation sensor web: An overview. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 3(4):415–417, 2010.

[28] Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, and Laurent Xu. Spot 2.0 — a framework for LTL and ω-automata manipulation. In *Proceedings of the 14th International Symposium on Automated Technology for Verification and Analysis (ATVA'16)*, volume 9938 of *Lecture Notes in Computer Science*, pages 122–129. Springer, October 2016.

[29] E. Allen Emerson. Temporal and modal logic. In JAN Van Leeuwen, editor, *Formal Models and Semantics*, Handbook of Theoretical Computer Science, pages 995–1072. Elsevier, Amsterdam, 1990.

[30] Amy Fang and Hadas Kress-Gazit. Automated task updates of temporal logic specifications for heterogeneous robots. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 4363–4369, 2022.

[31] Amy Fang and Hadas Kress-Gazit. High-level, collaborative task planning grammar and execution for heterogeneous agents. In *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '24, page 544–552, Richland, SC, 2024. International Foundation for Autonomous Agents and Multiagent Systems.

[32] Amy Fang, Tenny Yin, Jiawei Lin, and Hadas Kress-Gazit. Continuous execution of high-level collaborative tasks for heterogeneous robot teams. *arXiv preprint arXiv:2406.18019*, 2024.

[33] Fatma Faruq, David Parker, Bruno Laccrda, and Nick Hawes. Simultaneous task allocation and planning under uncertainty. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3559–3564, 2018.

[34] Ioannis Filippidis, Dimos V. Dimarogonas, and Kostas J. Kyriakopoulos. Decentralized multi-agent control from local ltl specifications. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pages 6235–6240, 2012.

[35] C. E. Fossa, R. A. Raines, G. H. Gunsch, and M. A. Temple. An overview of the iridium (r) low earth orbit (leo) satellite system. In *Proceedings of the IEEE 1998 National Aerospace and Electronics Conference. NAECON 1998. Celebrating 50 Years (Cat. No.98CH36185)*, pages 152–159, 1998.

[36] Virginie Gabrel and Cécile Murat. *Mathematical Programming for Earth Observation Satellite Mission Planning*, pages 103–122. Springer US, Boston, MA, 2003.

[37] Ximo Gallud and Daniel Selva. Agent-based simulation framework and consensus algorithm for observing systems with adaptive modularity. *Systems Engineering*, 21(5):432–454, 2018.

[38] Brian P. Gerkey and Maja J. Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research*, 23(9):939–954, 2004.

[39] Alessandro Golkar and Ignasi Lluch. The federated satellite systems

paradigm: Concept and business case evaluation. *Acta Astronautica*, 111:230 – 248, 2015.

[40] David Gundana and Hadas Kress-Gazit. Event-based signal temporal logic synthesis for single and multi-robot tasks. *IEEE Robotics and Automation Letters*, 6(2):3687–3694, 2021.

[41] Tyler Gunn and John Anderson. Dynamic heterogeneous team formation for robotic urban search and rescue. *J. Comput. Syst. Sci.*, 81(3):553–567, May 2015.

[42] Meng Guo, Karl H. Johansson, and Dimos V. Dimarogonas. Revising motion planning under linear temporal logic specifications in partially known workspaces. In *2013 IEEE International Conference on Robotics and Automation*, pages 5025–5032, 2013.

[43] Shu Guo, Quan Wang, Lihong Wang, Bin Wang, and Li Guo. Knowledge graph embedding with iterative guidance from soft rules, 2017.

[44] Eduardo Guzman, Beatriz Andres, and Raul Poler. Models and algorithms for production planning, scheduling and sequencing problems: A holistic framework and a systematic review. *Journal of Industrial Information Integration*, 27:100287, 2022.

[45] Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6, 02 1995.

[46] Pierre-Emmanuel Hladik, Hadrien Cambazard, Anne-Marie Déplanche, and Narendra Jussien. Solving a real-time allocation problem with constraint programming. *Journal of Systems and Software*, 81(1):132–149, 2008.

[47] T. Holvoet, D. Weyns, N. Boucke, and B. Demarsin. Dyncnet: A protocol for dynamic task assignment in multiagent systems. In *2007 First International Conference on Self-Adaptive and Self-Organizing Systems*, pages 281–284, Los Alamitos, CA, USA, jul 2007. IEEE Computer Society.

