Fast Decomposition of Temporal Logic Specifications for Heterogeneous Teams

Kevin Leahy[1], Austin Jones[1], and Cristian-Ioan Vasile[2]

***Abstract*—We focus on decomposing large multi-agent path planning problems with global temporal logic goals (common to all agents) into smaller sub-problems that can be solved and executed independently. Crucially, the sub-problems' solutions must jointly satisfy the common global mission specification. The agents' missions are given as Capability Temporal Logic (CaTL) formulas, a fragment of Signal Temporal Logic (STL) that can express properties over tasks involving multiple agent capabilities (i.e., different combinations of sensors, effectors, and dynamics) under strict timing constraints. We jointly decompose both the temporal logic specification and the team of agents, using a satisfiability modulo theories (SMT) approach and heuristics for handling temporal operators. The output of the SMT is then distributed to subteams and leads to a significant speed up in planning time compared to planning for the entire team and specification. We include computational results to evaluate the efficiency of our solution, as well as the trade-offs introduced by the conservative nature of the SMT encoding and heuristics.**

***Index Terms*—Formal Methods in Robotics and Automation, Multi-Robot Systems**

I. Introduction

MANY potential applications of robotics involve multiple agents with different capabilities working together to achieve common goals, such as aerial surveillance drones working with ground-based manipulators for disaster response. Temporal logics are increasingly used as a specification language for tasking such large heterogeneous teams [1], [2], [3], [4]. Decomposing the problem into sub-problems for parallel execution is attractive for improving computation time for solving such problems, and in some cases may reduce the need for communication and coordination during mission execution. This approach carries trade-offs. There is generally a large up-front computational cost to determine a feasible decomposition of the task and team, in exchange for faster planning. There may also be a trade-off in plan optimality. In this work, we introduce a system for quickly decomposing temporal logic specifications to allow large teams of heterogeneous agents to plan in near real-time. This framework serves for a large class of temporal logic planning problems, including those with heterogeneous teams, task interdependencies and strict timing requirements.

This work builds upon [1], which introduced Capability Temporal Logic (CaTL), a fragment of Signal Temporal Logic (STL) designed for tasking large teams of heterogeneous agents, each with varying capability to service requests. CaTL can specify tasks to be completed by the team, without specifying which agent should complete which task. More than one agent (or group of agents) may be able to accomplish the same task. In [1], a centralized mixed integer linear program (MILP) generates a plan for the entire team simultaneously from a given CaTL specification, but scales poorly with specification length. Indeed, planning problems of this type are known to be NP-complete [5]. We seek to reduce the computation time by decomposing the specification and team of agents into sub-specifications and subteams. Such a decomposition generates several smaller planning problems that can be solved in parallel, rather than one large problem. Decomposition also has potential for decentralized execution of the specification, by reducing online coordination.

Distributed synthesis has been well-studied in the literature [6], [7], [8]. For robotics applications, there are many works that focus on decomposition of linear temporal logic (LTL) or related untimed logics, e.g., [9], [10], [11]. The authors of [9] use a product automaton method to decompose specifications among teams of heterogeneous agents. In [10], [11], the authors decompose finite LTL specifications into *decomposition sets* that capture tasks that can be accomplished in any order without violating the specification. While these methods scale reasonably for untimed logics, they do not scale well for CaTL and other timed logics like STL, whose satisfaction depends on strict time bounds on task completion.

We perform the task assignment problem concurrently with the decomposition of the specification. Task assignment can be solved in a variety of manners, including auction and optimization-based methods [12], [13], [14]. Auctions and market-based solutions provide fast agent-to-task pairings, but they do not consider the interdependencies and coordination among tasks across the time horizon of a problem. They also often consider homogeneous agents that are able to service any task [13]. In our work, agents may only service certain tasks, multiple agents may be required to service a single task, and each task might be serviced by different classes of agents. We jointly consider the requirements of the specification and the capabilities of each agent when assigning agents to tasks.

Manuscript received: September 9, 2021; Revised December 11, 2021; Accepted January 5, 2022.

This paper was recommended for publication by Editor Lucia Pallottino upon evaluation of the Associate Editor and Reviewers' comments.

DISTRIBUTION STATEMENT A. Approved for public release. Distribution is unlimited. This material is based upon work supported by the Under Secretary of Defense for Research and Engineering under Air Force Contract No. FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Under Secretary of Defense for Research and Engineering.

[1]Kevin Leahy and Austin Jones are with Massachusetts Institute of Technology Lincoln Laboratory, Lexington, MA 02421, USA kevin.leahy@ll.mit.edu

[2]Cristian-Ioan Vasile is with Dept. of Mechanical Engineering and Mechanics, Lehigh University, Bethlehem, PA 18015, USA cvasile@lehigh.edu

Digital Object Identifier (DOI): see top of this page.

Our contributions are as follows: 1) we present a computational framework for simultaneous decomposition and assignment, splitting a team of agents and a specification into corresponding subteams and sub-specifications that can be solved and executed in parallel; 2) we provide a satisfiability modulo theories (SMT) approach together with a MILP that implements this framework; and 3) we perform extensive simulations that demonstrate our method and characterize its performance on problems of different scales. We are motivated by robotics applications with short planning times relative to the complexity of the planning problem. Therefore, we trade completeness in exchange for a fast decomposition process, while still ensuring any solution to the decomposed problem satisfies the original problem.

We present models and preliminaries in Sec. II and formalize our problem in Sec. III. The specification and agent assignment are encoded in an SMT problem [15] (Sec. V). The assignment is used to transform a syntax tree representation of the specification, from which a set of sub-specifications and subteams is extracted (Sec. VI). The sub-specification, subteam pairs can then be solved in parallel as MILPs using the methods presented in [1]. Because the MILP problem is NP-complete, we seek a fast process for decomposition to improve the tractability of the MILP. Our approach is conservative, using heuristics for temporal operators. The heuristics encode *sufficient* conditions for satisfying the original formula, but may exclude some solutions, resulting in an infeasible MILP. In that sense, out method is conservative with respect to *completeness*. In the event of infeasibility, feedback from the MILP to the decomposition process may be necessary to achieve a feasible decomposition. Here, we focus on encoding and efficiently solving the decomposition problem. This approach allows us to focus on increasing the tractability of the planning problem, while minimizing the computational overhead that we add, and we leave the question of efficient feedback from the MILP as future work.