[48] Matthew Honnibal and Ines Montani. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear, 2017.

[49] John N Hooker. Planning and scheduling by logic-based benders decomposition. *Operations research*, 55(3):588–602, 2007.

[50] Feifei Huang, Xiang Yin, and Shaoyuan Li. Failure-robust multi-robot tasks planning under linear temporal logic specifications. In *2022 13th Asian Control Conference (ASCC)*, pages 1052–1059, 2022.

[51] Ioana Hustiu, Marius Kloetzer, and Cristian Mahulea. Distributed path planning of mobile robots with ltl specifications. In *2020 24th International Conference on System Theory, Control and Computing (ICSTCC)*, pages 60–65, 2020.

[52] Xiao Jia and Max Q.-H. Meng. A survey and analysis of task allocation algorithms in multi-robot systems. In *2013 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 2280–2285, 2013.

[53] Austin M Jones, Kevin Leahy, Cristian Ioan Vasile, Sadra Sadradinni, Zachary Serlin, Roberto Tron, and Calin Belta. Scalable and Robust Deployment of Heterogenenous Teams from Temporal Logic Specifications. In *International Symposium on Robotics Research (ISRR)*, Hanoi, Vietnam, October 2019.

[54] Samarth Kalluraya, George J. Pappas, and Yiannis Kantaros. Resilient temporal logic planning in the presence of robot failures. In *62nd IEEE Conference on Decision and Control, CDC 2023, Singapore, December 13-15, 2023*, pages 7520–7526. IEEE, 2023.

[55] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[56] Yiannis Kantaros and Michael M Zavlanos. Stylus*: A temporal logic optimal control synthesis algorithm for large-scale multi-robot systems. *The International Journal of Robotics Research*, 39(7):812–836, 2020.

[57] Marius Kloetzer and Calin Belta. Automatic deployment of distributed teams of robots from temporal logic motion specifications. *IEEE Transactions on Robotics*, 26(1):48–61, 2010.

[58] M. Klusch and A. Gerber. Dynamic coalition formation among rational agents. *IEEE Intelligent Systems*, 17(3):42–47, 2002.

[59] Savas Konur, Clare Dixon, and Michael Fisher. Formal verification of probabilistic swarm behaviours. In Marco Dorigo, Mauro Birattari, Gianni A. Di Caro, René Doursat, Andries P. Engelbrecht, Dario Floreano,

Luca Maria Gambardella, Roderich Groß, Erol Şahin, Hiroki Sayama, and Thomas Stützle, editors, *Swarm Intelligence*, pages 440–447, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[60] G. Ayorkor Korsah, Anthony Stentz, and M. Bernardine Dias. A comprehensive taxonomy for multi-robot task allocation. *The International Journal of Robotics Research*, 32(12):1495–1512, 2013.

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

[62] Hanna Kurniawati and Vinay Yadav. An online pomdp solver for uncertainty planning in dynamic environment. In *Robotics Research: The 16th International Symposium ISRR*, pages 611–629. Springer, 2016.

[63] Marta Kwiatkowska, Gethin Norman, and David Parker. Prism 4.0: Verification of probabilistic real-time systems. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Computer Aided Verification*, pages 585–591, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[64] Kevin Leahy, Zachary Serlin, Cristian-Ioan Vasile, Andrew Schoer, Austin M. Jones, Roberto Tron, and Calin Belta. Scalable and robust algorithms for task-based coordination from high-level specifications (scratches). *IEEE Transactions on Robotics*, 38(4):2516–2535, 2022.

[65] Guoliang Li. Online scheduling of distributed earth observation satellite system under rigid communication constraints. *Advances in Space Research*, 65(11):2475 – 2496, 2020.

[66] Guoliang Li, Lining Xing, and Yingwu Chen. A hybrid online scheduling mechanism with revision and progressive techniques for autonomous earth observation satellite. *Acta Astronautica*, 140:308 – 321, 2017.

[67] Zhiyong Li, Bo Xu, Lei Yang, Jun Chen, and Kenli Li. Quantum evolutionary algorithm for multi-robot coalition formation. In *Proceedings of the First ACM/SIGEVO Summit on Genetic and Evolutionary Computation*, GEC '09, page 295–302, New York, NY, USA, 2009. Association for Computing Machinery.