II. Models and Specifications

We consider a team of agents operating in an environment consisting of a finite set of discrete locations (states) Q and weighted edges $\mathcal{E}$ between states, where the weights represent positive integer travel times. The states are labeled with atomic propositions from a set AP. We denote the labeling function by $L : Q \to 2^{AP}$, with inverse $L^{-1}(\pi)$ returning the set of states labeled with $\pi \in AP$.

We denote the finite set of all agents by J and the finite set of all agent capabilities[1] by Cap. An agent is a tuple $A_j = (q_{0,j}, Cap_j)$, where $q_{0,j} \in Q$ is the initial state of the agent and $Cap_j \subseteq Cap$ is its set of capabilities. As agent $j \in J$ moves about its environment, it induces a discrete-time trajectory $s_j : \mathbb{Z}^{\geq 0} \to Q \cup \mathcal{E}$ such that $s_j(t)$ returns the state or edge occupied by agent j at time t. Agents traverse edges according to their weights. For example, for an agent j in state q at time t entering edge $e = (q, q')$ with weight W, $s_j(t') = e \ \forall t' \in (t, t + W)$ and $s_j(t + W) = q'$. The team trajectory of a set of agents J, denoted by s_J, is the concatenation of the individual trajectories in J, such that $s_J(t) = [s_1(t), s_2(t), s_3(t), \ldots]$, where t is a global time index. We denote the number of agents with capability $c \in Cap$ at state $q \in Q$ at time $t \in \mathbb{Z}^{\geq 0}$ by $n_{q,c}(t) = \sum_{j \in J} I(s_j(t) = q) I(c \in Cap_j)$, where I is the indicator.

[1]In our examples, capabilities correspond to sensing modalities, but the set Cap can in general capture any important characteristics of the agents.

Example 1. *Fig. 1 shows a large farm that grows crops in different regions. Each color corresponds to a different type of crop from the set* $AP = \{\text{green}, \text{yellow}, \text{orange}, \text{blue}\}$. *Red regions are obstacles. A fleet of robots with different sensing modalities must monitor crop health. The fleet as a whole has visual (*Vis*), infrared (*IR*), ultraviolet (*UV*), and soil moisture (*Mo*) sensors. Each of the crops in the field has distinct monitoring requirements. List 1 shows the tasks that the team of robots must perform during a 24 hour deployment. For the tasks in List 1,* $Cap = \{Vis, IR, UV, Mo\}$, *corresponding to the types of sensing required. Individual agents have subsets of those capabilities, for example, agent 1 may have* IR *and* UV *capabilities* ($Cap_1 = \{IR, UV\}$)*, and agent 2 may have* UV *and* Vis *capabilities* ($Cap_2 = \{UV, Vis\}$)*.*

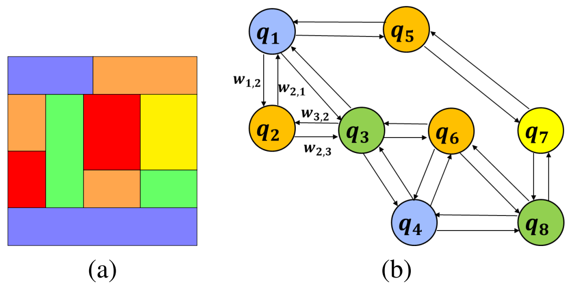


Fig. 1: (a) Schematic of the precision agriculture problem from Example 1. Colors of regions correspond to crop types, and red indicates obstacles. (b) Associated Environment with colors indicating region labels.

1) Within 10 hours of deployment, two visual and two IR sensors must remain within each green crop region at the same time for at least 1/2 hour
2) Half hour soil moisture readings must be made in each of the blue crop regions every 10 hours
3) Within 4 and 12 hours after deployment, two UV and two visual sensors must be in the yellow crop region for a half hour
4) Within 1 and 9 hours of deployment and within 10 and 15 hours of deployment, two visual sensors must be in each of the orange regions of the environment and remain there for 1 hour

LIST 1: Example list of precision agriculture tasks

The team of agents is tasked with a specification given as a CaTL formula, a fragment of STL [16], in which the core units are *tasks* rather than arbitrary predicates. Here, we define the syntax and semantics of CaTL.

Definition 1. *A* counting proposition *is a tuple* $cp = (c, m) \in Cap \times \mathbb{Z}^{\geq 0}$, *specifying the number of a given capability required to accomplish a task.*

Definition 2. *A* task *is a tuple* $T = (d, \pi, cp_T)$, *where* $d \in \mathbb{Z}^{\geq 1}$ *is a duration of time,* $\pi \in AP$ *is an atomic proposition labeling* *each region in which the task should be satisfied, and* cp_T *is a set of counting propositions* $\{cp_i\}_{i \in I_T}$ *with index set* I_T. $(c_i, m_i) \in cp_i$ *denotes that at least* m_i *agents with capability* $c_i \in Cap$ *are required for* T.

Definition 3. *The* syntax *of CaTL [1] is*

$$\phi := T \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \phi_1 \mathcal{U}_{[a,b)} \phi_2 \mid \Diamond_{[a,b)} \phi \mid \Box_{[a,b)} \phi$$

where ϕ *is a CaTL formula,* T *is a task,* $\wedge$ *and* $\vee$ *are the Boolean conjunction and disjunction operators,* $\mathcal{U}_{[a,b)}$, $\Diamond_{[a,b)}$, *and* $\Box_{[a,b)}$ *are the time-bounded until, eventually, and always operators, respectively.*

Definition 4. *The* qualitative semantics *of CaTL are defined over team trajectories* s_J. *At time* t,

$$\begin{aligned}
(s_J, t) \models T &\Leftrightarrow \forall t' \in [t, t+d), \forall q \in L^{-1}(\pi), \\
&\quad\ \forall cp_i \in cp_T,\ n_{q,c_i}(\tau) \geq m_i \\
(s_J, t) \models \phi_1 \wedge \phi_2 &\Leftrightarrow (s_J, t) \models \phi_1 \text{ and } (s_J, t) \models \phi_2 \\
(s_J, t) \models \phi_1 \vee \phi_2 &\Leftrightarrow (s_J, t) \models \phi_1 \text{ or } (s_J, t) \models \phi_2 \\
(s_J, t) \models \phi_1 \mathcal{U}_{[a,b)} \phi_2 &\Leftrightarrow \exists t' \in [t+a, t+b) (s, t') \models \phi_2 \\
&\quad\ \text{and } \forall t'' \in [t, t') s(t'') \models \phi_1 \\
(s_J, t) \models \Diamond_{[a,b)} \phi &\Leftrightarrow \exists t' \in [t+a, t+b) (s, t') \models \phi \\
(s_J, t) \models \Box_{[a,b)} \phi &\Leftrightarrow \forall t' \in [t+a, t+b) (s, t') \models \phi,
\end{aligned} \tag{1}$$

A team trajectory satisfies a CaTL formula ϕ, *denoted* $s_J \models \phi$, *if* $(s_J, 0) \models \phi$.

Note that a task $T = (d, \pi, cp_T)$ is semantically equivalent to the STL formula $\phi_T = \Box_{[0,d)} \bigwedge_{q \in L^{-1}\pi} \bigwedge_{cp_i \in Cap} n_{q,c_i} \geq m_i)$, [1]. CaTL allows intuitive, compact encodings of tasks for an operator deploying a team of robots, and like STL, can be efficiently encoded in a MILP.

Remark 1. *CaTL, unlike STL, does not include negation. Negating tasks does not have a single well-defined meaning (i.e., tasks can be false if there are not enough agents in a given region or if they are not present for a long enough duration). However, CaTL formulas are in positive normal form [1], [17], enabling efficient MILP encoding.*

Example 2. *The CaTL formula for List 1 is written*

$$\begin{aligned}
&\Diamond_{[0,10)} T(0.5, green, \{(Vis, 2), (IR, 2)\}) \\
&\wedge \Box_{[0,K)} \Diamond_{[0,10)} T(0.5, blue, \{(Mo, 1)\}) \\
&\wedge \Diamond_{[4,12)} T(0.5, yellow, \{(UV, 2), (Vis, 2)\}) \\
&\wedge \Diamond_{[1,9)} T(1, orange, (Vis, 2)) \\
&\wedge \Diamond_{[10,15)} T(1, orange, (Vis, 2)) \ ,
\end{aligned} \tag{2}$$

where K *is the planning horizon of the entire mission*

Definition 5 (Quantitative Semantics (Availability Robustness))**.** *Availability robustness of a task is computed as*

$$\rho_a(s_J, t, T) = \min_{cp_i \in cp_T} \min_{t' \in [t, t+d)} \min_{q \in L^{-1}(\pi)} n_{q,c}(t') - cp(c) \tag{3}$$

while for the other operators it is computed recursively as for STL [18].

The availability robustness of a task is the minimum difference between the available and required number of agents with a given capability considered over all capabilities and locations, and entire task duration. In other words, it is the maximum number of agents that can arbitrarily fail while still guaranteeing satisfaction of the task. Availability robustness is a measure of robustness that is therefore semantically meaningful, which is not necessarily true of STL in general.

III. Problem Statement

We wish to find a set of sub-missions ϕ_r and associated teams J_r to perform them, such that the satisfaction of the entire mission is guaranteed by the distributed and independent satisfaction of all ϕ_r, and whose solution can be computed more quickly than finding a trajectory that solves ϕ for the entire team. In other words, we would like an offline pre-processing framework for converting a planning problem into several smaller planning problems that can be solved and executed in parallel. Denote by $\mathrm{Synth}(J, \phi)$ a synthesis method that returns trajectories s_j for all $j \in J$ such that the team trajectory s_J satisfies ϕ[2]. We assume that there exists a solution to the problem for the entire team, i.e., $s_J^* = \mathrm{Synth}(J, \phi)$. We are now ready to formally state the decomposition problem.

Problem 1. *Assuming the synthesis problem is feasible, i.e.,* $\exists s_J^* \models \phi$, *given a set of agents* $\{A_j\}_{j \in J}$ *and a CaTL formula* ϕ, *find a team partition* R[3], *and a set of formulas* $\{\phi_r\}_{r \in R}$ *such that the original formula* ϕ *is satisfied if each subteam* r *satisfies its specification* ϕ_r.

Problem 1 seeks to find a partition of agents into disjoint subteams, each with their associated sub-specification, such that the original specification is satisfied if each subteam satisfies their sub-specification. No coordination is required between subteams. To solve Problem 1, we use syntax trees corresponding to CaTL formulas (Sec. IV). We use SMT to find an assignment α of agents to tasks that is amenable to decomposition (Sec. V), and then demonstrate options for decomposing a specification according to its assignment (Sec. VI). The resulting decomposed specifications and teams can each be handled concurrently using Synth. Because the synthesis problem for each subteam and sub-specification is smaller than the original problem, we expect that solving them concurrently will be faster than solving the original problem. However, we don't know ahead of time if the synthesis problem is feasible for each subteam and sub-specification, even if the original problem is feasible. This infeasibility can arise due to the spatial configuration of the agents and environment, if for example, agents in a subteam are unable to travel between assigned regions in adequate time. Thus, we gain a computational speedup at the price of potential infeasibility. We note that the potential for infeasibility exists in the original problem as well, and it is difficult to determine if a problem is infeasible without solving the planning problem (whether or not the proposed decomposition process is used), which is NP-complete [5].

The decomposition process is outlined in Alg. 1.

[2]There are many choices for the synthesis method Synth [19], [20]. We employ a MILP approach similar to [1].

[3]Here, we use the typical definition of the partition of a set: $\cup_{r \in R} J_r = J$ and $J_{r_1} \cap J_{r_2} = \emptyset \, \forall r_1, r_2 \in R$.

Algorithm 1: Solution overview.

```
Input: Agents $\{A_j\}_{j\in J}$, CaTL formula $\phi$
Output: Team partition $\{J_r\}_{r\in R}$, Formulas $\{\phi_r\}_{r\in R}$
1 $\alpha \leftarrow$ SolveSMT($\{A_j\}_{j\in J}, \phi$)            // Alg. 2
2 $\{\phi_r, J_r\}_{r\in R} \leftarrow$ DecomposeTree($\phi, \alpha$)   // Alg. 3
3 return $\{J_r\}_{r\in R}$, $\{\phi_r\}_{r\in R}$
```

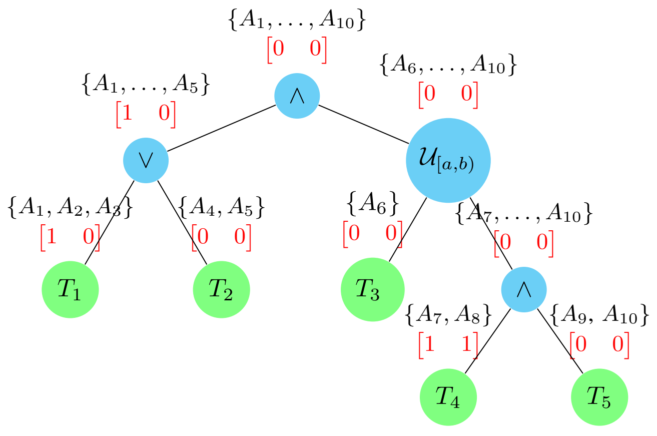


Fig. 2: Abstract syntax tree for (4) with assignment in black and capability excess in red.