[68] Lars Lindemann, George J Pappas, and Dimos V Dimarogonas. Reactive and risk-aware control for signal temporal logic. *IEEE Transactions on Automatic Control*, 67(10):5262–5277, 2021.

[69] Martina Lippi and Alessandro Marino. A mixed-integer linear programming formulation for human multi-robot task allocation. In *2021 30th IEEE International Conference on Robot & Human Interactive Communication (RO-MAN)*, pages 1017–1023, 2021.

[70] Wenliang Liu, Kevin Leahy, Zachary Serlin, and Calin Belta. Robust multi-agent coordination from catl+ specifications, 2023.

[71] Zhong Liu, Xiao-Guang Gao, and Xiao-Wei Fu. Coalition formation for multiple heterogeneous uavs cooperative search and prosecute with communication constraints. In *2016 Chinese Control and Decision Conference (CCDC)*, pages 1727–1734, 2016.

[72] Matt Luckcuck, Marie Farrell, Louise A. Dennis, Clare Dixon, and Michael Fisher. Formal specification and verification of autonomous robotic systems: A survey. *ACM Comput. Surv.*, 52(5), September 2019.

[73] Xusheng Luo and Michael M. Zavlanos. Temporal logic task allocation in heterogeneous multirobot systems. *IEEE Transactions on Robotics*, pages 1–20, 2022.

[74] Lilyana Mihalkova and Raymond J Mooney. Bottom-up learning of markov logic network structure. In *Proceedings of the 24th international conference on Machine learning*, pages 625–632, 2007.

[75] Stefan Mitsch, Khalil Ghorbal, David Vogelbacher, and André Platzer. Formal verification of obstacle avoidance and navigation of ground robots. *The International Journal of Robotics Research*, 36(12):1312–1340, 2017.

[76] Salar Moarref and Hadas Kress-Gazit. Decentralized control of robotic swarms from high-level temporal logic specifications. In *2017 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*, pages 17–23, 2017.

[77] Alessio Mosca, Cristian-Ioan Vasile, Calin Belta, and Davide M. Raimondo. Multi-robot routing and scheduling with temporal logic and synchronization constraints. In *Proceedings of the 2019 2nd International Conference on Control and Robot Technology*, ICCRT 2019, page 40–45, New York, NY, USA, 2019. Association for Computing Machinery.

[78] J Muylaert, R Reinhard, C Asma, J Buchlin, P Rambaud, and M Vetrano. Qb50: an international network of 50 cubesats for multi-point, in-situ

measurements in the lower thermosphere and for re-entry research. In *ESA Atmospheric Science Conference, Barcelona, Spain*, pages 7–11, 2009.

[79] Fatma A. Omara and Mona M. Arafa. Genetic algorithms for task scheduling problem. *Journal of Parallel and Distributed Computing*, 70(1):13–22, 2010.

[80] Dung Phan, Junxing Yang, Denise Ratasich, Radu Grosu, Scott A. Smolka, and Scott D. Stoller. Collision avoidance for mobile robots with limited sensing and limited information about the environment. In Ezio Bartocci and Rupak Majumdar, editors, *Runtime Verification*, pages 201–215, Cham, 2015. Springer International Publishing.

[81] Laxmi Poudel, Wenchao Zhou, and Zhenghui Sha. Resource-Constrained Scheduling for Multi-Robot Cooperative Three-Dimensional Printing. *Journal of Mechanical Design*, 143(7), 04 2021. 072002.

[82] G. Qu and N. Li. Exploiting fast decaying and locality in multi-agent mdp with tree dependence structure. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pages 6479–6486, 2019.

[83] Vasumathi Raman, Cameron Finucane, and Hadas Kress-Gazit. Temporal logic robot mission planning for slow and fast actions. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 251–256, 2012.

[84] Joshua D. Redding, N. Kemal Ure, Jonathan P. How, Matthew A. Vavrina, and John Vian. Scalable, mdp-based planning for multiple, cooperating agents. In *2012 American Control Conference (ACC)*, pages 6011–6016, 2012.

[85] Tim Rocktäschel, Sameer Singh, and Sebastian Riedel. Injecting logical background knowledge into embeddings for relation extraction. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1119–1129, 2015.

[86] Yunus Emre Sahin, Petter Nilsson, and Necmiye Ozay. Synchronous and asynchronous multi-agent coordination with cltl+ constraints. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 335–342, 2017.

[87] Hannes Schabauer, Erich Schikuta, and Thomas Weishaupl. Solving very

large traveling salesman problems by som parallelization on cluster architectures. In *Sixth International Conference on Parallel and Distributed Computing Applications and Technologies (PDCAT'05)*, pages 954–958. IEEE, 2005.

[88] Philipp Schillinger, Mathias Bürger, and Dimos V. Dimarogonas. Hierarchical ltl-task mdps for multi-agent coordination through auctioning and learning. *The international journal of robotics research*, 2019.

[89] Philipp Schillinger, Mathias Bürger, and Dimos V. Dimarogonas. Simultaneous task allocation and planning for temporal logic goals in heterogeneous multi-robot systems. *The International Journal of Robotics Research*, 37(7):818–838, 2018.

[90] Thomas Schmickl, Christoph Möslinger, and Karl Crailsheim. Collective perception in a robot swarm. In *International Workshop on Swarm Robotics*, pages 144–157. Springer, 2006.

[91] D. Selva and E. F. Crawley. Vassar: Value assessment of system architectures using rules. In *2013 IEEE Aerospace Conference*, pages 1–21, 2013.

[92] Travis Service and Julie Adams. Coalition formation for task allocation: Theory and algorithms. *Autonomous Agents and Multi-Agent Systems*, 22:225–248, March 2011.

[93] N E Shlyakhov, I V Vatamaniuk, and A L Ronzhin. Survey of methods and algorithms of robot swarm aggregation. *Journal of Physics: Conference Series*, 803(1):012146, jan 2017.

[94] Shuli Sun. Multi-sensor optimal information fusion kalman filter for discrete multichannel arma signals. In *Proceedings of the 2003 IEEE International Symposium on Intelligent Control*, pages 377–382, 2003.

[95] Himani Sinhmar and Hadas Kress-Gazit. Guaranteed encapsulation of targets with unknown motion by a minimalist robotic swarm. *IEEE Transactions on Robotics*, 40:816–830, 2024.

[96] Alisa Smirnova and Philippe Cudré-Mauroux. Relation extraction using distant supervision: A survey. *ACM Comput. Surv.*, 51(5), November 2018.

[97] Sagar Sudhakara, Dhruva Kartik, Rahul Jain, and Ashutosh Nayyar. Optimal communication and control strategies in a cooperative multi-agent mdp problem. *IEEE Transactions on Automatic Control*, pages 1–8, 2024.

[98] Elina Suslova and Pooyan Fazli. Multi-robot task allocation with time window and ordering constraints. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6909–6916, 2020.

[99] Avraam Th. Tolmidis and Loukas Petrou. Multi-objective optimization for dynamic task allocation in a multi-robot system. *Engineering Applications of Artificial Intelligence*, 26(5):1458–1468, 2013.

[100] Jana Tumova and Dimos V. Dimarogonas. Multi-agent planning under local ltl specifications and event-based synchronization. *Automatica*, 70:239–248, 2016.

[101] Alphan Ulusoy, Stephen L. Smith, Xu Chu Ding, and Calin A. Belta. Robust multi-robot optimal path planning with temporal logic constraints. *2012 IEEE International Conference on Robotics and Automation*, pages 4693–4698, 2012.

[102] Matti Vahs, Christian Pek, and Jana Tumova. Risk-aware spatio-temporal logic planning in gaussian belief spaces. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7879–7885, 2023.

[103] Michel Vasquez and Jin-Kao Hao. A "logic-constrained" knapsack formulation and a tabu algorithm for the daily photograph scheduling of an earth observation satellite. *Computational Optimization and Applications*, 20, 03 1999.

[104] Christos K. Verginis, Yiannis Kantaros, and Dimos V. Dimarogonas. Planning and control of multi-robot-object systems under temporal logic tasks and uncertain dynamics. *Robotics and Autonomous Systems*, 174:104646, 2024.