IV. Syntax Trees and Agent Assignments

We use syntax trees to reason about CaTL specifications, their decomposition, and the assignment of agents to tasks.

Definition 6. *A* CaTL syntax tree *of a CaTL formula ϕ is a tuple $\mathcal{T}_\phi = (V, v_0, Par)$, where V is the set of nodes associated with the operators and tasks of ϕ, v_0 is a root node, $Par : (V \setminus \{v_0\}) \to V \cup \{\bowtie\}$ maps each node to its parent in the tree, and $Par(v_0) = \bowtie$ denotes that the root has no parent. Each node is labeled from the set of operators $\{\wedge, \vee, \mathcal{U}_{[a,b)}, \Diamond_{[a,b)}, \Box_{[a,b)}, T = (d, \pi, cp)\}$. For brevity, we will refer to nodes by their operator. For example, if the root node v_0 is labeled $\wedge$, we will write $v_0 = \wedge$. For nodes v_1, $v_2 \in V$, we say that v_1 is* upstream *of v_2 (and v_2 is* downstream *of v_1) if v_1 is on the path from the root node to v_2.*

Let $Ch(v)$ denote the set of all children of a node $v \in V$, and $\Lambda \subset V$ the set of leaf nodes, i.e., nodes without children $Ch(v) = \emptyset$. The leaf nodes exactly correspond to the tasks of CaTL formulas.

Example 3. *Fig. 2 shows the syntax tree for the formula*

$$\psi = (T_1 \vee T_2) \wedge ((T_3)\,\mathcal{U}_{[a,b)}(T_4 \wedge T_5))) \,. \tag{4}$$

Definition 7. *An* assignment *of agents $\{A_j\}_{j\in J}$ in a CaTL syntax tree $\mathcal{T}_\phi$ is a mapping $\alpha : \Lambda \to 2^J$.*

An assignment α keeps track of agents assigned to tasks. Agents are assigned to other nodes in the tree according to the formula structure, such that agents assigned to an intermediate node v must be assigned to some child node of v. Thus, α is completely determined by the assignment over the leaves Λ

$$\alpha(v) = \bigcup_{v' \in Ch(v)} \alpha(v') \;\forall v \notin \Lambda \,. \tag{5}$$

Agent	c_1	c_2	Agent	c_1	c_2	Task	$cp(c_1)$	$cp(c_2)$
A_1	1	1	A_6	0	1	T_1	2	0
A_2	1	0	A_7	1	1	T_2	2	2
A_3	1	0	A_8	1	1	T_3	0	1
A_4	1	1	A_9	1	0	T_4	1	1
A_5	1	1	A_{10}	0	1	T_5	1	1

TABLE I: Numbers and requirements of capabilities for (4).

We note that an agent may be assigned to multiple tasks, and a task may have no agents assigned to it, but we require all agents to be assigned to at least one task.

We further define the notion of *capability excess* to aid in evaluating the assignment of agents to tasks.

Definition 8. *The* capability excess *of an assignment α to a node $v \in V$ is defined recursively as*

$$ce(\alpha, v) = \begin{cases} \min\limits_{cp_i \in cp_T} \frac{na_{c_i} - m_i}{|L^{-1}(\pi)|} & v = T \\ \max\limits_{v' \in Ch(v)} ce(\alpha, v') & v = \vee \\ \min\limits_{v' \in Ch(v)} ce(\alpha, v') & v \in \{\wedge, \mathcal{U}_{[a,b)}\}^4 \\ ce(\alpha, Ch(v)) & v \in \{\Box_{[a,b)}, \Diamond_{[a,b)}\}, \end{cases} \tag{6}$$

where $na_{c_i} = |\{j \in \alpha(v) \mid c_i \in Cap_j\}|$ is the number of agents with capability c_i assigned to node v.

Intuitively, an assignment's capability excess measures how many extra agents with a given capability are assigned to a node. A positive capability excess indicates that there are more than enough agents assigned to a node. A negative capability excess indicates that there are not enough agents assigned to the node. Capability excess of zero indicates that there are exactly as many agents assigned as needed. Our solution focuses on finding assignments that are eligible at the task level. Therefore, each task appearing in the specification should have at least as many agents as necessary to accomplish the task. The recursive Def. 6 allows us to check the capability excess for the entire specification by inspecting the root only. This property is formalized as eligibility in Def. 9 and shown to be necessary for satisfaction of the specification in Prop. 1.

Example 4. *Consider formula (4), with tasks involving two capabilities $Cap = \{c_1, c_2\}$. The capabilities assigned to each agent, and the ones required by each task are listed in Table I. An assignment of agents to tasks and the resulting capability excess are illustrated in Fig. 2.*

Definition 9. *An assignment α is* eligible *for the CaTL syntax tree $\mathcal{T}_\phi$, denoted $\alpha \models_e \mathcal{T}_\phi$, if*

$$ce(\alpha, v_0) \geq 0 \tag{7}$$

Proposition 1. *For a given team trajectory s_J, let $\alpha[s_J]$ denote the induced assignment such that $A_j \in \alpha(\lambda)$ if A_j participates in task $\lambda \in \Lambda$. It holds that*

$$s_J \models \phi \Rightarrow \alpha[s_J] \models_e \mathcal{T}_\phi \tag{8}$$

That is, eligibility is a necessary condition for satisfiability.

The proof of Prop. 1 follows from a consideration of capability excess. It is omitted due to space considerations.

[4]For $\phi_1 \mathcal{U}_{[a,b)} \phi_2$, we assume $a > 0$. Otherwise, we substitute $\phi_1 \mathcal{U}_{[0,b)} \phi_2$ with $\phi_1 \mathcal{U}_{[0,b)} \phi_2 \vee \phi_2$ to preserve the correctness of the ce computation.

We provide a brief sketch here. If a team trajectory satisfies ϕ, then it directly follows from the quantitative semantics of tasks (Def. 5) that $ce(\alpha[s_J], T) \geq 0$ for the tasks that are satisfied. Therefore $ce(\alpha[s_J], T) \geq 0 \implies \rho_a(s_J, 0, T) \geq 0$ for each task. By comparing the quantitative semantics of STL [16] or CaTL [1] with capability excess for the rest of the operators, it follows that $ce(\alpha[s_J], \phi) \geq 0$.

V. Agent Assignments as SMT

Now, we consider the problem of finding an assignment α for a given CaTL formula ϕ such that it can be decomposed. For each node in the syntax tree $\mathcal{T}$, we encode whether its assignment is eligible according to Def. 9, and whether the assignments to its children are independent (i.e., non-intersecting). We encode the problem using an SMT solver, allowing us to reason over agents individually. We pair agents to tasks and efficiently compare the effect of the assignment on our ability to decompose the formula.

We encode the Boolean variable $Ind(\cdot)$ as presented in Alg. 2 to capture independence under an assignment α as described above. The variable $Ind(\alpha, v)$ captures whether the assignment of agents to subtrees rooted at v are non-intersecting, i.e., $Ind(\alpha, v) = \texttt{True}$ if $\alpha(v') \cap \alpha(v'') = \emptyset$ for all v', $v'' \in Ch(v)$.

Alg. 2 presents the encoding of the SMT problem. For leaves of the $\mathcal{T}$, ce is computed based on the assignment, and Ind is assigned a value of `True` (lines 2-4). The values of ce and Ind for each remaining node in the syntax tree are determined by that node's children. Thus, the assignment of agents to tasks determines the values of ce and Ind for all of the other nodes in the tree. For the temporal operators eventually ($\Diamond_{[a,b)}$) and always ($\Box_{[a,b)}$), which each only have one child node, these properties are inherited directly (lines 11-12). For disjunction ($\vee$), only one downstream branch need be satisfying, so the disjunction in line 6 selects a satisfying branch and forces the other to have an empty assignment. The node inherits ce and Ind from the satisfying branch accordingly (lines 6-9). The other branch of the disjunction is forced to have an empty assignment to reduce the likelihood of agents being assigned to a branch that is not necessary for satisfaction. Finally, for until ($\mathcal{U}_{[a,b)}$) and conjunction ($\wedge$), the eligibility is the conjunction of the children, and independence is according to the pairwise independence of children (lines 14-18).

Top level SMT problem We solve whether

$$\psi_1 \wedge \psi_2 \wedge \psi_3 \tag{9}$$

is satisfiable, where

$$\psi_1 = ce(\alpha, v_0) \geq 0 \tag{10}$$

$$\psi_2 = Ind(\alpha, v_0) \tag{11}$$

$$\psi_3 = \bigvee_{\lambda \in \Lambda} A_j \in \alpha(\lambda),\ \forall j \in J\ , \tag{12}$$

Eq. (10) ensures an eligible assignment in all downstream nodes, a necessary condition for satisfiability via Prop. 1. Eq. 11 ensures that the formula can be split into more than one formula. All agents are assigned to at least one task via (12), because unassigned agents cannot contribute to satisfaction of a CaTL formula.

Algorithm 2: SMT encoding of assignment problem

```
Input: Syntax Tree $\mathcal{T}_\phi$
Output: Assignment $\alpha$
1  for $v \in V$ do
2      if $v \in \Lambda$ then
3          $ce(\alpha, v) \leftarrow \min_{cp_i \in cp_T} \frac{na_{c_i} - m_i}{|L^{-1}(\pi)|}$;
4          $Ind(\alpha, v) \leftarrow$ True;
5      else if $v = \vee$ then
6          $\alpha(v') = \emptyset \vee \alpha(v'') = \emptyset$ for $v', v'' \in Ch(v)$;
7          $ce(\alpha, v) \leftarrow \max_{v' \in Ch(v)} ce(\alpha, v')$;
8          $v^* \leftarrow v' \in Ch(v)$ s.t. $ce(\alpha, v') \geq 0$;
9          $Ind(\alpha, v) \leftarrow Ind(\alpha, v^*)$;
10     else if $v \in \{\Box_{[a,b)}, \Diamond_{[a,b)}\}$ then
11         $ce(\alpha, v) \leftarrow ce(\alpha, Ch(v))$;
12         $Ind(\alpha, v) \leftarrow Ind(\alpha, Ch(v))$;
13     else if $v \in \{\mathcal{U}_{[a,b)}, \wedge\}$ then
14         $ce(\alpha, v) \leftarrow \min_{v' \in Ch(v)} ce(\alpha, v')$;
15         if $\alpha(v') \bigcap \alpha(v'') = \emptyset\ \forall v', v'' \in Ch(v)$ then
16             $Ind(\alpha, v) \leftarrow$True;
17         else
18             $Ind(\alpha, v) \leftarrow$False;
19 Find $\alpha$ via (9);
```

Remark 2. *As noted in Sec. I, our approach is conservative with respect to feasibility. Our assignment is based on capability excess, which is an upper bound on robustness [1]. Decomposition cannot increase capability excess across sub-teams, and thus can lead to infeasibility of sub-problems, and suboptimality of overall robustness. Since even the feasibility problem is hard (NP-complete), we focus on how to speed up the computation of a team motion plan when one exists, which is the prevalent case in practice.*

VI. Formula Transformations

Next, we discuss how to use an assignment α to determine a partition R of the set of agents J and formula ϕ into a set of subteams and subformulas $\{J_r, \phi_r\}_{r \in R}$. We list some results on modifications to ϕ that make it more amenable to decomposition (Sec. VI-A), and then apply the modifications to ϕ according to the assignment α (Sec. VI-B).

A. Formula transformations for decomposition

In this section, we list some transformation to CaTL formulas that can make a formula more amenable to decomposition. We note that the transformations described in this section are conservative. The satisfaction of a transformed formula implies satisfaction of the original formula. The reverse is not necessarily true. It may therefore be necessary to find a centralized solution to a formula in the event that a solution to the decomposed problem cannot be found.

The transformations we describe below are conservative and non-exhaustive, i.e., there may be other transformations that apply to this problem. Crucially, these transformations were chosen because of two properties. First, they potentially reduce the amount of coordination required. For disjunction, this means pruning one of the subformulas and only requiring that one subformula be satisfied. For until and conjunction, temporal operators are moved inside of a conjunction, allowing the formula to be split into two formulas that can each be satisfied independently. The second important property is that transformations cannot introduce new tasks. That is, for a formula ϕ transformed to ϕ', let $\mathcal{T}$, Λ and $\mathcal{T}'$, Λ' be the corresponding syntax trees and tasks, respectively. Then $\Lambda' \subseteq \Lambda$. An assignment α induces and assigment α' on $\mathcal{T}'$ such that $\alpha'(\lambda) = \alpha(\lambda)$ for all $\lambda \in \Lambda'$.