[105] Hanlin Wang and Michael Rubenstein. Shape formation in homogeneous swarms using local task swapping. *IEEE Transactions on Robotics*, 36(3):597–612, 2020.

[106] Jin Wang, Yingfeng Zhang, Yang Liu, and Naiqi Wu. Multiagent and bargaining-game-based real-time scheduling for internet of things-enabled flexible job shop. *IEEE Internet of Things Journal*, 6(2):2518–2531, 2019.

[107] Quan Wang, Bin Wang, and Li Guo. Knowledge base completion using embeddings and rules. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[108] W. Wang, J. Jiang, B. An, Y. Jiang, and B. Chen. Toward efficient team formation for crowdsourcing in noncooperative social networks. *IEEE Transactions on Cybernetics*, 47(12):4208–4222, 2017.

[109] Xiaofeng Wang and Tuomas Sandholm. Reinforcement learning to play an optimal nash equilibrium in team markov games. In *Proceedings of the 15th International Conference on Neural Information Processing Systems*, NIPS'02, page 1603–1610, Cambridge, MA, USA, 2002. MIT Press.

[110] William Wolfe and Stephen Sorensen. Three scheduling algorithms applied to the earth observing systems domain. *Management Science*, 46:148–166, 01 2000.

[111] T. N. Wong, C. W. Leung, K. L. Mak, and R. Y. K. Fung. Integrated process planning and scheduling/rescheduling—an agent-based approach. *International Journal of Production Research*, 44(18-19):3627–3655, 2006.

[112] Robert Wright. Modvolc: 14 years of autonomous observations of effusive volcanism from space. *Geological Society, London, Special Publications*, 426, 08 2015.

[113] Weinan Wu, Xiaogang Wang, and Naigang Cui. Fast and coupled solution for cooperative mission planning of multiple heterogeneous unmanned aerial vehicles. *Aerospace Science and Technology*, 79:131–144, 2018.

[114] Bing Xie, Shaofei Chen, Jing Chen, and LinCheng Shen. A mutual-selecting market-based mechanism for dynamic coalition formation. *International Journal of Advanced Robotic Systems*, 15(1):1729881418755840, 2018.

[115] Bo Xu, Zhaofeng Yang, Yu Ge, and Zhiping Peng. Coalition formation in multi-agent systems based on improved particle swarm optimization algorithm. *International Journal of Hybrid Information Technology*, 8:1–8, 03 2015.

[116] Yang Xu, Paul Scerri, Bin Yu, Steven Okamoto, Michael Lewis, and Katia Sycara. An integrated token-based algorithm for scalable coordination. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '05, page 407–414, New York, NY, USA, 2005. Association for Computing Machinery.

[117] Vikas Yadav and Steven Bethard. A survey on recent advances in named entity recognition from deep learning models, 2019.

[118] Yongming Yang, Changjiu Zhou, and Yantao Tian. Swarm robots task allocation based on response threshold model. In *2009 4th International Conference on Autonomous Robots and Agents*, pages 171–176, 2009.

[119] Yuanjiang Yang, Xiang Yin, and Shaoyuan Li. A distributed framework for multi-robot task planning with temporal logic specifications. In *2020 IEEE 16th International Conference on Control & Automation (ICCA)*, pages 570–575, 2020.

[120] Yuanxi Yang and Weiguang Gao. An optimal adaptive kalman filter. *Journal of Geodesy*, 80(4):177–183, 2006.

[121] Boyan Yordanov, Jana Tumova, Ivana Cerna, Jiří Barnat, and Calin Belta. Temporal logic control of discrete-time piecewise affine systems. *IEEE Transactions on Automatic Control*, 57(6):1491–1504, 2012.

[122] Ziyang Zhen, Liangdong Wen, Bolan Wang, Zhou Hu, and Danmeng Zhang. Improved contract network protocol algorithm based cooperative target allocation of heterogeneous uav swarm. *Aerospace Science and Technology*, 119:107054, 2021.

[123] Ziyi Zhou, Dong Jae Lee, Yuki Yoshinaga, Stephen Balakirsky, Dejun Guo, and Ye Zhao. Reactive task allocation and planning for quadrupedal and wheeled robot teaming. In *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*, page 2110–2117. IEEE Press, 2022.