Disjunction $\vee$ Disjunction of two formulas is satisfied if at least one of its subformulas is satisfied. Therefore we may replace a disjunction of two subformulas with either subformula:

$$\phi_1 \vee \phi_2 \rightarrow \phi_i \text{ for } i = 1 \text{ or } i = 2 \tag{13}$$

Until $\mathcal{U}_{[a,b)}$ The formula $\phi_1 \mathcal{U}_{[a,b)} \phi_2$ is in general difficult to parallelize. However, we can substitute the until operator with a more conservative formula containing a conjunction whose subformulas can be independently satisfied:

$$\phi_1 \mathcal{U}_{[a,b)} \phi_2 \rightarrow \square_{[0,b)} \phi_1 \wedge \diamondsuit_{[a,b)} \phi_2 \tag{14}$$

Conjunction $\wedge$ **with upstream temporal operators** Temporal operators followed by a conjunction can be transformed to a conjunction of subformulas. This transformation is conservative for the eventually, but not for always:

$$\begin{aligned} \square_{[a,b)}(\phi_1 \wedge \phi_2) &\rightarrow \square_{[a,b)} \phi_1 \wedge \square_{[a,b)} \phi_2 \\ \diamondsuit_{[a,b)}(\phi_1 \wedge \phi_2) &\rightarrow \square_{[a,b)} \phi_1 \wedge \square_{[a,b)} \phi_2 \end{aligned} \tag{15}$$

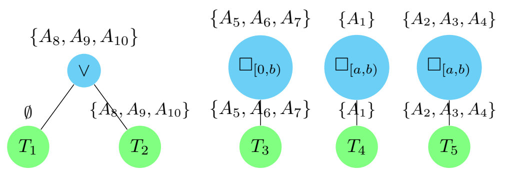
Remark 3. *Our choice of transformation for* $\diamondsuit_{[a,b)}(\phi_1 \wedge \phi_2)$ *to a conjunction of always operators (i.e.,* $\square_{[a,b)}\phi_1 \wedge \square_{[a,b)}\phi_2$*) is one of several choices that satisfy the original formula. It is also the most conservative. The formula could equivalently be transformed into* $\diamondsuit_{[a,b)}\phi_1 \wedge \square_{[a,b)}\phi_2$ *or* $\square_{[a,b)}\phi_1 \wedge \diamondsuit_{[a,b)}\phi_2$*. We choose the symmetric conjunction of always operators for simplicity of presentation.*

Proposition 2. *The transformations presented above transform the original specification* ϕ *into a new specification* ϕ'*. It follows by induction that the language of* ϕ' *is a subset of the language of* ϕ*. Therefore, for a given team trajectory,* s_J*,* $s_J \models \phi' \implies s_J \models \phi$*.*

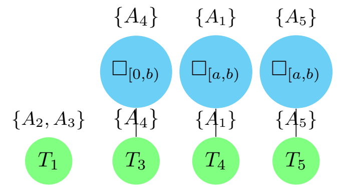
B. Decomposing Specification using Assignment and SMT

Given an assignment from the preceding SMT problem, we wish to decompose the tree as much as possible, returning a set of subformulas and subteams. Here we describe how to obtain these formulas and teams from the SMT solution. Based on the assignment results, we wish to only modify the formula as much as necessary. For example, if the same agents are assigned to both ϕ_1 and ϕ_2, then there is no need to modify the formula $\phi_1 \mathcal{U}_{[a,b)} \phi_2$.

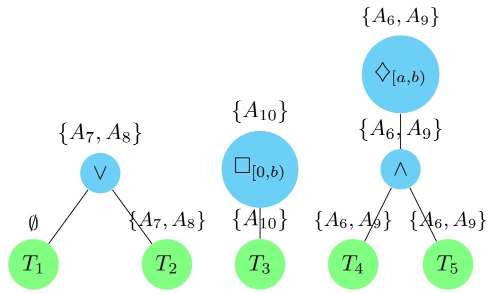
We note that any tree $\mathcal{T}$ with conjunction at the root ($v_0 = \wedge$) can be split into two trees $\mathcal{T}'$, $\mathcal{T}''$ whose formulas can be independently satisfied if the assignment of agents to $\mathcal{T}'$ and $\mathcal{T}''$ are disjoint, i.e., $Ind(\alpha, v_0) == \texttt{True}$. We will denote this



(a) Assignment and syntax trees for the entire team of agents $\{A_1, \ldots, A_{10}\}$



(b) Assignment and syntax trees for agents $\{A_1, \ldots, A_5\}$



(c) Assignment and syntax trees for agents $\{A_6, \ldots, A_{10}\}$

Fig. 3: Final syntax trees and assignments for varying subsets of agents $\{A_1, \ldots, A_{10}\}$ assigned to the initial abstract syntax tree in Fig. 2. Note that the assignment α determines the final structure of the specifications.

process of splitting a tree into its two subtrees by $\text{Split}(\mathcal{T}) = \{\mathcal{T}', \mathcal{T}''\}$.

The process is outlined in Alg. 3. Briefly, the syntax tree and assignment from the SMT are provided as input. The tree is pruned at nodes labeled $\vee$ for any children not selected in the SMT (lines 1-2). Then, subtrees are substituted at nodes labeled $\mathcal{U}_{[a,b)}$ or $\wedge$ if the assignments to their children are disjoint (lines 3-5). Finally, the tree is recursively split at the root (line 6) using the function Subtrees (lines 11-16). The resulting formulas and corresponding team assignments are extracted and returned (lines 7-10). They can then be solved in parallel as a set of MILPs using the method in [1].

Remark 4. *If the assignment is eligible but no satisfying parallel execution exists (because by Prop. 1, we find a necessary but not sufficient condition for satisfiability), we must add that information to the SMT problem so that the solver does not continue to investigate similar solutions that are unlikely to work. This can be accomplished using the irreducible inconsistent set (IIS) from the MILP solver, which is computed by most modern solvers, and provides constraints that can be used in the SMT. There are several technical issues that need to be addressed in this process, including converting MILP constraints to their corresponding SMT constraints. We leave it as future work. For now, we assume that if the decomposition solution contains an infeasible MILP, we will instead solve the centralized synthesis problem upon detecting infeasibility. Since the decomposition process is much faster than the centralized*

Algorithm 3: Decomposition using assignment α^* from SMT

```
Input: Assignment $\alpha^*$ from the SMT Problem, Syntax Tree $\mathcal{T}_\phi$
Output: Set of formulas $\{\phi_i\}_{i\in 1,\dots,N}$, Corresponding team partition $\{J_i\}_{i\in 1,\dots,N}$
1  for $v \in V | v = \vee$ do
2      transform according to (13);
3  for $v \in V | v \in \{\mathcal{U}_{[a,b)}, \wedge\}$ and $v \neq v_0$ do
4      if $Ind(\alpha, v) =$ True then
5          transform according to (14) or (15);
6  $Trees \leftarrow$ Subtrees($\mathcal{T}$);
7  for $\mathcal{T}_i \in Trees$ do
8      extract formula $\phi_i$ from subtree $\mathcal{T}_i$;
9      extract subteam $J_i$ from $\alpha_i$;
10 return $\{\phi_i\}_{i\in 1,\dots,N}$, $\{J_i\}_{i\in 1,\dots,N}$
11 Function Subtrees($\mathcal{T}$):
12     if $v_0 = \wedge$ and $Ind(\alpha, v_0) =$ True then
13         $\{\mathcal{T}', \mathcal{T}''\} =$ Split($\mathcal{T}$);
14         return { Subtrees($\mathcal{T}'$), Subtrees($\mathcal{T}''$) };
15     else
16         return $\mathcal{T}$;
```

synthesis, doing the two operations in sequence does not add much overhead in the case of an infeasible decomposition.

VII. Simulation and Results

To validate evaluate our method's performance, we performed computational experiments. The SMT problem was coded in Z3 [21]. Synthesis was performed using the Gurobi solver [22]. Experiments were run in Python 2.7 on Ubuntu 16.04 with a 2.5 GHz Intel i7 processor and 16 GB of RAM.

We evaluated the performance for 10, 20, 30, 40, and 50 agents, with increasing environment size for larger teams of agents. The environment was a grid, with edge weight of 1 and randomly assigned region labels. Timeout for the MILP was set at $120s$ and for the SMT solver at $60s$. In the following, "feasible" refers to running the MILP to obtain the first feasible solution without decomposition. "Decompose" refers to synthesis of the first feasible solution on the decomposed problem.

We performed simulations with a specification motivated by our agriculture example, with fifty runs per case. Agents were randomly generated with 1 or 2 of the following capabilities: harvest crops 1 (H_1), harvest crops 2 (H_2), water crops (W), spray pesticide (S), deter pests (D), and monitor crops (M). Here we evaluated the specification

$$\phi = \phi_1 \wedge \phi_2 \wedge \phi_3 \wedge \phi_4 \,, \tag{16}$$

where each ϕ_i is defined as

$$\phi_1 = \Diamond_{[15,25)}((T_1 \mathcal{U}_{[5,10)} T_2) \vee (T_3 \wedge T_4)) \tag{17}$$
$$\phi_2 = \Diamond_{[15,30)}(T_5 \mathcal{U}_{[5,10)} (T_6 \wedge T_7)) \tag{18}$$
$$\phi_3 = \Box_{[15,50)}(\Diamond_{[0,20)}(T_8 \wedge T_9)) \tag{19}$$
$$\phi_4 = \Diamond_{[15,30)}(T_{10} \wedge (T_{11} \vee T_{12}))\,. \tag{20}$$

TABLE II: Task definitions for (16).

Name	Task	cp_T	d	π
Spray	T_1	$\{(S,1)\}$	3	A
Harvest	T_2	$\{(H_1,1)\}$	3	B
Monitor	T_3	$\{(M,1)\}$	3	C
Spray	T_4	$\{(S,1)\}$	3	D
Spray	T_5	$\{(S,2)\}$	5	A
Wet Harvest	T_6	$\{(W,1),(H_1,1)\}$	3	B
Harvest	T_7	$\{(H_2,1)\}$	3	C
Monitor	T_8	$\{(M,1)\}$	3	D
Spray	T_9	$\{(S,2)\}$	3	A
Spray	T_{10}	$\{(S,1)\}$	3	B
Deter and Water	T_{11}	$\{(D,1),(W,1)\}$	3	C
Deter	T_{12}	$\{(D,2)\}$	3	D

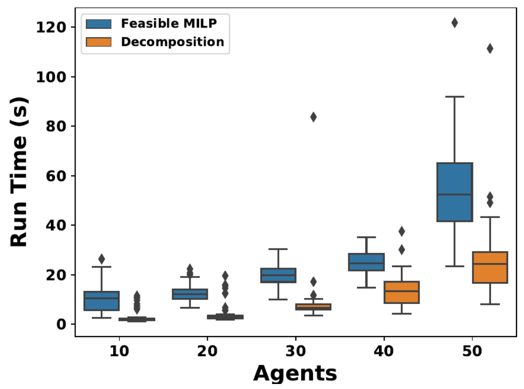


Fig. 4: Run time with and without decomposition for varying problem sizes with longer specification. Feasible MILP solutions are in blue and decomposition is in orange.

The tasks are defined in Table II.

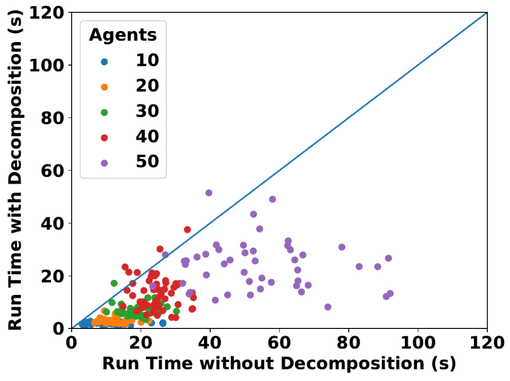
Results of the simulation are shown in Figs. 4-5b. From Fig. 4 it is clear that decomposition leads to a significant reduction in run time. Table III shows the percentage of overall runtime spent on decomposition. The majority of the run time is spent solving the MILP, while the time spent on decomposition remains fairly constant. Table IV presents statistics on the subteams and SMT performance. Some smaller problems returned UNSAT, while larger problems timed out. Since larger teams are harder for the MILP to solve, these results suggest decomposition is a tool better suited for problems that a user expects to be difficult for the MILP to solve in the first place. Fig. 5a plots run time for identical problem instances with and without decomposition. These results indicate that for nearly all individual instances, decomposition reduced run time. Finally, Fig. 5b plots the MILP size compared to the run time with and without decomposition, indicating that there is a reduction in MILP size after decomposition. Interestingly, decomposition appears to decrease the run time and the variance in run time.

TABLE III: Percentage of total runtime spent on decomposition for different numbers of agents.

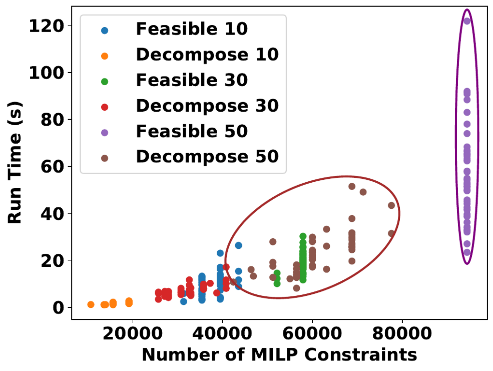
Agents	10	20	30	40	50
Percentage of Runtime	8.8%	15.2%	22.5%	7.2%	10.1%

VIII. Conclusion

We have proposed a method for the automatic decomposition of a team of agents and a formal specification into a set of



(a)



(b)

Fig. 5: (a) Run time with and without decomposition. Each point is an identical problem instance. Points below the diagonal indicate that the problem was solved faster with decomposition than without decomposition. (b) Run time compared to MILP size. Decomposition reduces variance in run time while reducing MILP size. Purple circle highlights the data for 50 agents without decomposition and the brown circle highlights the data for 50 agents with decomposition.

TABLE IV: Statistics for decomposition by numbers of agents. Number of subteams and subteam size are means.

Agents	Env. Size	Subteams	Subteam Size	SMT Timeout	UNSAT
10	5×5	5.18	3.15	0	9
20	5×5	5.40	5.33	0	6
30	6×6	5.90	5.50	1	0
40	7×7	6.00	6.67	0	0
50	8×8	5.90	9.17	1	0

subteams and sub-specifications using SMT. The decomposed problem can be solved in a distributed manner. This method significantly reduces the run time over a centralized approach. It represents a promising first step towards speeding up planning for large heterogeneous teams.

There are several directions for future work. For a decomposition assignment with no feasible MILP solution, it may be possible to use the IIS (see Remark 4) from the MILP solver in feedback with the SMT problem to remove any infeasible conditions. We would also like to investigate a decomposition that considers robustness. Such a system would likely require re-allocating agents across subteams as a post-processing step after solving the MILP. We hypothesize that any reduction in MILP problem size should improve performance. However, certain aspects of our problem, like length of subformula, lead to larger reductions in MILP size. Focusing our decomposition problem on those aspects in particular may further improve the performance.

References

[1] K. Leahy, Z. Serlin, C.-I. Vasile, A. Schoer, A. M. Jones, R. Tron, and C. Belta. Scalable and robust algorithms for task-based coordination from high-level specifications (ScRATCHeS). *IEEE Transactions on Robotics*, pages 1–20, 2021.

[2] M. Guo and D. Dimarogonas. Multi-agent plan reconfiguration under local LTL specifications. *The International Journal of Robotics Research*, 34(2):218–235, 2015.

[3] Y. Kantaros, M. Guo, and M. Zavlanos. Temporal logic task planning and intermittent connectivity control of mobile robot networks. *IEEE Transactions on Automatic Control*, 64(10):4105–4120, 2019.

[4] Y. Sahin, P. Nilsson, and N. Ozay. Multirobot coordination with counting temporal logics. *IEEE Transactions on Robotics*, 2019.

[5] C. H. Papadimitriou and K. Steiglitz. *Combinatorial optimization: algorithms and complexity*. Dover, 1998.

[6] M. Mukund. From global specifications to distributed implementations. In *Synthesis and control of discrete event systems*, pages 19–35. Springer, 2002.

[7] A. Ştefănescu, J. Esparza, and A. Muscholl. Synthesis of distributed algorithms using asynchronous automata. In *International Conference on Concurrency Theory*, pages 27–41. Springer, 2003.

[8] M. Karimadini and H. Lin. Guaranteed global performance through local coordinations. *Automatica*, 47(5):890–898, 2011.

[9] Y. Chen, X. C. Ding, A. Stefanescu, and C. Belta. Formal approach to the deployment of distributed robotic teams. *IEEE Transactions on Robotics*, 28(1):158–171, 2012.

[10] P. Schillinger, M. Bürger, and D. Dimarogonas. Decomposition of finite LTL specifications for efficient multi-agent planning. In *Distributed Autonomous Robotic Systems*, pages 253–267. Springer, 2018.

[11] C. Banks, S. Wilson, S. Coogan, and M. Egerstedt. Multi-agent task allocation using cross-entropy temporal logic optimization. In *International Conference on Robotics and Automation*, 2020.

[12] A. Khamis, A. Hussein, and A. Elmogy. Multi-robot task allocation: A review of the state-of-the-art. In *Cooperative Robots and Sensor Networks 2015*, pages 31–51. Springer, 2015.

[13] L. Luo, N. Chakraborty, and K. Sycara. Distributed algorithm design for multi-robot task assignment with deadlines for tasks. In *2013 IEEE International Conference on Robotics and Automation*, pages 3007–3013. IEEE, 2013.

[14] N. Michael, M. Zavlanos, V. Kumar, and G. J Pappas. Distributed multi-robot task assignment and formation control. In *2008 IEEE International Conference on Robotics and Automation*, pages 128–133. IEEE, 2008.

[15] C. Barrett and C. Tinelli. Satisfiability modulo theories. In *Handbook of Model Checking*, pages 305–343. Springer, 2018.

[16] O. Maler and D. Nickovic. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 152–166. Springer, 2004.

[17] S. Sadraddini. *Formal methods for resilient control*. PhD thesis, Boston University, 2018.

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

[19] H. Kress-Gazit, M. Lahijanian, and V. Raman. Synthesis for robots: Guarantees and feedback for robot behavior. *Annual Review of Control, Robotics, and Autonomous Systems*, 2018.

[20] C. Belta, B. Yordanov, and E. Gol. *Formal methods for discrete-time dynamical systems*, volume 89. Springer, 2017.

[21] L. De Moura and N. Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[22] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2021.